Custom Providers

by Louis Heredero

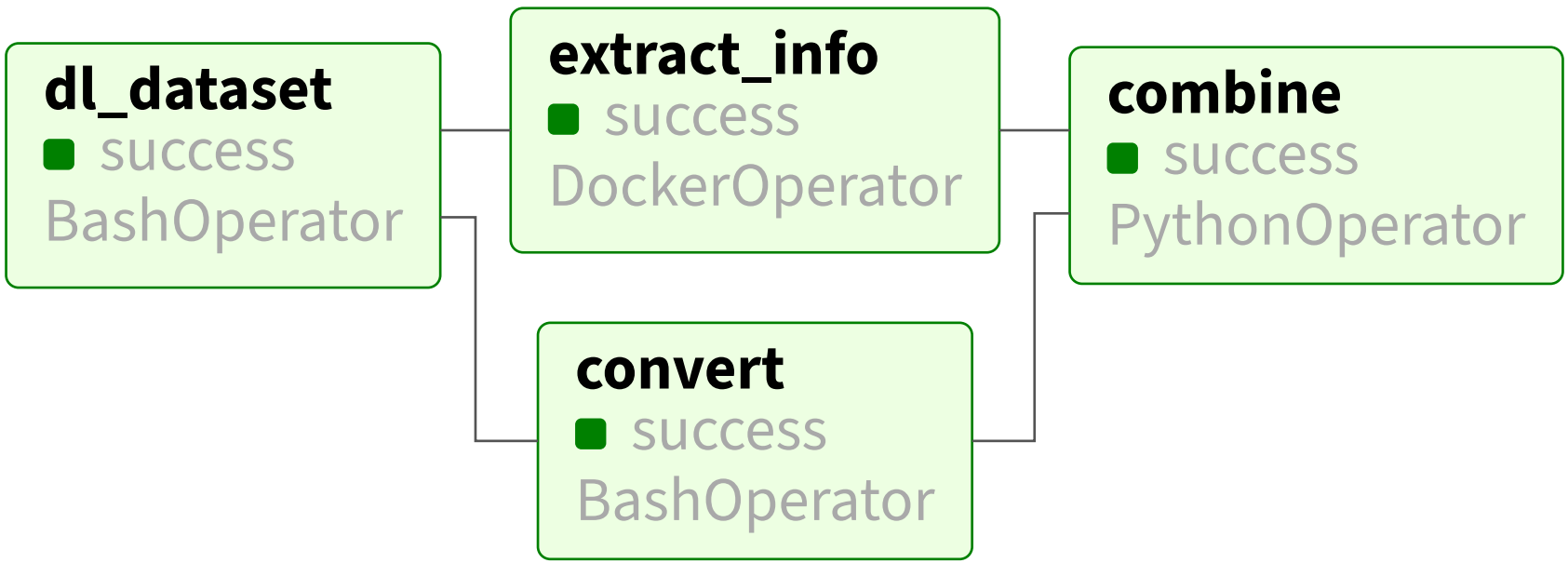
Modularity

Apache Airflow is very powerful thanks to its modularity

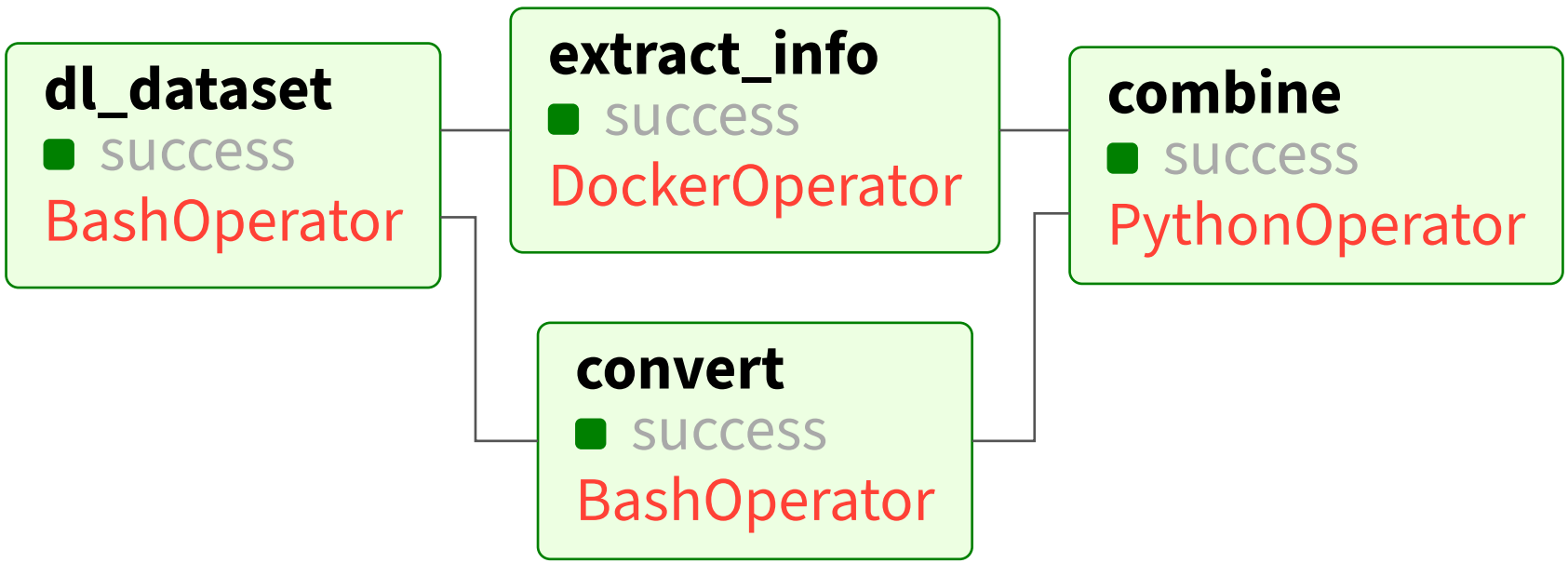
Small components which communicate between each other

- Operations
- Notifications
- Transfers
- Secret backend
- Logs
- And more

Operators


dl_dataset
success
BashOperator
extract_info
success
DockerOperator
combine
success
PythonOperator
convert
success
BashOperator

Operators



Operators are defined by **providers**

Notifiers

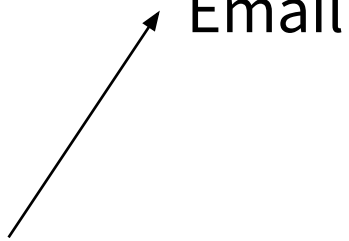
Triggered by events (`*_callback`):

- success
- failure
- skipped
- execute (before)
- retry
- …

Notifiers

Triggered by events (`*_callback`):

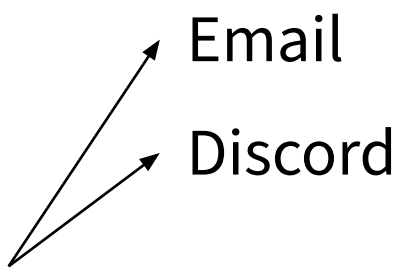
- success
- failure
- skipped
- execute (before)
- retry
- …



Notifiers

Triggered by events (`*_callback`):

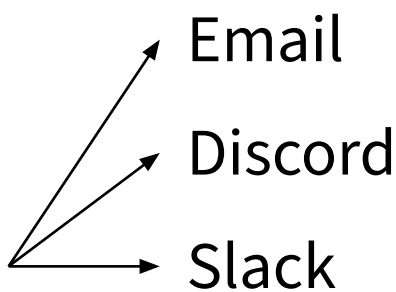
- success
- failure
- skipped
- execute (before)
- retry
- …



Notifiers

Triggered by events (`*_callback`):

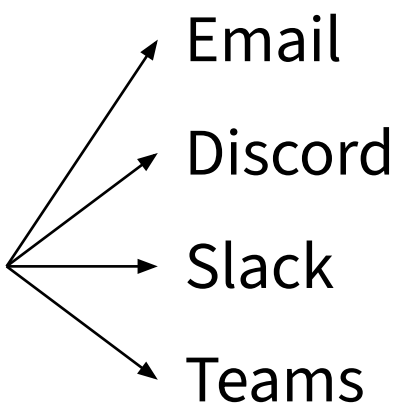
- success
- failure
- skipped
- execute (before)
- retry
- …



Notifiers

Triggered by events (`*_callback`):

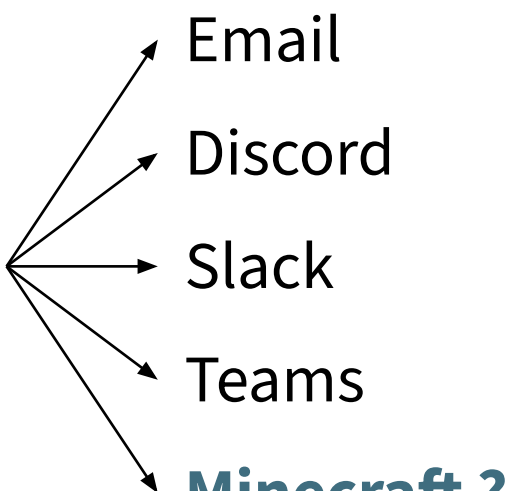
- success
- failure
- skipped
- execute (before)
- retry
- …



Notifiers

Triggered by events (`*_callback`):

- success
- failure
- skipped
- execute (before)
- retry
- …



Implementation

A provider is defined as a Python package

pyproject.toml [T] TOML

```toml
[project]
name = "airflow-provider-minecraft"
description = "An Apache Airflow provider to send notification to a Minecraft server"
readme = "README.md"
requires-python = ">=3.9"
dependencies = [ "apache-airflow" ]
dynamic = [ "version" ]
…                                                    …
[project.entry-points.apache_airflow_provider]
provider_info = "minecraft_provider.__init__:get_provider_info"
…                                                    …
```

Implementation

__init__.py

Python

```python
…
def get_provider_info():
    return {
        "package-name": "airflow-provider-minecraft",
        "name": "Minecraft",
        "description": "A short description",
…
        "hooks": [  # Connection manager
            {
                "integration-name": "Minecraft RCON",
                "python-modules": ["minecraft_provider.hooks.rcon"]
            }
        ],
        "connection-types": [  # UI connection settings
            {
                "hook-class-name": "minecraft_provider.hooks.rcon.RCONHook",
                "connection-type": "rcon"
            }
        ],
        "notifications": [  # Notifiers
            "minecraft_provider.notifications.minecraft.MinecraftNotifier"
        ],
…
```

Implementation

hooks/rcon.py — Python

```python
class RCONHook(BaseHook):
    conn_name_attr = "rcon_conn_id"
    default_conn_name = "rcon_default"
    conn_type = "rcon"
    hook_name = "Minecraft RCON"

    @classmethod
    def get_ui_field_behaviour(cls) -> dict:
        return {"hidden_fields": ["login", "schema", "extra"], "relabeling": {}}

    def __init__(
        self,
        rcon_conn_id: Optional[str] = None,
        host: str = "",
        port: Optional[int] = None,
        password: Optional[str] = None
    ) -> None:
        super().__init__()
        self.rcon_conn_id: Optional[str] = rcon_conn_id
        self.host: str = host
        self.port: Optional[int] = port
        self.password: Optional[str] = password

        if self.rcon_conn_id is not None:
            conn = self.get_connection(self.rcon_conn_id)
            if not self.host and conn.host:
                self.host = conn.host
            if self.port is None:
                self.port = conn.port
            if self.password is None:
                self.password = conn.password

        self.socket: socket.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.connected: bool = False
        self.connect()

    def connect(self) -> bool:
        self.socket.connect((self.host, self.port))
        packet: Packet = Packet(
            PacketType.LOGIN,
            self.password.encode() if self.password is not None else b""
        )
        res: Packet = self.send(packet)
        self.connected = res.request_id == packet.request_id
        if not self.connected:
            self.log.error(f"Could not connect to server: {res.payload.decode()}")
        return self.connected

    def disconnect(self):
        self.socket.close()
        self.connected = False

    def send(self, packet: Packet) -> Packet:
        self.socket.send(packet.to_bytes())
        return Packet.from_bytes(self.socket.recv(4110))

    def send_command(self, cmd: str) -> str:
        res: Packet = self.send(Packet(PacketType.COMMAND, cmd.encode()))
        return res.payload.decode()
```

Implementation

notifications/minecraft.py — Python

```python
class MinecraftNotifier(BaseNotifier):
    template_fields = ("message", "name")

    def __init__(
        self,
        rcon_conn_id: str = "minecraft_rcon_default",
        message: str = "",
        name: Optional[str] = None
    ) -> None:
        super().__init__()
        self.rcon_conn_id: str = rcon_conn_id
        self.message: str = message
        self.name: str = name or "Apache-Airflow"

    @cached_property
    def rcon(self) -> RCONHook:
        return RCONHook(rcon_conn_id=self.rcon_conn_id)

    def _build_command(self) -> str:
        return " ".join([
            "tellraw", "@a", json.dumps([
                f"[{self.name}] ",
                self.message
            ])
        ])

    def notify(self, context):
        cmd: str = self._build_command()
        self.rcon.send_command(cmd)
```

Using our provider

1. Install it in our docker container
2. Import it in a DAG file
3. Use the notifier as a callback

```python
from airflow import DAG
from airflow.operators.bash import BashOperator
from minecraft_provider.notifications.minecraft import MinecraftNotifier

with DAG(dag_id="minecraft-test", schedule_interval=None) as dag:
    task1 = BashOperator(
        task_id="task1",
        bash_command="echo Hello World!",
        on_success_callback=MinecraftNotifier(
            rcon_conn_id="my_server",
            message="Task {{ task_instance.task_id }} finished with
            state {{ task_instance.state }} at {{ ts }}"
        )
    )

    task2 = BashOperator(
        task_id="task2",
        bash_command="exit 1"
    )

    task1 >> task2
```

4. Create a connection in the Web UI
5. Run the DAG