

Course: Software Engineering

Service Bus Durable Task Framework

Submitted To

Prof. Dr. Andreas Pech

Damir Dobric

Submitted By

Setu Moye Biswas

1099793

Date of Submission 20.04.2016

Contents

1	Overview	3
2	Task of the project	3
3	Description	3
3.1	Core Concepts	3
3.1.1	Task Hub	3
3.1.2	Task Activities	4
3.1.3	Task Orchestrations	4
3.1.4	Task Hub Worker	4
3.1.5	Task Hub Client	4
4	Organize Module	5
4.1	Writing Task Orchestrations	6
4.3	Orchestration Instance Management	6
4.4	Error Handling & Compensation	7
4.5	Task Hub Management	7
5	Scenarios	7
5.1	Upload File in Google Drive	7
5.1.1	User Case 1	7
5.1.2	User Case 2	8
6	Conclusion:	9

Document Version Control:

Date	Version	Change Reference	Author
20.04.2016	V1.0	Final Draft	Setu Moye Biswas

1 Overview

The Service Bus Durable Task Framework provides developers a means to write code orchestrations in C# using the .Net Task framework and integration projects can be implemented without using any Enterprise Service Bus product.

The main features of the durable task framework are-

- Definition of code orchestrations in simple C# code
- Automatic persistence and check-pointing of program state
- Versioning of orchestrations and activities
- Async timers, orchestration composition, user aided check pointing

It is used heavily within various teams at Microsoft to reliably orchestrate long running provisioning, monitoring and management operations. The orchestrations scale out linearly by simply adding more worker machines. The framework itself is very light weight and only requires an Azure Service Bus namespace and optionally an Azure Storage account. Running instances of the orchestration and worker nodes are completely hosted by the user. No user code is executing 'inside' Service Bus.

2 Task of the project

Google Drive is a file storage and synchronization service. It allows users to store files in the cloud, share files, and edit documents, spread sheets, and presentations with collaborators. In this project, files can be uploaded to Google Drive using C# console application and it is done by using Durable Task Framework. First, it is needed to ask for access token for authentication to Google Drive and secondly, upload file task to upload file to that access token. . Durable task framework ensures the state of the execution of these two tasks is preserved durably.

3 Description

The Service Bus Durable Task Framework allows users to write C# code and encapsulate it within 'durable' .Net Tasks. These durable tasks can then be composed with other durable tasks to build complex task orchestrations.

3.1 Core Concepts

There are a few fundamental concepts in the framework.

3.1.1 Task Hub

The Task Hub is a logical container for Service Bus entities within a namespace. These entities are used by the Task Hub Worker to pass messages reliably between the code orchestrations and the activities that they are orchestrating.

3.1.2 Task Activities

Task Activities are pieces of code that perform specific steps of the orchestration. A Task Activity can be 'scheduled' from within some Task Orchestration code. This scheduling yields a plain vanilla .Net Task which can be (asynchronously) awaited on and composed with other similar Tasks to build complex orchestrations.

3.1.3 Task Orchestrations

Task Orchestrations schedule Task Activities and build code orchestrations around the Tasks that represent the activities.

3.1.4 Task Hub Worker

The worker is the host for Task Orchestrations and Activities. It also contains APIs to perform CRUD operations on the Task Hub itself.

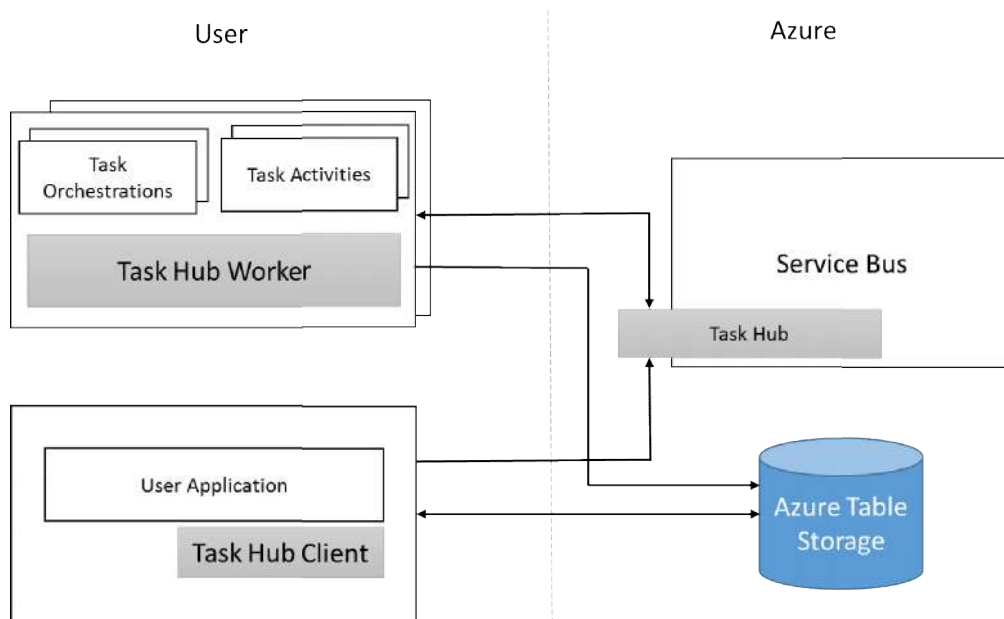
3.1.5 Task Hub Client

The Task Hub Client provides:

- APIs for creating and managing Task Orchestration instances
- APIs for querying the state of Task Orchestration instances from an Azure Table

Both the Task Hub Worker and Task Hub Client are configured with connection strings connection strings for Service Bus and optionally with connection strings for a storage account.

Service Bus is used for storing the control flow state of the execution and message passing between the task orchestration instances and task activities. However Service Bus is not meant to be a database so when a code orchestration is completed, the state is removed from Service Bus. If an Azure Table storage account was configured then this state would be available for querying for as long as it is kept there by the user.



For Durable Task Implementation we have

Task Orchestration: TaskOrchestration.cs

Task Activities: GetAccessTokenTask.cs, UploadFileTask.cs

User Applications: Google Drive

The framework provides TaskOrchestration and TaskActivity base classes which users can derive from to specify their orchestrations and activities. They can then use the TaskHub APIs to load these orchestrations and activities into the process and then start the worker which starts processing requests for creating new orchestration instances.

The TaskHubClient APIs are used to create new orchestration instances, query for existing instances and then terminate those instances if required.

4 Organize Module

Assume that user wants to build a code orchestration that will first request for an access token and using this access token user will be able to upload file in Google Drive. To implement this task by using Service Bus Durable Task Framework, the user has to write two Task Activities, firstly, user should get the access token for user authentication. Secondly, upload file task which will upload file to Google Drive by using that access token .Here, Task Orchestration is done that orchestrates between the Task activities.

In this orchestration process, the user is scheduling get access token task activity, and then waiting for the response and then scheduling the upload file task activity. The framework will ensure that the state of the execution is preserved durably. E.g., if the node hosting the task orchestration above

crashed before scheduling the get access token task activity, on restart it will know to schedule this activity. If the node crashed after it had scheduled the activity but before the response came back, on restart it will be smart enough to know that the activity was already scheduled and it will directly start waiting for the response of the get access token task activity.

By this process users can load these orchestration and activity classes in a worker and start processing requests to create new orchestration instances. Multiple instances of these workers can be running concurrently against the same task hub to provide load balancing as required. The framework guarantees that a particular orchestration instance code would only be executing on a single worker at one time.

4.1 Writing Task Orchestrations

Task orchestrations basically invoke Task Activities and define how the control flows from one activity to another. The code that can be written within an orchestration is plain C# but with a few constraints. These constraints exist because of how the framework replays the orchestration code. This is described in a nutshell below.

Every time new work needs to be processed by an orchestration (e.g. a Task Activity finished or a timer fired), the framework replays the user's TaskOrchestration code from scratch. Whenever this user code attempts to schedule a TaskActivity, the framework intercepts this call and consults the 'execution history' of the orchestration. If it finds that the particular TaskActivity had already been executed and yielded some result, it would replay that Activity's result immediately and the TaskOrchestration would continue. This would continue happening until the user code has executed to a point where either it is finished or it has scheduled a new Activity. If it is the latter case then the framework would actually schedule and execute the specified Activity. After this Activity is completed its result also becomes part of the execution history and the value would be used in subsequent replays.

The Task Orchestration code is always executed in a single thread. This means that if the code was awaiting multiple tasks and one of them completed followed immediately by another one, the framework is guaranteed to run the continuations for both of these tasks serially.

4.2 Writing Task Activities

Task Activities are the 'leaf' nodes of an orchestration. This is the code which actually performs a unit of operation within the orchestration. This is plain C# code with no constraints. Task Activity code is guaranteed to be called at least once. However in error cases it might be invoked multiple times so idempotence is desirable.

4.3 Orchestration Instance Management

The TaskHubClient API allows users to create new orchestration instances, query for the state of created orchestration instances and terminate these instances. The API for creating an orchestration instance will return the instance information. This information can be used in subsequent APIs to query for the state of the instance.

4.4 Error Handling & Compensation

Any exception that is thrown in the TaskActivity code is marshalled back and thrown as a TaskFailedException in the TaskOrchestration code. Users can write the appropriate error handling and compensation code that suits their needs around this. Refer Failure scenario's below for Error handling.

4.5 Task Hub Management

The TaskHubWorker has APIs that can be used to perform CRUD operations on the TaskHub itself.

5 Scenarios

5.1 Upload File in Google Drive

5.1.1 User Case 1

User Case 1	Create a API key
Description	At first, a user has to register his application for Google Drive API in Google Developers Console and automatically turn on the API. After requesting for a new client ID, user will get API Secret, API Key and call back URL. This credential will save to a json file which should be imported to the working directory. After running the program the user will get access token from this json file and ready for uploading files in Google Drive.

API API Manager

Credentials

Overview

Credentials

←

Create client ID

Application type

☐ Web application
 ☐ Android [Learn more](#)
☐ Chrome App [Learn more](#)
☐ iOS [Learn more](#)
☐ PlayStation 4
 ☒ Other

Name

Drive API Quickstart

Create


Cancel

API API Manager

Credentials

Overview

Credentials


Email address 

realuser32@gmail.com


Product name shown to users

Product name

Homepage URL (Optional)

Product logo URL (Optional) 

http://www.example.com/logo.png




This is how your logo will look to end users
 Max size: 120x120 px

Privacy policy URL (Optional)

Terms of service URL (Optional)

Save

Cancel

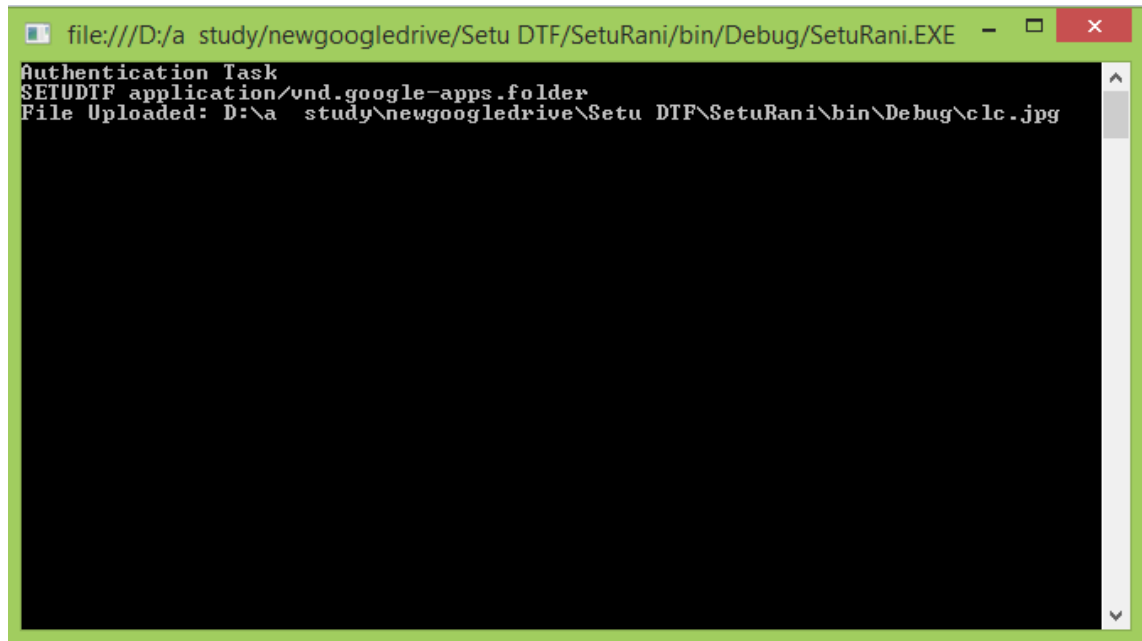


The consent screen will be shown to users whenever you request access to their private data using your client ID. It will be shown for all applications registered in this project.

You must provide an email address and product name for OAuth to work.

5.1.2 User Case 2

User Case 2	Successfully Upload File
Description	After the successful authentication of the user, the input File should be uploaded to Google Drive and the response can be viewed in the screen.



```
file:///D:/a study/newgoogledrive/Setu DTF/SetuRani/bin/Debug/SetuRani.EXE
Authentication Task
SETUDTF application/vnd.google-apps.folder
File Uploaded: D:\a study\newgoogledrive\Setu DTF\SetuRani\bin\Debug\c1c.jpg
```

6 Conclusion:

Durable task framework allows users to write long running persistent workflows in C# using the `async/await` capabilities. It is used heavily within various teams at Microsoft to reliably orchestrate long running provisioning, monitoring and management operations. The orchestrations scale out linearly by simply adding more worker machines. In this project I have implemented a console application which is used to upload file on Google Drive. Durable task framework is used to orchestrate the access token and uploading file task to make the system efficient. By open sourcing this project I hope to give the community a very cost-effective alternative to heavy duty workflow systems. I also hope to build an ecosystem of providers and activities around this simple yet incredibly powerful framework.

References:

[1] *Service Bus Durable Task Framework*, p. 2-10. [Online]. Available: <https://abhishekrlal.com/2013/06/27/durable-task-framework-preview-with-azure-service-bus/>. Accessed: Oct. 21, 2015

[2] "Durable Task Framework – Episode III: Coded Orchestrations,". [Online]. Available: http://developers.de/blogs/damir_dobric/archive/2015/09/15/durable-task-framework-episode-iii-coded-orchestrations.aspx. Accessed: Nov. 16, 2015.