The Essentials of Custom Executor Development

Dennis Ferruzzi

Syed Hussain

What is an Executor?

"Executors are the mechanism by which task instances get run."

Local

Runs inside the Scheduler process

- Debug
- Local
- Sequential

Remote

Scheduler triggers an external worker

- Celery
- Dask
- ECS
- Kubernetes
- etc

Why Create A Custom Executor?

- Are tasks executed locally or remote?
- Noisy neighbors
- Task startup time
- Preferred cloud provider
- Location-sensitive restrictions

Six Main Parts of an Executor

- Start
- Sync running tasks
- Try to run new tasks
- Try to adopt task instances [OPTIONAL]
- Handle failed tasks
- End (graceful exit) / Terminate (forced exit)

Real-world Example

AWS ECS Executor

Each task that Airflow schedules for execution is run within its own ECS container.

Advantages

- Resources like CPU, memory and disk are isolated to each individual task
- You can build different container images per task
- Compute resources only exist for the lifetime of the Airflow task itself

Disadvantages

- Every task runs on a separate container, which takes time
- Config must be consistent across deployment
- Requires an existing ECS Cluster

AWS ECS Executor

What is it actually doing?

- Start
  - Ensures ECS Cluster is active and responding to requests

```python
    def start(self):
        """Call this when the Executor is run for the first time by the scheduler."""
        check_health = conf.getboolean(...)

        if not check_health:
            return

        self.log.info("Starting ECS Executor and determining health...")
        try:
            self.check_health()
        except AirflowException:
            self.log.error("Stopping the Airflow Scheduler from starting until the issue is resolved.")
            raise
```

```python
    def check_health(self):
        """Make a test API call to check the health of the ECS Executor...."""
        success_status = "succeeded."
        status = success_status

        try:
            invalid_task_id = "a" * 32
            self.ecs.stop_task(cluster=self.cluster, task=invalid_task_id)

            ...
            status = "failed for an unknown reason. "
        except ClientError as ex:
            error_code = ex.response["Error"]["Code"]
            error_message = ex.response["Error"]["Message"]

            if ("InvalidParameterException" in error_code) and ("task was not found" in error_message):
                # This failure is expected, and means we're healthy
                pass
            else:
                # Catch all for unexpected failures
                status = f"failed because: {error_message}. "
        except Exception as e: ...
        finally:
            msg_prefix = "ECS Executor health check has %s"
            if status == success_status:
                self.IS_BOTO_CONNECTION_HEALTHY = True
                self.log.info(msg_prefix, *args: status)
            else:
                msg_error_suffix = ( ... )
                raise AirflowException(msg_prefix % status + msg_error_suffix)
```

AWS ECS Executor

What is it actually doing?

- Start
  - Ensures ECS Cluster is active and responding to requests
- Sync
  - Update state on all running tasks

```python
    def sync(self):
        if not self.IS_BOTO_CONNECTION_HEALTHY:
            exponential_backoff_retry(...)
            if not self.IS_BOTO_CONNECTION_HEALTHY:
                return
        try:
            self.sync_running_tasks()
            self.attempt_task_runs()
        except (ClientError, NoCredentialsError) as error:...

        except Exception:
            # We catch any and all exceptions because otherwise they would bubble
            # up and kill the scheduler process
            self.log.exception( msg: "Failed to sync %s", *args: self.__class__.__name__)
```

```python
    def sync_running_tasks(self):
        """Check and update state on all running tasks."""
        all_task_arns = self.active_workers.get_all_arns()
        if not all_task_arns:
            self.log.debug("No active Airflow tasks, skipping sync.")
            return

        describe_tasks_response = self.__describe_tasks(all_task_arns)

        self.log.debug( msg: "Active Workers: %s",  *args: describe_tasks_response)

        if describe_tasks_response["failures"]:
            for failure in describe_tasks_response["failures"]:
                self.__handle_failed_task(failure["arn"], failure["reason"])

        updated_tasks = describe_tasks_response["tasks"]
        for task in updated_tasks:
            self.__update_running_task(task)

```

AWS ECS Executor

What is it actually doing?

- Start
  - Ensures ECS Cluster is active and responding to requests
- Sync
  - Update state on all running tasks
- Try to run new tasks
  - Loads all submitted tasks and attempts to run them

```python
332     def attempt_task_runs(self):
333         """Take tasks from the pending_tasks queue, and attempts to find an instance to run it on...."""
342 ...
344         for _ in range(queue_len):
345             ecs_task = self.pending_tasks.popleft()
346             ...
352             if timezone.utcnow() < ecs_task.next_attempt_time:
353                 self.pending_tasks.append(ecs_task)
354                 continue
355             try:
356                 run_task_response = self._run_task(task_key, cmd, queue, exec_config)
357             except NoCredentialsError: ...
360             except ClientError as e:
361                 error_code = e.response["Error"]["Code"]
362                 if error_code in INVALID_CREDENTIALS_EXCEPTIONS:
363                     self.pending_tasks.append(ecs_task)
364                     raise
365                 failure_reasons.append(str(e))
366             except Exception as e: ...
372             else:
373                 ...
376                 if run_task_response["failures"]:
377                     failure_reasons.extend([f["reason"] for f in run_task_response["failures"]])
378
379             if failure_reasons: ...
404             elif not run_task_response["tasks"]: ...
414             else:
415                 task = run_task_response["tasks"][0]
416                 self.active_workers.add_task(task, task_key, queue, cmd, exec_config, attempt_number)
417                 try:
418                     self.running_state(task_key, task.task_arn)
419                 except AttributeError: ...
```

AWS ECS Executor

What is it actually doing?

- Start
  - Ensures ECS Cluster is active and responding to requests
- Sync
  - Update state on all running tasks
- Try to run new tasks
  - Loads all submitted tasks and attempts to run them
- Try to adopt task instances
  - Check ECS Cluster for existing running tasks

```python
520 def try_adopt_task_instances(self, tis: Sequence[TaskInstance]) -> Sequence[TaskInstance]:
521     """Adopt task instances which have an external_executor_id (the ECS task ARN)...."""
526     with Stats.timer("ecs_executor.adopt_task_instances.duration"):
527         adopted_tis: list[TaskInstance] = []
528
529         if task_arns := [ti.external_executor_id for ti in tis if ti.external_executor_id]:
530             task_descriptions = self.__describe_tasks(task_arns).get("tasks", [])
531
532             for task in task_descriptions:
533                 ti = next(ti for ti in tis if ti.external_executor_id == task.task_arn)
534                 self.active_workers.add_task( ... )
542                 adopted_tis.append(ti)
543
544         if adopted_tis:
545             tasks = [f"{task} in state {task.state}" for task in adopted_tis]
546             task_instance_str = "\n\t".join(tasks)
547             self.log.info( ... )
552
553         not_adopted_tis = [ti for ti in tis if ti not in adopted_tis]
554         return not_adopted_tis
555
```

AWS ECS Executor

What is it actually doing?

- Start
  - Ensures ECS Cluster is active and responding to requests
- Sync
  - Update state on all running tasks
- Try to run new tasks
  - Loads all submitted tasks and attempts to run them
- Try to adopt task instances
  - Check ECS Cluster for existing running tasks
- Handle failed tasks
  - Attempt any retries if applicable, otherwise log the error and remove the task from queue

```
289     def __handle_failed_task(self, task_arn: str, reason: str):
290         """If an API failure occurs, the task is rescheduled...."""
298         ...
303         failure_count = self.active_workers.failure_count_by_key(task_key)
304         if int(failure_count) < int(self.__class__.MAX_RUN_TASK_ATTEMPTS):
305             self.log.warning( ... )
313             self.pending_tasks.append(
314                 EcsQueuedTask( ... )
322             )
323         else:
324             self.log.error( ... )
329             self.fail(task_key)
330         self.active_workers.pop_by_key(task_key)
```

AWS ECS Executor

What is it actually doing?

- Start
  - Ensures ECS Cluster is active and responding to requests
- Sync
  - Update state on all running tasks
- Try to run new tasks
  - Loads all submitted tasks and attempts to run them
- Try to adopt task instances
  - Check ECS Cluster for existing running tasks
- Handle failed tasks
  - Attempt any retries if applicable, otherwise log the error and remove the task from queue
- End (graceful exit) / Terminate (forced exit)

```python
    def end(self, heartbeat_interval=10):
        """Wait for all currently running tasks to end, and don't launch any tasks."""
        try:
            while True:
                self.sync()
                if not self.active_workers:
                    break
                time.sleep(heartbeat_interval)
        except Exception:
            # We catch any and all exceptions because otherwise they would bubble
            # up and kill the scheduler process.
            self.log.exception( msg: "Failed to end %s", *args: self.__class__.__name__)

    def terminate(self):
        """Kill all ECS processes by calling Boto3's StopTask API."""
        try:
            for arn in self.active_workers.get_all_arns():
                self.ecs.stop_task(
                    cluster=self.cluster, task=arn, reason="Airflow Executor received a SIGTERM"
                )
            self.end()
        except Exception:
            # We catch any and all exceptions because otherwise they would bubble
            # up and kill the scheduler process.
            self.log.exception( msg: "Failed to terminate %s", *args: self.__class__.__name__)
```

AWS ECS Executor

What is it actually doing?

- Start
  - Ensures ECS Cluster is active and responding to requests
- Sync
  - Update state on all running tasks
- Try to run new tasks
  - Loads all submitted tasks and attempts to run them
- Try to adopt task instances
  - Check ECS Cluster for existing running tasks
- Handle failed tasks
  - Attempt any retries if applicable, otherwise log the error and remove the task from queue
- End (graceful exit) / Terminate (forced exit)

Questions?

Executor Source

These Slides