.NET (https://cloud.google.com/dotnet/) Guides

Using Pub/Sub with .NET

Many apps do background processes outside of a web request. In this sample, the Bookshelf app sends tasks to a separate background worker for execution. The worker gathers information from the Google Books API (https://developers.google.com/books/) and updates the book information in the database. This sample demonstrates how to set up separate services in App Engine, how to run a worker process in the App Engine flexible environment, and how to deal with lifecycle events.

This page is part of a multipage tutorial. To start from the beginning and read the setup instructions, go to .NET Bookshelf app (https://cloud.google.com/dotnet/docs/getting-started/tutorial-app).

Configuring settings

1. To open the sample app in Visual Studio, in the `getting-started-dotnet\aspnet\5-pubsub` directory, double-click **5-pubsub**.
2. In the **Solution Explorer** pane, click **Bookshelf** > **Web.config**.
3. In `bookshelf\Web.config`, complete the following steps:
    a. Set `GoogleCloudSamples:ProjectId` to your project ID.
    b. Set the value of `GoogleCloudSamples:BookStore` to the same value you used during the Using Structured Data (https://cloud.google.com/dotnet/docs/getting-started/using-structured-data) part of this tutorial.
    c. If you used Cloud SQL (https://cloud.google.com/dotnet/docs/getting-started/using-cloud-sql) or SQL Server

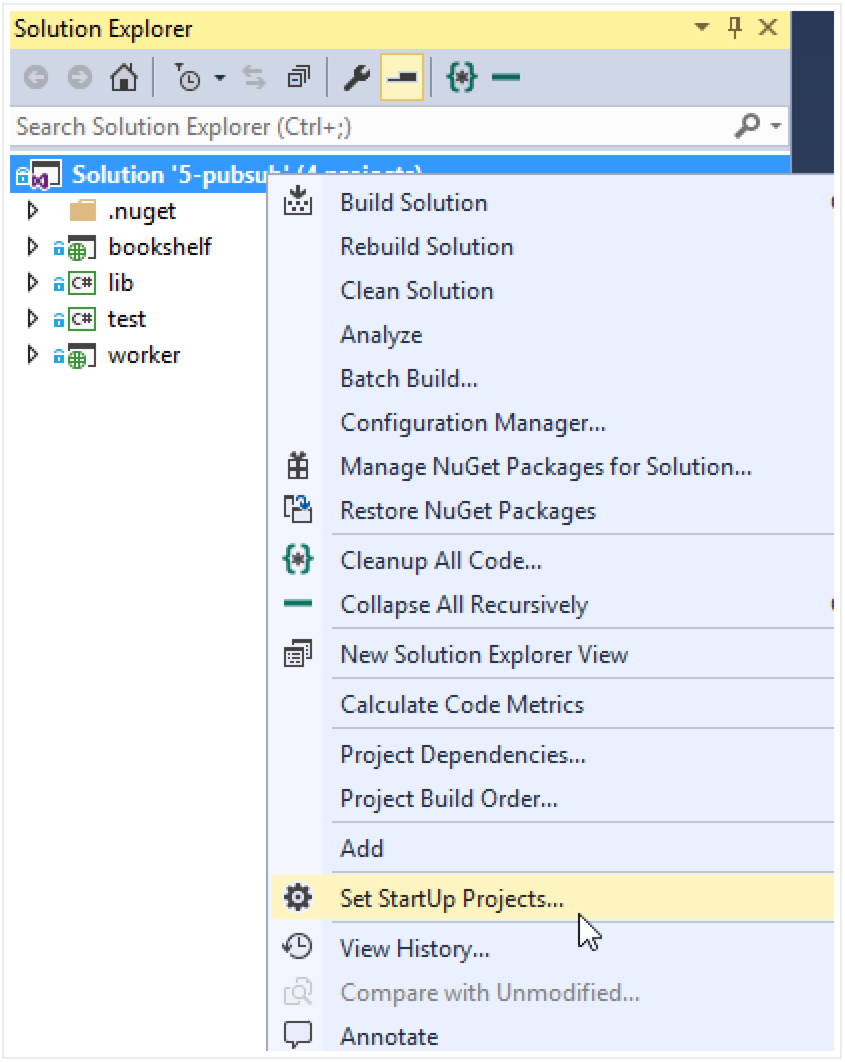
(https://cloud.google.com/dotnet/docs/getting-started/using-sql-server) during the structured data step, find the `<connectionStrings>` XML element and set the **connectionString** to the same value you used during that step.

d. Set `GoogleCloudSamples:BucketName` to the name of the Cloud Storage bucket (https://cloud.google.com/dotnet/docs/getting-started/using-cloud-storage) you created previously.

4. Save and close `bookshelf\Web.config`.
5. In the **Solution Explorer** pane, go to **Worker** > **Web.config**.
6. In `worker\Web.config`, complete the following steps:
   a. Set `GoogleCloudSamples:ProjectId` to your project ID.
   b. Set the value of `GoogleCloudSamples:BookStore` to the same value you used during the Using Structured Data (https://cloud.google.com/dotnet/getting-started/using-structured-data) step of this tutorial.
   c. If you used Cloud SQL (https://cloud.google.com/dotnet/docs/getting-started/using-cloud-sql) or SQL Server (https://cloud.google.com/dotnet/docs/getting-started/using-sql-server) during the structured data step, find the `<connectionStrings>` XML element and set the **connectionString** to the same value you used during that step.
   d. Set `GoogleCloudSamples:BucketName` to the name of the Cloud Storage bucket (https://cloud.google.com/dotnet/docs/getting-started/using-cloud-storage) you created previously.
7. Save and close `worker\Web.config`.

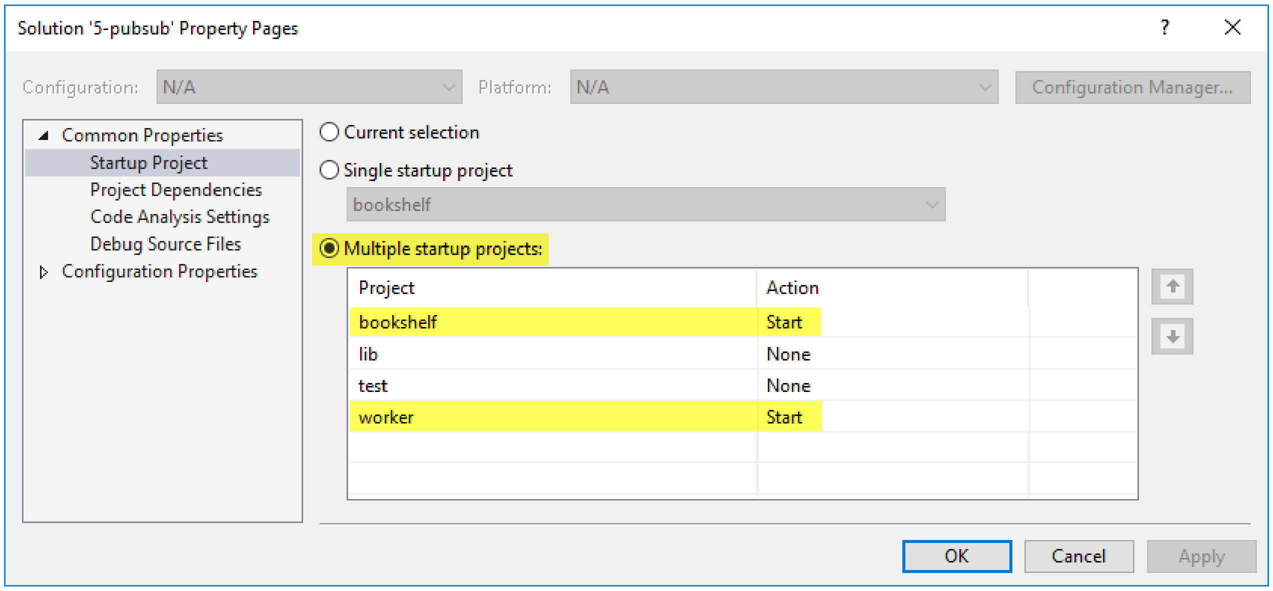
Running the app on your local machine

1. In Visual Studio, in the **Solution Explorer** pane, right-click **Solution**, and click **Set StartUp Projects**.



2. Click **Multiple startup projects**.

3. For the **Bookshelf** and **Worker** rows, set the **Action** to **Start**, and then click **OK**.

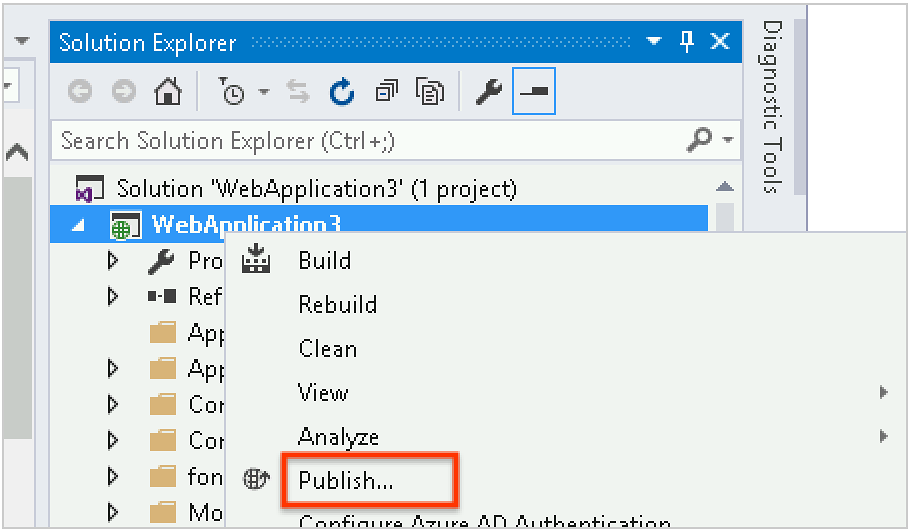


Project	Action
bookshelf	Start
lib	None
test	None
worker	Start

4. Press **F5** to run the projects.
5. Add some books to the bookshelf. If you have both the app and worker instance running locally, you can watch the worker update the book information in the background.

Deploying the Bookshelf app to Compute Engine

1. In Visual Studio, in the **Solution Explorer** pane, right-click **Bookshelf**, and then click **Publish**.


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'WebApplication3' (1 project)
WebApplication3
Build
Rebuild
Clean
View
Analyze
Publish...
Configure Azure AD Authentication
Diagnostic Tools

2. Create a new custom profile as you did in the Using Datastore (https://cloud.google.com/dotnet/docs/getting-started/using-cloud-datastore#deploying_the_app_to_compute_engine) part of this tutorial.
3. Click **Publish**.

Deploying the worker to Compute Engine

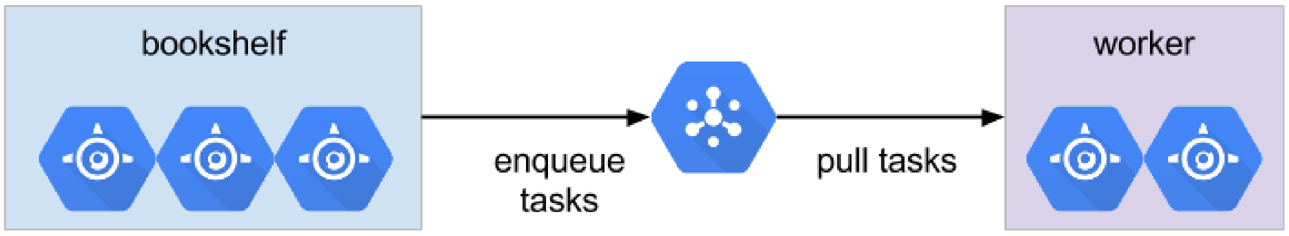
1. In Visual Studio, in the **Solution Explorer** pane, right-click **Worker**, and click **Publish**.
2. Create a new custom profile as you did in the Using Datastore (https://cloud.google.com/dotnet/docs/getting-started/using-cloud-datastore#deploying_the_app_to_compute_engine) part of this tutorial.
3. Click **Publish**.

Running the app on Compute Engine

In your web browser, enter the address of your first Compute Engine instance.

App structure

This diagram shows the app's components and how they fit together.

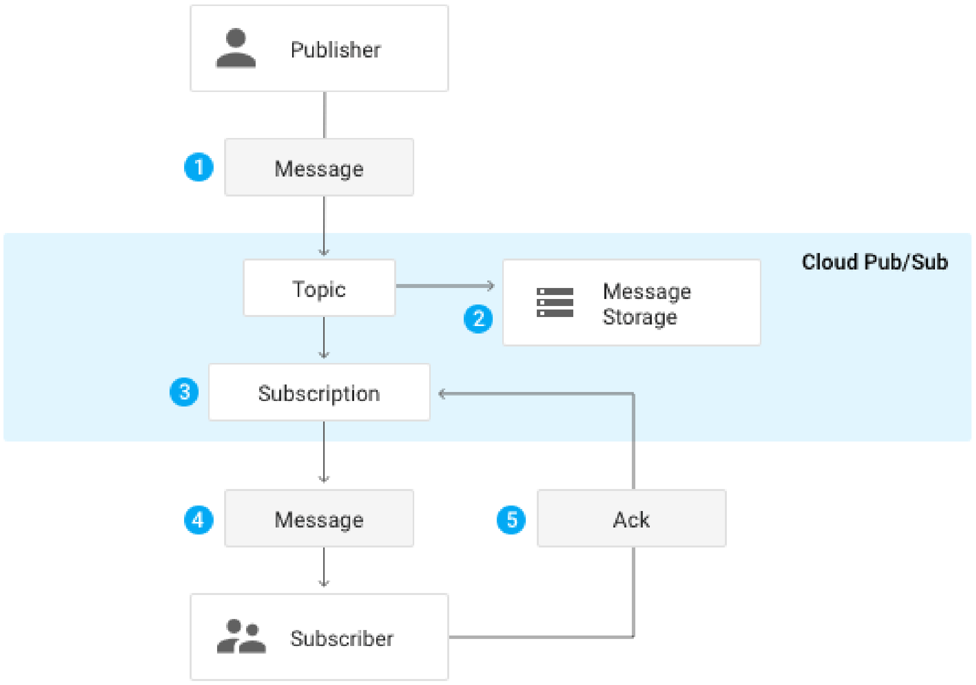


Understanding the code

This section walks you through how to create a queue, add tasks to the queue, and use the worker to process tasks.

Create a queue

A Pub/Sub topic and subscription together form a queue.



A **`QueueMessage`** contains the ID of a book to look up in the Google Books API.

[aspnet/5-pubsub/lib/Services/BookDetailLookup.cs](https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/aspnet/5-pubsub/lib/Services/BookDetailLookup.cs) (https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/aspnet/5-pubsub/lib/Services/BookDetailLookup.cs)

GETTING-STARTED-DOTNET/BLOB/MASTER/ASPNET/5-PUBSUB/LIB/SERVICES/BOOKDETAILLOOKUP.CS)

```
private class QueueMessage
{
    public long BookId;
};
```

A book ID is added to a topic named **`book-process-queue`**. A subscription named **`shared-worker-subscription`** subscribes to this topic. The worker watches this subscription for tasks to execute.

The full topic and subscription paths include the project name.

[aspnet/5-pubsub/lib/Services/BookDetailLookup.cs](https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/aspnet/5-pubsub/lib/Services/BookDetailLookup.cs) (https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/aspnet/5-pubsub/lib/Services/BookDetailLookup.cs)

GETTING-STARTED-DOTNET/BLOB/MASTER/ASPNET/5-PUBSUB/LIB/SERVICES/BOOKDETAILLOOKUP.CS)

```
_topicName = new TopicName(projectId, options.TopicId);
_subscriptionName = new SubscriptionName(projectId, options.SubscriptionId);
```

`CreateTopicAndSubscription()` attempts to create a topic and subscription in Pub/Sub, but first checks to see if it already exists.

aspnet/5-pubsub/lib/Services/BookDetailLookup.cs (https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/aspnet/5-pubsub/lib/Services/BookDetailLookup.cs)

GETTING-STARTED-DOTNET/BLOB/MASTER/ASPNET/5-PUBSUB/LIB/SERVICES/BOOKDETAILLOOKUP.CS)

```
public void CreateTopicAndSubscription()
{
    try
    {
        _pub.CreateTopic(_topicName);
        _logger.LogVerbose("Created topic " + _topicName);
    }
    catch (Grpc.Core.RpcException e)
    when (e.Status.StatusCode == Grpc.Core.StatusCode.AlreadyExists)
    {
        // The topic already exists.  Ok.
        _logger.LogError(_topicName + " already exists", e);
    }
    try
    {
        _sub.CreateSubscription(_subscriptionName, _topicName, null, 0);
        _logger.LogVerbose("Created subscription " + _subscriptionName);
    }
    catch (Grpc.Core.RpcException e)
    when (e.Status.StatusCode == Grpc.Core.StatusCode.AlreadyExists)
    {
        // The subscription already exists.  Ok.
        _logger.LogError(_subscriptionName + " already exists", e);
    }
}
```

Queue tasks

The `QueueMessage` is JSON-encoded, and the resulting JSON is base64-encoded. While this is excessive for encoding a simple `long`, this is the preferred way to encode messages so that they are compatible with the Pub/Sub API.

aspnet/5-pubsub/lib/Services/BookDetailLookup.cs (https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/aspnet/5-pubsub/lib/Services/BookDetailLookup.cs)

ETTING-STARTED-DOTNET/BLOB/MASTER/ASPNET/5-PUBSUB/LIB/SERVICES/BOOKDETAILLOOKUP.CS)

```
public void EnqueueBook(long bookId)
{
    var message = new QueueMessage() { BookId = bookId };
    var json = JsonConvert.SerializeObject(message);
    _pub.Publish(_topicName, new[] { new PubsubMessage()
    {
        Data = Google.Protobuf.ByteString.CopyFromUtf8(json)
    } });
}
```

The worker

The worker is a separate app that listens to Pub/Sub events. This splits the app into two independent processes that communicate by using Pub/Sub, instead of directly with each other.

Process books

To process a book, the task retrieves the book by its ID, finds additional information, and then saves the updated information in the database.

aspnet/5-pubsub/lib/Services/BookDetailLookup.cs (https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/aspnet/5-pubsub/lib/Services/BookDetailLookup.cs)

ETTING-STARTED-DOTNET/BLOB/MASTER/ASPNET/5-PUBSUB/LIB/SERVICES/BOOKDETAILLOOKUP.CS)

```
public void ProcessBook(IBookStore bookStore, long bookId)
{
    var book = bookStore.Read(bookId);
    _logger.LogVerbose($"Found {book.Title}.  Updating.");
    var query = "https://www.googleapis.com/books/v1/volumes?q="
```

```
        + Uri.EscapeDataString(book.Title);
    var response = WebRequest.Create(query).GetResponse();
    var reader = new StreamReader(response.GetResponseStream());
    var json = reader.ReadToEnd();
    UpdateBookFromJson(json, book);
    bookStore.Update(book);
}
```

The function `PullOnce` reads messages from the subscription and calls `ProcessBook` for every message.

aspnet/5-pubsub/lib/Services/BookDetailLookup.cs
 (https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/aspnet/5-pubsub/lib/Services/BookDetailLookup.cs)

GETTING-STARTED-DOTNET/BLOB/MASTER/ASPNET/5-PUBSUB/LIB/SERVICES/BOOKDETAILLOOKUP.CS)

```
        private void PullOnce(Action<long> callback, CancellationToken cancellationT
        {
            _logger.LogVerbose($"Pulling messages from {_subscriptionName}...");
            // Pull some messages from the subscription.

            var response = _sub.Pull(_subscriptionName, false, 3,
                CallSettings.FromCallTiming(
                    CallTiming.FromExpiration(
                        Expiration.FromTimeout(
                            TimeSpan.FromSeconds(90)))));
            if (response.ReceivedMessages == null)
            {
                // HTTP Request expired because the queue was empty.  Ok.
                _logger.LogVerbose("Pulled no messages.");
                return;
            }
            _logger.LogVerbose($"Pulled {response.ReceivedMessages.Count} messages."
            if (response.ReceivedMessages.Count == 0)
            {
                return;
            }
            foreach (var message in response.ReceivedMessages)
            {
                try
                {
                    // Unpack the message.
                    byte[] json = message.Message.Data.ToByteArray();
                    var qmessage = JsonConvert.DeserializeObject<QueueMessage>(
                        Encoding.UTF8.GetString(json));
                    // Invoke ProcessBook().
                    callback(qmessage.BookId);
                }
                catch (Exception e)
                {
                    _logger.LogError("Error processing book.", e);
                }
            }
            // Acknowledge the message so we don't see it again.
            var ackIds = new string[response.ReceivedMessages.Count];
            for (int i = 0; i < response.ReceivedMessages.Count; ++i)
                ackIds[i] = response.ReceivedMessages[i].AckId;
            _sub.Acknowledge(_subscriptionName, ackIds);
        }
```

< PREV (HTTPS://CLOUD.GOOGLE.COM/DOTNET/DOCS/GETTING-STARTED/USING-CLOUD-STORAGE)

NEXT > (HTTPS://CLOUD.GOOGLE.COM/DOTNET/DOCS/GETTING-STARTED/AUTHENTICATE-USERS)