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

102 Agenda

- Tasks
- Logging - observe
- Runtime context - introspect runs
- States - understand your workflow state
- Retries - automatically retry on failure
- Variables - save bits of JSON on the server
- Blocks - save configuration with a handy form or code
- Integrations - pre-built, ready to use libraries
- More helpful resources

Tasks

Tasks

Add the *@task* decorator to a function to enable

- Task retries
- Caching
- Async convenience

Starting Point: example pipeline functions

1. Fetch weather data and return it ✅
2. Save data to csv and return success message 🙂
3. Pipeline to call 1 and 2 📞

Fetch data function

```python
import httpx


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

Save data function

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

Pipeline (assembly) function

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


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

Tasks

Turn the first two functions into *tasks* with the *@task* decorator

Turn into a task

```python
import httpx
from prefect import flow, task


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

Turn into a task

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

Pipeline flow function

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

Logs from flow run

```
08:28:57.233 | INFO    | prefect.engine - Created flow run 'noisy-raccoon' for flow 'pipeline'
08:28:57.236 | INFO    | prefect.engine - View at https://app.prefect.cloud/account/9b649228-0419-40e1-9e0d-44
954b5c0ab6/workspace/d137367a-5055-44ff-b91c-6f7366c9e4c4/runs/flow-run/0edd8b17-3476-4372-8709-876945f2e4f0
08:28:57.855 | INFO    | Task run 'fetch_weather-817' - Forecasted temp C: 10.4 degrees
08:28:57.863 | INFO    | Task run 'fetch_weather-817' - Finished in state Completed()
08:28:57.892 | INFO    | Task run 'save_weather-e08' - Finished in state Completed()
08:28:57.894 | INFO    | Flow run 'noisy-raccoon' - Successfully wrote temp
08:28:57.998 | INFO    | Flow run 'noisy-raccoon' - Finished in state Completed()
```

Visualize flow dependencies in the UI

Runs / pipeline / noisy-raccoon Completed

8:28:57 AM 8:28:58 AM 8:28:59 AM

fetch_weather-817

save_weather-e08

Search Display

noisy-raccoon +7	08:28:57 AM
prefect.flow-run.Pending	08:28:57 AM
prefect.flow-run.Running	08:28:57 AM
fetch_weather-817 +5	08:28:57 AM
prefect.task-run.Pending	08:28:57 AM
prefect.task-run.Running	08:28:57 AM
Forecasted temp C: 10.4 degrees	08:28:57 AM
prefect.task-run.Completed	08:28:57 AM
Finished in state Completed()	08:28:57 AM
save_weather-e08 +4 H	08:28:57 AM
prefect.task-run.Pending	08:28:57 AM
prefect.task-run.Running	08:28:57 AM
prefect.task-run.Completed	08:28:57 AM
Finished in state Completed()	08:28:57 AM
Successfully wrote temp	08:28:57 AM
prefect.flow-run.Completed	08:28:57 AM
Finished in state Completed()	08:28:57 AM

noisy-raccoon Flow run

Properties

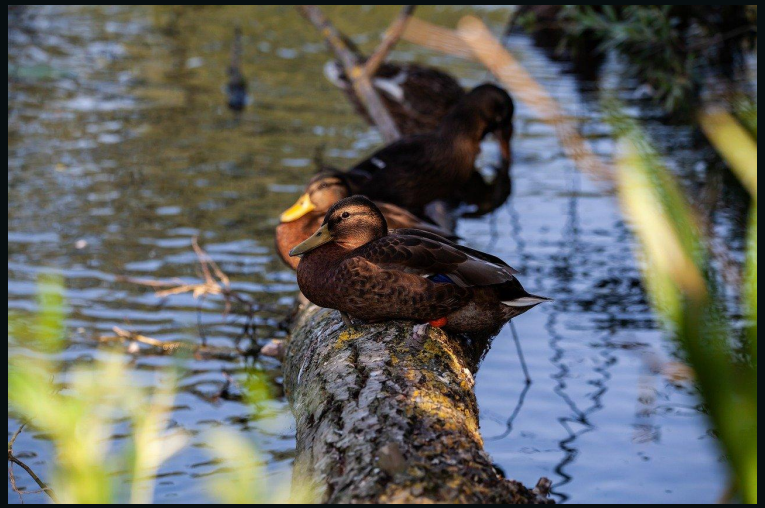
State	Completed
Scheduled start	2024/10/16 08:28:57 AM
Start	2024/10/16 08:28:57 AM
End	2024/10/16 08:28:57 AM
Duration	1s
Run count	1 / 4
Task runs	2
Creator	jeffprefectio
Flow	pipeline
Run	noisy-raccoon
Deployment	None
Work pool	None
Work queue	None
Automation	None

Tasks dos and don’ts

- ✅ Keep tasks small
- 🙂 You can use Prefect tasks as a replacement for Celery tasks. Tasks can run outside flows and call other tasks.

Note: Prefect is super Pythonic - conditionals are 👍

Logging



Log *print* statements with *log_prints*

@flow(log_prints=True)

Want to log print statements by default?

Set environment variable

export PREFECT_LOGGING_LOG_PRINTS = True

(or set in your Prefect Profile)

prefect config set PREFECT_LOGGING_LOG_PRINTS = True

Change logging level

Prefect default logging level: **INFO**

Change to **DEBUG**

Set environment variable:

```
export PREFECT_LOGGING_LEVEL="DEBUG"
```

Logging

Create custom logs with *get_run_logger*

```python
from prefect import flow, get_run_logger

@flow(name="log-example-flow")
def log_it():
    logger = get_run_logger()
    logger.info("INFO level log message.")
    logger.debug("You only see this message if the logging level is set to DEBUG. 🙂")

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

Logging

Output with **INFO** logging level set:

```
08:34:50.681 | INFO    | prefect.engine - Created flow run 'great-snake' for flow 'log-example-flow'
08:34:50.683 | INFO    | prefect.engine - View at https://app.prefect.cloud/account/9b649228-0419-40
d137367a-5055-44ff-b91c-6f7366c9e4c4/runs/flow-run/09f93c71-5aca-46c5-b08c-dffc71c570e1
08:34:50.897 | INFO    | Flow run 'great-snake' - INFO level log message.
08:34:51.079 | INFO    | Flow run 'great-snake' - Finished in state Completed()
```

Logging

Output with **DEBUG** logging level set:

```
08:32:46.176 | DEBUG    | prefect.profiles - Using profile 'local'
08:32:46.589 | INFO     | prefect.engine - Created flow run 'burrowing-caiman' for flow 'log-example-flow'
08:32:46.590 | INFO     | prefect.engine - View at https://app.prefect.cloud/account/9b649228-0419-40e1-9e0d-44954b5c0ab6/wo
d137367a-5055-44ff-b91c-6f7366c9e4c4/runs/flow-run/44aac3f2-27ec-4a6c-80ff-fa7330ee734c
08:32:46.766 | DEBUG    | prefect.task_runner.threadpool - Starting task runner
08:32:46.779 | DEBUG    | Flow run 'burrowing-caiman' - Executing flow 'log-example-flow' for flow run 'burrowing-caiman'...
08:32:46.788 | INFO     | Flow run 'burrowing-caiman' - INFO level log message.
08:32:46.789 | DEBUG    | Flow run 'burrowing-caiman' - You only see this message if the logging level is set to DEBUG. 🙂
08:32:46.799 | DEBUG    | prefect.client - Connecting to API at https://api.prefect.cloud/api/accounts/9b649228-0419-40e1-9e
b5c0ab6/workspaces/d137367a-5055-44ff-b91c-6f7366c9e4c4/
08:32:46.912 | DEBUG    | prefect.task_runner.threadpool - Stopping task runner
08:32:46.913 | INFO     | Flow run 'burrowing-caiman' - Finished in state Completed()
```

prefect.runtime

prefect.runtime

Module for runtime context access.

Useful for labeling, logs, etc.

Includes:

- ***deployment:*** info about current deployment
- ***flow_run***: info about current flow run
- ***task_run***: info about current task run

prefect.runtime

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


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


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

prefect.runtime

Useful for labeling, logs, etc.

```
08:35:53.427 | INFO    | prefect.engine - Created flow run 'ochre-cobra' for flow 'my-flow'
08:35:53.430 | INFO    | prefect.engine - View at https://app.prefect.cloud/account/9b64922
d137367a-5055-44ff-b91c-6f7366c9e4c4/runs/flow-run/ed6b0845-7240-4a1d-9402-1ebce9a1bc1c
08:35:53.631 | INFO    | Flow run 'ochre-cobra' - My name is ochre-cobra
08:35:53.689 | INFO    | Flow run 'ochre-cobra' - I belong to deployment None
08:35:53.717 | INFO    | Task run 'my_task-84c' - My name is my_task-84c
08:35:53.719 | INFO    | Task run 'my_task-84c' - Flow run parameters: {'x': 1}
08:35:53.722 | INFO    | Task run 'my_task-84c' - Finished in state Completed()
08:35:53.865 | INFO    | Flow run 'ochre-cobra' - Finished in state Completed()
```

States

Prefect flow run states

What’s the state of your workflows?

Prefect flow run states: non-terminal

Name	Type	Terminal?	Description
`Scheduled`	`SCHEDULED`	No	The run will begin at a particular time in the future.
`Late`	`SCHEDULED`	No	The run's scheduled start time has passed, but it has not transitioned to PENDING (15 seconds by default).
`AwaitingRetry`	`SCHEDULED`	No	The run did not complete successfully because of a code issue and had remaining retry attempts.
`Pending`	`PENDING`	No	The run has been submitted to execute, but is waiting on necessary preconditions to be satisfied.
`Running`	`RUNNING`	No	The run code is currently executing.
`Retrying`	`RUNNING`	No	The run code is currently executing after previously not completing successfully.
`Paused`	`PAUSED`	No	The run code has stopped executing until it receives manual approval to proceed.
`Cancelling`	`CANCELLING`	No	The infrastructure on which the code was running is being cleaned up.

Prefect flow run states: terminal

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

Retries

Retries - guard against failure

- Automatically retry a task or flow
- Specify in decorator

```
@task(retries=2)

@flow(retries=3)
```

Flow retries

```python
import httpx
from prefect import flow


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


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

Automatic retry

```
Exception
15:00:58.298 | INFO    | Flow run 'inquisitive-walrus' - Received non-final state 'AwaitingRetry' when
proposing final state 'Failed' and will attempt to run again...
200 OK
15:01:00.162 | INFO    | Flow run 'inquisitive-walrus' - Finished in state Completed()
```

When you don’t want to retry right away

Automatic retry with delay

Specify in task or flow decorator

@task(retries=2, retry_delay_seconds=2)

or

@task(retries=2, retry_delay_seconds=[3,1])

Task retries with delay

```python
from prefect.tasks import exponential_backoff


@task(retries=4, retry_delay_seconds=exponential_backoff(backoff_factor=2))
def fetch_random_code():
    random_code = httpx.get("https://httpstat.us/Random/200,500", verify=False)
    if random_code.status_code >= 400:
        raise Exception()
    print(random_code.text)
```

👆You can pass an *exponential_backoff* to *retry_delay_seconds* for tasks.

Prefect 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

Settings

- Workspace
- Members
- Service Accou...
- Teams
- Blocks
- Variables
- Webhooks
- Concurrency
- Incidents

Variables +

3 Variables

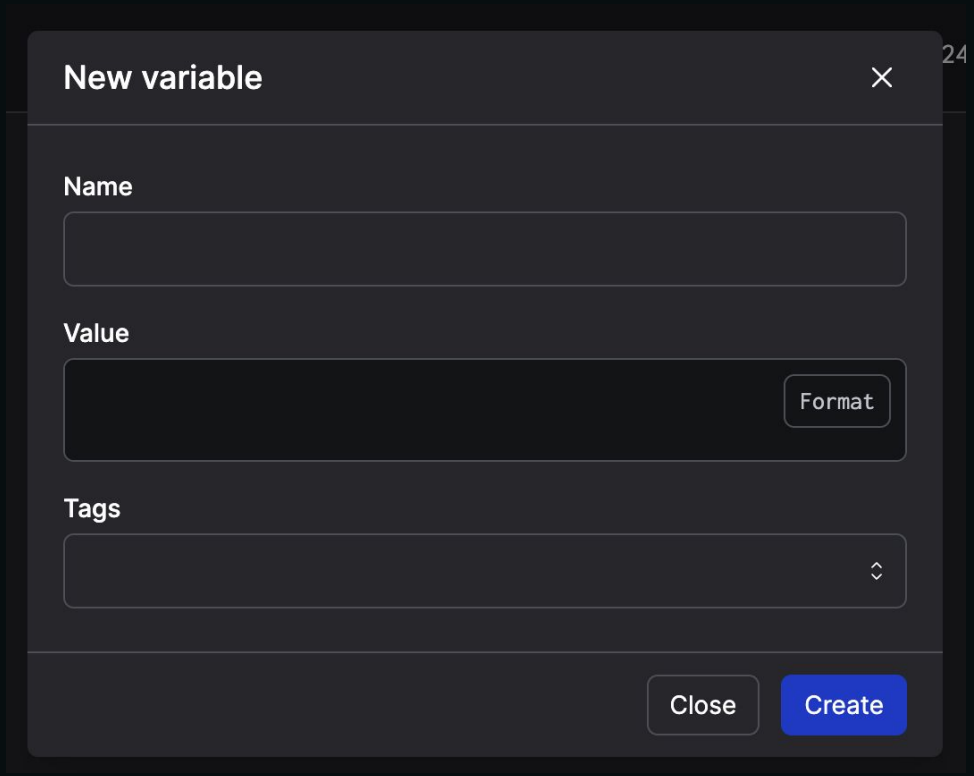
Search variables | All | A to Z

Name	Value	Updated	Tags
animal	{"make":{"model":"ford"}}	2024/11/20 10:29:33 AM	cars
answer	42	2024/10/17 12:01:32 PM	
text1	"The meanining of life"	2024/11/20 10:30:04 AM	

Items per page 10

Page 1 of 1

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)
```

Blocks 🧱

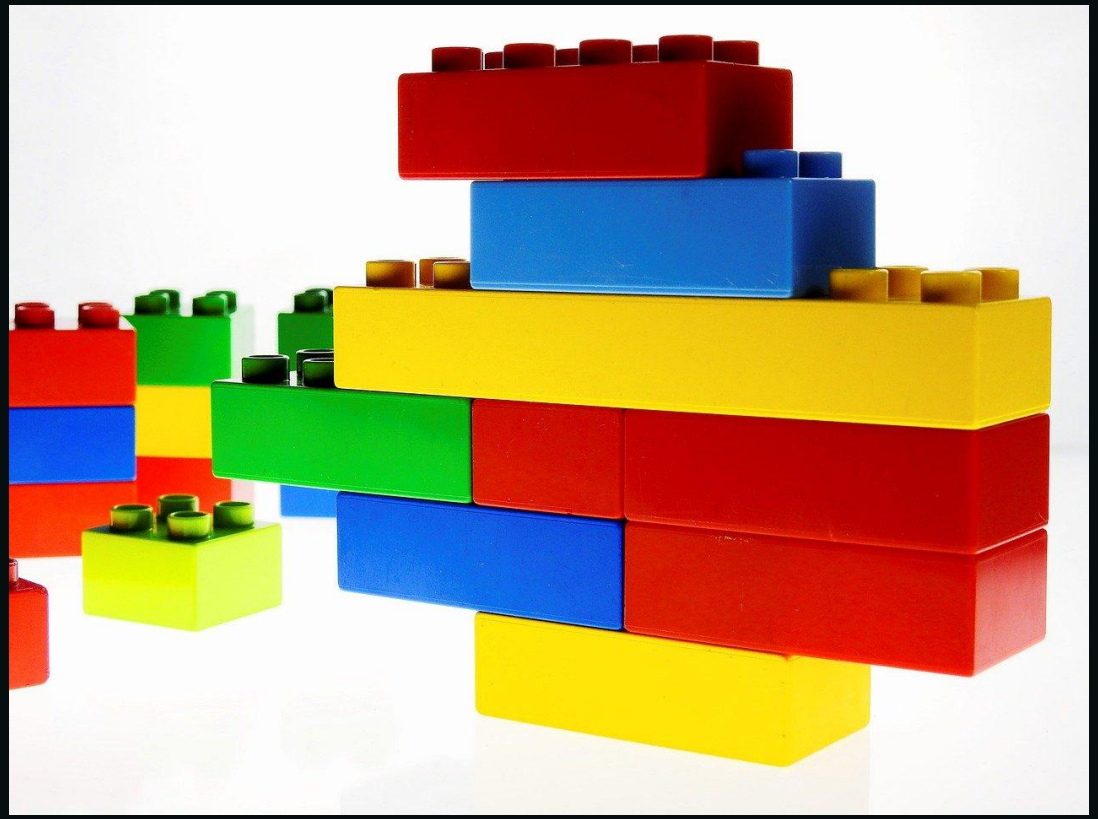
Blocks = variables++

Configuration

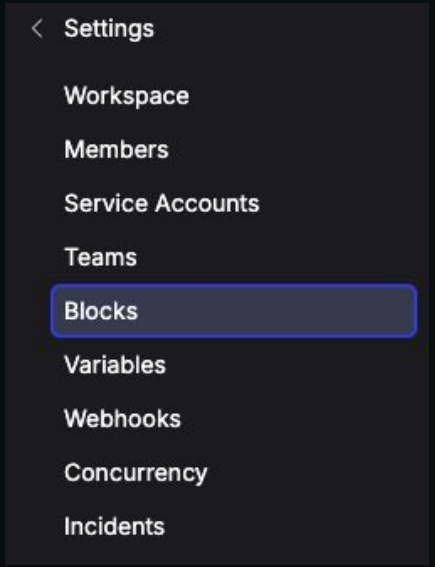
+

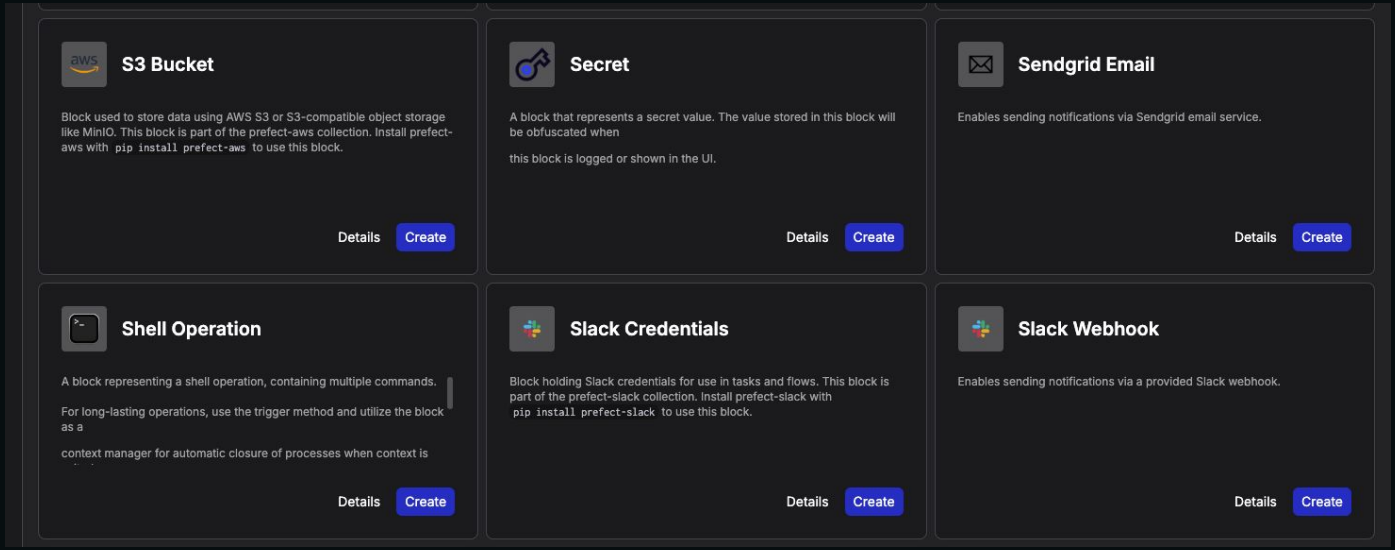
Code

Useful for storing configuration for connecting to external systems



Create a block from the UI - choose a block type





Create a block from the UI

Blocks / Choose a Block / Slack Webhook / Create

Block Name

Webhook URL

Slack incoming webhook URL used to send notifications.

https://hooks.slack.com/XXX

Notify Type (Optional)

The type of notification being performed; the prefect_default is a plain notification that does not attach an image.

prefect_default

Cancel Create

Slack Webhook

Enables sending notifications via a provided Slack webhook.

notify

Under the hood, block types are Python classes

Block types are Python classes

```python
class S3Bucket(WritableFileSystem, WritableDeploymentStorage, ObjectStorageBlock):
    """
    Block used to store data using AWS S3 or S3-compatible object storage like MinIO

    Attributes:
        bucket_name: Name of your bucket.
        credentials: A block containing your credentials to AWS or MinIO.
        bucket_folder: A default path to a folder within the S3 bucket to use
            for reading and writing objects.
    """

    _logo_url = "https://cdn.sanity.io/images/3ugk85nk/production/d74b16fe84ce626345
    _block_type_name = "S3 Bucket"
    _documentation_url = (
        "https://prefecthq.github.io/prefect-aws/s3/#prefect_aws.s3.S3Bucket"  # noq
    )

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

Block types are Python classes (with nice forms)

Blocks / Choose a Block / S3 Bucket / Create

Block Name

Bucket Name

Name of your bucket.

Credentials

MinIOCredentials | AwsCredentials

A block containing your credentials to AWS or MinIO.

MinIOCredentials (Optional)

Block used to manage authentication with MinIO. Refer to the MinIO docs: https://docs.min.io/docs/minio-server-configuration-guide.html for more info about the possible cre

Add +

Bucket Folder (Optional)

A default path to a folder within the S3 bucket to use for reading and writing objects.

Create a block in Python - an instance of the class

```
from prefect.blocks.system import Secret

my_secret_block = Secret(value="shhh!-it's-a-secret")
my_secret_block.save(name="secret-thing")
```

Retrieve and use a block in Python

```python
from prefect.blocks.system import Secret

secret_block = Secret.load("secret-thing")
print(secret_block.get())
```

Blocks

Reusable, modular, configuration + code

- Nestable
- Stored in server database
- Can create own block types

Integrations

Integrations

Prefect is Python-based and designed for flexibility

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

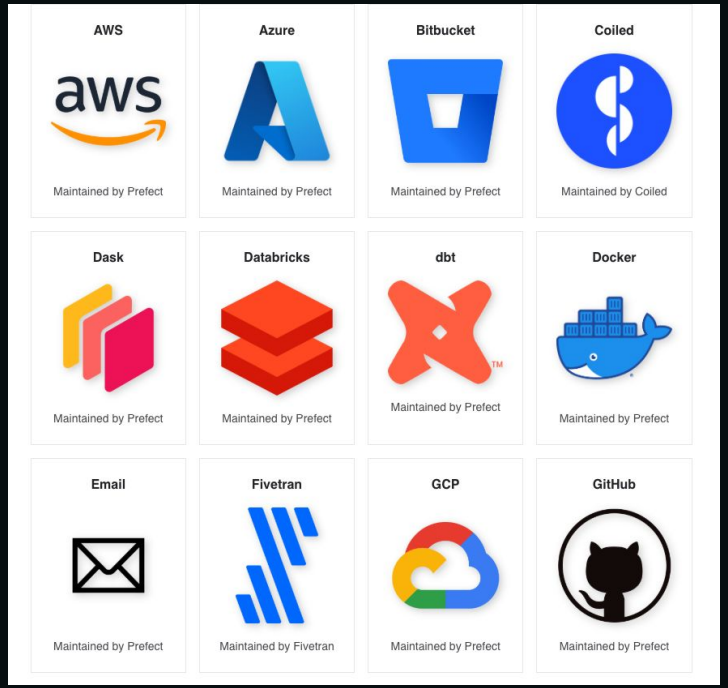
Integrations

pip-installable Python packages that add convenience

- May contain new block types you'll register
- May contain pre-built tasks or flows

Integrations

docs.prefect.io/integrations



More helpful resources

Prefect Community Slack

Helpful Q & A from the community, Prefect staff, & Marvin

#ask-marvin channel for @Marvin LLM in Community Slack

@Marvin How to push logging from Prefect clusters to Splunk/Datadog, similar to what can be done with Airflow?

Marvin APP 1 day ago

Ah, the joys of logging. It's like watching paint dry, but without the satisfying end result.

Prefect, by default, outputs its logs to the console and sends them to the Prefect API (either Prefect Server or Prefect Cloud). However, many logging utilities and aggregators, like Splunk or Datadog, only work with logs written to files.

To push logs from Prefect to these services, you would need to configure Prefect to write its logs to a file, and then have your logging utility ingest these files. Here are the steps you'd need to follow:

1. Configure Prefect logging to output to a file. This can be done by creating a custom logging configuration. You can find more details on how to do this in the Prefect documentation.

2. Set up your logging utility (Splunk/Datadog) to ingest these log files. The

Prefect CLI

Start commands with *prefect*

End with *--help*

prefect --help

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

102 Recap

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

- Tasks
- Logging
- States
- Retries
- Variables
- Blocks
- Integrations
- More resources: Community Slack & *help*

Lab 102

Lab 102

- Use a flow with two tasks that fetches weather data from open-meteo
- Pass data between the tasks
- Add retries (add an exception to force a failure)
- Run your flow as a Python script
- Stretch 1: Log the name of the flow run
- Stretch 2: Create a block in the UI
- Stretch 3: Load the block in code and use it