103 - Work with data and create automatic alerts

103 Agenda

- Work with data to save time and money
  - Save results
  - Use caching
  - Create artifacts to communicate insights
- Learn about user management in Prefect Cloud
- Set up automatic notifications for workflow states

Results

Results

The data returned by a flow or a task

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

1 is the result

Passing results

Pass results from one task to another so Prefect can discover dependency relationships at runtime

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

Results

👆 By default, Prefect returns a result that is ***not*** persisted to disk. It is only stored in memory.

Persist results with *persist_result=True*

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


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


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


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

Persisted results

- Stored in **.PREFECT/*storage*** folder by default
- Pickled by default 🥒
- You can use other serializer or compress

```
.PREFECT
  storage
    {} c65d28dcc374424ba7212a39dd19418b
  memo_store.toml
  prefect.db
  profiles.toml
```

storage > {} c65d28dcc374424ba7212a39dd19418b > ...


```
1  {"serializer": {"type": "pickle", "picklelib": "cloudpickle", "picklelib_version": "2.2.1"},
2  "data": "gAWVxwIAAAAAAACMEXBhbmRhcy5jb3JlLmZyYW1llIwJRGF0YUZyYW1llJOUKYGUfZQojARfbWdy\nlIwec
3  "prefect_version": "2.10.3"}
4
5
6
```

Results - remote data storage

Store results in cloud provider storage - use a block

```python
from prefect import flow, task
import pandas as pd
from prefect_gcp.cloud_storage import GCSBucket

# install module with: pip install prefect-gcp
# register block type
# create block


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


@flow(result_storage=GCSBucket.load("my-bucket-block"))
def my_flow():
    df = my_task()
```

Working with big data

Read and write data to cloud provider without passing the data around.

See discussion of options:

docs.prefect.io/latest/resources/big-data

Caching

Caching

What?

Why?

⚠️ task only

Requires persisting results (so must be serializable)

cache_policy - for computing cache keys

INPUTS - rerun only if inputs change

```python
from prefect import task
from prefect.cache_policies import INPUTS


@task(cache_policy=INPUTS, log_prints=True)
def my_cached_task(x: int):
    print(f"Result is: {x + 42}")


my_cached_task(8)  # Task runs
my_cached_task(8)  # Task doesn't run, uses the cached result
my_cached_task(33)  # Task runs
```

Caching with *INPUTS* policy

Task runs two times:

```
12:51:32.389 | INFO    | Task run 'my_cached_task' - Result is: 50
12:51:32.398 | INFO    | Task run 'my_cached_task' - Finished in state Completed()
12:51:32.416 | INFO    | Task run 'my_cached_task' - Finished in state Cached(type=COMPLETED)
12:51:32.441 | INFO    | Task run 'my_cached_task' - Result is: 75
12:51:32.443 | INFO    | Task run 'my_cached_task' - Finished in state Completed()
```

What happens if run the task again with same three inputs?

Caching

```
12:53:48.663 | INFO    | Task run 'my_cached_task' - Finished in state Cached(type=COMPLETED)
12:53:48.684 | INFO    | Task run 'my_cached_task' - Finished in state Cached(type=COMPLETED)
12:53:48.702 | INFO    | Task run 'my_cached_task' - Finished in state Cached(type=COMPLETED)
```

If you delete the cached result (the file stored in *~/.prefect/storage* by default) and rerun the flow, the task will run.

Cache policy options

DEFAULT: task inputs, code definition, and current flow run ID.

INPUTS: only task inputs.

TASK_SOURCE: only task code definition.

FLOW_PARAMETERS: only parameter values provided to parent flow run.

Can be combined and can create customized functions.

Caching: *cache_expiration* ⏳

```python
from datetime import timedelta
from time import sleep
from prefect import flow, task
from prefect.cache_policies import INPUTS


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


@flow(log_prints=True)
def hello_flow(name_input: str):
    hello_task(name_input)


if __name__ == "__main__":
    hello_flow(name_input="world")
    sleep(100)
    hello_flow(name_input="world")
```

Transaction rollbacks ↩️

Transactions

- Can roll back a side effect if code fails
- Like an undo button to return to previous state

docs.prefect.io/latest/develop/transactions

Transactions

In a flow, call tasks using a *with transaction():* context block

```
@flow
def pipeline(contents: str):
    with transaction():
        write_file(contents)
        quality_test()
```

Transactions

- Can specify action to take if an error is raised
- Decorate a function with *my_task.on_rollback()*

```
@write_file.on_rollback
def del_file(transaction):
    "Deletes file."
    os.unlink("side-effect.txt")
```

Transactions

```python
from prefect import task, flow
from prefect.transactions import transaction

@task
def write_file(contents: str):
    "Writes to a file."
    with open("side-effect.txt", "w") as f:
        f.write(contents)

@write_file.on_rollback
def del_file(transaction):
    "Deletes file."
    os.unlink("side-effect.txt")

@task
def quality_test():
    "Checks contents of file."
    with open("side-effect.txt", "r") as f:
        data = f.readlines()

    if len(data) < 2:
        raise ValueError("Not enough data!")

@flow
def pipeline(contents: str):
    with transaction():
        write_file(contents)
        quality_test()
```

Transaction lifecycle stages

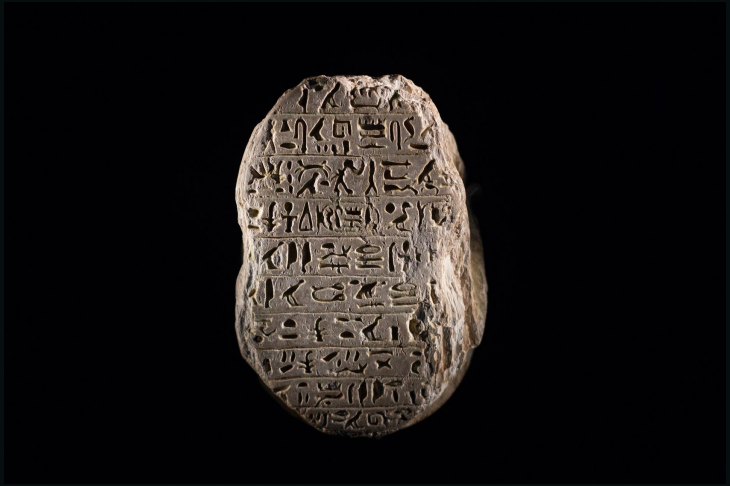
Each transaction goes through at most four lifecycle stages:

- **BEGIN:** transaction's key is computed and looked up. If a record already exists at the key location the transaction considers itself committed.
- **STAGE**: stages a piece of data to be committed to its result location.
- **ROLLBACK**: if the transaction encounters *any* error after staging, it rolls itself back and does not commit anything.
- **COMMIT**: the transaction writes its record to its configured location.

Artifacts

Artifacts

Communicate with team members through persisted outputs such as Markdown, tables, images, and links



Artifacts

- Meant for human consumption
- Examples:
  - Model scores
  - Data quality checks
  - Reports
- Stored in database & shown in UI
- Custom RBAC can limit user access (Enterprise)

Artifacts - Markdown example of weather report

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


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

## Recent weather

| Time        | Temperature |
|:-------------|-------:|
| Temp Forecast  | {temp} |
"""
    create_markdown_artifact(
        key="weather-report",
        markdown=markdown_report,
        description="Very scientific weather report",
    )
```

Artifacts - Markdown example of weather report

Access from UI: *Runs* timeline

Key

weather-report

Description

Very scientific weather report

Weather Report

Recent weather

Time	Temperature
Temp Forecast	25.9

Artifacts are versioned

Prefect Cloud

Prefect Cloud

- Prefect takes care of the server
- User account management (some at higher tiers)
  - Workspaces
  - Service accounts
  - RBAC
  - SSO
  - Audit logs
- Additional features

Prefect Cloud workspaces

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

Prefect Cloud - Default Roles (Pro + Enterprise)

Account level

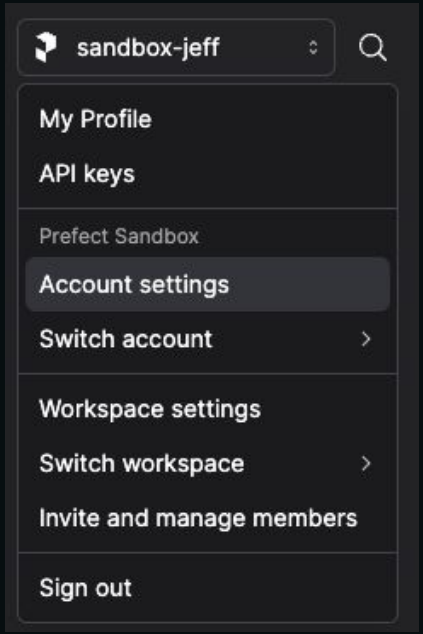
- Owner
- Admin
- Member

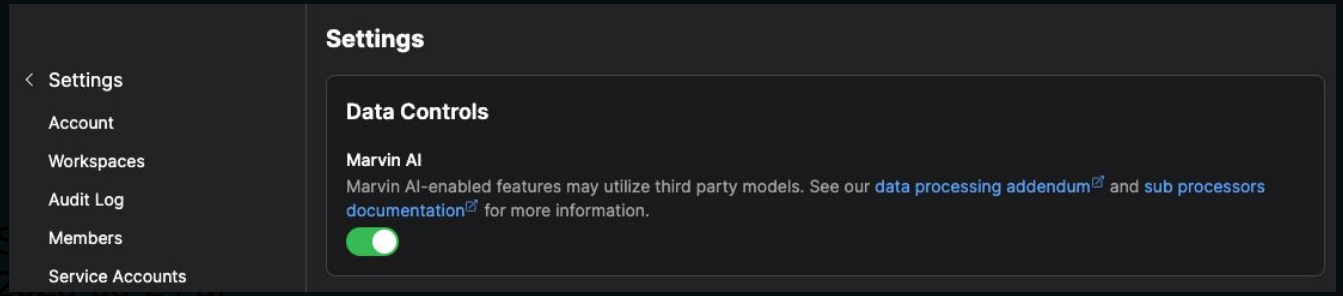
Workspace level

- Owner
- Developer
- Runner
- Viewer
- Worker

Error summaries by Marvin AI

Error summaries by Marvin AI





12.11.58 PM

Error summaries by Marvin AI

get-info > zircon-tapir

Failed 2023/09/25 03:29:38 PM 1s None

Failed due to a `IndexError` in the `get_info` task; range object index out of range.

ml-flow > translucent-pogona

Failed 2023/09/25 03:16:42 PM 1s 1 task run

Failed due to a `ZeroDivisionError` in the `compute` task with message 'division by zero'.

ml-flow > wealthy-firefly

Completed 2023/09/25 03:16:25 PM 2s 1 task run

Events

Events

- Record of what happened

Represent:

- API calls
- State transitions
- Changes in environment

Event feed

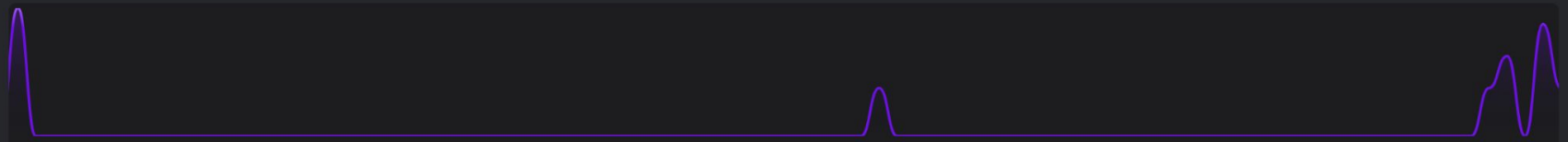
Workspace Events

Resource

All resources

Events

All events



Past day

08:49:29 AM
May 21st, 2024

Automation deleted

prefect-cloud.automation.deleted

Resource

prefect-cloud.automation.9e091fa8-29dc-4754-9452-699d1deb6c0e

Related Resources

prefect-cloud.actor.9d33d732-99f5-4330-b9dd-3f95a6154afa prefect-cloud.account.9b649228-0419-40e1-9e0d-44954b5c0ab6

prefect-cloud.workspace.d137367a-5055-44ff-b91c-6f7366c9e4c4

Events

Power several Cloud features:

- Flow run logs
- Audit logs
- Automations (triggers)

Automations ⚡

Automations

Flexible framework

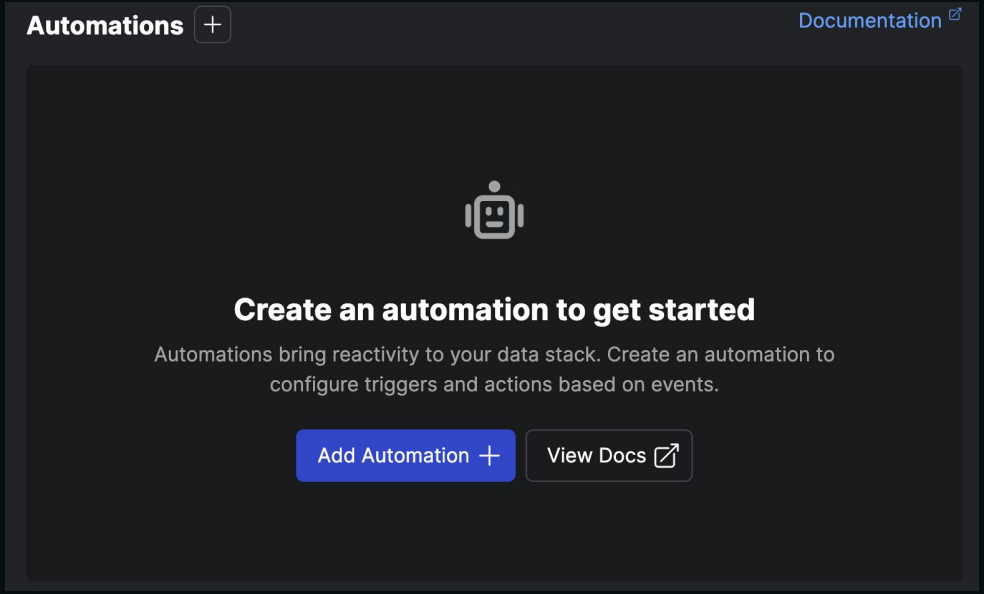
- If *Trigger* happens, do *Action*
- If *Trigger* doesn’t happen in a time period, do *Action*

Automation examples

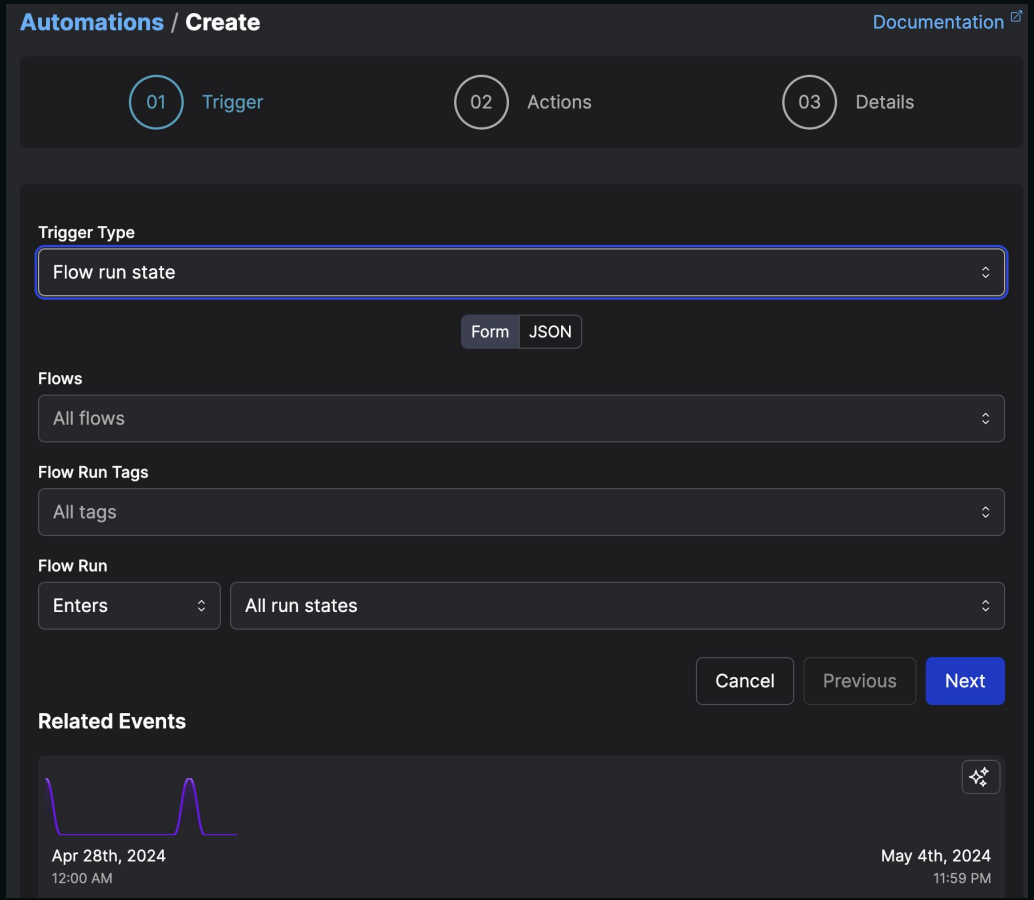
- If a flow run with tag **prod** fails, send an email 📧
- If a data quality check fails, run a deployment to fetch more data 📊
- If a work pool changes state to *Not Ready*, call a webhook 🕸️ 🪝

Send an email when a flow run fails with an Automation

Trigger: Flow run failure
Action: Send an email



Automation trigger: *Flow run state*



Automation action: *Send an email*

Trigger | 02 Actions | 03 Details

Action 1

Action Type

Send an email

Emails

jeff@prefect.io ×

Subject

```
Prefect flow run notification
```

Body

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

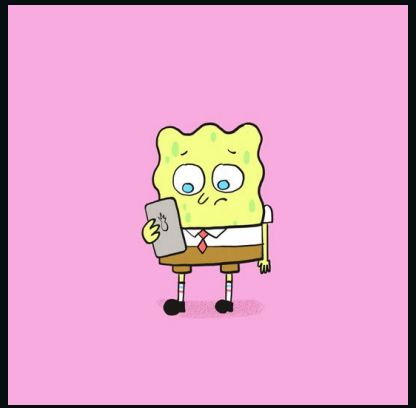
In addition to any fields present on the triggering event, the following objects can be used in notification templates: `flow`, `deployment`, `flow_run`, `work_pool`, `work_queue`, and `metric`.

+ Add Action

Cancel | Previous | Next

Name and save your automation

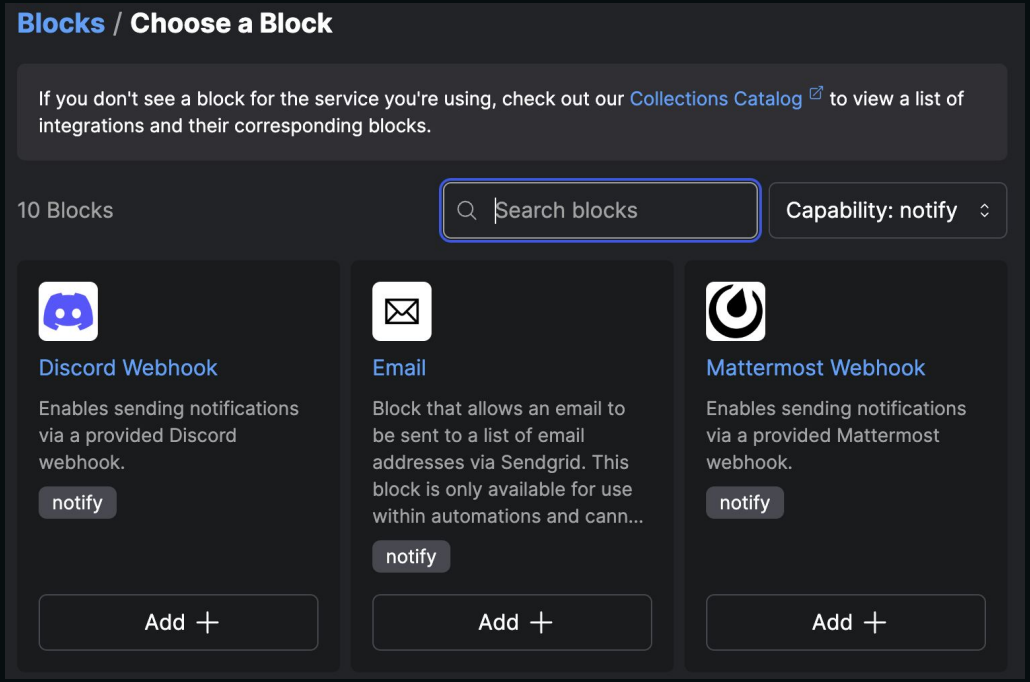
You'll receive an email when a flow run fails!



👆 You can toggle automations off and on

More automation action options (e.g. Slack): *Send a notification*

Create a block with **notify** capability



103 Recap

You've learned about

- Working with data
- Prefect Cloud
- Events
- Automations

Lab 103

103 Lab

- In the UI, make an email automation that fires when flow runs complete
- Then toggle it off
- See the event feed in the UI
- Create a Markdown artifact that prints a weather forecast in a nicely formatted table
- Stretch 1: Create a flow that contains a task that uses caching
- Stretch 2: Change the cache policy, does the task run or not run as you expect?