Deploy HttpPostSender to Azure App Service using Visual Studio Code

FACTS

Size:	n/a
Software:	ArcGIS Velocity, ArcGIS GeoEvent Server
Application/Extension:	n/a
Database:	n/a
Other Software Required:	Azure Portal, Microsoft Visual Studio Code, Azure App Service extension for VS Code, Git
Platforms Tested:	Windows 10, Mac OS X
Special Functionality:	Azure App Service
Audience:	Industry and sales teams

PURPOSE

HttpPostSender is a C#.Net console app that can be configured and run to send event messages to a REST endpoint. Intended to be deployed to an Azure portal as an Azure WebJob running in an App Service to support real-time demonstrations for Velocity and GeoEvent Server, the app provides a simulated stream of event messages so that Velocity or GeoEvent Server can receive them as if from remote sensors sending updates.

These instructions will guide you through the process of deploying your own HttpPostSender app to support your industry demos of Velocity and GeoEvent Server. Specifically, it will lead you through the following steps:

- Set up your deployment environment
- Clone the HttpPostSender app
- Configure it to use your demo simulation data and REST endpoint
- Create an App Service resource to host the HttpPostSender app
- Deploy your HttpPostSender to your App Service

DATA SOURCES AND DESCRIPTION (Refer to Data Distribution Permission statement at the end of this document)

For this demo resource one csv file is hosted in an Amazon Web Service (AWS) S3 bucket which you may use to set up your HttpPostSender . This is provided for illustrative purposes only and we generally expect most users will replace our sample csv file with their own simulation data:

1. Gather your deployment resources

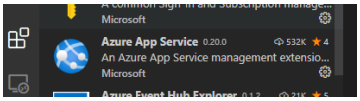
You'll need:

- An Azure account with an active subscription. Create one for free.
- An HTTP Receiver feed in Velocity or an HTTP Receiver input in GeoEvent Server.
- Visual Studio Code (VS Code) installed on your local machine. Get it here.
- The Azure App Service extension for VS Code (Get it here or install from within VS Code)
- Git installed on your local machine. Get it here.
- A delimited text file of events you wish to simulate. It must be hosted in a place where it is accessible by URL such as an Amazon S3 bucket or GitHub repo. To get started you may use the sample file hosted at https://a4iot-test-data.s3.us-west-2.amazonaws.com/point/Charlotte_Simulations/Buses_in_CharlotteNC.csv

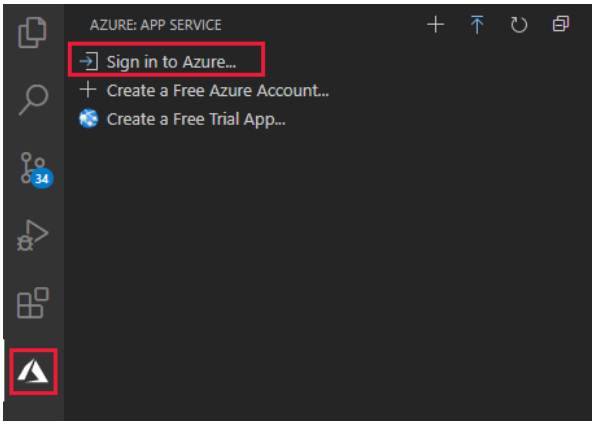
2. Sign in to Azure in VS Code

If you already use the Azure service extensions, you should already be logged in and can skip this step. If you don't use the Azure service extensions, continue in this section to install it.

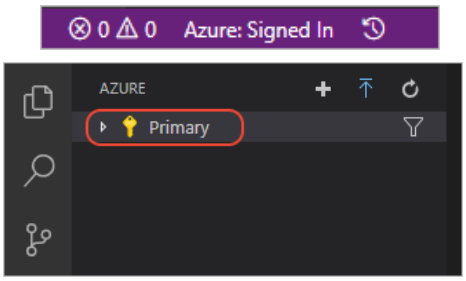
1. In VS Code click **View > Extensions**
2. In the search bar type **Azure App Service**
3. Click **Install** on the Azure App Service extension



Once you've installed the Azure App Service extension in VS Code, you need to sign into your Azure account by navigating to the **Azure Explorer**, select **Sign in to Azure**, and follow the prompts. (If you have multiple Azure extensions installed, select **App Service**.)



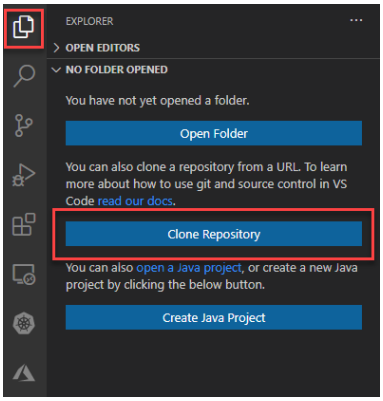
After signing in, verify that the email address of your Azure account (or "Signed In") appears in the Status Bar and your subscription(s) appears in the Azure Explorer:



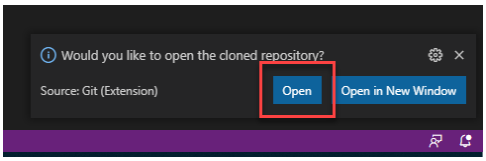
3. Clone the http-post-sender repo

Create the app by cloning a Git repository. Two methods for doing so are illustrated below.

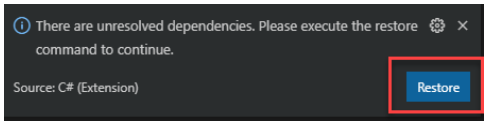
Method 1: Use Git in VS Code:



1. In VS Code, select the explorer icon to open the Files and Folders Explorer, then click **Clone Repository**
2. Paste **https://github.com/Esri/http-post-sender** into the search bar and click '**Clone from URL https://github.com/Esri/http-post-sender**'
3. Navigate to the folder where you will save the cloned application files and click **Select Repository Location**.



4. Click **Open**.



5. If a message appears indicating unresolved dependencies click **Restore**.

6. The application files appear in the **Files and Folders Explorer**.

Method 2: Use Git directly:

1. Open a terminal command prompt and change directories to the location where you want to create the app folder.

   Or

1. In Windows Explorer in the folder where you want to create the app, right click and click **Git Bash Here** to open a Git Bash command window.
2. Enter the following git command in the terminal or Git Bash window to clone the repository:

   git clone https://github.com/Esri/http-post-sender HttpPostSender

3. Change into the new *HttpPostSender* directory by running the following command:

   cd HttpPostSender

4. Install the application's dependencies by running the command:

   npm install

5. Start VS Code with the following command:

   code .

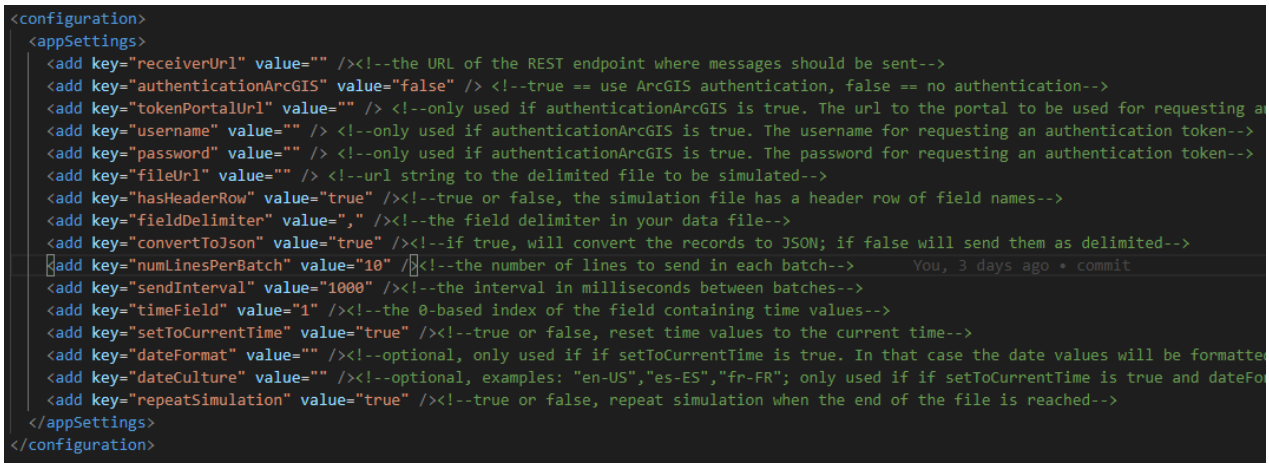
4. Update app.config to reflect your REST endpoint, credentials (if required) and simulation file

For this step you'll need the REST endpoint URL for your feed or input. To obtain this:

- In Velocity go to your feed's details page and copy the URL for HTTP endpoint path to your system clipboard.
- In GeoEvent Server copy the URL property for your input path to your system clipboard.

The app.config file in the HttpPostSender code files contains a set of key/value pairs that the app uses to initialize settings for your deployment. You will need to update some of the values in this file in order to configure the app for your needs.

4. In VS Code's **File and Folders Explorer** click app.config to open the file in the VS Code editor.



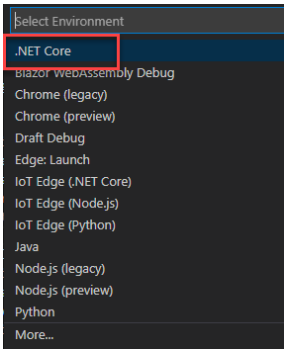
5. Enter or update values for the keys as follows:
   - receiverUrl – In the empty quotes after 'value=' paste the complete URL you copied above.
   - authenticationArcGIS – True if your Velocity feed requires ArcGIS authentication, false if not.
   - tokenPortalUrl – Used only if authenticationArcGIS is true. The root url to the ArcGIS portal to be used for obtaining a token.
   - username – Used only if authenticationArcGIS is true. The username for generating a token.
   - password – Used only if authenticationArcGIS is true. The password for generating a token.
   - fileUrl – Enter the URL to the simulation delimited file containing the data to be sent. If using our sample file, set this value to "**https://a4iot-test-data.s3.us-west-2.amazonaws.com/point/Charlotte_Simulations/Buses_in_CharlotteNC.csv"**.
   - hasHeaderRow – Enter true or false to indicate whether the simulation file has a header row of field names. If using our sample csv file, set this value to **"true"**.
   - fieldDelimiter – the delimiter between fields in the simulation file. If using our sample csv file, set this value to **","**.
   - convertToJson - Enter true to convert the delimited records to JSON, false to leave them in delimited format.

- numLinesPerBatch – Enter the number of lines to send with each batch. The app will read this number of lines from the simulation file, bundle them into a batch of events and send them to the REST endpoint all at once. Then it will read the next set of lines into a batch, send them and repeat until the end of the simulation file is reached and all lines have been sent. You might set this value to be equal to the number of unique track IDs in your data or use it in conjunction with the sendInterval to simply control the rate of events into your REST endpoint. If using our sample csv file, there are **57** unique track IDs.
- sendInterval – Enter the number of milliseconds between batches sent to the REST endpoint. This time includes the time required to send a batch. Thus, if this value is set to 1000ms, and it takes 700ms to send a batch, the app will wait 300ms before sending the next batch. If it takes longer than this value to send a batch, it will not wait before sending the next batch.
- timeField – The zero-based index of the field in the simulation file containing date values. If using our sample csv file, set this value to **"0"**.
- setToCurrentTime – Enter true or false to indicate whether to update the values in the date field to the date and time the event is sent to the REST endpoint. If using our sample csv file, set this value to "**true"**.
- dateFormat – Optional, only used if setToCurrentTime is true. In that case the date values will be formatted as strings according to this formatter. If this value is empty, date values will be epochs. Formatting string can be standard or custom. See https://docs.microsoft.com/en-us/dotnet/standard/base-types/standard-date-and-time-format-strings and https://docs.microsoft.com/en-us/dotnet/standard/base-types/custom-date-and-time-format-strings
- dateCulture - Optional, examples: "en-US", "es-ES", "fr-FR"; only used if setToCurrentTime is true and dateFormat is not empty. In that case date strings will be formatted according to the culture specified in this setting or the default culture if empty
- repeatSimulation – Enter true or false to indicate if the app, upon reaching the end of the simulation file, should return to the top of the file and repeat the simulation.

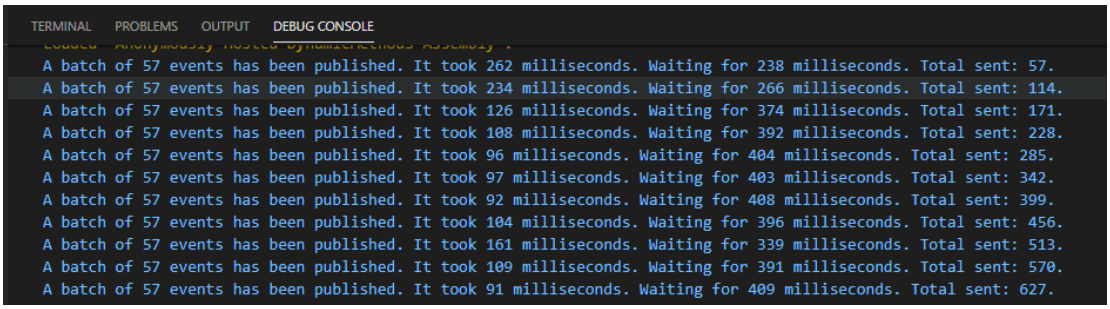
6. Click File > Save.

5. (Optional) Run your local HttpPostSender app

1. In VS Code click Run > Start Debugging.
2. If a message appears prompting you to select an Environment, select **.NET Core**.



After a pause while the app initializes and loads in the simulation file you configured, in the VS Code Debug Console you should see a scrolling list of messages indicating that the app has sent messages to your REST endpoint in the format "A batch of <<*numLinesPerBatch*>> events has been sent. It took *n*

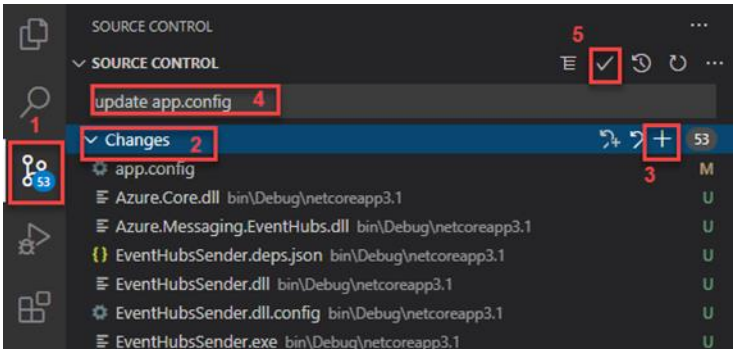


milliseconds. Waiting for <<*sendInterval* - *n*>> milliseconds. Total sent: <<*numLinesPerBatch* * number of batches sent>>.

3. Commit changes in the app.config file to the local repo

Having edited the app.config file, you must commit the changes to your local repository so that they will be reflected in the published App Service.

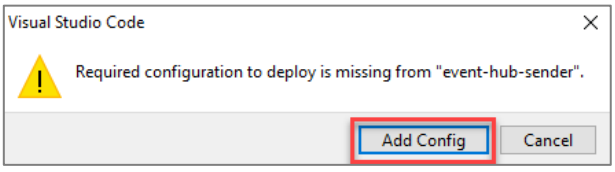
1. Open the Source Control explorer.
2. Select the Changes list.
3. Click the **+** button to stage all changes for a commit.
4. Enter a comment such as “updated app.config” to indicate the reason for the commit.
5. Click the checkmark button ✓ to commit the changes.



4. Create App service resource in VS Code

1. From the command palette (**Ctrl**+**Shift**+**P** on Windows**, Cmd**+**Shift**+**P** on Mac), type "create web" and select **Azure App Service: Create New Web App...Advanced**. You use the advanced command to have full control over the deployment including resource group, App Service Plan, and operating system rather than use Linux defaults.

2. If a message appears indicating a required configuration to deploy is missing, click **Add Config**.

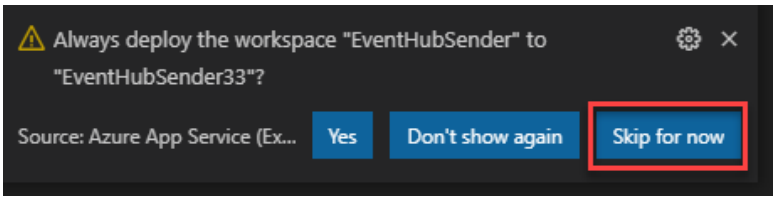


3. Respond to the prompts as follows:

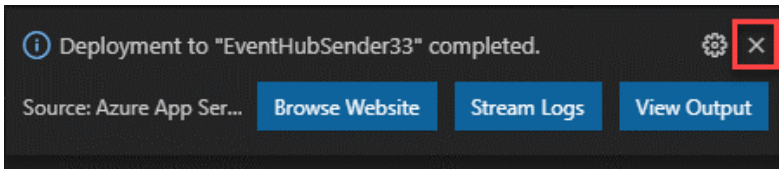
   - Select your **Subscription** account.
   - For **Enter a globally unique name**, enter a name that's unique across all of Azure. Use only alphanumeric characters ('A-Z', 'a-z', and '0-9') and hyphens ('-')

- Select **Create new resource group** and provide a name like HttpPostSender-rg.
- Select the **.Net Core 3.1 (LTS)** runtime stack.
- Select an operating system (Windows or Linux).
- Select **Create a new App Service plan**, provide a name like HttpPostSender-plan, and select any pricing tier. The app will incur costs on any pricing tier other than **F1 Free**.
- Select **Skip for now** for the Application Insights resource.
- Select a location near you or where you want the app to run.

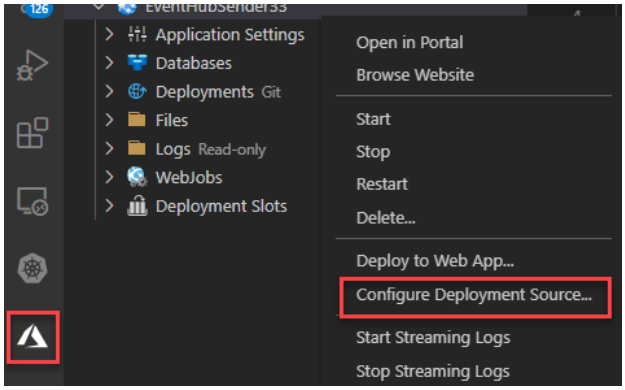
If a message asks to 'Always deploy the workspace HttpPostSender to ' this App Service click **Skip for now**.



4. After a short time, VS Code notifies you that creation is complete. Close the notification with the **X** button:



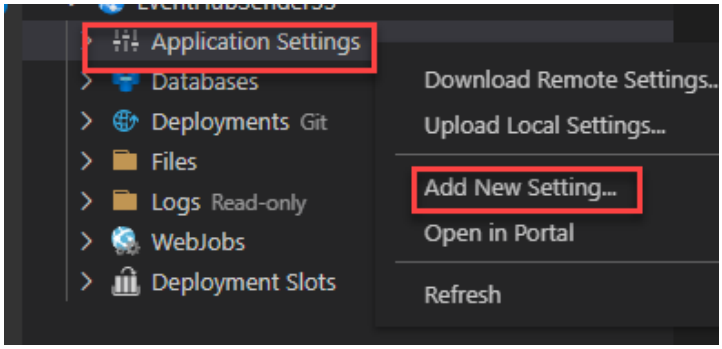
5. With the web app in place, you next instruct VS Code to deploy your code from the local Git repo. Select the Azure icon to open the **Azure App Service** explorer, expand your subscription node, right-click the name of the web app you just created, and select **Configure Deployment Source**.



6. When prompted, select **LocalGit**.

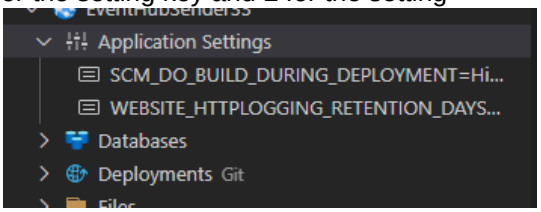
If deploying to an App Service on Windows, you need to create an additional setting before deployment:

i. In VS Code, expand the node for the new App Service, right-click **Application Settings**, and select **Add New Setting**:



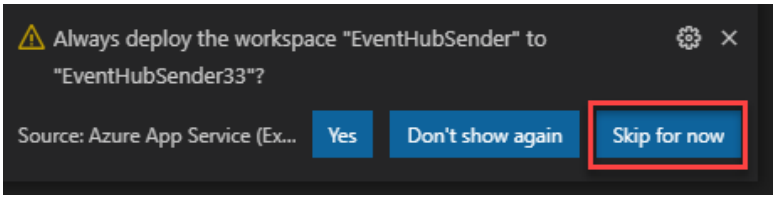
ii. Enter **SCM_DO_BUILD_DURING_DEPLOYMENT** for the setting key and **1** for the setting value. This setting forces the server to run npm install upon deployment.

iii. Expand the **Application Settings** node to verify the setting is in place.



7. Select the blue up arrow icon to deploy your code to Azure:

8. At the prompts, select your **subscription** account again and then select the name of the web app created earlier.
9. If a message appears about uncommitted change(s) in your local repo, click **Deploy Anyway**.
10. When deploying to Linux, select **Yes** when prompted to update your configuration to run npm install on the target server.
11. If a message asks to 'Always deploy the workspace HttpPostSender to' this App Service click **Skip for now**.



Congratulations. You have deployed an Azure App Service and WebJob that continuously sends event messages to the REST endpoint for your Velocity HTTP Receiver feed or GeoEvent Server input.

DATA DISTRIBUTION PERMISSION

Esri – the sample data and related material(s) may be used for **Esri internal use only**:

_X_ Demonstrations/Benchmarks

_X_ Trade Shows/Esri User Conferences

_X_ Esri World Wide Web

_X_ Esri Educational/Training Materials

_X_ General Marketing, Advertising, and Promotion (i.e. brochures, slides, videos, etc.)

Esri Domestic Business Partners – the sample data and related material(s) may be forwarded to Esri Business Partners for:

_X_ Demonstrations/Benchmarks

_X_ Trade Shows/Esri User Conferences

Esri International Distributors – the sample data and related material(s) may be forwarded to Esri International Distributors for:

_X_ Demonstrations/Benchmarks

_X_ Trade Shows/Esri User Conferences

