Practical WebJobs SDK and Extensions

Demo Script

Setup – Do before demo

1. Create 2 storage accounts for WebJobs to use
2. Create SendGrid account to use

Demo 1 - Triggering a function's logic by watching a queue

1. Create console app
2. Add nuget package Microsoft.Azure.WebJobs
3. Add following code to the main:

```
    var config = new JobHostConfiguration();
    config.UseDevelopmentSettings();

    var host = new JobHost(config);
    host.RunAndBlock();
```

4. Create Functions.cs
5. Add following to file

```
public class Functions
{
    public static void ProcessMessage([QueueTrigger("messages")] string message)
    {
        Console.WriteLine($"Message arrived: {message}");
    }
}
```

6. Add following to app config

```
<connectionStrings>
  <add name="AzureWebJobsDashboard"
connectionString="DefaultEndpointsProtocol=https;AccountName=<your account
name>;AccountKey=<your storage key>" />
  <add name="AzureWebJobsStorage"
connectionString="DefaultEndpointsProtocol=https;AccountName=<your account
name>;AccountKey=<your storage key>" />
</connectionStrings>
```

7. Run console app
8. Open Azure Management Studio and connect to storage account
9. Set breakpoint in function
10. Create Queue named "messages"
11. Add message to queue

Demo 2 - How to send a file to blob storage

1. Start with where Demo1 leaves off
2. Replace ProcessMessage function with the following:

```
public static void ProcessMessage_Demo2(
  [QueueTrigger("messages")] string message,
  [Blob("bostonazure/messages.txt", FileAccess.Write)] out string messagelog)
  {
      Console.WriteLine($"Message arrived: {message}");

      messagelog = message;
  }
```

3. Run console app
4. Open Azure Management Studio and connect storage account
5. Create Queue messages if it isn't there
6. Add a message to the queue
7. Go to the blobs
8. Notice the new created container
9. Find the messages.txt file and open it
10. Add another message and repeat
11. Notice the file contents have been replaced

Demo 3 - How to customize the serialization when using the Blob binding

1. Start where Demo2 left off
2. Add class Message to the project with the following implementation

```
public class Message
{
    public string Title { get; set; }
    public string Text { get; set; }
}
```

3. Add a class MessageBlobBinder to the project with the following implementation

```
public class MessageBlobBinder : ICloudBlobStreamBinder<Message>
{
    public Task<Message> ReadFromStreamAsync(Stream input,
        CancellationToken cancellationToken)
    {
        using (StreamReader reader = new StreamReader(input))
        {
            var jsonString = reader.ReadToEnd();
            var order = JsonConvert.DeserializeObject<Message>(jsonString);
            return Task.FromResult(order);
        }
    }

    public Task WriteToStreamAsync(Message value, Stream output,
        CancellationToken cancellationToken)
```

```
        {
            var serializer = new JsonSerializer();
            using (var writer = new StreamWriter(output))
            using (var jsonTextWriter = new JsonTextWriter(writer))
            {
                serializer.Serialize(jsonTextWriter, value);
            }
            return Task.FromResult<object>(null);
        }
    }
```

4. In the Functions class, change the ProcessMessage method to the following

```
    public static void ProcessMessage_Demo3(
        [QueueTrigger("messages")] Message message,
        [Blob("bostonazure/{title}.txt")] out Message messagelog)
    {
        Console.WriteLine($"Message arrived: {message.Title} {message.Text}");

        messagelog = message;
    }
```

5. Open Azure Management Studio, go to the messages queue and add the following message (notice the Title is spelt wrong):

```
{ Tilte: "test1", Text: "this is a binding test" }
```

6. Watch the error go through the console
7. Find the Messages_Poison queue and look at the message
8. Go back to the messages queue and put the following correct message in:

```
{ Title: "test1", Text: "this is a binding test" }
```

9. Go to the blob container and find the message and open it

Demo 4 - How to surface your own logging from within a WebJob

1. Start with where Demo 3 ended
2. Add a class named ColorConsoleTraceWriter to the project with the following contents:

```
    public class ColorConsoleTraceWriter : TraceWriter
    {
        public ColorConsoleTraceWriter(TraceLevel level)
            : base(level)
        {
        }

        public override void Trace(TraceEvent traceEvent)
        {
            var holdColor = Console.ForegroundColor;
```

```
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.WriteLine(traceEvent.Message);

            Console.ForegroundColor = holdColor;
        }
    }
```

3. Add the following line to the Program.cs under the config:

```
config.Tracing.Tracers.Add(new ColorConsoleTraceWriter(TraceLevel.Info));
```

4. Run the console app and notice the yellow output
5. Go to Azure management studio and put a message on the messages queue

```
{ Title: "test1", Text: "this is a binding test" }
```

6. Notice the text written to the Console.Writeline is not in yellow
7. Modify the ProcessMessages function in Functions class to the following:

```
    public static void ProcessMessage_Demo4(
        [QueueTrigger("messages")] Message message,
        [Blob("bostonazure/{title}.txt")] out Message messagelog,
        TextWriter log)
    {
        log.WriteLine($"Message arrived: {message.Title} {message.Text}");

        messagelog = message;
    }
```

8. Go back to Azure Management studio and put another message on the messages queue

```
    { Title: "test1", Text: "this is a binding test" }
```

9. Notice the Console.WriteLine is now duplicated in yellow like the other messages

Demo 5 - Work with multiple Azure storage accounts

1. Start off with the end of Demo 4
2. Add the following connection string to the app.config file

```
<add name="SecondaryStorageAccount"
connectionString="DefaultEndpointsProtocol=https;AccountName=<your secondary storage
account name>;AccountKey=<your secondary storage account key>"/>
```

3. Modify the ProcessMessages function in the Functions class to the following

```

```

```
public static void ProcessMessage_Demo4(
    [QueueTrigger("messages")] Message message,
    [Blob("bostonazure/{title}.txt"), StorageAccount("Secondary")] out Message
messagelog,
    TextWriter log)
{
    log.WriteLine($"Message arrived: {message.Title} {message.Text}");

    messagelog = message;
}
```

4. Open Azure Management Studio add a new message to the messages queue:

```
{ Title: "test2", Text: "this should go to the secondary account" }
```

5. Look in the blob container and verify it isn't there
6. Go to the secondary storage account
7. Look into the blob container and find the message

Demo 6 - How to use the Table binding to store data in table storage

1. Start with where demo 5 ended
2. Add a class to the project named LogItemEntity with the following implementation:

```
public class LogItemEntity : TableEntity
{
    public LogItemEntity()
    { }

    public LogItemEntity(string level, string message)
    {
        Level = level;
        Message = message;
        LogUtc = DateTime.UtcNow;

        SetRowKey();
    }

    public string Level { get; set; }
    public string Message { get; set; }
    public DateTime LogUtc { get; set; }

    public void SetRowKey()
    {
        // Use a reverse data schema to keep rows sorted chronologically
        long ticks = DateTimeOffset.MaxValue.Ticks - LogUtc.Ticks;

        RowKey = $"{ticks}_{Level}";
        PartitionKey = Level;
    }
}
```

3. Change the contents of the ProcessMessage function to the following

```
    public static void ProcessMessage_Demo6(
        [QueueTrigger("messages")] Message message,
        [Blob("bostonazure/{title}.txt"), StorageAccount("Secondary")] out Message
    messagelog,
        [Table("WebJobLog")] ICollector<LogItemEntity> logTable,
        TextWriter log)
    {
        log.WriteLine($"Message arrived: {message.Title} {message.Text}");

        logTable.Add(new LogItemEntity("Info", "New message Recevied"));

        messagelog = message;

        logTable.Add(new LogItemEntity("Info", "Sent message to Secondary account"));
    }
```

4. Run the console application
5. Open Azure management studio and add a new message to the messages queue

```
{ Title: "Demo6", Text: "trying the table binding" }
```

6. Go to the tables and locate the new WebJobsLog table and look at the contents

Demo 7 - Writing your own NameResolver and why you would want to

1. Start with end of demo 6
2. Add a class named NameResolver with the following implementation

```
public class NameResolver : INameResolver
{
    public string Resolve(string name)
    {
        var settingValue = ConfigurationManager.AppSettings[name];
        if (string.IsNullOrEmpty(settingValue))
        {
            settingValue = Environment.GetEnvironmentVariable(name);
        }
        return settingValue;
    }
}
```

3. Add a reference to System.Configuration
4. Add the following line in the Main of the Program.cs below the config declaration

```
    config.NameResolver = new NameResolver();
```

5. Modify the ProcessMessage in the Functions class to the following:

```
public static void ProcessMessage_Demo7(
            [QueueTrigger("%MessagesQueueName%")] Message message,
```

```
            [Blob("%BlobConatainer%/{title}.txt"), StorageAccount("Secondary")] out
Message messagelog,
            [Table("%WebJobLogTable%")] ICollector<LogItemEntity> logTable,
            TextWriter log)
```

6. Go to the app.config and add the following keys

```
    <appSettings>
      <add key="MessagesQueueName" value="prodmessages" />
      <add key="BlobConatainer" value="prodcontainer" />
      <add key="WebJobLogTable" value="prodWebJobsLog" />
    </appSettings>
```

7. Open Azure management studio and create the queue and add a new message with it
8. Verify the newly named blob container and the table were created

Demo 8 -Executing a function on a cron schedule

1. Start with where Demo 7 ended
2. Add a nuget package for Microsoft.Azure.WebJobs.Extensions
3. Add the following method to the functions class

```
  public static void SendMessage([TimerTrigger("00:00:05")] TimerInfo timer,
      [Queue("%MessagesQueueName%")] out Message message,
      TextWriter log)
  {
      log.WriteLine("sending message");

      message = new Message()
      {
          Title = "TimedMessage",
          Text = "Sent from timer"
      };

      log.WriteLine("message sent");
  }
```

4. In the main.cs add the following below the config declaration

```
config.UseTimers();
```

5. Run the console application
6. Open Azure management studio

Demo 9 -Sending a message via SendGrid in a WebJob with very little code

1. Start were demo 8 stopped
2. Add the following to the app.config

```
  <add key="AdminEmail" value="<your email address>" />
  <add key="AzureWebJobsSendGridApiKey" value="<your send grid api key>" />
```

3. Comment out the timer function in the Functions class
4. Add a nuget rerefernce for Microsoft.Azure.WebJobs.Extensions.SendGrid
5. Modify the ProcessMessaged to look like the following

```
public static void ProcessMessage_Demo9(
    [QueueTrigger("%MessagesQueueName%")] Message message,
    [Blob("%BlobConatainer%/{title}.txt"), StorageAccount("Secondary")] out Message
messagelog,
    [Table("%WebJobLogTable%")] ICollector<LogItemEntity> logTable,
    [SendGrid] SendGridMessage email,
    TextWriter log)
{
    log.WriteLine($"Message arrived: {message.Title} {message.Text}");

    logTable.Add(new LogItemEntity("Info", "New message Recevied"));

    messagelog = message;

    logTable.Add(new LogItemEntity("Info", "Sent message to Secondary account"));

    email.AddTo("<your email address>");
    email.Subject = "Message Recieved";
    email.Text = message.Text;
}
```

6. Add the following to the program.cs file after the config declaration

```
config.UseSendGrid(new SendGridConfiguration()
{
  FromAddress = new MailAddress(
    ConfigurationManager.AppSettings["AdminEmail"])
});
```

7. Open Azure Management Studio and send a new message

```
{ Title: "Demo9", Text: "send grid testing" }
```