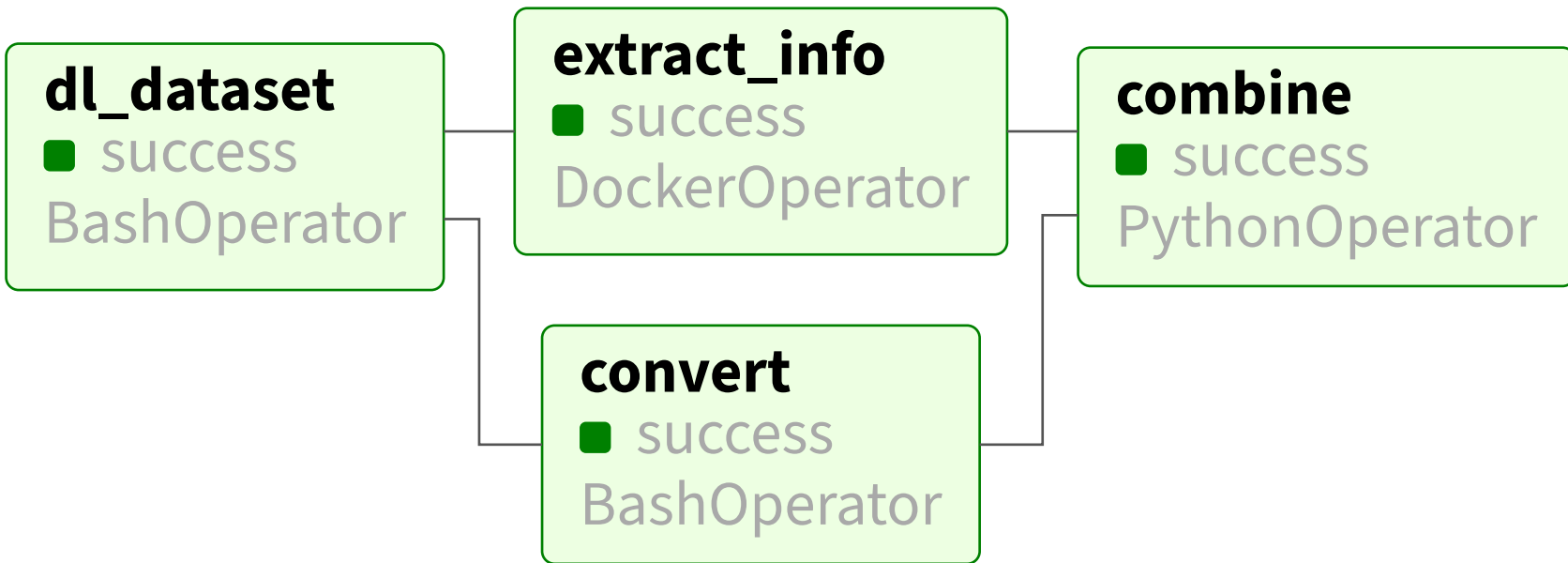# Custom Providers

by Louis Heredero

Modularity

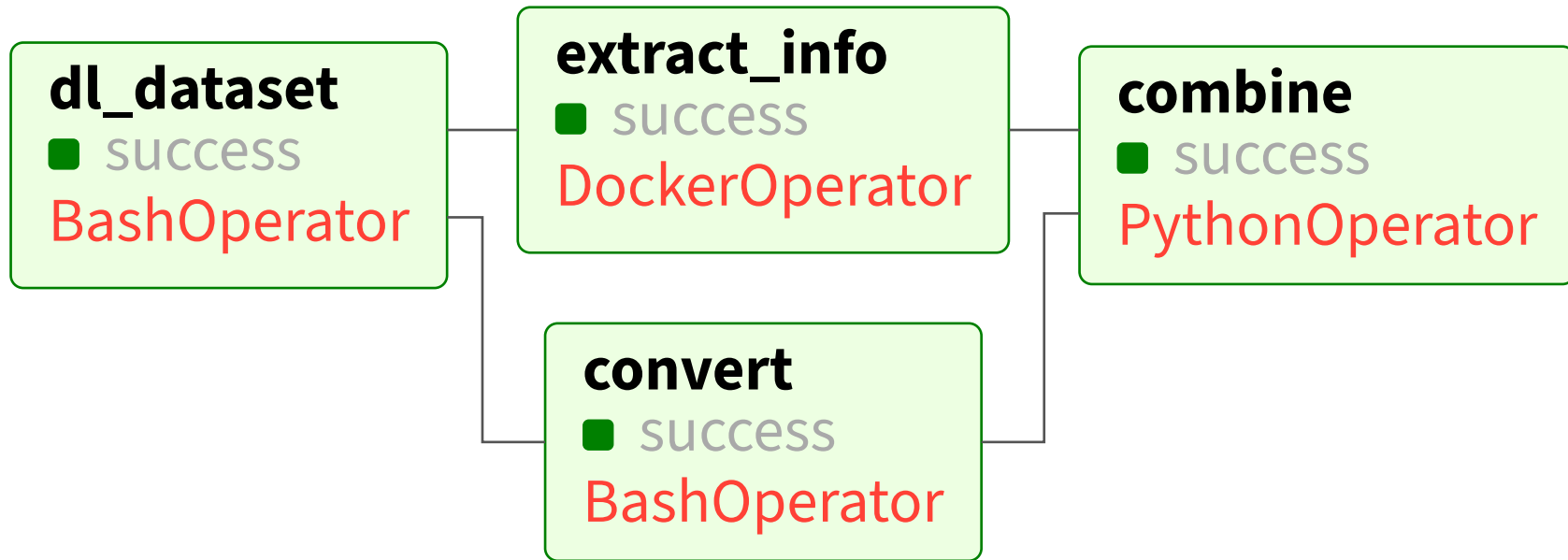Apache Airflow is very powerful thanks to its modularity

Small components which communicate between each other

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

Operators

Operators

Operators are defined by **providers**

Notifiers

Triggered by events (*_callback):

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

Notifiers
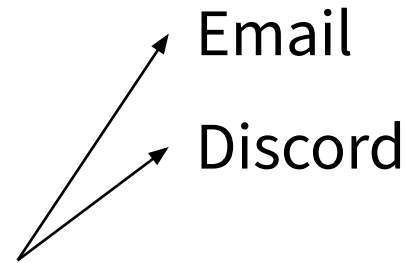
Triggered by events (*_callback):   Email

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

Notifiers

Triggered by events (*_callback):
- success
- failure
- skipped
- execute (before)
- retry
- …

Email

Discord

Notifiers

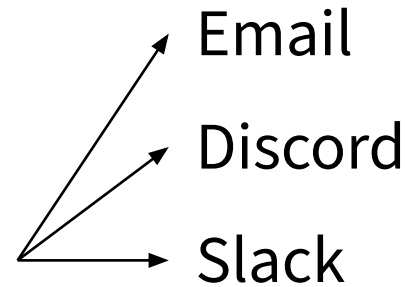Triggered by events (*_callback):
- success
- failure
- skipped
- execute (before)
- retry
- …

Email

Discord

Slack

Notifiers

Triggered by events (*_callback):
- success
- failure
- skipped
- execute (before)
- retry
- …

Email

Discord

Slack

Teams

Notifiers

Triggered by events (*_callback):
- success
- failure
- skipped
- execute (before)
- retry
- …

Email

Discord

Slack

Teams

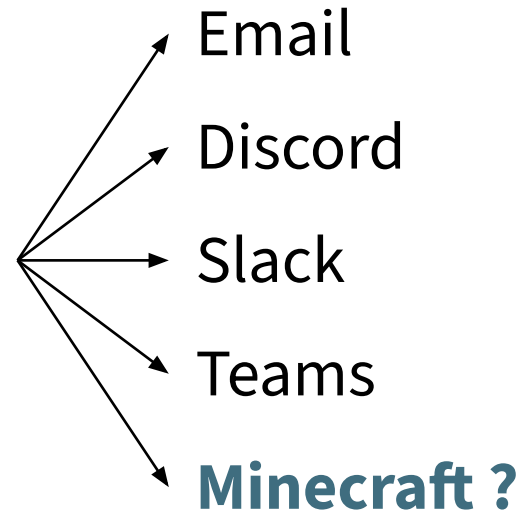**Minecraft ?**

Implementation

A provider is defined as a Python package

```toml
pyproject.toml                                                    [T] TOML

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

Implementation

```python
__init__.py                                                                    🐍 Python
…                                                    …
3   def get_provider_info():
4       return {
5           "package-name": "airflow-provider-minecraft",
6           "name": "Minecraft",
7           "description": "A short description",
…                                                    …
15          "hooks": [  # Connection manager
16              {
17                  "integration-name": "Minecraft RCON",
18                  "python-modules": ["minecraft_provider.hooks.rcon"]
19              }
20          ],
21          "connection-types": [  # UI connection settings
22              {
23                  "hook-class-name": "minecraft_provider.hooks.rcon.RCONHook",
24                  "connection-type": "rcon"
25              }
26          ],
27          "notifications": [  # Notifiers
28              "minecraft_provider.notifications.minecraft.MinecraftNotifier"
29          ],
…                                                    …
```

Implementation

```python
# hooks/rcon.py                                          Python
8   class RCONHook(BaseHook):
9       conn_name_attr = "rcon_conn_id"
10      default_conn_name = "rcon_default"
11      conn_type = "rcon"
12      hook_name = "Minecraft RCON"
13
14      @classmethod
15      def get_ui_field_behaviour(cls) -> dict:
16          return {"hidden_fields": ["login", "schema", "extra"], "relabeling": {}}
17
18      def __init__(
19          self,
20          rcon_conn_id: Optional[str] = None,
21          host: str = "",
22          port: Optional[int] = None,
23          password: Optional[str] = None
24      ) -> None:
25          super().__init__()
26          self.rcon_conn_id: Optional[str] = rcon_conn_id
27          self.host: str = host
28          self.port: Optional[int] = port
29          self.password: Optional[str] = password
30
31          if self.rcon_conn_id is not None:
32              conn = self.get_connection(self.rcon_conn_id)
33              if not self.host and conn.host:
34                  self.host = conn.host
35              if self.port is None:
36                  self.port = conn.port
```

```python
37              if self.password is None:
38                  self.password = conn.password
39
40          self.socket: socket.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
41          self.connected: bool = False
42          self.connect()
43
44      def connect(self) -> bool:
45          self.socket.connect((self.host, self.port))
46          packet: Packet = Packet(
47              PacketType.LOGIN,
48              self.password.encode() if self.password is not None else b""
49          )
50          res: Packet = self.send(packet)
51          self.connected = res.request_id == packet.request_id
52          if not self.connected:
53              self.log.error(f"Could not connect to server: {res.payload.decode()}")
54          return self.connected
55
56      def disconnect(self):
57          self.socket.close()
58          self.connected = False
59
60      def send(self, packet: Packet) -> Packet:
61          self.socket.send(packet.to_bytes())
62          return Packet.from_bytes(self.socket.recv(4110))
63
64      def send_command(self, cmd: str) -> str:
65          res: Packet = self.send(Packet(PacketType.COMMAND, cmd.encode()))
66          return res.payload.decode()
```

Implementation

```python
# notifications/minecraft.py                            Python
7   class MinecraftNotifier(BaseNotifier):
8       template_fields = ("message", "name")
9
10      def __init__(
11          self,
12          rcon_conn_id: str = "minecraft_rcon_default",
13          message: str = "",
14          name: Optional[str] = None
15      ) -> None:
16          super().__init__()
17          self.rcon_conn_id: str = rcon_conn_id
18          self.message: str = message
19          self.name: str = name or "Apache-Airflow"
20
21      @cached_property
22      def rcon(self) -> RCONHook:
23          return RCONHook(rcon_conn_id=self.rcon_conn_id)
24
25      def _build_command(self) -> str:
26          return " ".join([
27              "tellraw", "@a", json.dumps([
28                  f"[{self.name}] ",
29                  self.message
30              ])
31          ])
32
33      def notify(self, context):
34          cmd: str = self._build_command()
35          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
1   from airflow import DAG
2   from airflow.operators.bash import BashOperator
3   from minecraft_provider.notifications.minecraft import MinecraftNotifier
4
5   with DAG(dag_id="minecraft-test", schedule_interval=None) as dag:
6       task1 = BashOperator(
7           task_id="task1",
8           bash_command="echo Hello World!",
9           on_success_callback=MinecraftNotifier(
10              rcon_conn_id="my_server",
11              message="Task {{ task_instance.task_id }} finished with
                state {{ task_instance.state }} at {{ ts }}"
12          )
13      )
14
```

```
15  task2 = BashOperator(
16      task_id="task2",
17      bash_command="exit 1"
18  )
19
20  task1 >> task2
```

4. Create a connection in the Web UI

5. Run the DAG

References

- Apache Airflow tutorial:
  https://airflow.apache.org/docs/apache-airflow-providers/howto/create-custom-providers.html

- Astronomer example provider:
  https://github.com/astronomer/airflow-provider-sample

- Minecraft RCON protocol:
  https://minecraft.wiki/w/RCON