Data Engineering with Dagster — Week 3

Schedules and Sensors

Kicking Off a Pipeline

We have only been launching runs of our pipelines from the CLI or the UI. Ad hoc runs are very helpful, but they are usually not how we would be running things in production. Two other ways to invoke our pipelines are with `schedules` (cron based) and `sensors` (event driven) patterns. The reason we have not talked about these until now is both rely on the Dagster daemon which we introduced last week. Now that we have all the necessary infrastructure to use these processes, we can add in the code to leverage them.

Schedules

A schedule acts as you would expect. You define some cron syntax for when we want our pipeline to run and the job it should be associated with.

```
our_schedule = ScheduleDefinition(job=our_job, cron_schedule="0 0 * * *")
```

This schedule will run `our_job` at midnight every day.

Sensors

A sensor is a little more complicated. A sensor will monitor and wait for changes in an external component. For example, you could create a sensor that looks for new files to be added in a local directory. When a file is added, it would trigger a run of our pipeline:

```
@sensor(job=our_job, minimum_interval_seconds=30)
def local_directory_sensor(context):
    new_files = get_new_files(
        directory=".",
        suffix=".csv"
    )
    if not new_files:
        yield SkipReason("No new files")
        return
    for new_file in new_files:
        yield RunRequest(
            run_key=new_file,
            run_config={
                "ops": {
                    "parameter": {"config": {"some_value": "pasta"}},
                },
            },
        )
```

With this bit of code, we can get a sense of what Dagster is doing under the hood. We have a function `get_new_files()` which returns a list of csv files in the current directory. If no new files are found, the sensor is instructed to skip. If new files are found, we will iterate over the list and launch a job for each file in the list. You will notice a `run_key` is associated with each new file to ensure we do not execute duplicate runs.

The polling process for the sensor is executed by the daemon in background. If a sensor is enabled, the daemon polls our external source. Previous runs of the sensor are recorded in the storage layer based on their run key. For each request, the process will compare the run key against past runs, and if a match is not found, the job will execute.

Summary

The ability to trigger our pipeline in different ways gives us a lot more flexibility in how we design and implement data applications. There are also some advanced ways to use sensors, besides simply triggering a pipeline to begin executing.

Object	Relationship	Description
Schedule	A schedule can be associated with a job. A job can have any number of schedules (or sensors).	A definition of a fixed time to submit a Dagster job.
Sensor	A sensor can be associated with a job. A job can have any number of sensors (or schedules).	A definition to submit a Dagster job based on the change in an external state.

LECTURE PREVIEW

You still might be wondering when it is best to trigger a run using schedules or sensors (and the other ways you can trigger a run). We will discuss use cases in person.