Writing and Responding to Pub/Sub Messages

Python (/appengine/docs/flexible/python/writing-and-responding-to-pub-sub-messages) | Java (/appengine/docs/flexible/java/writing-and-responding-to-pub-sub-messages) | Node.js (/appengine/docs/flexible/nodejs/writing-and-responding-to-pub-sub-messages) | Go (/appengine/docs/flexible/go/writing-and-responding-to-pub-sub-messages) | Ruby (/appengine/docs/flexible/ruby/writing-and-responding-to-pub-sub-messages) | PHP (/appengine/docs/flexible/php/writing-and-responding-to-pub-sub-messages) | **.NET**

Pub/Sub (/pubsub/docs) provides reliable, many-to-many, asynchronous messaging between applications. Publisher applications can send messages to a *topic*, and other applications can subscribe to that topic to receive the messages.

This document describes how to use the Cloud Client Library (https://googleapis.github.io/google-cloud-dotnet/docs/Google.Cloud.PubSub.V1/) to send and receive Pub/Sub messages in a .NET app.

Prerequisites

- Follow the instructions in "Hello, World!" for .NET on App Engine (/appengine/docs/flexible/dotnet/quickstart) to set up your environment and project, and to understand how App Engine .NET apps are structured.
- Write down and save your project ID, because you will need it to run the sample application described in this document.

* To download credentials from the Google Cloud Console to create a service account and add the service account key to your application, follow step 3 outlined on GitHub (https://github.com/GoogleCloudPlatform/dotnet-docs-samples/blob/927526578de0644a09a1e6611c848bd4f78f7a65/README.md#build-and-run)
. * Enable the Google Cloud Pub/Sub API.

Enable the API (https://console.cloud.google.com/flows/enableapi?apiid=pubsub)

Cloning the sample app

Copy the sample apps to your local machine, and navigate to the `pubsub` directory:

```
lone https://github.com/GoogleCloudPlatform/dotnet-docs-samples
tnet-docs-samples/appengine/flexible/pubsub
```

Creating a topic and subscription

Create a topic and subscription, which includes specifying the endpoint to which the Pub/Sub server should send requests:

```
d pubsub topics create YOUR_TOPIC
d pubsub subscriptions create YOUR_SUBSCRIPTION `
-topic YOUR_TOPIC `
-push-endpoint `
ttps://YOUR_PROJECT_ID.REGION_ID(#pubsub-urls).r.appspot.com/pubsub/push?token=YOUR_SE
-ack-deadline 10
```

Setting environment variables

Edit the `appsettings.json` file to set your project ID:

appengine/flexible/Pubsub/appsettings.json (https://github.com/GoogleCloudPlatform/dotnet-docs-samples/blob/master/appengine/flexible/Pubsub/appsettings.json)

om/GoogleCloudPlatform/dotnet-docs-samples/blob/master/appengine/flexible/Pubsub/appsettings.json)

```
{
  "Pubsub": {
    "ProjectId": "your-project-id",
    "VerificationToken": "your-secret-token",
    "TopicId": "your-topic",
    "SubscriptionId": "your-subscription"
  },

  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    }
  }
}
```

Code review

The sample app uses the Cloud Client Library (https://googleapis.github.io/google-cloud-dotnet/docs/Google.Cloud.PubSub.V1/).

appengine/flexible/Pubsub/Controllers/HomeController.cs (https://github.com/GoogleCloudPlatform/dotnet-docs-samples/blob/master/appengine/flexible/Pubsub/Controllers/HomeController.cs)

udPlatform/dotnet-docs-samples/blob/master/appengine/flexible/Pubsub/Controllers/HomeController.cs)

```
[HttpGet]
[HttpPost]
public IActionResult Index(MessageForm messageForm)
{
    var model = new MessageList();
    if (!_options.HasGoodProjectId())
    {
        model.MissingProjectId = true;
        return View(model);
    }
    if (!string.IsNullOrEmpty(messageForm.Message))
    {
        // Publish the message.
        var pubsubMessage = new PubsubMessage()
        {
            Data = ByteString.CopyFromUtf8(messageForm.Message)
        };
        pubsubMessage.Attributes["token"] = _options.VerificationToken;
        _publisher.PublishAsync(pubsubMessage);
        model.PublishedMessage = messageForm.Message;
    }
    // Render the current list of messages.
    lock (s_lock) model.Messages = s_receivedMessages.ToArray();
    return View(model);
}
```

Running the sample locally

When running locally, you can use the Cloud SDK to provide authentication to use Google Cloud APIs. Assuming you set up your environment as described in Prerequisites (#prerequisites), you have already run the `gcloud init` command, which provides this authentication.

To run the sample app locally:

Visual StudioCommand line (#command-li...

1. Open `dotnet-docs-samples\appengine\flexible\AppEngineFlex.sln` with Visual Studio.
2. In Solution Explorer, right-click **Pubsub**, and choose **Debug** > **Start new instance**.

Simulating push notifications

The application can send messages locally, but it is not able to receive push messages locally. You can, however, simulate a push message by making an HTTP request to the local push notification endpoint. The sample includes the file `sample_message.json`.

To send an HTTP `POST` request:

```
ontent -Raw .\sample_message.json | Invoke-WebRequest -Uri
//localhost:5000/Push?token=your-secret-token -Method POST -ContentType
/json' -OutFile out.txt
```

After the request completes, you can refresh `localhost:5000` and see the message in the list of received messages.

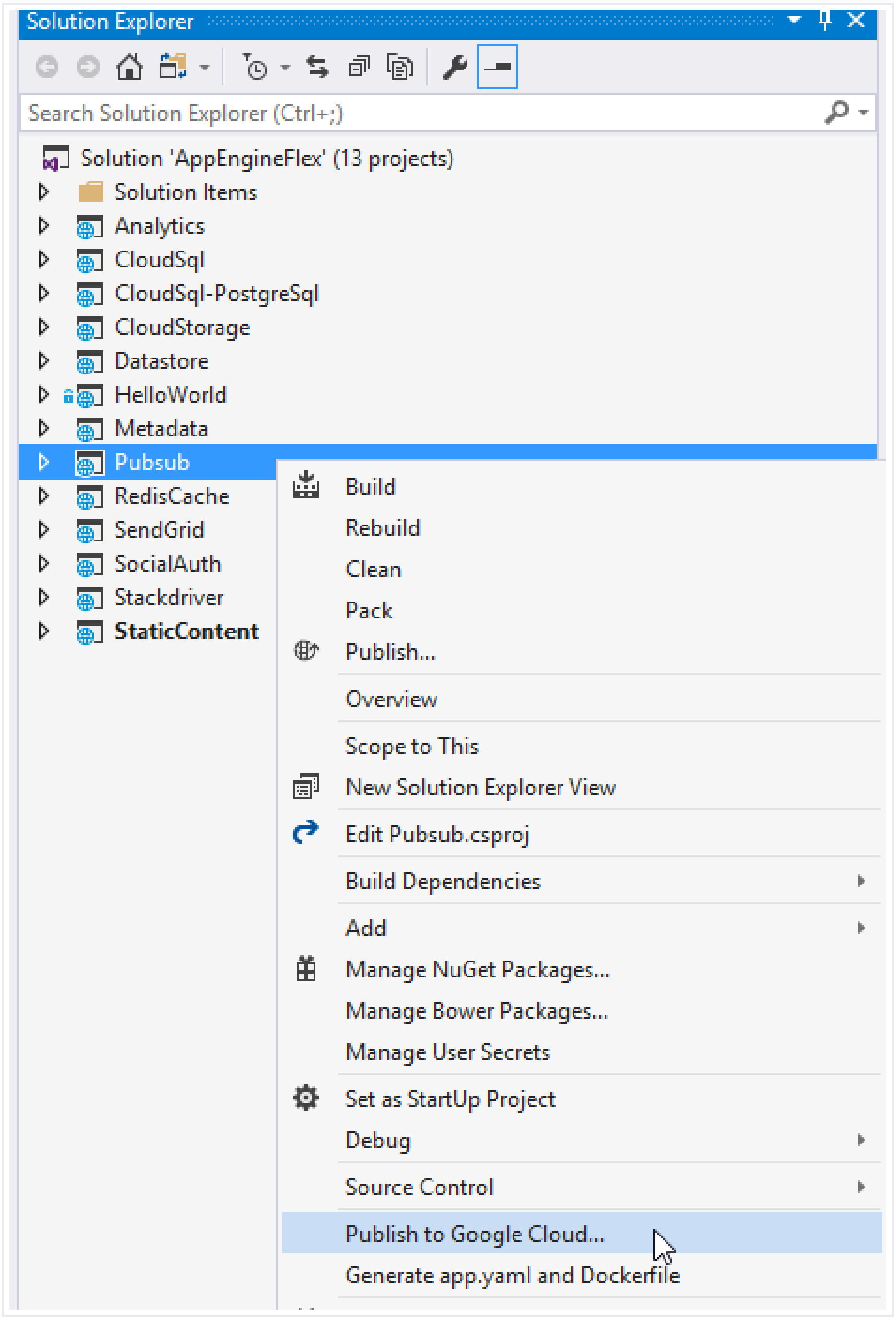
Running on App Engine

To deploy the demo app to App Engine by using the `gcloud` command-line tool, you run the following command from the directory where your `app.yaml` file is located:

Visual StudioCommand line (#command-li...

To deploy the Hello World app:

1. Open `dotnet-docs-samples\appengine\flexible\AppEngineFlex.sln` with Visual Studio.
2. In Solution Explorer, right-click **Pubsub**, and choose **Publish to Google Cloud...**



3. Click **App Engine Flex**.

4. Click **Publish**.

You can now access the application at `https://PROJECT_ID.REGION_ID`(#appengine-urls) `.r.appspot.com`. You can use the form to submit messages, but there's no guarantee of which instance of your application will receive the notification. You can send multiple messages and refresh the page to see the received message.