## Debugging Your Locally Hosted v4 Bot Using Azure Relays

by Steven Kanberg

March 26, 2019

This blog explains how to set up an Azure Relay resource, create the necessary relays in Visual Studio, and connect your locally hosted v4 bot for debugging across the various channels available in the Bot Framework ecosystem.

### Prerequisites:

* An Azure account (create a new account [here](https://azure.microsoft.com/en-us/free/))
* Access to Visual Studio
  + .Net Framework 4.7 or .Net Core 2.1 installed (project dependent)
* Access to VS Code
  + dotnet (.Net Core CLI - read about and install from [here](https://docs.microsoft.com/en-us/dotnet/core/tools/?tabs=netcore2x))
* A bot developed using one of the Bot Framework v4 SDKs (create your first bot [here](https://docs.microsoft.com/en-us/azure/bot-service/?view=azure-bot-service-4.0))

### Azure Relay

[Azure Relay](https://docs.microsoft.com/en-us/azure/service-bus-relay/) has several features that make it an excellent option and alternative to ngrok:

* You can debug your local v4 bot with Azure Relay.
* The resource owner manages and controls the environment.
* It allows for multiple application connections.
* It is a suitable alternative to ngrok (read this [blog](https://blog.botframework.com/2017/10/19/debug-channel-locally-using-ngrok/) to learn more about using ngrok).

Please note the following:

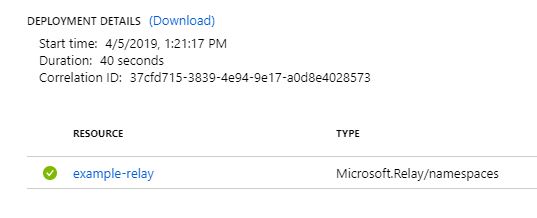
Azure Relay and Azure Service Bus relays are two closely related yet different resources. The steps provided here are specific to creating an Azure Relay resource.

Corporate customers need to consider their organizational IT and network policies before using in a production environment.

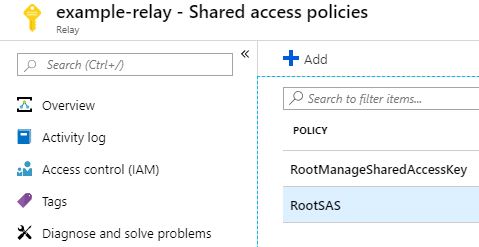
An equivalent JavaScript solution does not exist, yet. However, the relevant NPM libraries do and can be accessed here for development projects: [azure-sb](https://www.npmjs.com/package/azure-sb) and [azure-arm-relay](https://www.npmjs.com/package/azure-arm-relay).

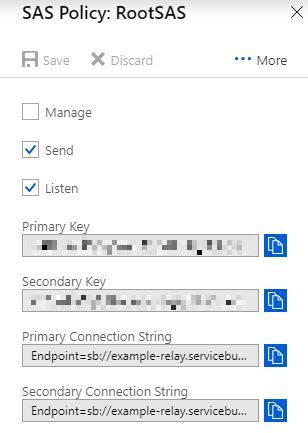
To get started, follow these steps:

1. Navigate to the [Azure portal](https://portal.azure.com/) and log in.
2. Select the **Create a resource** option in the sidebar menu.
3. Search for **Relay** in the search field and select this option from the list once displayed.
4. Click **Create** to begin building your relay resource.
5. Complete each field (Name, Subscription, Resource group, Location) to create the relay namespace.
6. Click **Create** to finalize the creation process.
7. Once completed, an overview of the deployment is shown. Click the resource name you just created, under **Resource**.

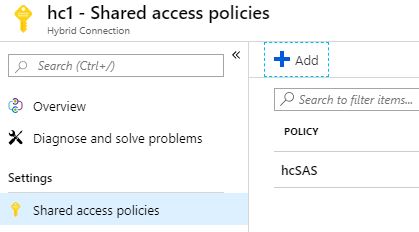
[](https://github.com/stevkan/blogs/blob/master/Using%20Azure%20Service%20Bus%20With%20Your%20Bot/images/service_bus_deployment.jpg)

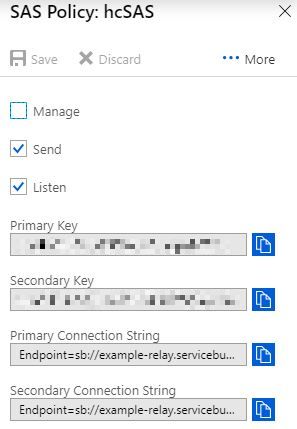
1. Select the **Shared access policies** blade located under **Settings**.
2. Click the **+ Add** button to create a new policy. Enter a **Policy name** and check both **Send** and **Listen**. For security reasons, be sure to leave **Manage** unchecked.
3. Copy the policy name down and Click **Create**.
4. Once the new policy has been created, select it and copy down the **Primary Key**.

[](https://github.com/stevkan/blogs/blob/master/Using%20Azure%20Service%20Bus%20With%20Your%20Bot/images/root_sas.jpg)



1. (Optional) If you plan on compiling the .Net Core version of the project, a Hybrid Connection will need to be created manually.
2. Select **Hybrid Connections** to open the blade.
3. Click **+ Hybrid Connection**. Enter a **Name** and leave all other values empty. If **Requires Client Authentication** is checked, clear the checkbox as client authentication is not required.
4. Copy the name down and click **Create**.
5. Once the hybrid connection has been created, select it. This will open the respective blade.
6. Select the **Shared access policies** blade located under **Settings**.
7. Click the **+ Add** button to create a new policy. Enter a **Policy name** and check both **Send** and **Listen**. For security reasons, be sure to leave **Manage** unchecked.
8. Copy the policy name down and click **Create**.
9. Once the new policy has been created, select it and copy down the **Primary Key**.

[](https://github.com/stevkan/blogs/blob/master/Using%20Azure%20Service%20Bus%20With%20Your%20Bot/images/hc_sas.jpg)



1. Return to the Azure portal dashboard.

### Azure Bot Service

An Azure Web Bot is required for routing between your locally hosted bot and the relay. You can use a bot you have already created or follow these instructions on how to [Create a bot with Azure Bot Service](https://docs.microsoft.com/en-us/azure/bot-service/bot-service-quickstart?view=azure-bot-service-4.0) to get started.

### Azure Service Bus Bot Relay

The Azure Service Bus Bot Relay is a GitHub project that can be compiled in either Visual Studio or VS Code and run locally to monitor your localhost address. It utilizes Azure Relay, using Azure Service Bus as its backbone, and serves as the connection between your locally hosted bot and the relay routing through your Azure Web Bot.

It functions like ngrok’s tunneling service by constructing a messaging service on Azure from which the bot can route through. In this way, you can debug your locally hosted bot on different channels.

However, unlike ngrok, the relay service streams user activity by displaying the full activity object. This is particularly useful in debugging scenarios.

This solution offers two options for compiling: .Net Framework and .Net Core. Using either will provide the same connectivity, however, note that:

* Visual Studio can run and compile both .Net Framework and .Net Core.
* VS Code only supports running and compiling .Net Core projects via the **dotnet** command.
* .Net Framework programmatically creates a WCF relay when started. This relay is viewable online in your Azure Relay resource for the duration the relay is running locally.
* .Net Core requires manually creating the hybrid connection on Azure. Hybrid connections should not be shared by multiple users as the activity from each will conflict with each other. Instead, create a separate hybrid connection for each user.
* .Net Core provides cross-platform functionality and will run in both the Windows and Mac OS environments.

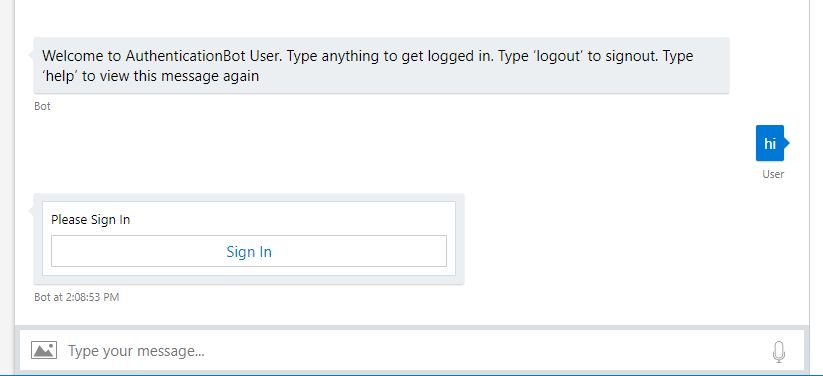
Before you can get started you will need a local copy of the solution. Navigate to the project’s GitHub [page](https://github.com/gabog/AzureServiceBusBotRelay) in a browser and clone the solution to your machine.

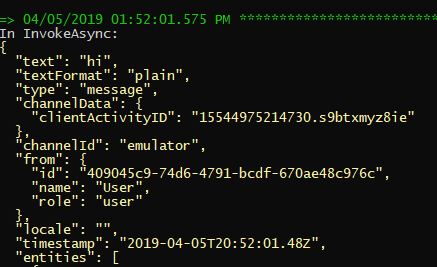
### Building with .Net Framework

1. Once the solution has been cloned to your machine, open the solution in Visual Studio.
2. In Solution Explorer, expand the **ServiceBusRelayUtil** folder.
3. Open the **App.config** file and replace the following values with those from your relay (not the hybrid connection).
4. " RelayNamespace" is the name of your relay created earlier. Enter the value in place of **[Your Namespace]**.
5. " RelayName" is the name of the shared access policy created in steps 9 through 11 during the relay set up process. Enter the value in place of **[Your Relay Name]**.
6. " PolicyName" is the value to the shared access policy created in steps 9 through 11 during the relay set up process. Enter the value in place of **[Your Shared Access Policy Name]**.
7. " PolicyKey" is the WCF relay to be used. Remember, this relay is programmatically created and only exists on your machine. Create a new, unused name and enter the value in place of **[Your Policy's Key]**.
8. "TargetServiceAddress" sets the port to be used for localhost. The address and port number should match the address and port used by your bot. Enter a value in place of the [PORT] string part. For example, "<http://localhost:[PORT]>".

[](https://github.com/stevkan/blogs/blob/master/Using%20Azure%20Service%20Bus%20With%20Your%20Bot/images/appConfig.jpg)

1. Before testing the relay, your Azure Web Bot's messaging endpoint must be updated to match the relay.
2. Login to the Azure portal and open your Web App Bot.
3. Select **Settings** under Bot management to open the settings blade.
4. In the **Messaging endpoint** field, enter the relay namespace and relay. The relay should match the relay name entered in the **App.config** file and should not exist in Azure.
5. Append **"/api/messages"** to the end to create the full endpoint to be used. For example, “<https://example-service-bus.servicebus.windows.net/wcf-example-relay/api/messages>".
6. Click **Save** when completed.
7. In Visual Studio, press **F5** to run the project.
8. Open and run your locally hosted bot.
9. Test your bot on a channel (Test in Web Chat, Skype, Teams, etc.). User data is captured and logged as activity occurs.
   * When using the Bot Framework Emulator: The endpoint entered in Emulator must be the relay endpoint saved in your Azure Web Bot **Settings** blade, under **Messaging Endpoint**.



[](https://github.com/stevkan/blogs/blob/master/Using%20Azure%20Service%20Bus%20With%20Your%20Bot/images/wcf-results.jpg)

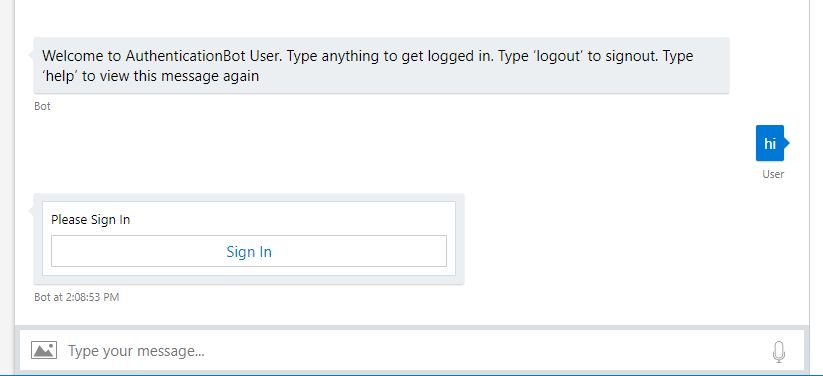
1. Once testing is completed, you can compile the project into an executable.
2. Right click the project folder in Visual Studio and select **Build**.
3. The .exe will output to the **/bin/debug** folder, along with other necessary files, located in the project’s directory folder. All the files are necessary to run and should be included when moving the .exe to a new folder/location.
   * The **app.config** is in the same folder and can be edited as credentials change without needing to recompile the project.

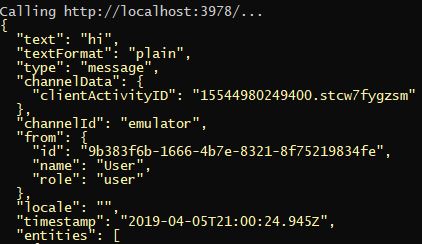
### Building with .Net Core

1. Once the solution has been cloned to your machine, open the solution in Visual Studio.
2. In Solution Explorer, expand the **ServiceBusRelayUtilNetCore** folder.
3. Open the **appsettings.json** file and replace the following values with those from your relay hybrid connection.
4. "RelayNamespace" is the name of your relay created earlier. Enter the value in place of **[Your Namespace]**.
5. " RelayName" is the name of the hybrid connection created in step 12. Enter the value in place of **[Your Relay Name]**.
6. " PolicyName" is the name of the shared access policy created in steps 9 through 11 during the relay set up process. Enter the value in place of **[Your Shared Access Policy Name]**.
7. " PolicyKey" is the value to the shared access policy created in steps 9 through 11 during the relay set up process. Enter the value in place of **[Your Policy's Key]**.
8. "TargetServiceAddress" sets the port to be used for localhost. The address and port number should match the address and port used by your bot. Enter a value in place of the **"http://localhost:[PORT]"**. For example, "[http://localhost:3978](http://localhost:3978/)".

[](https://github.com/stevkan/blogs/blob/master/Using%20Azure%20Service%20Bus%20With%20Your%20Bot/images/appsettings.jpg)

1. Before testing the relay, your Azure Web Bot's messaging endpoint must be updated to match the relay.
2. Login to the Azure portal and open your Web App Bot.
3. Select **Settings** under Bot management to open the settings blade.
4. In the **Messaging endpoint** field, enter the relay namespace and relay.
5. Append **"/api/messages"** to the end to create the full endpoint to be used. For example, “<https://example-service-bus.servicebus.windows.net/hc1/api/messages>".
6. Click **Save** when completed.
7. In Visual Studio, press **F5** to run the project.
8. Open and run your locally hosted bot.
9. Test your bot on a channel (Test in Web Chat, Skype, Teams, etc.). User data is captured and logged as activity occurs.
   * When using the Bot Framework Emulator: The endpoint entered in Emulator must be the relay endpoint saved in your Azure Web Bot **Settings** blade, under **Messaging Endpoint**.



[](https://github.com/stevkan/blogs/blob/master/Using%20Azure%20Service%20Bus%20With%20Your%20Bot/images/hc-results.jpg)

1. Once testing is completed, you can compile the project into an executable.
2. Right click the project folder in Visual Studio and select **Publish**.
3. For **Pick a publish Target**, select **Folder**.
4. For **Folder or File Share**, choose an output location or keep the default.
5. Click **Create Profile** to create a publish profile.
6. Click **Configure...** to change the build configuration and change the following:
   * **Configuration** to "Debug | Any CPU"
   * **Deployment Mode** to "Self-contained"
   * **Target Runtime** to "win-x64"
7. Click **Save** and then **Publish**
8. The .exe will output to the **/bin/debug** folder, along with other necessary files, located in the project’s directory folder. All the files are necessary to run and should be included when moving the .exe to a new folder/location.
   * The **appsettings.json** is in the same folder and can be edited as credentials change without needing to recompile the project.