Scheduling Tasks With Cron for Java

Note: This page uses Cron with the XML cron configuration file, which you use if you use App Engine SDK tooling, such as `appcfg`. If you use Cloud SDK-based tooling, such as the `gcloud` command line, or Cloud SDK-based Maven (https://cloud.google.com/appengine/docs/standard/java/tools/using-maven), Gradle (https://cloud.google.com/appengine/docs/standard/java/tools/gradle), Eclipse (https://cloud.google.com/eclipse/docs/), or IntelliJ (https://cloud.google.com/tools/intellij/docs/) plugins, refer to the YAML (https://cloud.google.com/appengine/docs/standard/java/config/cron-yaml) version of this page.

The App Engine Cron Service allows you to configure regularly scheduled tasks that operate at defined times or regular intervals. These tasks are commonly known as *cron jobs*. These cron jobs are automatically triggered by the App Engine Cron Service. For instance, you might use a cron job to send out an email report on a daily basis, or to update some cached data every 10 minutes, or refresh summary information once an hour.

A cron job makes an HTTP `GET` request to a URL as scheduled. The handler for that URL executes the logic when it is called. A cron job request is subject to the same limits as those for push task queues (https://cloud.google.com/appengine/docs/quotas#Task_Queue).

Creating a cron job

1. Create the `cron.xml` file in the `WEB-INF/` directory of your application (alongside `appengine-web.xml`).

2. Add one or more **<cron>** entries to your file and define the necessary elements for your job, including the required **<url>** and **<schedule>** elements.

   The following example creates a basic cron job that runs daily:

```
<?xml version="1.0" encoding="UTF-8"?>
<cronentries>
  <cron>
    <url>/tasks/summary</url>
    <target>beta</target>
    <description>daily summary job</description>
    <schedule>every 24 hours</schedule>
  </cron>
</cronentries>
```

   The target specification is optional and is the name of a service/version. If present, the target is prepended to your app's hostname, causing the job to be routed to that service/version. If no target is specified, the job will run in the versions of the **default** service that are configured for traffic.

3. Create a handler for the cron job URL. The handler should execute any tasks that you want scheduled. The handler should respond with an HTTP status code between 200 and 299 (inclusive) to indicate success. Other status codes can be returned and can be used to trigger.

The handler can be as simple as a Servlet in the app. The Servlet URL mapping in **web.xml** should be the same as the cron job URL. For more information about the syntax and parameters of the cron.xml, see the cron.xml reference (https://cloud.google.com/appengine/docs/standard/java/config/cronref).

Testing cron jobs in the development server

The local development server doesn't automatically run your cron jobs. You can make requests directly to your cron job's URL to test your functionality. You can use your local cron or scheduled tasks interface to trigger the URLs of your jobs with curl (http://curl.haxx.se/) or a similar tool.

Deploying cron jobs

Option 1: Upload your whole application

To upload your whole application, which also updates the Cron Service with the entries from your `cron.xml` file, run the following command:

```
./appengine-java-sdk/bin/appcfg.sh -A your-app-id -V app-version updat
```

Option 2: Upload only your cron updates

To update just the cron configuration without uploading the rest of the application, run the following command:

```
./appengine-java-sdk/bin/appcfg.sh -A your-app-id -V app-version updat
```

Deleting all cron jobs

To delete all cron jobs:

1. Edit the contents of the `cron.xml` file to:

```
<?xml version="1.0" encoding="UTF-8"?>
<cronentries/>
```

2. Deploy the `cron.xml` file to App Engine.

Retrying cron jobs that fail

If a cron job's request handler returns a status code that is not in the range 200–299 (inclusive) App Engine considers the job to have failed. By default, failed jobs are not retried unless a 503 status code is returned, in which case it is retried every minute until it succeeds or returns a 200-299 status code.

To set failed jobs to be retried:

1. Include a `retry-parameters` block in your `cron.xml` file.

2. Choose and set the retry parameters (https://cloud.google.com/appengine/docs/standard/java/config/cronref#retry) in the **`retry-parameters`** block.

   For example, this sample **`cron.xml`** file contains a single cron job that is configured to retry up to five times (the default) with a starting backoff of 2.5 seconds that doubles each time.

```
<cronentries>
  <cron>
    <url>/retry</url>
    <description>Retry on jsdk</description>
    <schedule>every 10 minutes</schedule>
    <retry-parameters>
      <min-backoff-seconds>2.5</min-backoff-seconds>
      <max-doublings>5</max-doublings>
    </retry-parameters>
  </cron>
</cronentries>
```

Learn more about the cron retry options (https://cloud.google.com/appengine/docs/standard/java/config/cronref#retry)

Securing URLs for cron

You can prevent users from accessing URLs used by scheduled tasks by restricting access to administrator accounts. Scheduled tasks can access admin-only URLs. You can read about restricting URLs at Security and Authentication (https://cloud.google.com/appengine/docs/standard/java/config/webxml#Security_and_Authentication)

. An example you would use in **`web.xml`** to restrict everything starting with **`/cron/`** to admin-only is:

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>cron</web-resource-name>
        <url-pattern>/cron/*</url-pattern>
    </web-resource-collection>
```

```
    <auth-constraint>
        <role-name>admin</role-name>
    </auth-constraint>
</security-constraint>
```

Note: While cron jobs can use URL paths restricted with `<role-name>admin</role-name>`, they *cannot* use URL paths restricted with `<role-name>*</role-name>` because cron scheduled tasks are not run as any user. The `admin` restriction is satisfied by the inclusion of the `X-Appengine-Cron` header described below.

For more on the format of `web.xml`, see the documentation on the deployment descriptor (https://cloud.google.com/appengine/docs/standard/java/config/webxml).

To test a cron job, sign in as an administrator and visit the URL of the handler in your browser.

Requests from the Cron Service will also contain a HTTP header:

```
X-Appengine-Cron: true
```

The `X-Appengine-Cron` header is set internally by Google App Engine. If your request handler finds this header it can trust that the request is a cron request. If the header is present in an external user request to your app, it is stripped, except for requests from logged in administrators of the application, who are allowed to set the header for testing purposes.

Google App Engine issues Cron requests from the IP address `0.1.0.1`.

Calling Google Cloud Endpoints

You cannot specify a Google Cloud Endpoint (https://cloud.google.com/appengine/features/#Endpoints) in the `url` field of a cron job. If you want your cron job to call a Google Cloud Endpoint, issue a request to a target that is

served by a handler in your app, and call the endpoint class and method from the handler code.

Viewing cron jobs in the GCP Console

The GCP Console Task queues page (https://console.cloud.google.com/project/_/appengine/taskqueues) has a tab that shows the tasks that are running cron jobs.

You can also visit the Logs page (https://cloud.google.com/appengine/docs/developers-console/#logs) see when cron jobs were added or removed.
