Sending emails from Azure Database for PostgreSQL and Azure SQL Database with Azure Functions and SendGrid on Azure

Prepared by

Data SQL Ninja Engineering Team (datasqlninja@microsoft.com)

Disclaimer

The High-Level Architecture, Migration Dispositions and guidelines in this document is developed in consultation and collaboration with Microsoft Corporation technical architects. Because Microsoft must respond to changing market conditions, this document should not be interpreted as an invitation to contract or a commitment on the part of Microsoft.

Microsoft has provided generic high-level guidance in this document with the understanding that MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THE INFORMATION CONTAINED HEREIN.

This document is provided "as-is". Information and views expressed in this document, including URL and other Internet Web site references, may change without notice.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2019 Microsoft. All rights reserved.

Note: The detail provided in this document has been harvested as part of a customer engagement sponsored through the [Data SQL Ninja Engineering](#).

Table of Contents

Introduction

Applications may need to send emails directly from the database server engine in response to completing a stored procedure execution, a maintenance task, send query results, or any other database engine operation.

Migrated databases from Oracle or other database services to Azure Database for PostgreSQL or Azure SQL Database may need to keep the capacity to send emails from the database engine to maintain application feature parity.

This document explains how to setup a solution that uses Azure Functions and the SendGrid Service on Azure to create a solution for sending emails from Azure Database for PostgreSQL or Azure SQL Database.

The accompanying .Net C# solution leverages Azure Functions to create a reliable, scalable, asynchronous, and lightweight email solution.

Solution Architecture

The solution uses [SendGrid on Microsoft Azure](#) to provide reliable email services, and Azure Functions provide the necessary code to invoke the SendGrid API.

The cloud infrastructure provides up-to-date servers to your application without the need to provision Virtual Machines or any other solution and with reduced costs.

The solution can scale vertically or horizontally by adding more resources or executing multiple Azure Functions in parallel. The frequency for pooling the database for new messages also can be adjusted. Nevertheless, all the source code is provided for custom expansion.

The integration with the databases is based on a table queue where we insert one row for each message we want to send. You can have as many database threads writing to the queue tables as well as multiple Azure Functions sending emails concurrently from the same queues.

The component that delivers email runs outside of the database engine, in a separate process. The database will continue to queue e-mail messages even if the external process stops or fails.

Solution Components:

Setup

1. Create a SendGrid Account

    a. Sign in to the [Azure portal](#)

    b. Create a SendGrid resource

    c. Create a SendGrid account and obtain an API KEY

After you follow the online instructions to create your SendGrid resource click "Manage" to manage send grid and continue the process to obtain an API KEY, including the Sender Identity confirmation that is now required.

2. Create a queue table

    a. You can create one queue table on each database you want to send messages or create a table on a central database and have other databases write to this table.

    b. Processed rows are deleted from the queue table after the email is sent. You can create triggers on the queue table if you want to save history.

    c. The message is deleted from the queue table only after its guaranteed delivery to the SendGrid API. Hence in case of API failure no messages will be lost.

d. You can run as many Azure Functions reading from the same queue tables. There is no conflict, and no emails will not be lost or sent more than once.

Create Queue Table Script for Azure Database for PostgreSQL

```sql
CREATE TABLE SendGridQueue (
    Sender                  varchar(200),
    Tos_comma_separated     varchar(2000),
    Subject                 varchar(200),
    PlainTextContent        varchar(4000),
    HtmlContent             varchar(4000),
    id                      int GENERATED ALWAYS AS IDENTITY,
    PRIMARY KEY (id)
);
```

Create Queue Table Script for Azure SQL Database

```sql
CREATE TABLE SendGridQueue (
    Sender                  varchar(200),
    Tos_comma_separated     varchar(2000),
    Subject                 varchar(200),
    PlainTextContent        varchar(4000),
    HtmlContent             varchar(4000),
    id                      int PRIMARY KEY IDENTITY
);
```

e. Here is how to insert an email message into the Queue Table for sending:

```sql
INSERT INTO sendgridqueue (sender, tos_comma_separated, subject, plaintextcontent, htmlcontent)

VALUES ('test@example.com', 'test@example.com, test@live.com', 'Test Email Subject', 'Test Email PlainText Body', '<strong>Test Email HTML Body</strong>');
```

Fields:

Sender	The "from:" email we are sending
tos_comma_separated	List of "to:" emails we are sending separate by commas
Subject	The email subject
Plaintextcontent	Plain text email body
Htmlcontent	Html email body

3. Create and Publish your Azure Function with Visual Studio 2019

    a. Open project ![icon] **AFSendMail.sln** in Visual Studio 2019

    b. Solution is a C# project of project type Azure Functions and the Target Framework is .NET CORE 3.1

    c. Required nugget references

Npgsql by Shay Rojansky, Yoh Deadfall, Austin Drenski, Emil Lenngren, Francisco Figueiredo Jr., Kenji Uno
Npgsql is the open source .NET data provider for PostgreSQL.

SendGrid by Elmer Thomas, Twilio DX Team
C# client library and examples for using Twilio SendGrid API's to send mail and access Web API v3 endpoints with .NET Standard 1.3 and .NET Core support.

System.Data.SqlClient by Microsoft
Provides the data provider for SQL Server. These classes provide access to versions of SQL Server and encapsulate database-specific protocols, including tabular data stream (TDS)

    d. If you want to run the solution locally, configure for local testing your API Key and Connection String in the solution project file local.settings.json

```
{
    "IsEncrypted": false,
    "Values": {
        "AzureWebJobsStorage": "UseDevelopmentStorage=true",
        "FUNCTIONS_WORKER_RUNTIME": "dotnet",
        "SENDGRID_API_KEY": "SG.XPP1RhuyRBu60TnOs9ekYg.VJSlQ_pG
        "CONNECTION_STRING": "Server=tcp:jmnetodb.database.wind
    }
}
```

    e. Your connection string will determine if we are using ADO.NET for Azure SQL Database or Azure Database for PostgreSQL.

Sample connections strings below:

```
Server={your_servername}.postgres.database.azure.com;Database={your_database};Port=5432;User Id={your_username}@{your_servername};Password={your_password};Ssl Mode=Require;
```

Or

```
Server=tcp:{your_servername}.database.windows.net,1433;Initial Catalog={your_database};Persist Security Info=False;User ID={your_username};Password={your_password};MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;
```

Note:

Only one type of database connection is allowed per Azure Function. If you have multiple databases servers, you must use one Azure Function for each.

This is defined in this fragment of the code:

```csharp
// Gather the connection string
string connString = Environment.GetEnvironmentVariable("CONNECTION_STRING");
if (connString == null)
    throw new Exception("CONNECTION_STRING is undefined ");

// Based of connection string hint determine if we are connecting to Azure SQL DB or Azure DB for PostgreSQL
if (connString.ToLower().Contains("postgres.database.azure.com"))
{
    ds = PostgreSQLDBHelper.RetrieveMessages(connString);
}
else if (connString.ToLower().Contains("database.windows.net"))
{
    ds = SQLDBHelper.RetrieveMessages(connString);
}
else
    throw new Exception("CONNECTION_STRING is invalid, must connect to AzureSQL Database for PostgreSQL or Azure SQL Database");
```

f. In Visual Studio 2019 Select the Option to Publish to Azure
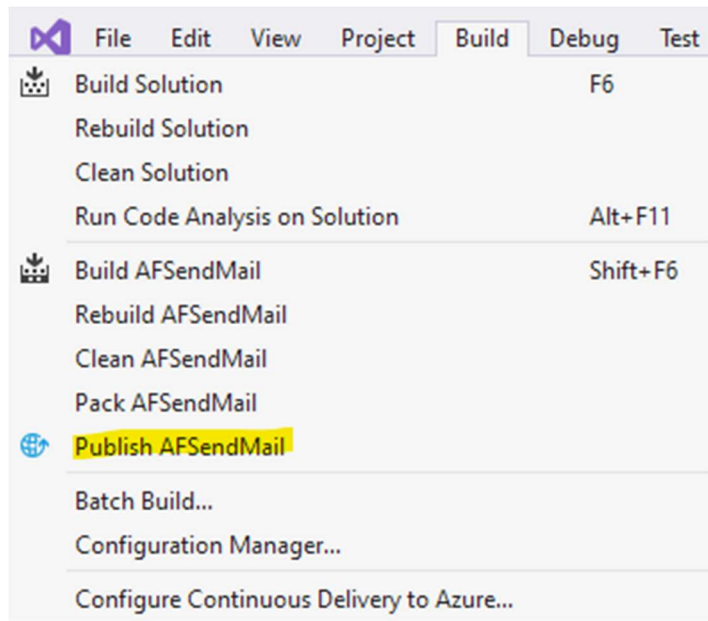
g. Before Publishing for the First Time, you will have to configure your Azure Environment to run Azure Functions, including selecting a Resource Group, Azure Function Plan and Storage Account.

More details:

[Quickstart: Create your first function in Azure using Visual Studio](#)
[Develop Azure Functions using Visual Studio](#)
[Deployment technologies in Azure Functions](#)

Note:

"Always On "option is required. To make sure your Azure Function continues running constantly, you must select **App Service Plan** when creating the host for the Azure Function. On an App Service plan, the functions runtime goes idle after a few minutes of inactivity, so only HTTP triggers will "wake up" your functions. Always on is available only on an App Service plan.
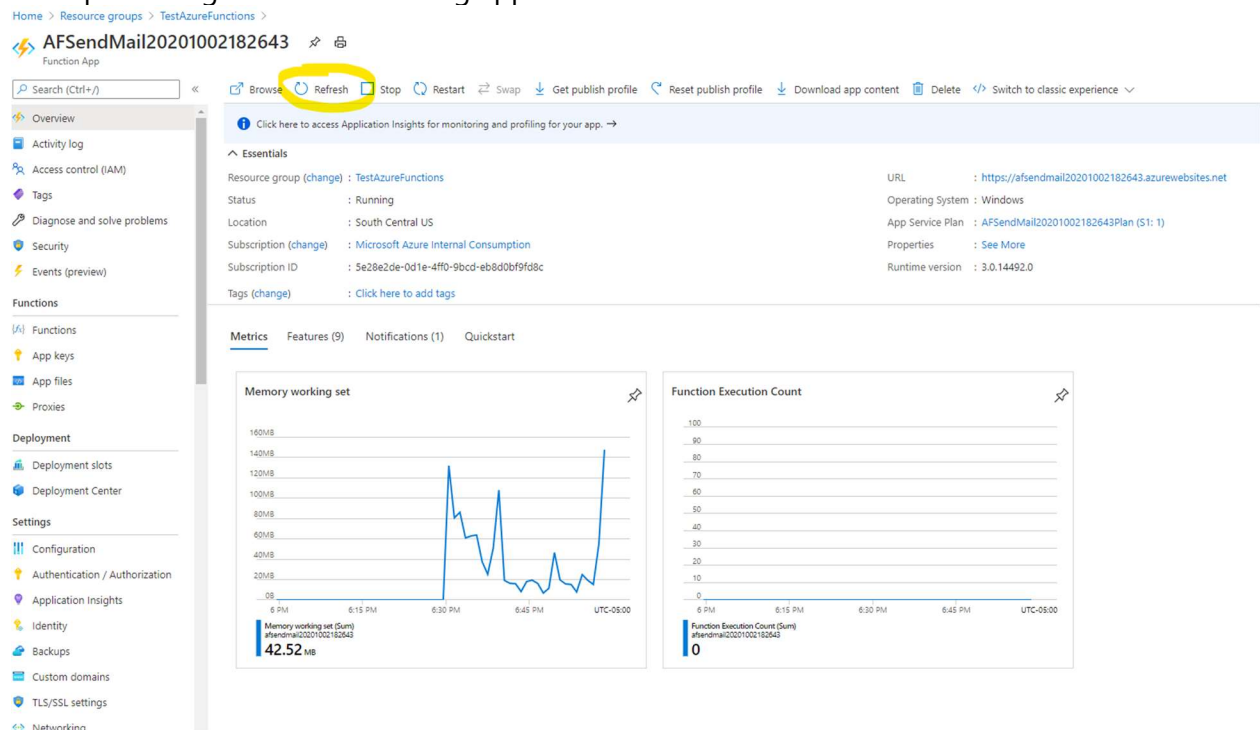
[Enable Always On when running on dedicated App Service Plan](#)

Make sure you select App Service Plan to make sure your Function stays on and running.
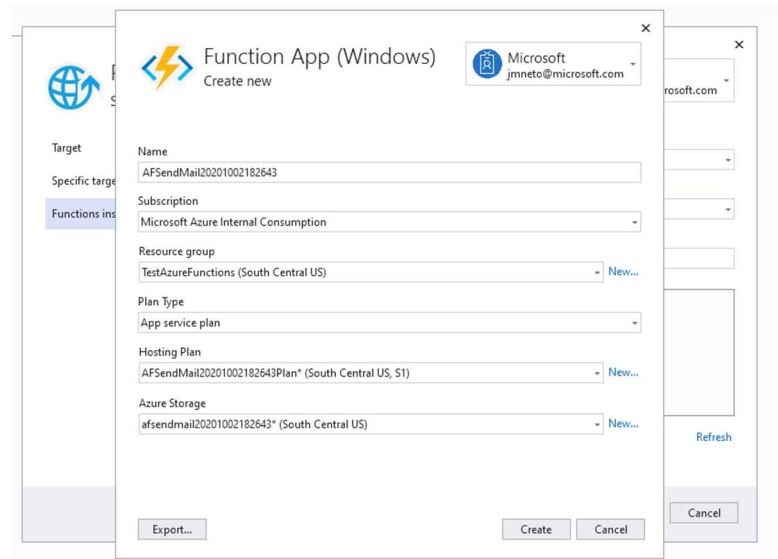
h. After publishing refresh the hosting app

4. Configure SENDGRID_API_KEY and CONNECTION_STRING in Azure Portal

a. The solution requires two Application Settings to be configured:

CONNECTION_STRING

SENDGRID_API_KEY

b. Open your function configuration screen and add Application Settings SENDGRID_API_KEY and CONNECTION_STRING

c. Configure the interval for SendGrid Azure Function to pool the Queue table and send emails.

The timer trigger is defined in this portion of the code and is defined as a NCRONTAB expression.

```
]namespace AFSendMail
{
]    public static class SendMail
    {
        [FunctionName("SendMail")]
        public static void Run([TimerTrigger("0 */1 * * * *")] TimerInfo myTimer, ILogger log)
        {
            try
            {
```
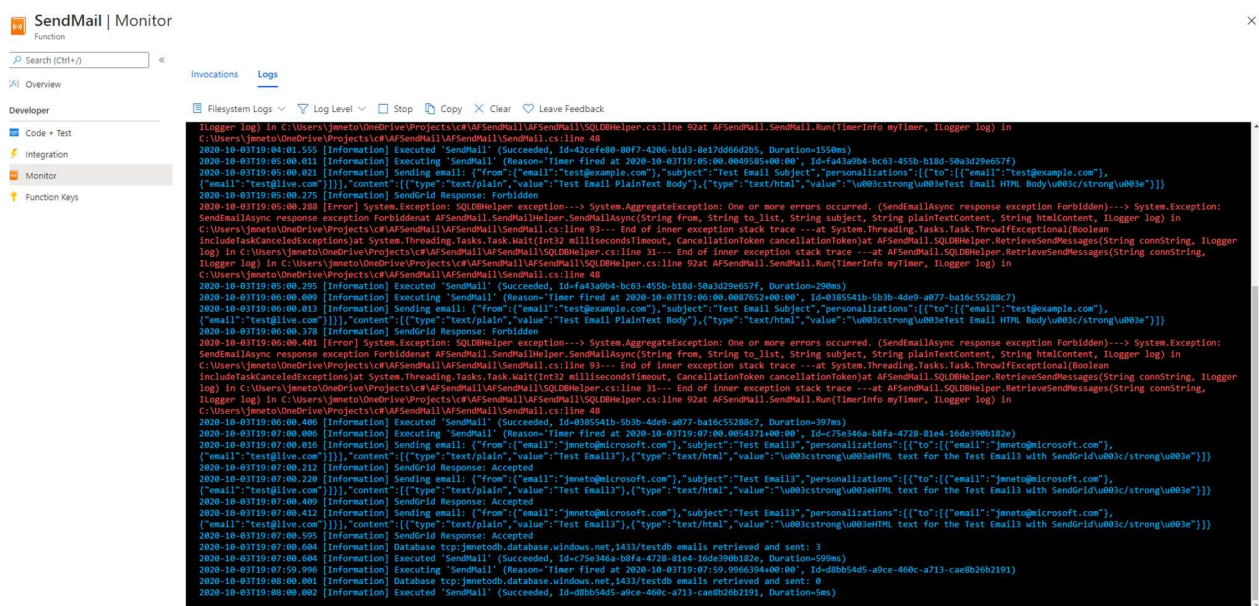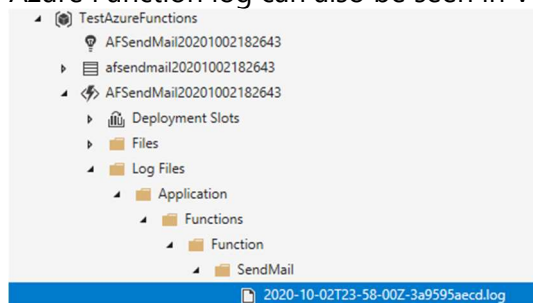
5. Monitoring

a. You can see the Azure Function console function to follow operation. (Application insights must be activated)

1.

b. Azure Function log can also be seen in Visual Studio Cloud Explorer

Feedback and suggestions

If you have feedback or suggestions for improving this data migration asset, please contact the Data SQL Ninja Engineering Team (datasqlninja@microsoft.com). Thanks for your support!

Note: For additional information about migrating various source databases to Azure, see the Azure Database Migration Guide.