Program Status Based Scheduling

Checklist

- [x] User Stories Documented
- [x] User Stories Reviewed
- [x] Design Reviewed
- [ ] APIs reviewed
- [ ] Release priorities assigned
- [ ] Test cases reviewed
- [ ] Blog post

Introduction

CDAP data pipelines currently allows time based triggering of pipelines. Having only the time based trigger to start the pipelines has a few drawbacks - this doesn't allow for logically separating and operating a lot of smaller pipelines which can depend on each other. Even if the pipeline can be separated into a smaller set of pipelines which can be triggered periodically, the first step in the pipeline has to check if the rest of the pipeline needs to be executed. There is a need to trigger pipelines based on states of a different pipeline.

Use Cases

FTP Download use-case

1. As an ETL developer, I would like to process bloomberg and reuters feeds. This feed is downloaded from an FTP site and is already being processed by another team and stored in a staging area. I do not wish to download the feed again, but would like to run my pipeline as soon as the feed data is available in the staging area. To run my pipeline, I would like to know the list of files that are freshly downloaded.
2. As an ETL developer, consuming the bloomberg and reuters feed, I do not wish to reprocess the same feed multiple times. I would like to process the feed only if the upstream pipeline has successfully completed.

Gold dataset consumption use-case

1. As an ETL developer, I would like to read a golden data set that is created from another team. The gold data is read from a database incrementally, cleansed, anonymized and standardized by our IT ops team via ETL process. I wish to consume the data that is generated by the IT ops team and do not wish to rerun the standardization process myself since that might change. I only wish to consume the golden data set that is created for my purposes and as an input for my pipeline. I would like to know the directory location of the golden data set. I would like to run the pipeline as soon as the new dataset is available.
2. As an ETL developer, I would like to know what feeds were responsible for creating the golden dataset, since I have to look up metadata for some feeds in my pipeline. I have configured the feeds that are present in the golden data set as a runtime argument.
3. As an ETL developer, I would like to reprocess my pipeline that consumes golden dataset automatically up to two times if it has failed. If the re-processing has failed more than twice, then I do not wish to re-process further.

High Level Requirements

Platform

1. Platform should allow event notification for all program states
2. Event notification (EN) should be published to a system topic in TMS
3. Event notification should have a well defined event structure
4. Event notification should be published for all types of programs on Completion, failure and if killed by YARN
5. Event notification should have a payload to pass in information from one program to another
6. Pipelines should be configured to be triggered based on event from another program
7. Pipelines should be triggered based on either
    1. Another Pipeline state
    2. Another Pipeline state and payload
        1. Conditions can be specified either if certain key exists, or if key equals a value or if key matches a condition (ex: runcount < 2)

Data pipeline

1. Users should be able to configure what parameters are emitted in the event payload from a pipeline. Hence the plugin must expose what parameters are emitted. Example: file-path, files-processed
2. Users configuring a pipeline (Golden Feed Consumer) should be able to select the fields from upstream pipeline (Gold Feed Generator) and set the the run time arguments needed for the current pipeline (Golden Feed Consumer)

Plugin

1. Plugins should expose what parameters it can expose to the event notification
2. The parameters that can be exposed is defined in pipeline configuration stage (pipeline studio) and not at runtime

UI

1. Capabilities to add event based triggers in studio mode
2. Capabilities to specify condition for triggering pipeline (ex: files.processed from Gold Feed Generator should not be empty)
3. Capabilities to select what fields are published in event notification

Design

Assumptions

- When we say "program" this refers to any program that can be scheduled. While the primary use case may be starting other pipelines after one pipeline finishes, MapReduce and Spark jobs can also be started.
- We will only allow programs to be triggered with any combination of the following program statuses: ProgramStatus.COMPLETED, ProgramStatus.FAILED, and ProgramStatus.KILLED.
    - There is no known use case for triggering a program when another program is ProgramStatus.INITIALIZING or ProgramStatus.RUNNING, so this will be restricted to prevent unexpected behavior.
- All combinations of existing constraints are possible.

Approach

- Implementing Program Status Based Scheduling is a two step process:
  1. We need to determine when the program status has changed.
  2. Based on the program status change, the scheduler must schedule the program.

Both of these steps will be covered below.

1. Determining a Program State Change:

- Whenever a program status changes, we can send program status based notifications to the Transactional Messaging System (TMS).
- These messages can be emitted to topic `program.status.event.topic` within TMS.
- The approach here will be to send a notification whenever the program status changes are persisted to the store. This occurs in each program runner.

Concerns:

- What happens if a YARN container crashes?
    - **Response:** We already monitor running programs and mark the status as failed when it crashes.

2. Scheduling Based on the Program Status Change:

- The scheduler will subscribe to the program status notifications with topic `program.status.event.topic`
- Based on the notification received, it will add a new job to the job queue.
- The existing scheduler implementation will handle data persistence and starting the job when all of its constraints have been satisfied, so the rest of the scheduler implementation remains unchanged and will work with the proposed changes.

Program Status Notification Payload Design:

Every time a program's status changes, the following will be sent in a payload through TMS (not used for scheduling, but the inherent design):

- programRunId <the specific run of a program>
- twillRunId: <only in distributed mode>
- programRunStatus <the updated run status of the program>
- stateChangeTime: <when the program has transitioned to starting, running, or 'terminal' state
- userArguments and systemArguments: <only when the program has transition to starting>

Program Status Scheduling Notification Payload Design:

The program status notifications received from TMS will be forwarded to the scheduler if and only if that program status notification can trigger another program. The following parameters will be passed in *this* notification between a triggering program and a triggered program:

- all userArguments (runtimeArguments) of the triggering program with no renaming (renaming to be done later)
- all USER-scoped workflow token key-value pairs of the triggering program with no renaming (renaming to be done later)
- triggering program run properties (programRunId, startTs, runTs, stopTs)

API changes

New Programmatic APIs

New Java APIs introduced (both user facing and internal)

Triggers

```
public class ProgramStatusTrigger extends Trigger {
  private final ProgramId programId;
  private final String key;
  private final Predicate<Value> condition;
  private final Set<ProgramStatus> statuses; // Use Set since it
removes duplicates.

  public ProgramStatusTrigger(ProgramId programId, @Nullable String
key, @Nullable Predicate<Value> condition, ProgramStatus... statuses) {
        this.programId = programId;
         this.key = key;
        this.condition = condition;
        this.statuses = new HashSet<ProgramStatus>(Arrays.asList
(statuses));
  }
}
```

External Methods (in ScheduleBuilder)

Trigger Definition

```
public class ScheduleBuilder {
        ...

        public ScheduleCreationBuilder triggerOnProgramStatus(String
programNamespace, String application, String applicationVersion,

ProgramType programType, String program,

ProgramStatus... programStatuses);

        // Assume same application version as starting program
        public ScheduleCreationBuilder triggerOnProgramStatus(String
programNamespace, String application, ProgramType programType,

String program, ProgramStatus...  programStatuses);

        // Assume same application name and version as starting program
        public ScheduleCreationBuilder triggerOnProgramStatus(String
programNamespace, ProgramType programType,

String program, ProgramStatus... programStatuses);

        // Assume same namespace, application, and application version
as starting program
        public ScheduleCreationBuilder triggerOnProgramStatus
(ProgramType programType, String program,

ProgramStatus...  programStatuses);
};
```

There needs to be a way to restrict these defined constraints to only schedules triggered by a program status (specific program status constraints)

Deprecated Programmatic APIs

There are none.

New REST APIs

Path	Method	Description	Request Body	Response Code	Response

/v3/namespaces
/<namespace-id>/apps
/<app-id>/schedules
/<schedule-id>
	PUT	To add a schedule for a program to an application	The body is entirely the same as the existing API endpoint already documented, but the trigger specified in the body is different.
			
			Request
			
			{
			"name": "<name of the schedule>",
			"description": "<schedule description>",
			"program": {
			"programName": "<name of the program>",
			"programType": "WORKFLOW"
			},
			"properties": {
			...
			},
			"constraints": [
			{
			...
			},
			...
			],
			"trigger": {
			"type": "PROGRAM_STATUS",
			"statuses": ["<status of the program; derived from ProgramStatus, only COMPLETED, FAILING, KILLED>"],
			"program": {
			"namespace": "<namespace of the program>",
			"application": "<application of the program>",
			"applicationVersion": "<application version of the program>",
			"programType": "<type of the program>",
			"programName": "<name of the program>",
			}
			}
				200 - On success
				
				404 - When application is not available
				
				409 - Schedule with the same name already exists
				
				500 - Any internal errors	

Deprecated REST API

- There are no API routes to be deprecated.

CLI Impact or Changes

- ```
  add program schedule <schedule-name> for <program-type> <program> [version <version>]
  [description "optional"] after program <program-name> type <program-type> status <program-status>
  [or <program-status>] [namespace <program-namespace>] [application <program-application>]
  [version <program-application-version>] [concurrency <concurrency>] [properties "optional map of
  properties"]

  +-> PUT /v3/namespaces/default/apps/<app-id>/versions/<app-version>/schedules/<schedule-id>

  update program schedule <schedule-name> for <program-type> <program> [version <version>]
  [description "optional"] after program <program-name> type <program-type> status <program-status>
  [or <program-status>] [namespace <program-namespace>] [application <program-application>]
  [version <program-application-version>] [concurrency <concurrency>] [properties "optional map of
  properties"]

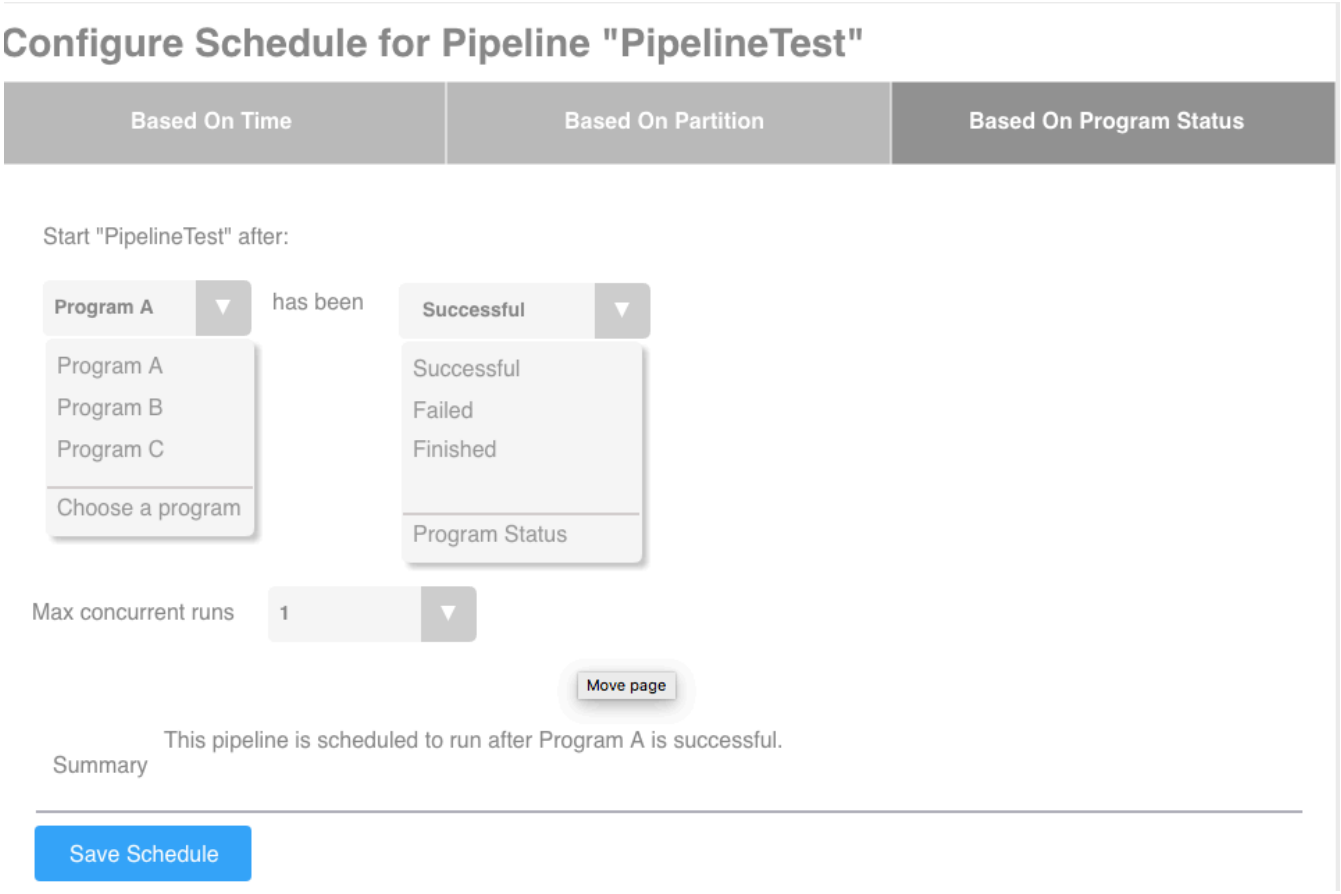
  +-> POST /v3/namespaces/default/apps/<app-id>/versions/<app-version>/schedules/<schedule-id>
  /update
  ```

- The namespace, application, and applicationVersion for the triggering program can all be optional in the CLI because they will inherit from the program that is being scheduled if they are not included.

UI Impact or Changes

- Add ProgramSchedule options to the "Schedule" panel, along with a dropdown for pipeline selection and a dropdown for program status selection

Potential Mockup:



Note that the program statuses in screenshot should be a checkbox, allowing you to select multiple statuses.
  ```

Security Impact

- Can a program A be triggered by a program B if the user running program A does not have read access to program B?
- If we allow cross namespace triggers, how will we make this secure?

Impact on Infrastructure Outages

- Since this change does not change the existing Scheduler architecture, the outage impact is the same as what we have today.
- Any authorization added to TMS will also affect this feature.

Test Scenarios

Test ID	Test Description	Expected Results
1	Configure a schedule to run every 5 minutes and always complete successfully. Configure a second schedule to run if the first schedule has succeeded.	Both schedules run and finish successfully.
2	Configure a schedule to fail. Configure a second schedule to run only if the first schedule has finished and has written the key-value pair "key", "value" in the workflow token.	The first schedule should run. The second schedule should run and see "key", "value" in the workflow token.
3	Configure a schedule to run for 10 seconds before it completes. Configure a second schedule to run only if the first schedule has succeeded. 500 milliseconds before the first schedule completes, delete the second schedule.	The first schedule should run and finish successfully.
4	Configure a schedule to run once and start at 9:50 PM. This schedule should finish in less than 10 minutes. Configure a second schedule to run only between 10:00 PM and 11:00 PM when the first schedule has been completed successfully.	The first schedule should run and finish before 10:00 PM. The second schedule should not run

Open Questions

1. Is there any meaning to starting a program manually (i.e by a user) that is dependent on other pipelines / programs? How should this behavior be handled?

Releases

Release 4.3.0

Related Work

Future work