

PACC

Prefect Associate
Certification Course



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

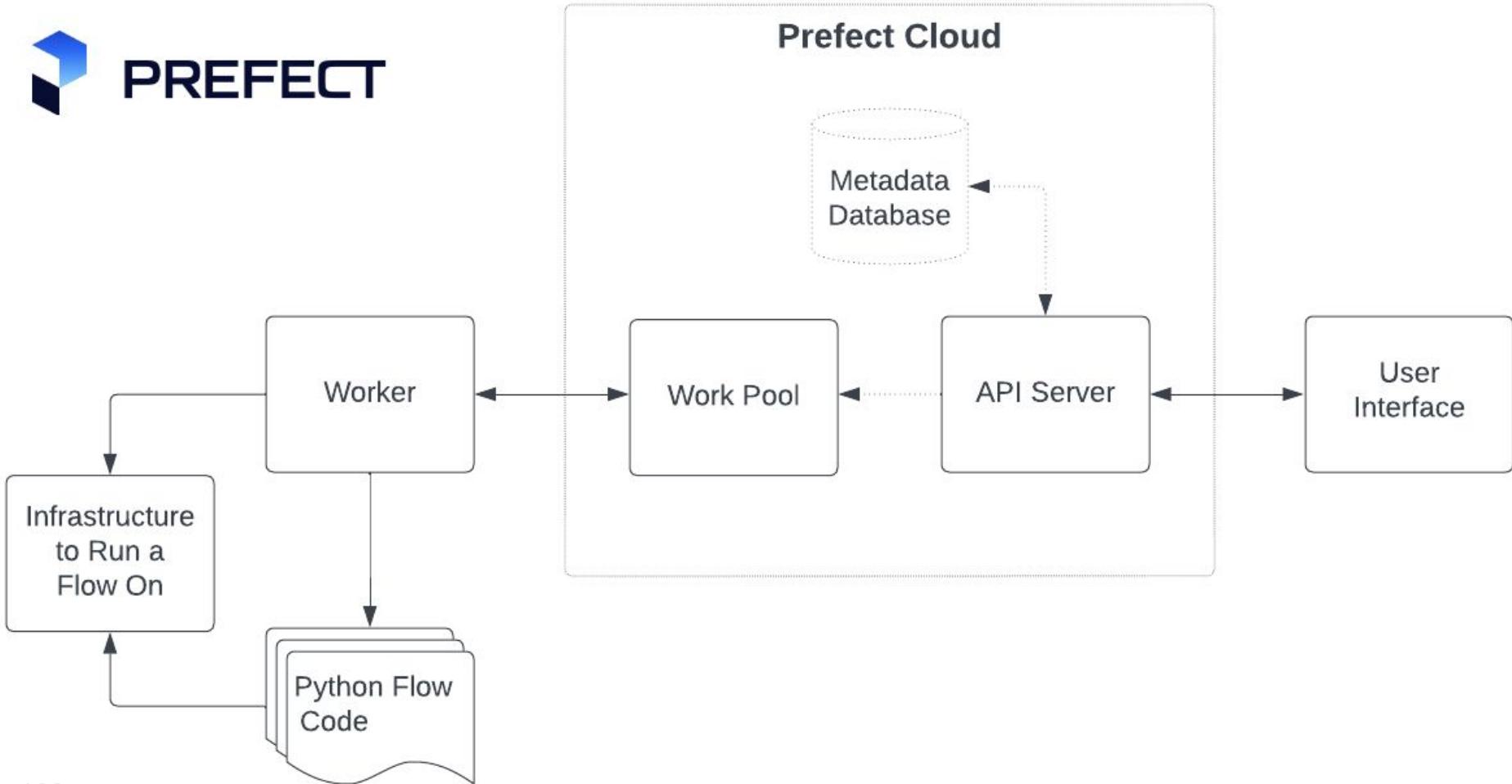


Hybrid Model





PREFECT

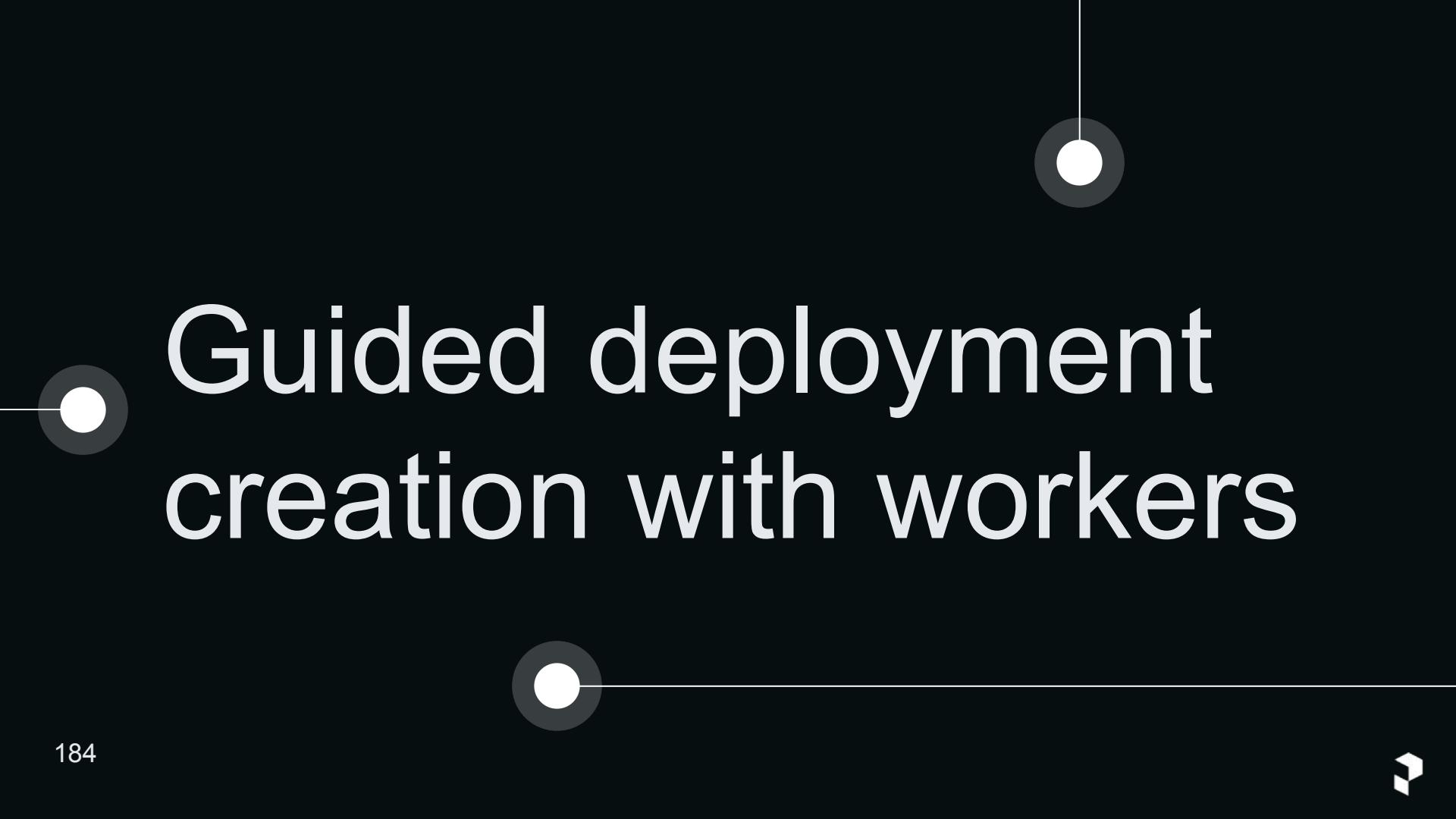


Hybrid model

- 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

```
@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 local-work my-pool prod-pool staging-pool zoompool | docker process process kubernetes kubernetes process | |



Flow code storage



Specify flow code storage



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

No auto-push.

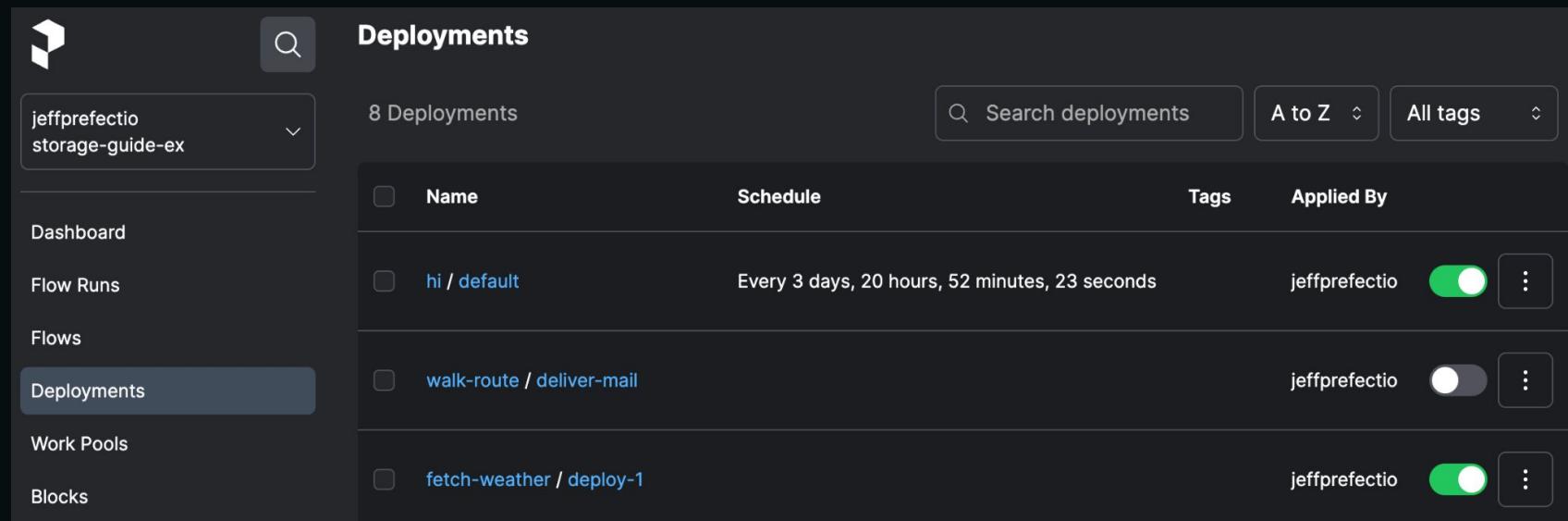
```
? Your Prefect workers will need access to this flow's code in order to run it. Would you like your workers to pull your flow code from its remote repository when running this flow? [y/n] (y): y
? Is https://github.com/discover/pacc-2023.git the correct URL to pull your flow code from? [y/n] (y):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

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



Deployments in the UI

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



The screenshot shows the Prefect UI interface with the following details:

- Left Sidebar:** Includes a logo, a search icon, and a dropdown menu set to "jeffprefectio storage-guide-ex". Below the dropdown are links: Dashboard, Flow Runs, Flows, **Deployments** (which is highlighted in blue), Work Pools, and Blocks.
- Header:** "Deployments" is displayed prominently. Below it are filters: "Search deployments" (with a magnifying glass icon), "A to Z" (with a dropdown arrow), and "All tags" (with a dropdown arrow).
- Table:** A table titled "8 Deployments" lists three entries:
 - hi / default: Schedule is "Every 3 days, 20 hours, 52 minutes, 23 seconds". Applied By: jeffprefectio (green toggle switch). More options: three-dot menu icon.
 - walk-route / deliver-mail: Applied By: jeffprefectio (gray toggle switch). More options: three-dot menu icon.
 - fetch-weather / deploy-1: Applied By: jeffprefectio (green toggle switch). More options: three-dot menu icon.

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



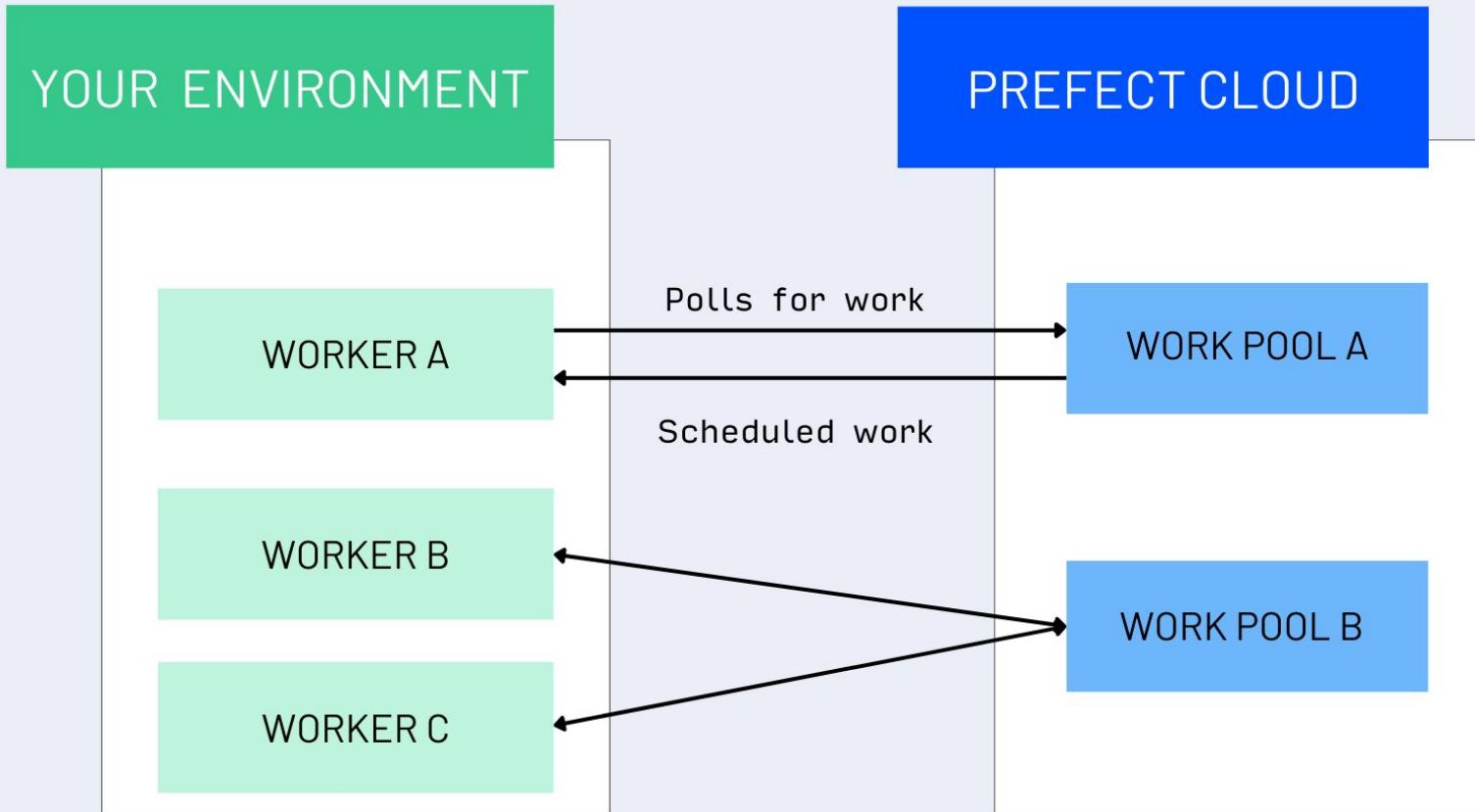
Workers

- 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

| Details | Runs | Work Queues | Workers |
|---|------------------------|-------------|-------------------------------------|
| 1 Worker | | | <input type="text"/> Search workers |
| | | | |
| Name | Last Polled | | |
| KubernetesWorker b52f4122-4ed2-4354-afa8-86b6cd1b6dde | 2023/08/25 03:24:04 PM | | <input type="button"/> |
| | | | |



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.

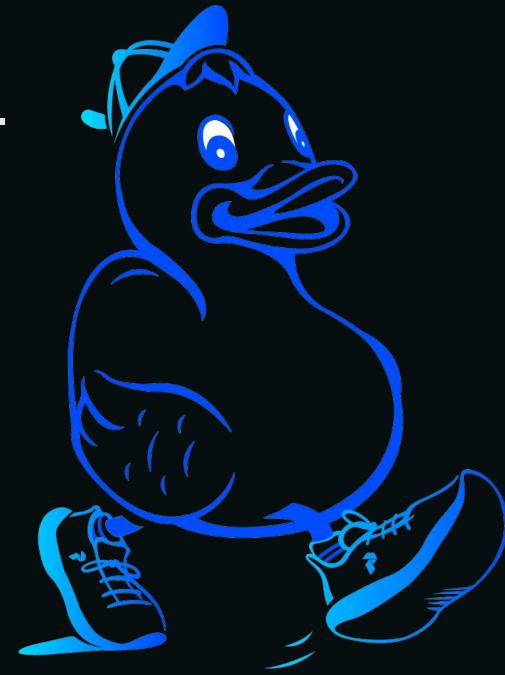


Parameters



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

Run Deployment X

Form JSON

lat

lon

Run Cancel



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

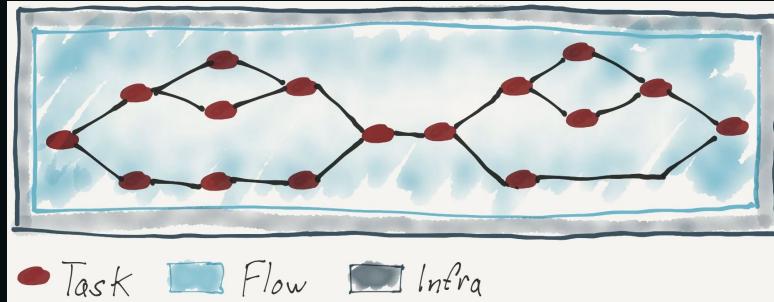
- Workflow patterns with
 - subflows
 - *run_deployment*
 - event webhooks with automations
- Explore *prefect.yaml* file for deployments
- Create schedules
- Pause and resume schedules and work pools



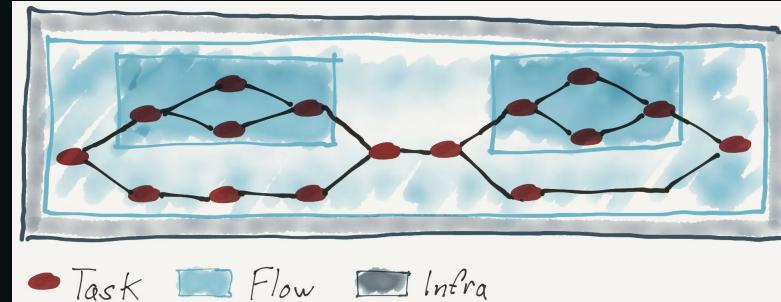
Workflow Patterns



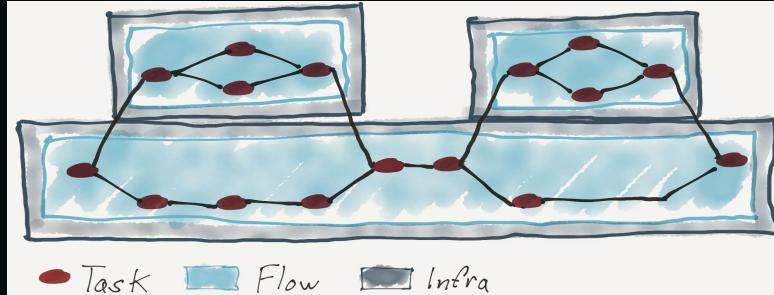
Workflow patterns - prefect.io/blog/workflow-design-patterns



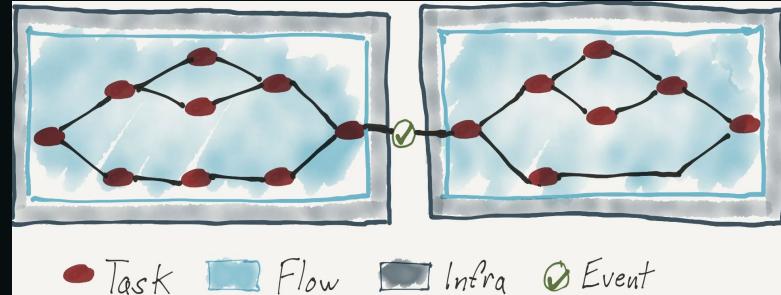
Monoflow



Flow of subflows



Flow of deployments



Event triggered flow

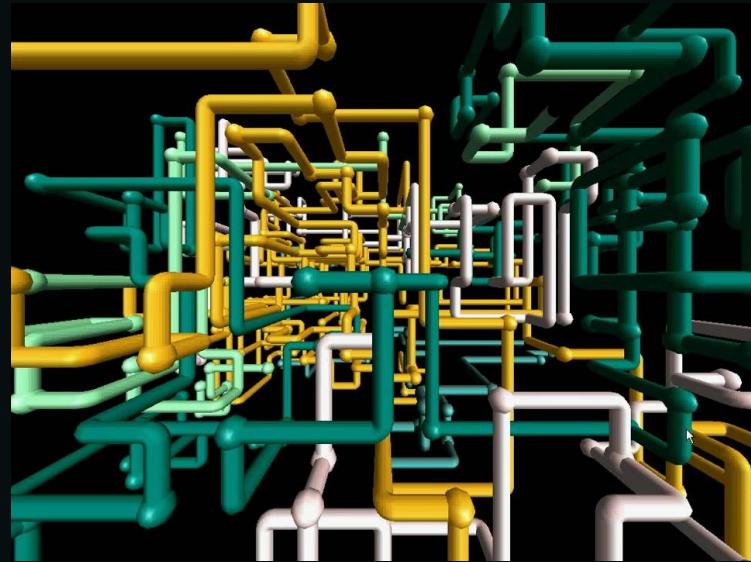


Subflows



Subflows

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



Subflows



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

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

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

Subflows



```
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

The screenshot displays a list of completed subflows in a dark-themed interface. Each item consists of a title, a completion status, and a timestamp.

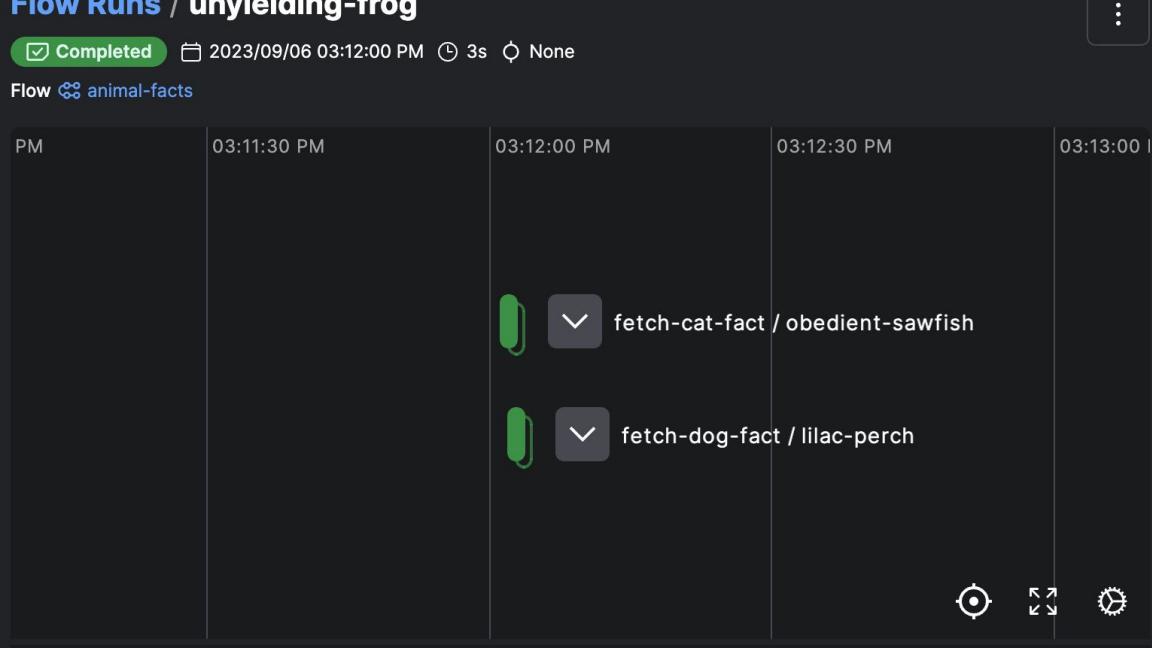
- fetch-dog-fact > lilac-perch**
Completed | 2023/09/06 03:12:01 PM | 1s | None
- fetch-cat-fact > obedient-sawfish**
Completed | 2023/09/06 03:12:01 PM | 1s | None
- animal-facts > unyielding-frog**
Completed | 2023/09/06 03:12:00 PM | 3s | None

Timeline view

Flow Runs / unyielding-frog

Completed 2023/09/06 03:12:00 PM ⌚ 3s ∅ None

Flow  animal-facts



The timeline view displays a sequence of events. At 03:11:30 PM, a green event icon (representing a flow run) is followed by a grey checkmark icon and the text "fetch-cat-fact / obedient-sawfish". At 03:12:00 PM, another green event icon is followed by a grey checkmark icon and the text "fetch-dog-fact / lilac-perch". The timeline is marked with vertical grid lines at 03:11:30 PM, 03:12:00 PM, 03:12:30 PM, and 03:13:00 PM.

PM 03:11:30 PM 03:12:00 PM 03:12:30 PM 03:13:00 PM

fetch-cat-fact / obedient-sawfish

fetch-dog-fact / lilac-perch

Events

⋮

⋮

Events

⋮



Logs in CLI

Sep 6th, 2023

| | | | |
|------|--|-------------|--------------------------------|
| INFO | Created subflow run 'obedient-sawfish' for flow 'fetch-cat-fact' | 03:12:01 PM | <code>prefect.flow_runs</code> |
| INFO | Created subflow run 'lilac-perch' for flow 'fetch-dog-fact' | 03:12:01 PM | <code>prefect.flow_runs</code> |
| INFO | <p>😺: 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.</p> <p>🐶: Puppies are blind, deaf and toothless when born.</p> | 03:12:02 PM | <code>prefect.flow_runs</code> |
| INFO | Finished in state Completed('All states completed.') | 03:12:02 PM | <code>prefect.flow_runs</code> |



run_deployment

| run_deployment <small>async</small>  | | | |
|---|------------------|---|-----------------|
| 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: <code><slugified-flow-name>/<slugified-deployment-name></code> | <i>required</i> |
| parameters | Optional[dict] | Parameter overrides for this flow run. Merged with the deployment defaults. | None |



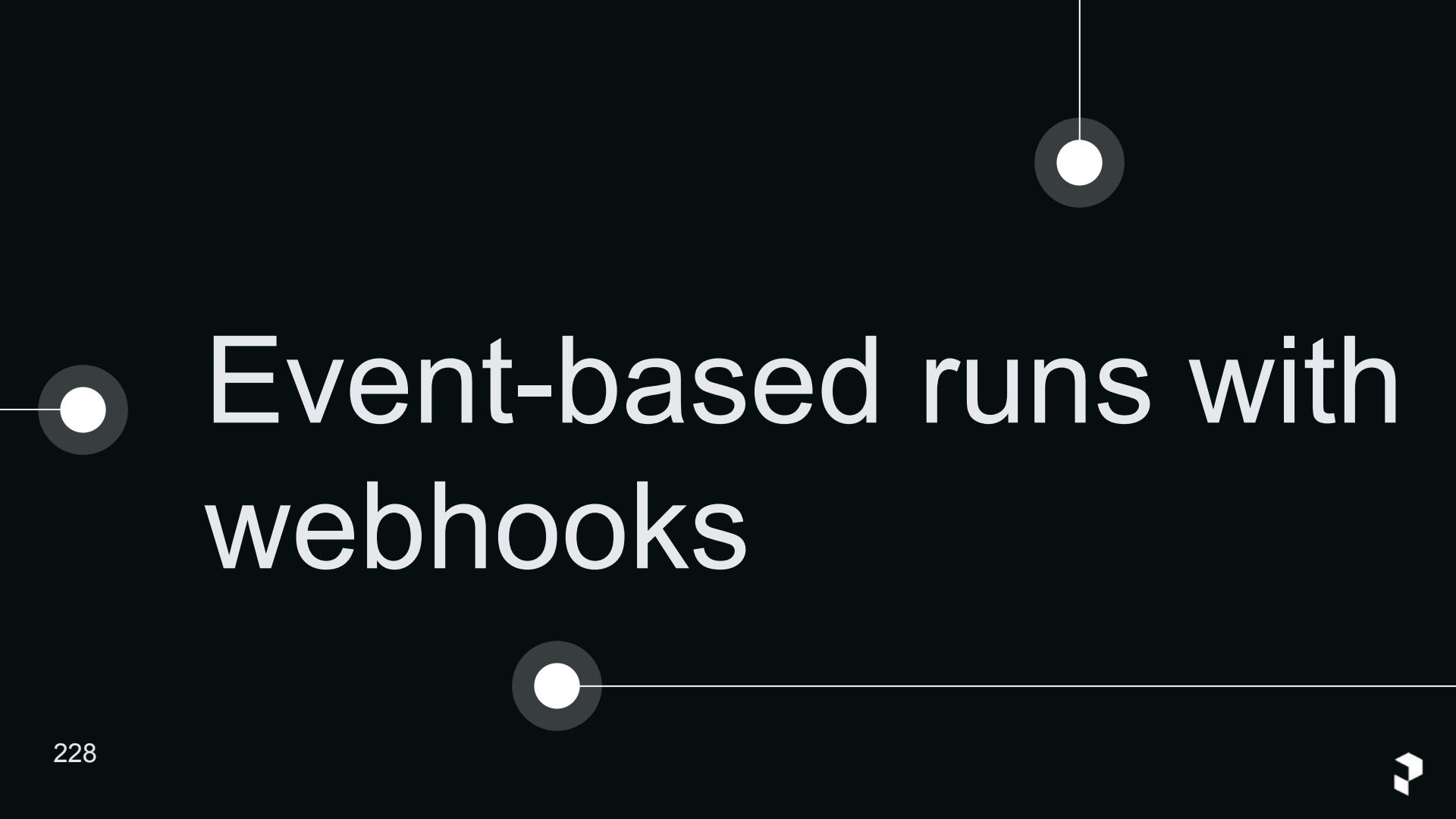
run_deployment

```
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

10:24:54 PM
Jun 19th, 2023



Demo event

demo.event

Resource

demo.alert.2

Related Resources

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



Webhooks

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



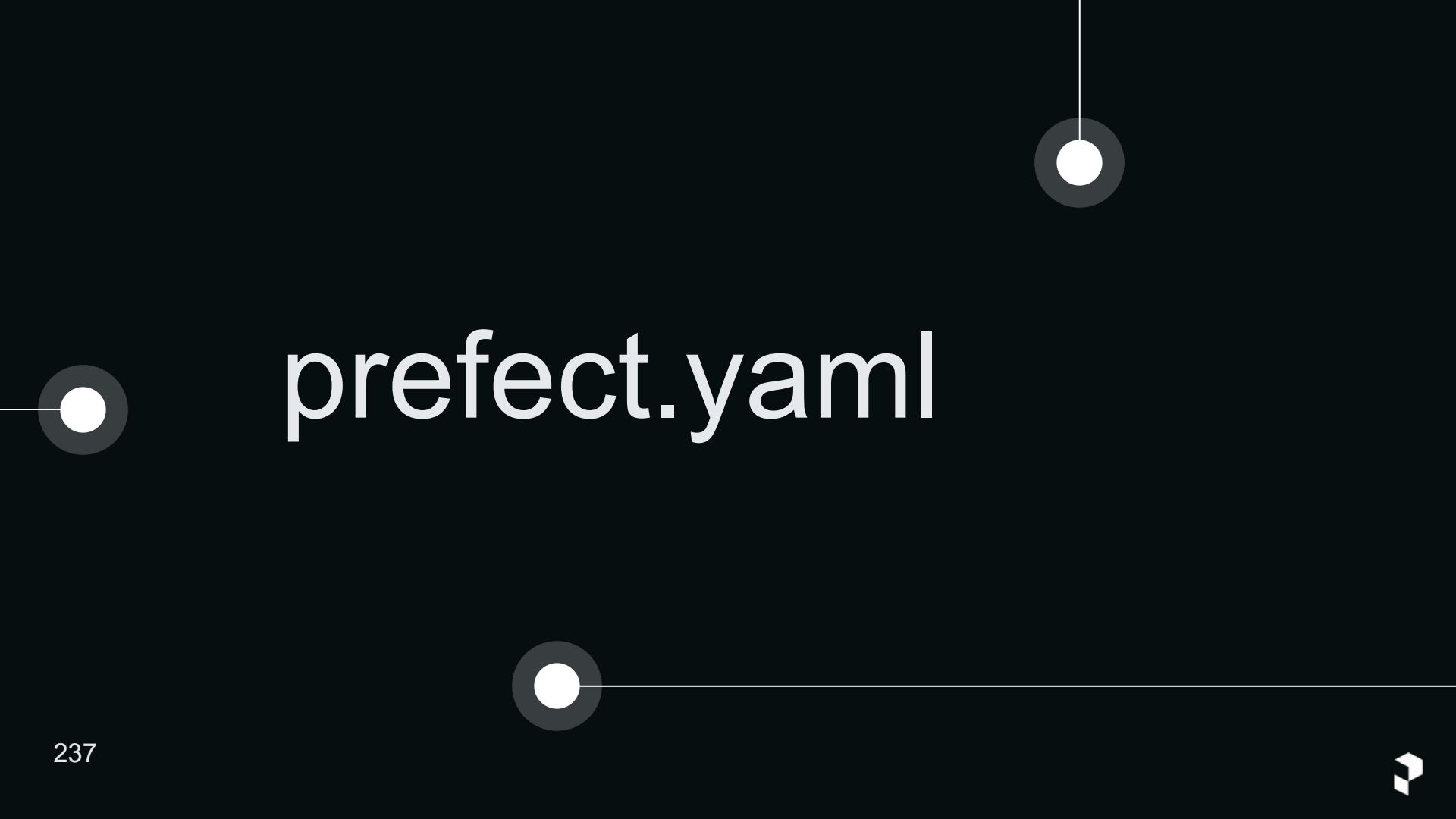
Webhooks

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.





A decorative graphic at the top of the slide features three white circles with thin gray outlines. One circle is positioned on the left edge, another is near the top right, and a third is at the bottom center. Thin white lines connect the top-right and bottom-center circles to the left edge, while a thicker horizontal line connects the bottom-center circle to the left edge.
`prefect.yaml`

prefect.yaml

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

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

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

# pull section allows you to provide instructions for cloning
pull:
- prefect.deployments.steps.git_clone:
    repository: https://github.com/dsdiscover/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*

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



Can override steps above on per-deployment basis

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

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



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. |

248 ? Seconds between scheduled runs (3600): 100



In the UI

The screenshot shows the Prefect UI interface. On the left, there's a sidebar with navigation links: Dashboard, Flow Runs, Flows, Deployments (which is highlighted with a blue bar), and Work Pools. The main area is titled "Deployments / deploy-1". It has tabs for Details, Runs, Parameters, Infra Overrides, and Description, with "Details" selected. Below the tabs, there are sections for "Flow" (showing a flow named "fetch-weather") and "Schedule" (with an "Add" button). Under "Triggers", there's a "+ Add" button. A red arrow points to this "+ Add" button.

jeffprefectio
storage-guide-ex

Deployments / deploy-1

Details Runs Parameters Infra Overrides Description

Flow

fetch-weather

Schedule

Add

Triggers

+ Add

In the UI

Add schedule X

Schedule type

Interval Cron RRule

Value Interval

60 Minutes ▼

Start date Timezone

Sep 6th, 2023 at 2:57 PM ▼

Cancel Save



RRule

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

Or ask [Marvin](#) (another Prefect package) *pip install*

```
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

```
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

| | Name | Schedule | Tags | Applied By | |
|--------------------------|--------------|--|------|---------------|---|
| <input type="checkbox"/> | hi / default | Every 3 days, 20 hours, 52 minutes, 23 seconds | | jeffprefectio |   |

Pause/resume schedule from CLI



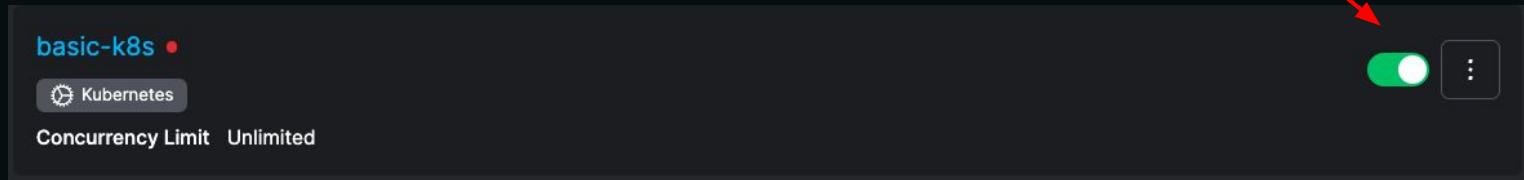
Pause

*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



| 1 Work Queue | | | | + | <input type="text"/> Search |
|--------------|-------------------|----------------------------|------------------------|-------------------------------------|-----------------------------|
| Name | Concurrency Limit | Priority ? | Status | | |
| default | 1 | | Unhealthy | <input checked="" type="checkbox"/> | ⋮ |

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

Prefect integrations are organized into collections of pre-built [tasks](#), [flows](#), [blocks](#) and more that are installable as PyPI packages.

| | | | | |
|--------------------------|-----------------------------------|--------------------------|--------------------------|--------------------------|
| Airbyte | Alert | AWS | Azure | Bitbucket |
| Maintained by Prefect | Maintained by Khuyen Tran | Maintained by Prefect | Maintained by Prefect | Maintained by Prefect |
| Census | CubeJS | Dask | Databricks | dbt |
| Maintained by Prefect | Maintained by Alessandro Lollo | Maintained by Prefect | Maintained by Prefect | Maintained by Prefect |



Integrations



Python packages that add convenience

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



Integrations



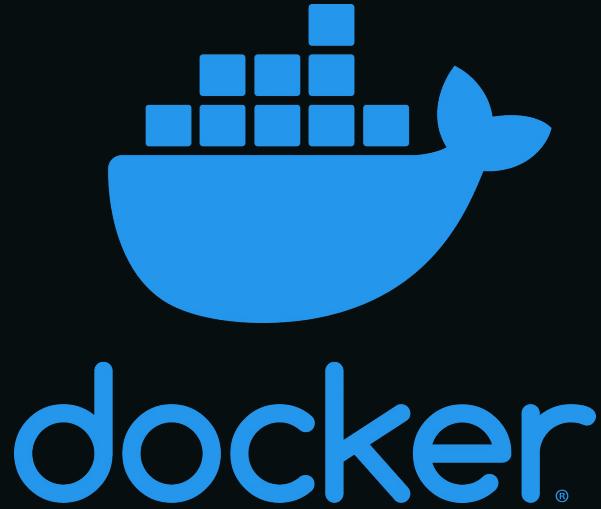
Prefect is Python-based and designed for flexibility

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





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
 - AWS logo
 - Execute flow runs within containers on AWS ECS. Works with EC2 and Fargate clusters. Requires an AWS account.
- Azure Container Instances
 - Azure logo
 - Execute flow runs within containers on Azure's Container Instances service. Requires an Azure account.
- Docker
 - Docker logo
 - 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
 - Google Cloud logo
 - Execute flow runs within containers on Google Cloud Run. Requires a Google Cloud Platform account.
- Kubernetes
 - Kubernetes logo
 - Execute flow runs within jobs scheduled on a Kubernetes cluster. Requires a Kubernetes cluster.



Container-based work pools

The screenshot shows the 'Work Pools / Create' interface. At the top, there are three tabs: 'Infrastructure Type' (marked with a checkmark), 'Details' (marked with a checkmark), and 'Configuration' (marked with '03'). Below these tabs, a message states: "Below you can configure workers' behavior when executing flow runs from this work pool. You can use the editor in the **Advanced** section to modify the existing configuration options if you need additional configuration options. If you don't need to change the default behavior, hit **Create** to create your work pool!"

The 'Base Job Template' section has two tabs: 'Defaults' (selected) and 'Advanced'. A note below the tabs says: "The fields below control the default values for the base job template. These values can be overridden by deployments."

Command (Optional)
The command to use when starting a flow run. In most cases, this should be left blank and the command will be automatically generated by the worker.
[Empty input field]

Environment Variables (Optional)
Environment variables to set when starting a flow run.
[Table with 3 rows: 1, 2, 3] Format



Container-based work pools

Can specify a particular image your team created

Name (Optional)
Name given to infrastructure created by the worker using this job configuration.

Image (Optional)
The image reference of a container image to use for created jobs. If not set, the latest Prefect image will be used.
`docker.io/prefecthq/prefect:2-latest`

Image Pull Policy (Optional)
The image pull policy to use when pulling images.
`None`

Networks (Optional)
Docker networks that created containers should be connected to.

| | |
|---|--------|
| 1 | Format |
| 2 | |
| 3 | |

Network Mode (Optional)
The network mode for the created containers (e.g. host, bridge). If 'networks' is set, this cannot be set.

Auto Remove (Optional)
If set, containers will be deleted on completion.



Container-based work pools

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

Below you can configure workers' behavior when executing flow runs from this work pool. You can use the editor in the **Advanced** section to modify the existing configuration options if you need additional configuration options. If you don't need to change the default behavior, hit **Create** to create your work pool!

Base Job Template

Defaults Advanced

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

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

```
{  
  "job_configuration": {  
    "command": "{{ command }}",  
    "env": "{{ env }}",  
    "labels": "{{ labels }}",  
    "name": "{{ name }}",  
    "image": "{{ image }}",  
    "image_pull_policy": "{{ image_pull_policy }}",  
    "networks": "{{ networks }}",  
    "network_mode": "{{ network_mode }}",  
    "auto_remove": "{{ auto_remove }}",  
    "volumes": "{{ volumes }}",  
  },  
  "resources": {  
    "limits": {  
      "cpu": 1,  
      "memory": "1Gi"  
    },  
    "requests": {  
      "cpu": 1,  
      "memory": "1Gi"  
    }  
  },  
  "secret_references": []  
}
```



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 my-pool | docker process | |

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



Method 1: *prefect deploy*



Use the defaults for the work pool

OR

Build a custom Docker image with flow code

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

Follow the prompts. 😊



Resulting *prefect.yaml*

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



Development environment

Where you write your flows

1 Create a project in a new directory

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

my-project

```
prefect.yaml  
prefect-version: 2.10  
name: my-project
```

```
build:  
  - step  
push:  
  - step  
pull:  
  - step
```

Deploy your flow

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

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

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

Orchestration environment

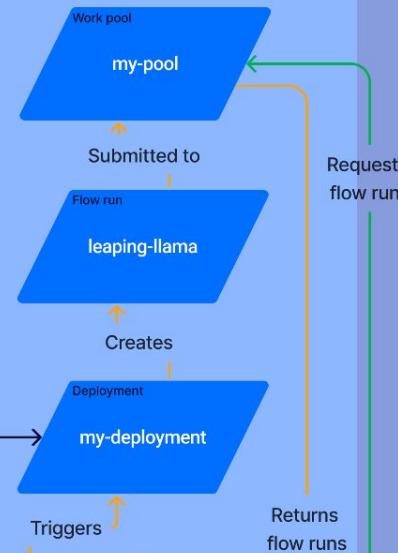
Where Prefect Cloud or server runs

3 Log in to Prefect Cloud

```
$ prefect cloud login
```

OR Start an open source Prefect server

```
$ prefect server start
```



OR

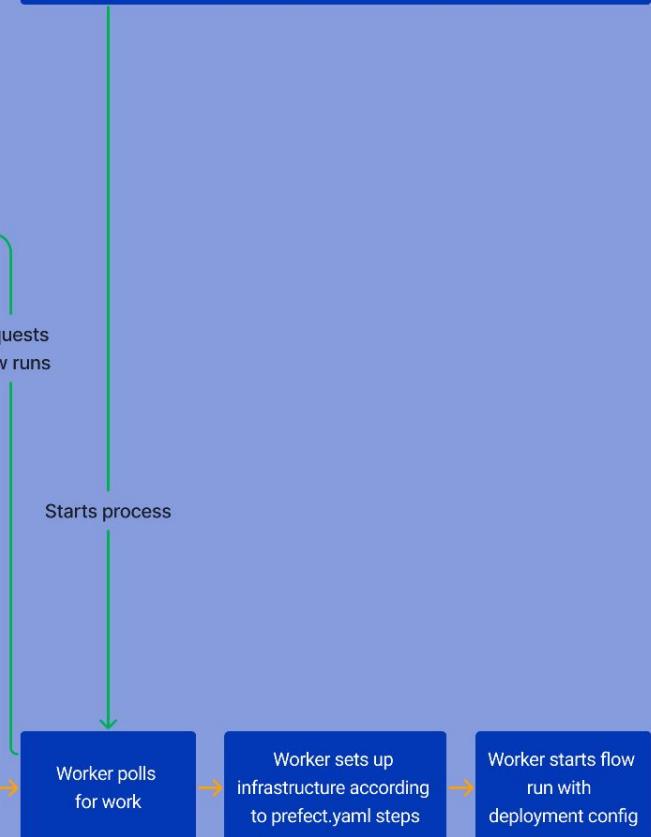
Page 6

Execution environment

Where your flows run

Start a worker for a corresponding work pool

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



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

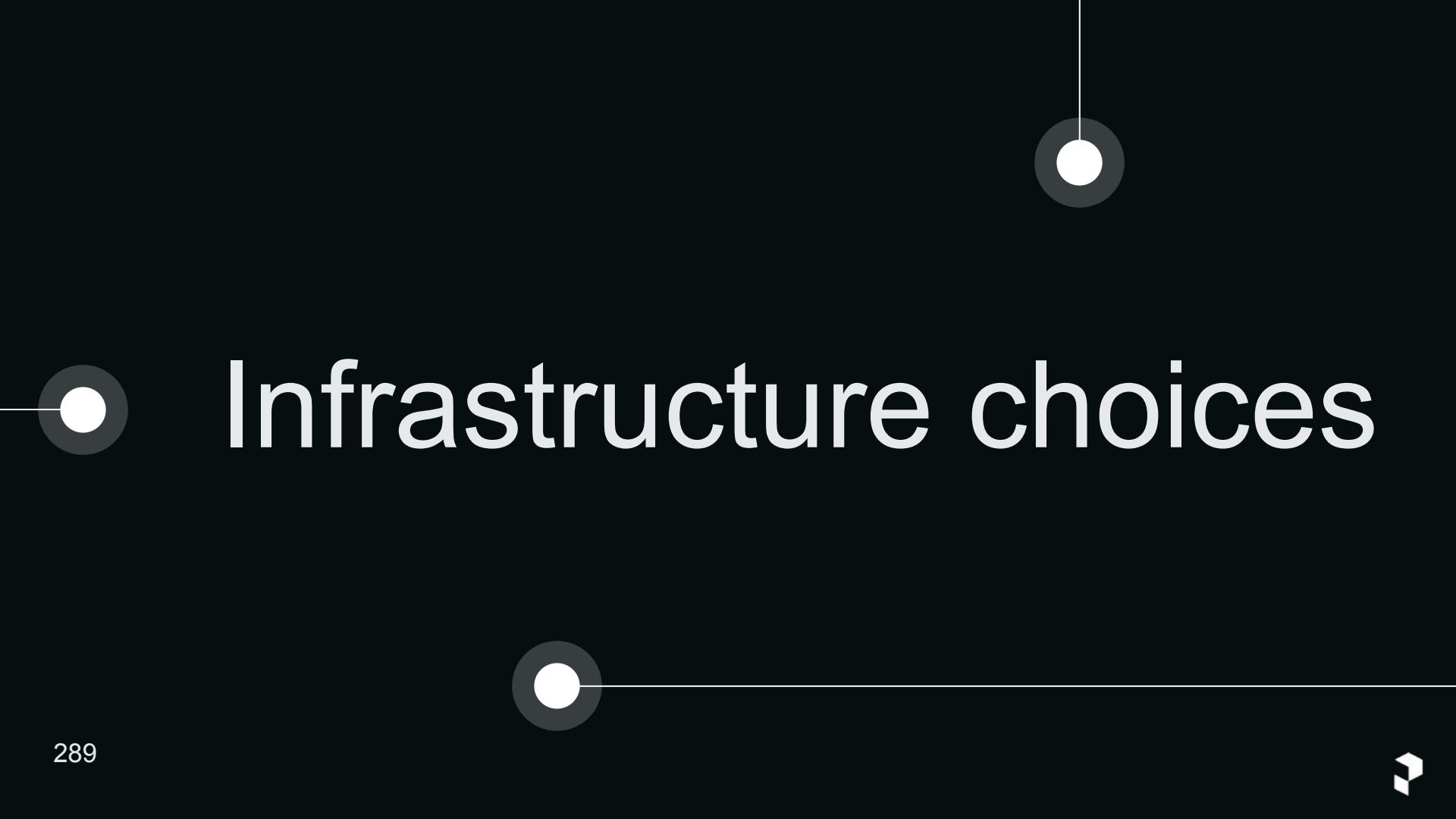
Containers [Give feedback](#)

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. [Learn more](#)

Only show running containers Search ⋮

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





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

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.



AWS Elastic Container Service - Push

Execute flow runs within containers on AWS ECS. Works with existing ECS clusters and serverless execution via AWS Fargate. Requires an AWS account. Flow runs are pushed directly to your environment, without the need for a Prefect worker.



Azure Container Instances

Execute flow runs within containers on Azure's Container Instances service. Requires an Azure account.



Azure Container Instances - Push

Execute flow runs within containers on Azure's Container Instances service. Requires an Azure account. Flow runs are pushed directly to your environment, without the need for a Prefect worker.



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.



Google Cloud Run - Push

Execute flow runs within containers on Google Cloud Run. Requires a Google Cloud Platform account. Flow runs are pushed directly to your environment, without the need for a Prefect worker.



Google Vertex AI

Execute flow runs within containers on Google Vertex AI. Requires a Google Cloud Platform account.



Kubernetes

Execute flow runs within jobs scheduled on a Kubernetes cluster. Requires a Kubernetes cluster.

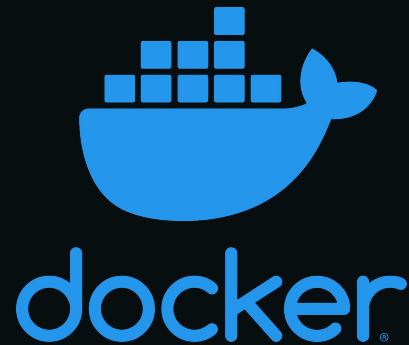


201 Recap



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



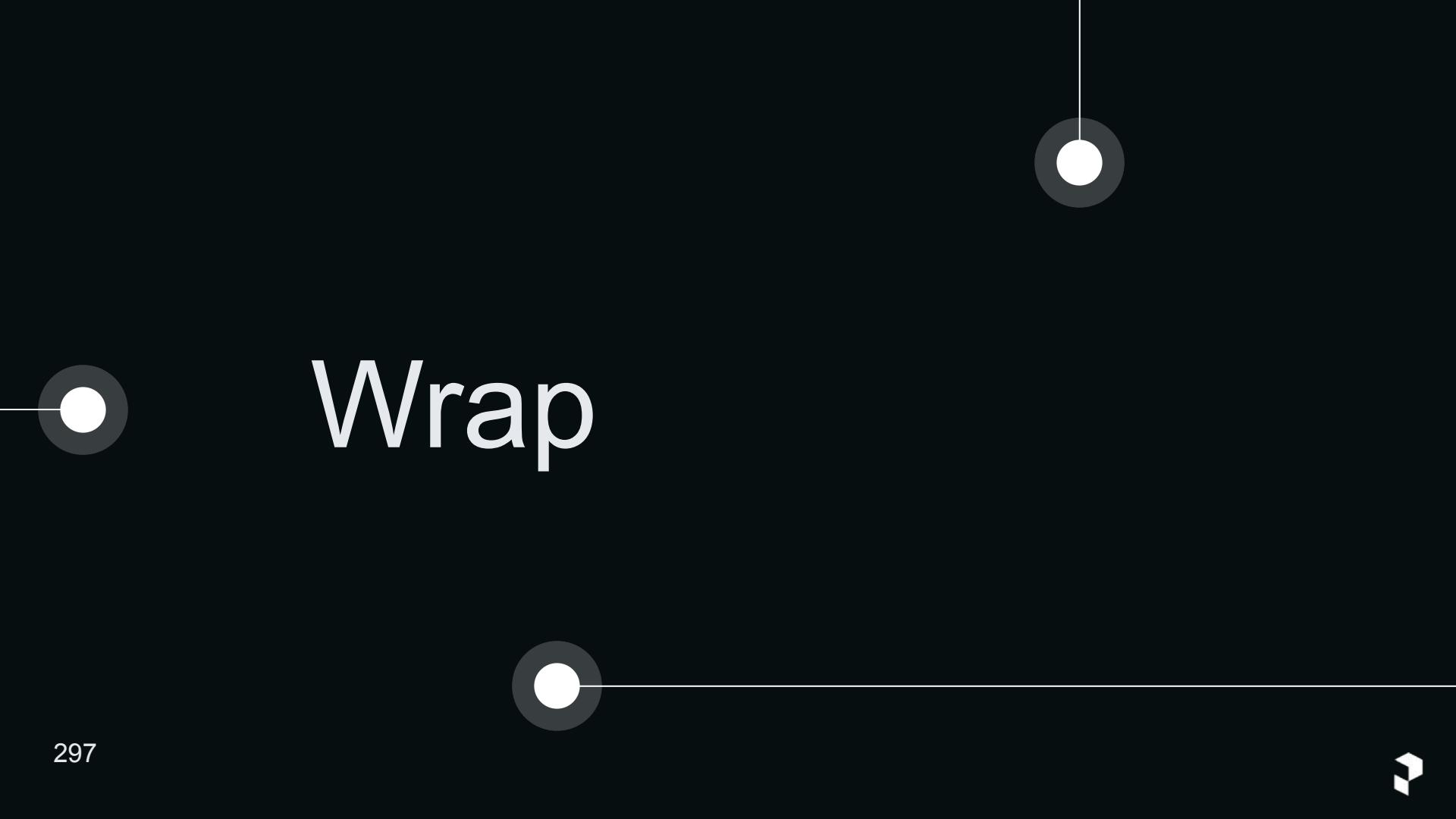
If you give an engineer a job...



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





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

New variable X

Name

Value

Tags

Cancel Create



Prefect variables

The screenshot shows the Prefect Variables page. On the left, a sidebar lists navigation options: Flow Runs, Flows, Deployments, Work Pools, Blocks, **Variables** (which is selected and highlighted in blue), Notifications, and Task Run Concurrency. The main area is titled "Variables" and shows "3 Variables". It includes a search bar, a sorting dropdown set to "A to Z", and a "Filter by tags" button. A table lists the variables:

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



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

```
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

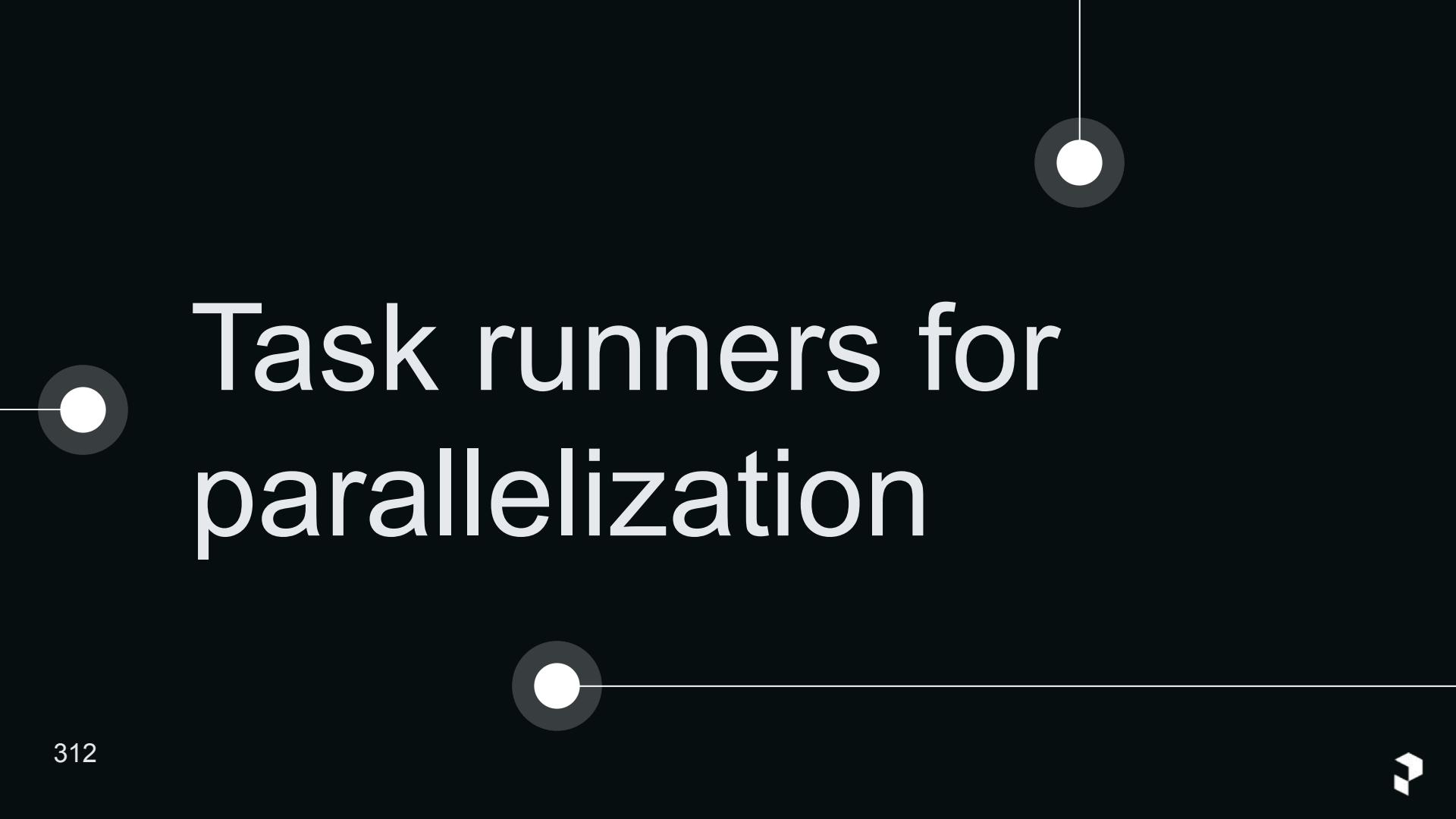
```
elevator moving to floor 3
elevator stops on floor 3
elevator moving to floor 1
elevator stops on floor 1
elevator moving to floor 2
elevator stops on floor 2
```



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

```
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)

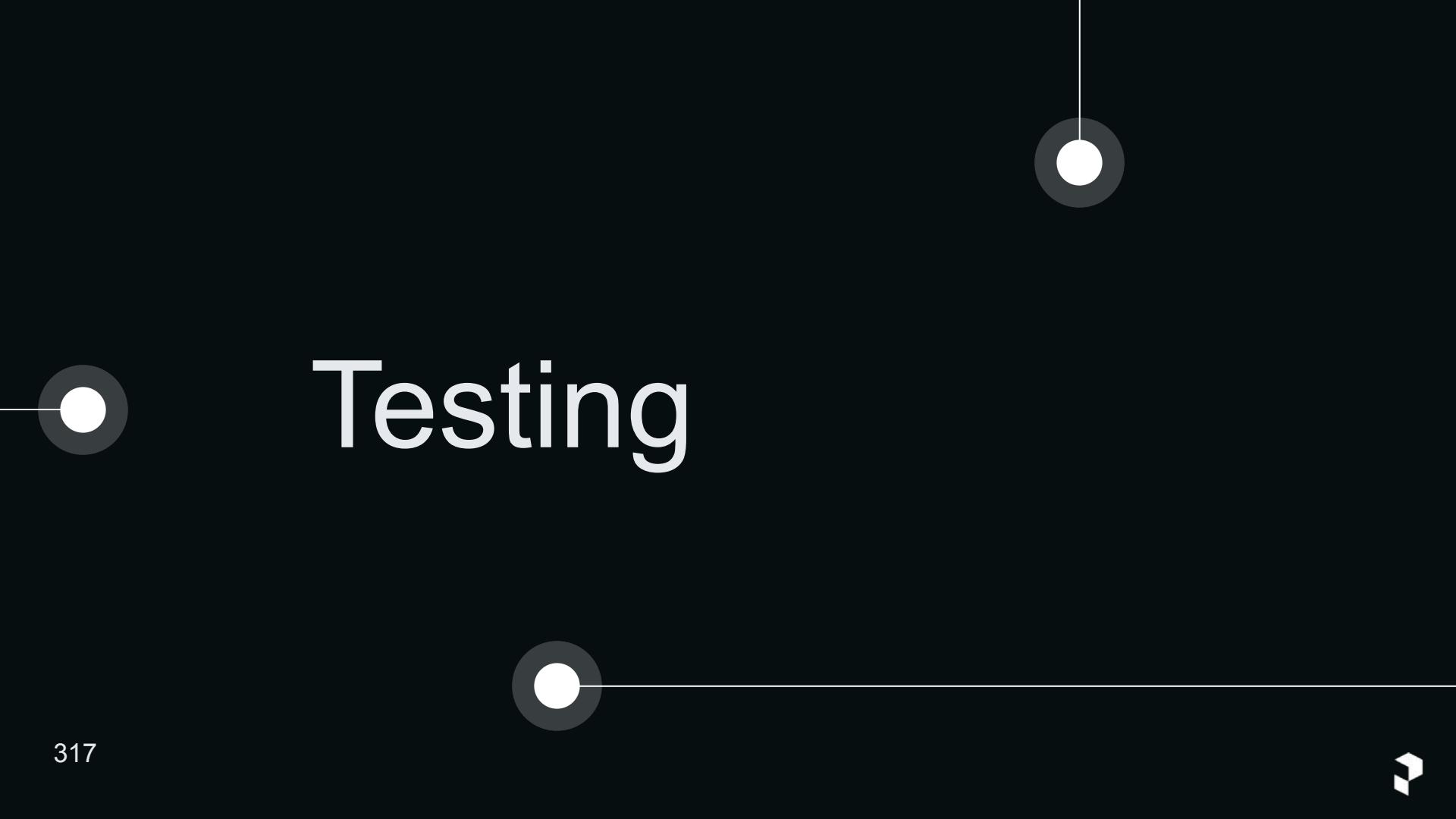
if __name__ == "__main__":
    greetings(["arthur", "trillian", "ford", "marvin"])
```



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

```
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_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

```
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

```
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

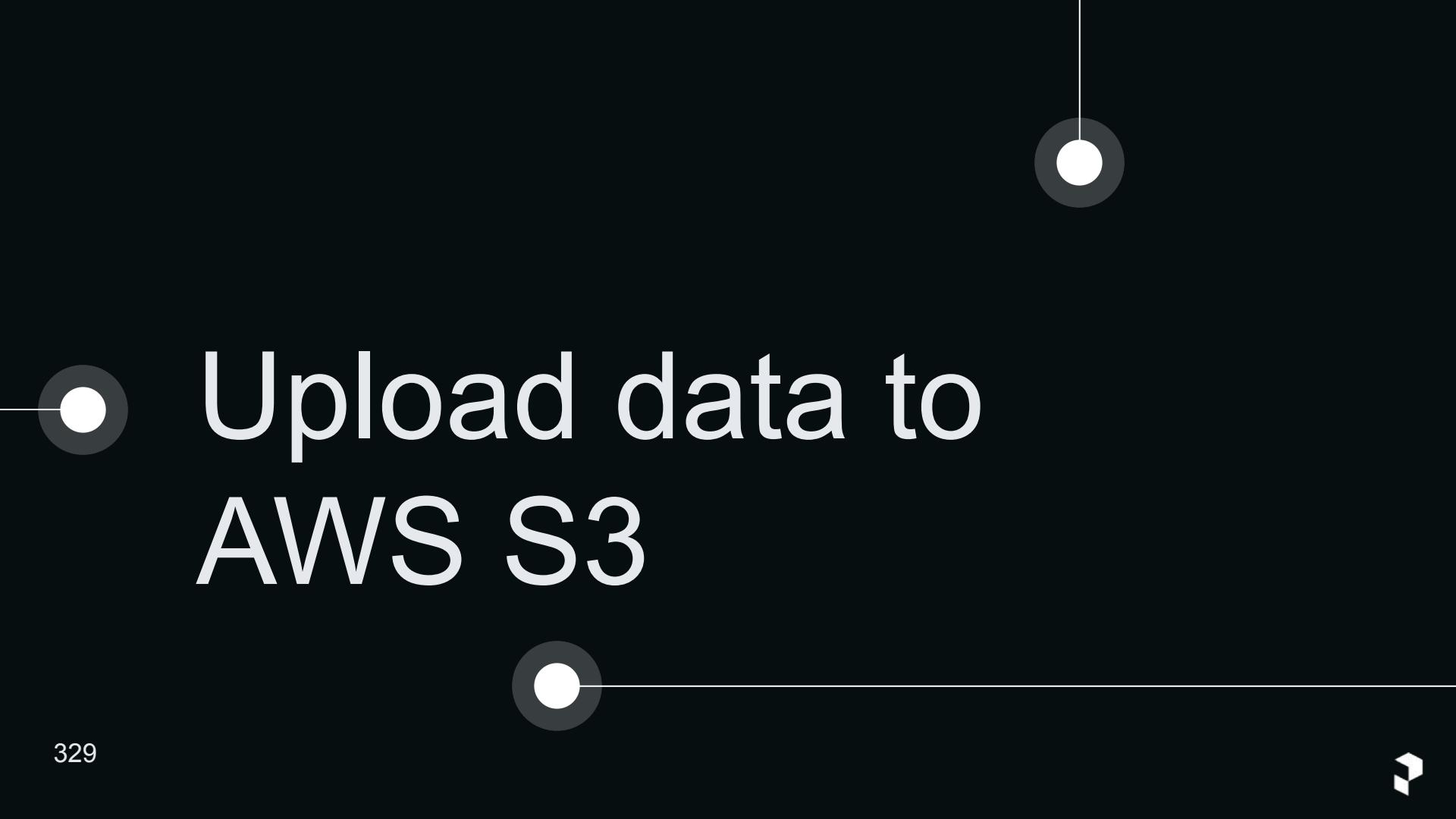
```
from prefect import flow, task
import prefect.runtime

@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

```
Successfully registered 5 blocks

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



See block types & blocks from CLI

prefect block type ls

prefect block ls



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 *S3Bucket* block



Create S3 Bucket

Amazon S3 > Buckets > Create bucket

Create bucket Info

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

General configuration

Bucket name

myawsbucket

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

AWS Region

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



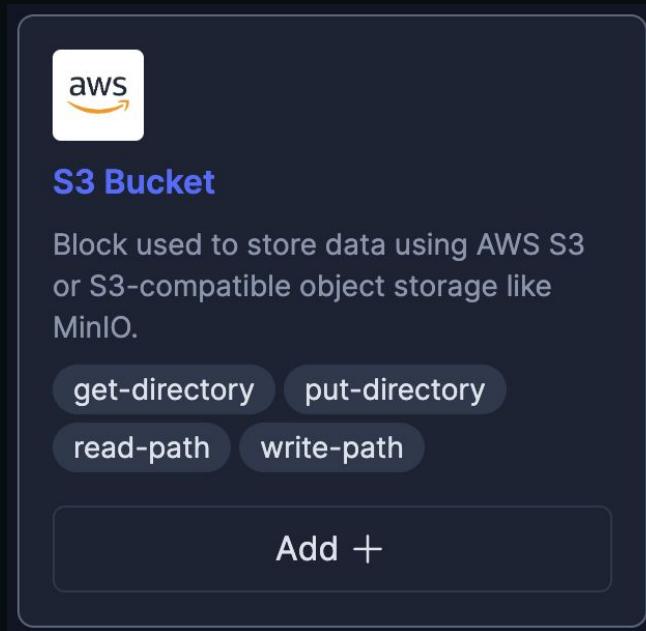
Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

[Choose bucket](#)



Create S3Bucket block from UI



Create S3Bucket block from UI

Blocks / Choose a Block / S3 Bucket / Create

Block Name

Bucket Name
Name of your bucket.

Credentials

[AwsCredentials](#) [MinIOCredentials](#)

A block containing your credentials to AWS or MinIO.
AwsCredentials (Optional)
Block used to manage authentication with AWS. AWS authentication is handled via the `boto3` module. Refer to the [boto3 docs](#) for more info about the possible credential configurations.

 Add +

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

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



AWS Credentials block from UI



Use the nested AWS Credentials block as needed

[Blocks](#) / [Choose a Block](#) / [AWS Credentials](#) / Create

Block Name

Region Name (Optional)

The AWS Region where you want to create new connections.

Profile Name (Optional)

The profile to use when creating your session.

AWS Access Key ID (Optional)

A specific AWS access key ID.



AWS Credentials

Block used to manage authentication with AWS. AWS authentication is handled via the `boto3` module. Refer to the [\[boto3 docs\]](#)...



AWS Credentials block from UI



Leave most fields blank.

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

The screenshot shows a dark-themed UI dialog box. At the top left, it says "AWS Access Key Secret (Optional)". Below that, a placeholder text "A specific AWS secret access key." is displayed. A large, empty input field follows. At the bottom right are two buttons: "Cancel" and a blue "Create" button.

AWS Access Key Secret (Optional)
A specific AWS secret access key.

Cancel Create



Or create blocks with Python code

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

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

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

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



View block in the UI

Blocks / my-aws-creds-block

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

Block document  my-aws-creds-block 2 events

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

```
from prefect_aws import AwsCredentials

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

Region Name
None

Profile Name
None

AWS Access Key ID
123abc

AWS Session Token
None

AWS Client Parameters
{ "config": null, "verify": true, "use_ssl": true, "api_version": "", "endpoint_url": "", "verify_cert_path": "" }

AWS Access Key Secret

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



Flow code loads S3 block and uploads data file

```
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

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

2020/

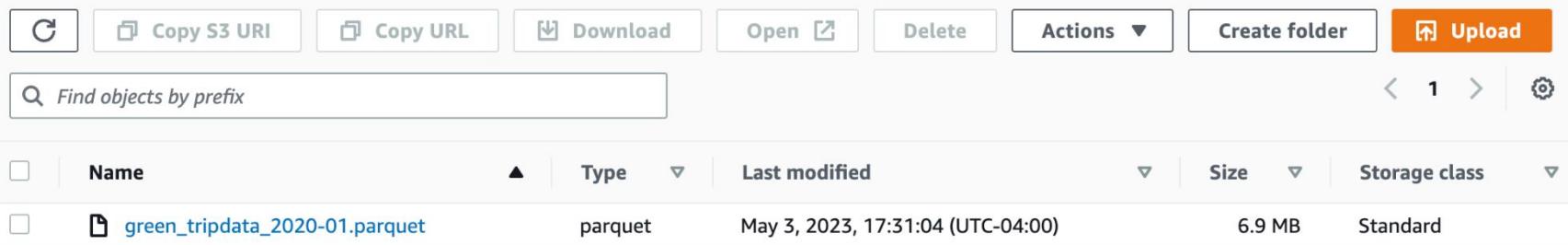
 Copy S3 URI

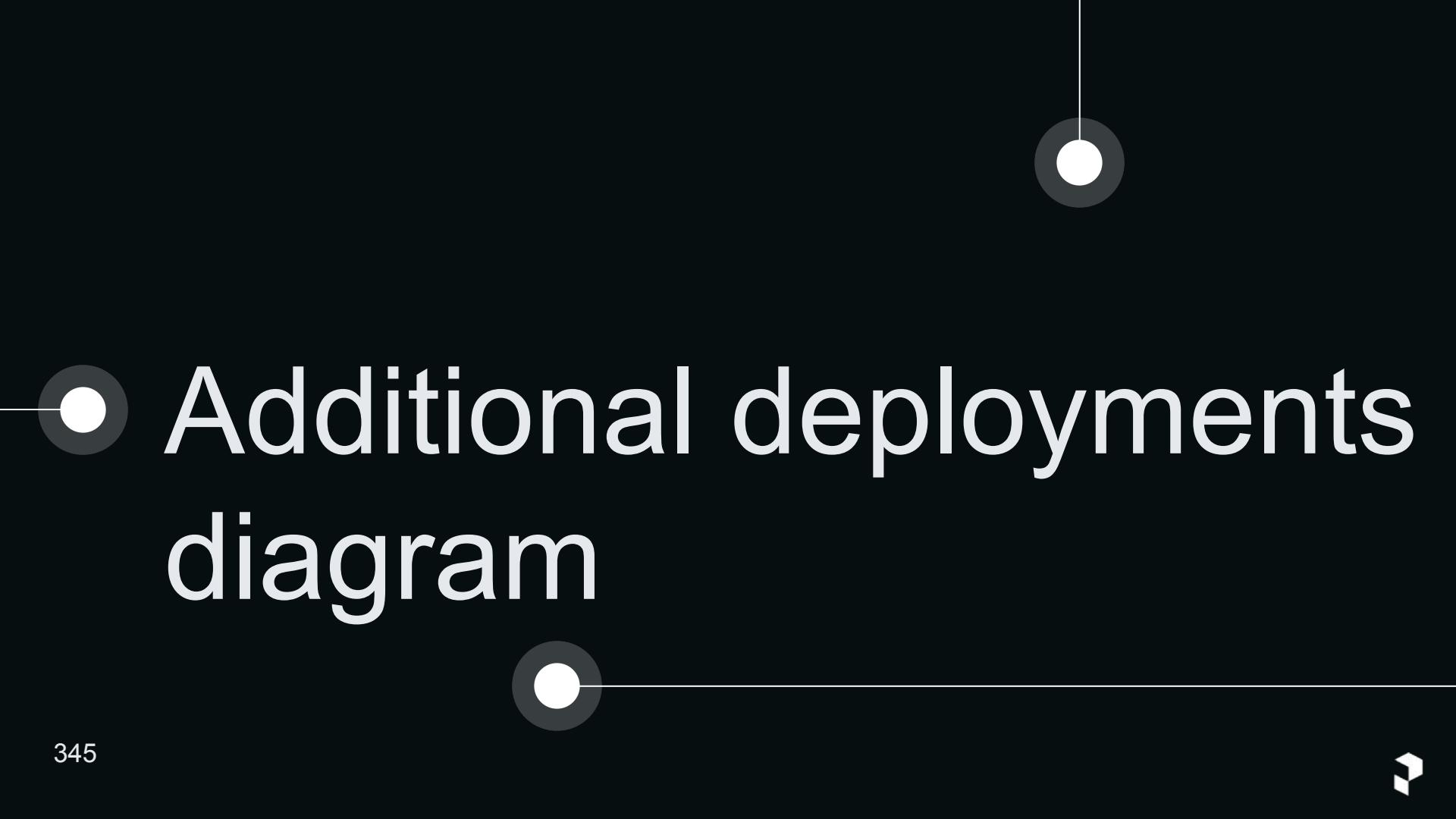
Objects

Properties

Objects (1)

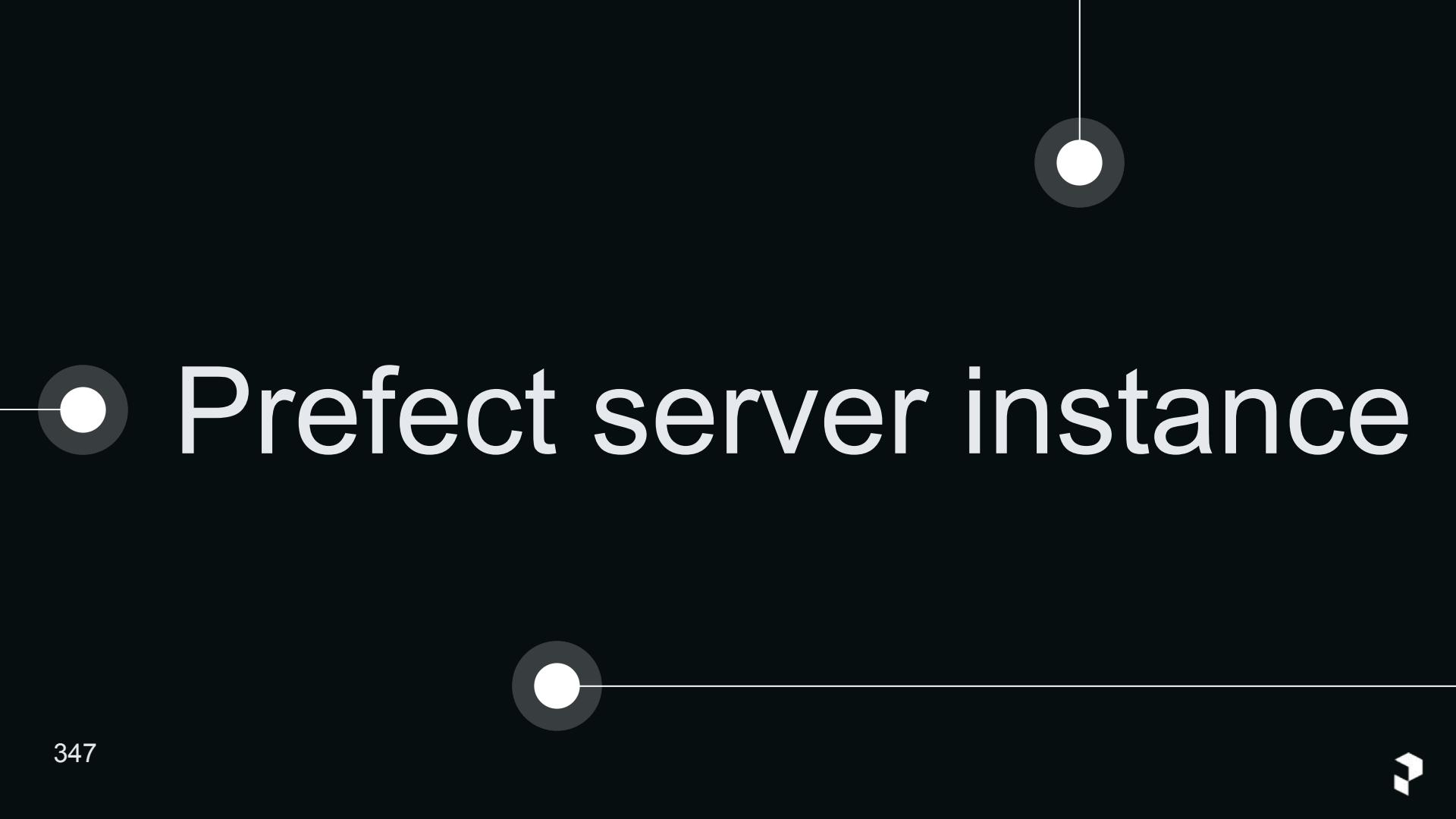
Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)





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>

The screenshot shows the Prefect UI Dashboard with a dark theme. On the left is a sidebar with navigation links: Dashboard (selected), Flow Runs, Flows, Deployments, Work Pools, Blocks, Variables, Notifications, Concurrency, and Artifacts. The main area has three sections: "Flow Runs" (3 total), "Task Runs" (0 completed), and "Active Work Pools".

Flow Runs: Shows 3 total runs. A bar chart indicates 0 failed, 0 pending, 1 succeeded, 2 late, and 0 canceled.

Task Runs: Shows 0 completed tasks.

Active Work Pools:

- default-agent-pool**: 0 total runs. Metrics: Polled (N/A), Work Queues (red circle), Late runs (0), Completes (N/A).
- docker-pool**: 0 total runs. Metrics: Polled (N/A), Work Queues (red circle), Late runs (0), Completes (N/A). A red dot next to the pool name indicates an error.
- my-pool**: 0 total runs. Metrics: Polled (17d 7h ago), Work Queues (red circle), Late runs (0 (43s avg)), Completes (N/A). A red dot next to the pool name indicates an error.

At the bottom, there is a "Settings" link and a small Prefect logo.

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

