Codelab: Task Queues and Cron

Updated: March 20, 2015

This exercise leads you through creating an App Engine application that illustrates usage of the Task Queues and Cron Jobs.

Contents

What You Will Learn

You will learn:

- How to create, configure, and use push queues
- How to setup and administer cron jobs

What You Will Do

Push Queues

- Use a push queue to send out email for newly created conference

- When a conference is created, add a task to the task queue to notify all users about the new conference
- Write the handler to notify the appropriate users.

Cron Jobs

- Set up a cron job that runs daily to send out an email about conferences

Get Set Up

1. If you need to deploy your application to App Engine production, you will need to have an application ID (project ID). If you already have a project to test, you can skip this section. Otherwise, create a new project as follows:

a. Visit Google Developers Console in your web browser, and click **Create Project**.
b. Supply an appropriate project name and accept the project ID that is auto-generated for you. Please note down the application id that has been generated. You will need that if you deploy the application.
c. Click **Create**
d. Enable billing.

Creating the Application

Here is what we are going to do:

1. We are going to extend the application that we created in the Datastore Introduction hands-on exercise.
2. To recap the Module 6 hands-on , this is what we did:
    a. We created a web application that created a **Conference** entity, with attributes and data types as follows:
    b. Conference Title (**title)** : String
    c. City in which the conference will be held (**city)** : String
    d. Maximum number of Attendees allowed (**maxAttendees**) : Integer
    e. Conference Start Date (**startdate**) : Date
    f. Conference End Date (**enddate**) : Date
    g. A HTML Web Form (**Create Conference Web Form)** that will provide a simple data entry mechanism for the above Conference entity attributes. On submission of the form, a handler will extract out the HTTP request parameters, create the Datastore entity (**Conference)** and **persist** it.
    h. We will then redirect the form back to the original Web Form, where we shall use the Datastore Query API to retrieve all the current Conference entities present in the Datastore.

What we shall be doing in this hands-on exercise is the following:

1. We would like to send out an email to all registered users of the application, whenever we create a new Conference (Step 2.g) above. Instead of introducing the code to send out email (or

any other task as you see fit), we do not want the same HTTP Request to do any tasks that could be long running. **So we will introduce a Task Queue Handler here that will consume a task from the Task Queue and do the work i.e. send out email.**

2. A Cron Job is a task that is scheduled at regular intervals. We shall configure a Cron Job that will send out email to all registered users. We will simply demonstrate how to configure a Configure Job, write the Handler for it and see it work.

Do note that we will not be sending out actual email in the code that we shall write for this exercise. We will simply print out a message for that. You can take up the additional exercise of integrating the App Engine Mail API. Refer to the Mail API documentation for (Java) and (Python).

You will need to work with the final project files that we have for the Datastore Introduction Hands-on Exercises. They are available here:

- Java Datastore Basics Project ZIP file
- Python Datastore Basics Project ZIP file

You can also clone the projects directly from GitHub:

```
git clone https://github.com/GoogleCloudPlatformTraining/cp300-gae-datastore-java.git
```

OR

```
git clone https://github.com/GoogleCloudPlatformTraining/cp300-gae-datastore-python.git
```

Alternately, you can work with the final Project ZIP files available for both Java and Python. You can use them if you want but we do encourage you to try it out on your own. The files are given below:

- Java Task Queues and Cron Project ZIP file
- Python Task Queues and Cron Project ZIP file

You can also clone the projects directly from GitHub:

```
git clone https://github.com/GoogleCloudPlatformTraining/cp300-gae-taskqueue-java.git
```

OR

```
git clone https://github.com/GoogleCloudPlatformTraining/cp300-gae-taskqueue-python.git
```

Task Queue Implementation

We shall first look at the Task Queue Handler implementation.

Java Instructions

Create the **queue.xml** file in **WEB-INF** folder as shown below:

```
<queue-entries>
  <queue>
```

```
    <name>default</name>
    <rate>1/s</rate>
  </queue>
</queue-entries>
```

We are using the default queue here.

Add the Servlet entries in the WEB-INF/web.xml file as shown below:

```
<servlet>
   <servlet-name>EmailWorker</servlet-name>
  <servlet-class>com.mycompany.app.EmailWorker</servlet-class>
</servlet>

<servlet-mapping>
      <servlet-name>EmailWorker</servlet-name>
      <url-pattern>/worker</url-pattern>
</servlet-mapping>
```

The next step is to **invoke the Task Queue Handler** in the **/create** Conference handler. The lines in bold are the ones to be added:

Note the following:

1. We are using **/worker** URL since we have mapped that in the **web.xml** file
2. We are passing a request parameter **key** that contains the Entity Key for the Conference entity that we just created. This will be used by the worker to retrieve the conference details from the Datastore.

CreateConference.java

```
package com.mycompany.app;

import java.io.IOException;
import java.io.PrintWriter;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;
import java.util.logging.Logger;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.google.appengine.api.datastore.DatastoreService;
import com.google.appengine.api.datastore.DatastoreServiceFactory;
```

```
import com.google.appengine.api.datastore.Entity;
import com.google.appengine.api.datastore.Key;
import com.google.appengine.api.datastore.KeyFactory;

import com.google.appengine.api.taskqueue.Queue;
import com.google.appengine.api.taskqueue.QueueFactory;
import com.google.appengine.api.taskqueue.TaskOptions;

@SuppressWarnings("serial")

public class CreateConference extends HttpServlet {

  public static final Logger _LOG = Logger.getLogger(CreateConference.class.getName());

  // Get a handle to the DatastoreService

  public static DatastoreService datastore =
DatastoreServiceFactory.getDatastoreService();

  // Get a handle to the Task Queue

  Queue queue = QueueFactory.getDefaultQueue();

  public void doPost(HttpServletRequest req, HttpServletResponse resp) throws
IOException {

    PrintWriter writer = resp.getWriter();

    // Get the values sent by the form

    String title = req.getParameter("title");
    String city = req.getParameter("city");
    String maxAttendees = req.getParameter("maxAttendees");
    String confStartDate = req.getParameter("startdate");
    String confEndDate = req.getParameter("enddate");

    try {

       // Convert maxAttendees to a Long before saving it

        Long mL = Long.parseLong(maxAttendees);

        // Convert the startDate and endDate to Dates

        // The format is 2013-04-26

        Date startDate;

        try {
```

```
          startDate = new SimpleDateFormat("yyyy-MM-dd",
Locale.ENGLISH).parse(confStartDate);
        } catch (ParseException e) {
          startDate = null;
        }
        Date endDate;
        try {
          endDate = new SimpleDateFormat("yyyy-MM-dd",
Locale.ENGLISH).parse(confEndDate);
        } catch (ParseException e) {
          endDate = null;
        }
        // Create an entity of kind "Conference"
        Entity confEntity = new Entity("Conference");
       confEntity.setProperty("title", title);
       confEntity.setProperty("city", city);
       confEntity.setProperty("maxAttendees", mL);
       confEntity.setProperty("startdate", startDate);
       confEntity.setProperty("enddate", endDate);
       // Save the entity in the datastore
        Key confKey = datastore.put(confEntity);
           // Get a string of the conference entity's key
           String confKeyString = KeyFactory.keyToString(confKey);
           _LOG.info("Conference Entity with Key = " + confKeyString + " created");
           // Add the task to the default queue.
           queue.add(TaskOptions.Builder.withUrl("/worker").param("key",
confKeyString));
      resp.sendRedirect("/");
    } catch (Exception e) {
```

```
      resp.setContentType("text/html");
      writer.println("Exception in saving entity. Reason : " + e.getMessage());
    }
  }

  /*

   * If a user come to this page as a GET, redirect to the Create Conference page

   */

  public void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
    resp.sendRedirect("/createconference");
  }
}
```

The Task Queue Handler (EmailWorker) is shown below. Note the following:

1. The POST method of the Task Queue Handler is invoked
2. We extract out the Key from the Request parameters and then query the Datastore for the Conference Record with that Entity Key.
3. We are using a dummy list of registered emails. We will simulate sending out an email to each of the addresses (we are simply printing a log message here) with the conference details (note the title attribute used here). You could extend this code to send out an actual Email and utilize other attributes of the Conference Entity.

EmailWorker.java

```
package com.mycompany.app;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.google.appengine.api.datastore.DatastoreService;
import com.google.appengine.api.datastore.DatastoreServiceFactory;
import com.google.appengine.api.datastore.Entity;
import com.google.appengine.api.datastore.EntityNotFoundException;
import com.google.appengine.api.datastore.Key;
import com.google.appengine.api.datastore.KeyFactory;
```

```
@SuppressWarnings("serial")

public class EmailWorker extends HttpServlet {

  public static final Logger _LOG = Logger.getLogger(EmailWorker.class.getName());

  // Get a handle to the DatastoreService

  public static DatastoreService datastore =
DatastoreServiceFactory.getDatastoreService();

  public void doPost(HttpServletRequest req, HttpServletResponse resp) throws
IOException {

    // Get the key value

    String param_key = req.getParameter("key");

    Key conference_key = KeyFactory.stringToKey(param_key);

    try {

             //Retrieve the Entity from the Datastore

            Entity conference = datastore.get(conference_key);

           //Get List of Registered Users. Ideally you will read this from the
Datastore

           String[] registered_users = {"a@a.com","b@b.com","c@c.com"};

           for (String user : registered_users) {

                    _LOG.info("Sending email to " + user + " on " +
conference.getProperty("title"));

              }


       } catch (EntityNotFoundException e) {
              _LOG.log(Level.WARNING, "Conference Entity not found with Key = " +
conference_key);
       }
  }
}
```

To build and run the Java App Engine project, execute the following Maven commands from the command line / terminal. These commands should be executed from the **cp300-gae-taskqueue-java**

folder i.e. where your pom.xml file is present:

1. To build the project

   ```
   mvn package
   ```

2. To run the application in the local devserver :

   ```
   mvn appengine:devserver
   ```

To test the application locally, try the following in your local browser:

1. Visit http://localhost:<PORT_NUMBER>.
2. This should bring up the Create Conference Form.
3. Go ahead and create a conference. Fill out all the values (do not leave any empty values). Click on Schedule Conference to submit the form.
4. Once the conference has been successfully created, the browser will be redirected back to the same page and the newly created record will be shown in the list.
5. **Observe the console output. You should see several log statements that show that the email is being sent out to the Users : a , b and c.**

Python Instructions

Here is the **app.yaml** for the application:

```
application: your-app-id
version: 1
runtime: python27
api_version: 1
threadsafe: true

handlers:

- url: /.*
   script: taskqueue.application

libraries:
- name: webapp2
   version: latest
- name: jinja2
   version: latest
```

Create a **queue.yaml** file as shown below:

```
queue:
- name: default
  rate: 1/s
```

Add a request handler to the Main Application **taskqueue.py** as shown below:

```
application = webapp2.WSGIApplication([

    ('/', MainPage),

    ('/create', CreateConference),

    ('/worker', EmailWorker),

], debug=True)
```

Update the CreateConfrence Request Handler with the lines in bold as shown below:

```
class CreateConference(webapp2.RequestHandler):

    def post(self):

        conference = Conference()
        conference.title = self.request.get('title')
         conference.city = self.request.get('city')
        conference.startdate =
datetime.datetime.strptime(self.request.get('startdate'), '%Y-%m-%d').date()
        conference.enddate = datetime.datetime.strptime(self.request.get('enddate'),
'%Y-%m-%d').date()
        conference.maxAttendees = int(self.request.get('maxAttendees'))
        key = conference.put()

        # Add the task to the default queue.

        taskqueue.add(url='/worker', params={'key': key.urlsafe()})

        self.redirect('/')
```

Note the following:

1. We are using **/worker** URL since we have mapped that in the request URLs
2. We are passing a request parameter **key** that contains the Entity Key for the Conference entity that we just created. This will be used by the worker to retrieve the conference details from the Datastore.

The Task Queue Handler (**EmailWorker**) is shown below. Add that to the above application file:

```
class EmailWorker(webapp2.RequestHandler):

    def post(self): # should run at most 1/s due to entity group limit
        param_key = self.request.get('key')
```

```
        conference_key = ndb.Key(urlsafe=param_key)

        #Retrieve the Item from the database using the key
        if (conference_key is not None):
            conference = conference_key.get()

            #Get List of Registered Users.

            #Ideally you will read this from the Datastore
            registered_users = ['a@a.com','b@b.com','c@c.com']

            for user in registered_users:
                logging.info('Sending email to ' + user + ' on ' +
conference.title)
```

Note the following in the above implementation:

1. We extract out the Key from the Request parameters and then query the Datastore for the Conference Record with that Entity Key.
2. We are using a dummy list of registered emails. To each of the emails, we send out an email (we are simply printing a log message here) with the conference details (note the title attribute used here). You could extend this code to send out an actual Email and utilize other attributes of the Conference Entity.

Run the Python Application locally via your IDE or if you prefer via the terminal/ command line using the following:

```
dev_appserver.py /<your_python_app_dir>
```

To test out the application locally, try the following in your local browser:

1. Visit http://localhost:<PORT_NUMBER>. (The default is 8080)
2. This should bring up the Create Conference Form.
3. Go ahead and create a conference. Fill out all the values (do not leave any empty values). Click on Schedule Conference to submit the form.
4. Once the conference has been successfully created, the browser will be redirected back to the same page and the newly created record will be shown in the list.
5. Once the conference has been successfully created, the browser will be redirected back to the same page and the newly created record will be shown in the list.
6. **Observe the console output. You should see several log statements that show that the email is being sent out to the Users : a , b and c.**

Cron Job Implementation

We shall now look at introducing the Cron Job into our applications.

Java Instructions

Create the **cron.xml** file in **WEB-INF** directory as shown below:

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

<cronentries>
   <cron>
      <url>/cron/email</url>
      <description>Upcoming Conferences Daily Email</description>
      <schedule>every 2 minutes</schedule>
   </cron>
</cronentries>
```

Note that we have used every 2 minutes, so that you can see it work. This might be changed in production to a daily (once in 24 hours) execution. For more information, refer to the Cron Schedule format.

Add the following servlet entries to **WEB-INF\web.xml** format:

```
<servlet>
      <servlet-name>EmailCron</servlet-name>
      <servlet-class>com.mycompany.app.EmailCron</servlet-class>
</servlet>
<servlet-mapping>
      <servlet-name>EmailCron</servlet-name>
      <url-pattern>/cron/email</url-pattern>
</servlet-mapping>

<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>
```

Notice that we have also added a security constraint so that users cannot invoke the Cron Job URL

directly.

Finally, we have the **EmailCron.java** file shown below:

```
package com.mycompany.app;

import java.io.IOException;
import java.util.logging.Logger;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.google.appengine.api.datastore.DatastoreService;
import com.google.appengine.api.datastore.DatastoreServiceFactory;

@SuppressWarnings("serial")

public class EmailCron extends HttpServlet {

  public static final Logger _LOG = Logger.getLogger(EmailCron.class.getName());

 // Get a handle to the DatastoreService

  public static DatastoreService datastore =
DatastoreServiceFactory.getDatastoreService();

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
     _LOG.info("Sending out email");
}
}
```

Notice that the code is meant to show you the mechanics of a Cron Job. You can complete the code as an exercise.

The Development Server does not support automatically scheduling and running the Cron Jobs for you. To test them out, you will need to invoke the Cron Job URL directly in the local browser. To do that, follow these steps:

1. To build and run the Java App Engine project, execute the following Maven commands from the command line / terminal. These commands should be executed from the **cp300-gae-taskqueue-java** folder i.e. where your pom.xml file is present:
2. To build the project

   ```
   mvn package
   ```

3. To run the application in the local devserver :

```
mvn appengine:devserver
```

4. In your local browser, visit http://localhost:<PORT_NUMBER>/cron/email . This should invoke the GET method of your Cron Job and you should see a log statement that indicates that Email is being sent out.

Python Instructions

Create a **cron.yaml** file as shown below:

```
cron:
- description: Upcoming Conferences Daily Email
  url: /cron/email
  schedule: every 2 minutes
```

Add a request handler to the Main Application **taskqueue.py** as shown below:

```
application = webapp2.WSGIApplication([
    ('/', MainPage),
    ('/create', CreateConference),
    ('/worker', EmailWorker),
    ('/cron/email',EmailCron)
], debug=True)
```

Add the **EmailCron Request Handler** in the main Application file as shown below:

```
class EmailCron(webapp2.RequestHandler):
    def get(self):
        logging.info('Sending out Email')
```

Run the Python Application locally via your IDE or if you prefer via the terminal/ command line using the following:

```
dev_appserver.py /<your_python_app_dir>
```

In your local browser, visit http://localhost:<PORT_NUMBER>/cron/email . This should invoke the GET method of your Cron Job and you should see a log statement that indicates that Email is being sent out.

Deploying the Application and using the Admin Console

Optionally, if you want, you can deploy the application to App Engine and test it out. You can view Datastore records in the Admin Console.

1. Deploy the application to App Engine. The application will be available over http://<APP_ID>.appspot.com

2. Go to Main -> Task Queues. It should show the Task Queue that you have configured.
3. Go to Main -> Cron Jobs. It should show the Cron Job that you have configured.
4. Create a few Conference Records.
5. Visit the Logs in the Admin console. You should see invocation of the Task Queue Handler and Log Statements that indicate that the email was sent out.
6. To view the Cron Jobs execution, ensure that you have modified the frequency to 2 minutes or 1 minute, so that the Cron Job executes and you can see the log statements in the Logs section in Admin console.

Additional Resources

- Configuring Queues (Java) (Python)
- Using Push Queue (Java) (Python)
- Using Pull Queues (Java) (Python)
- Article: Getting Started with Task Queues
- Scheduling tasks with cron (Java) (Python)
- App Engine Mail API (Java) (Python)

Summary

In this exercise, you learned:

- How to use push queues to execute tasks.
- How to use cron jobs to run regularly scheduled activities.