Migrating an ASP.NET Core Web API Project to Azure Function

This blog will introduce the key migration steps and practices to assist you in migrating similar Web API projects.

Background of the project

This Web API project is part of my blog system and is used to send emails to administrators or readers. In the overall blog architecture, it is separate from the main blog site and runs on an independent service with Azure App Service. It makes use of a custom API Key for authentication. Passwords for email accounts are stored in Azure Key Vault. The backend has no database support. When the main blog site requires push notifications, it will send the payload of the notification to the API via REST request using the website's proprietary API Key to complete the email sending.

Why is Azure Function used?

Even though the Web API lacks event and queue functionality that is typically found in a serious notification system, my simple notification system has been running well and can meet business needs. The most serious issue, however, is the cost, which includes operation and maintenance costs as well as development costs.

First and foremost, the App Service Plan that hosts the Web API is a significant complexity. There is currently no way to select Consumption Plan for an ASP.NET Core application, and as such the API will be billed even when it is inactive. According to the company, the API is only used about ten times per day, with a total processing time of less than two minutes. However, I must pay for the remaining 1438 minutes every day. I must also free up computing resources so that Microsoft can assign them to those who require them in COVID-19, as well as reduce my carbon footprint to contribute to global environmental improvement.

Second, even a basic and straightforward Email notification feature needs a full framework to support it. Although ASP.NET Core offers a framework that is incredibly adaptable and robust, Azure Function may entirely free up these fundamental chores. Infrastructure is irrelevant to Azure Function; it only cares about business code. In other words, developers typically just need to write the function's business logic and not the code necessary to start, route, or assign API Keys.

Finally, Azure Function enables me to keep using my existing knowledge of the.NET technology stack. To create business logic, Azure Function supports.NET, Java, Python, Node.js, and even PowerShell. This indicates that my code can run on a new platform with very little modification.

Am I really stuck with Azure forever?

Is the question of whether Azure Functions will force your application to run exclusively on Azure the most crucial one for everyone? The response is no. Since the infrastructure of Azure Function is now open source and its apps can be containerized and deployed to any cloud, including those of rivals, such as China's Alibaba Cloud and other platforms, it has significantly aided rivals in building world-class serverless platforms. You can run the full design in a local data centre if you don't want to use the cloud. Therefore, there is no need to be concerned about falling victim to Microsoft's traps.

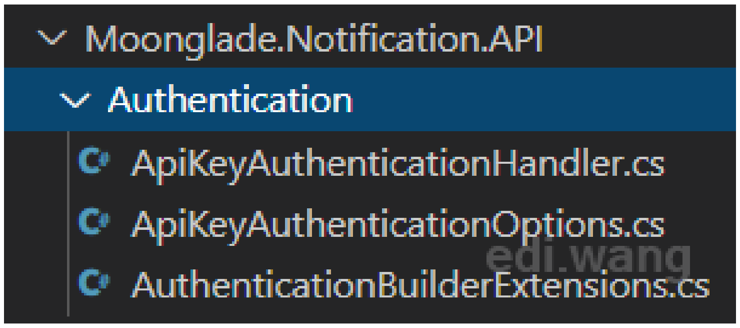
Migration procedures and essentials

The basics of using Azure Function are not covered in this tutorial.

Learn the fundamentals.

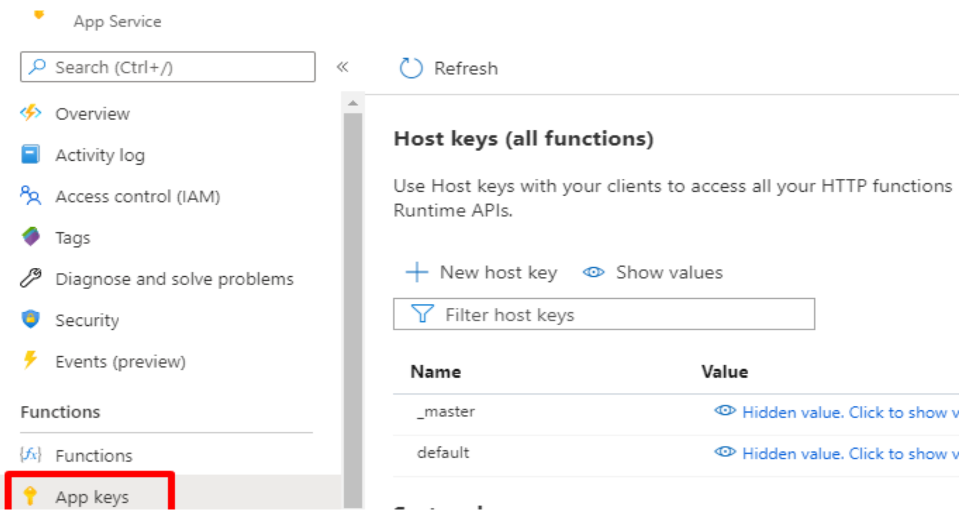
API Key verification

This is a component of the original project framework for the ASP.NET Core Web API. To prevent the API from being called without authorization, I utilized a special API key.



```
Moonglade.Notification.API
  Authentication
    ApiKeyAuthenticationHandler.cs
    ApiKeyAuthenticationOptions.cs
    AuthenticationBuilderExtensions.cs
```

Azure Function frees us from this section of code and offers a very similar App Key concept to manage authentication with only a few mouse clicks and no lines of code!



Host keys (all functions)

Use Host keys with your clients to access all your HTTP functions Runtime APIs.

Name	Value
_master	Hidden value. Click to show v
default	Hidden value. Click to show v

These App Keys can be sent with the query string to the function endpoint. I therefore erased the entire API Key code that I had laboriously written when porting the programme. There will be no more maintenance fees for this section of the code.

Azure Key Vault for reading

The first application made use of Microsoft.Azure.

To read data from Azure Key Vault, use the KeyVault package and the System Assigned Identity of the App Service. It is clear that this relationship with Azure is true.

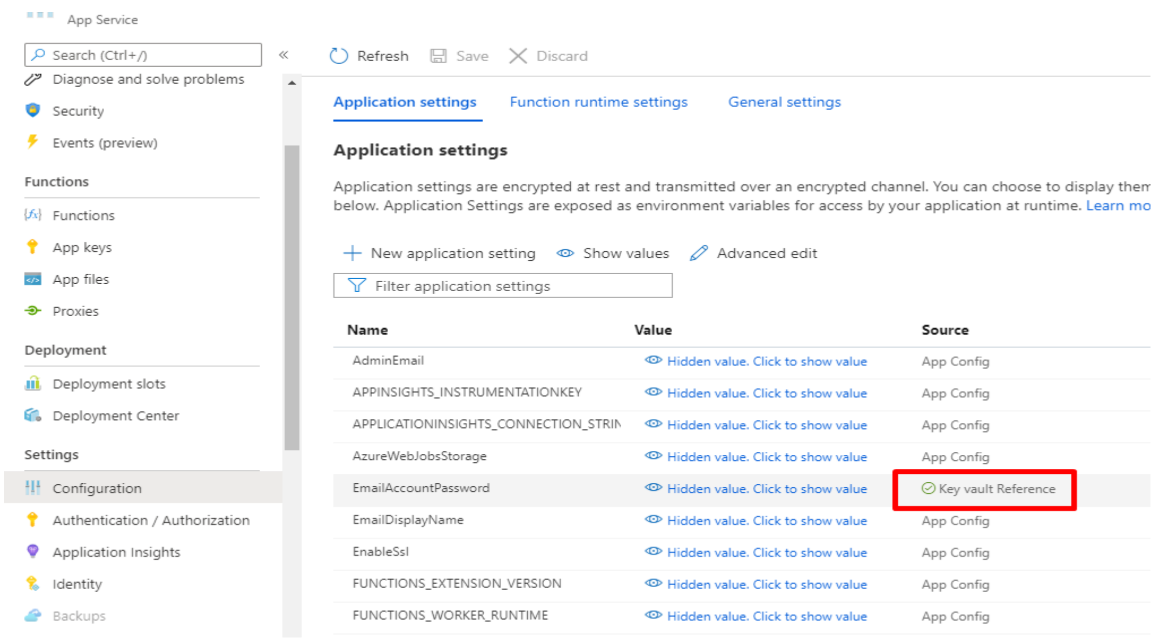
```
var keyVaultClient = new KeyVaultClient(new
KeyVaultClient.AuthenticationCallback(azureServiceTokenProvider.KeyVaultTo
kenCallback));

builder.AddAzureKeyVault(

    $"https://{builtConfig["AzureKeyVault:Name"]}.vault.azure.net/",
```

To avoid coupling the code with Azure and to enable me to quickly set up certain configuration items to read from Key Vault, I chose to use environment variables in Azure Function. The application still views Azure Key Vault values as environmental variables even if the method of reading them is transparent to the application.

Set the configuration items that need to read from the Key Vault to the format shown below on the Configuration page of Azure Function.

```
@Microsoft.KeyVault(SecretUri=<Azure Key Vault Secret Uri>)
```

My EmailAccountPassword, for instance, is an environment variable. When configuration is successful, Source will be shown as the key vault reference.



Application settings

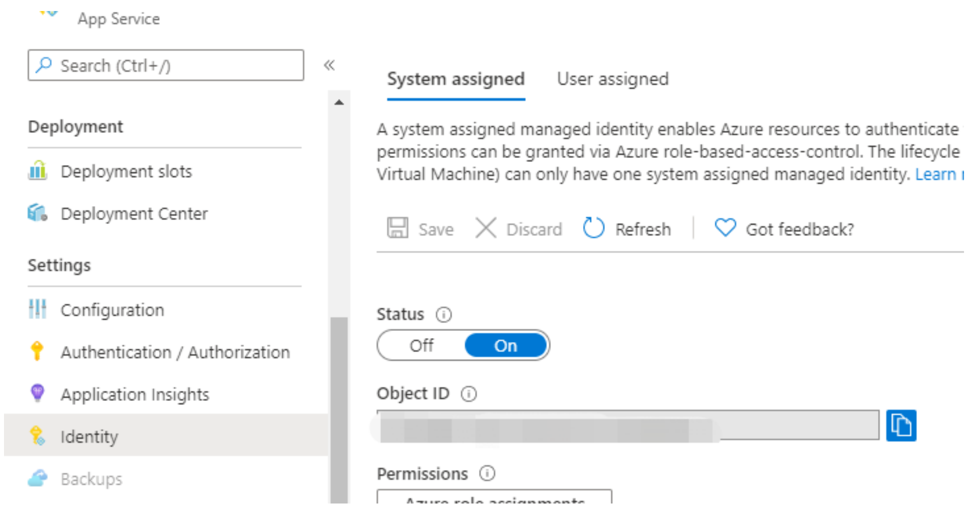
Application settings are encrypted at rest and transmitted over an encrypted channel. You can choose to display them below. Application Settings are exposed as environment variables for access by your application at runtime. Learn mo

Name	Value	Source
AdminEmail	Hidden value. Click to show value	App Config
APPINSIGHTS_INSTRUMENTATIONKEY	Hidden value. Click to show value	App Config
APPLICATIONINSIGHTS_CONNECTION_STRIN	Hidden value. Click to show value	App Config
AzureWebJobsStorage	Hidden value. Click to show value	App Config
EmailAccountPassword	Hidden value. Click to show value	✓ Key vault Reference
EmailDisplayName	Hidden value. Click to show value	App Config
EnableSsl	Hidden value. Click to show value	App Config
FUNCTIONS_EXTENSION_VERSION	Hidden value. Click to show value	App Config
FUNCTIONS_WORKER_RUNTIME	Hidden value. Click to show value	App Config

The application's code for accessing this environment variable is identical to that for reading the standard environment variable:

```
Environment.GetEnvironmentVariable("EmailAccountPassword",
```

Because Azure Key Vault is incredibly versatile, you may continue to maintain the deployment on the local development environment or other clouds independent. Your code is not required to use solely Azure by Microsoft.
Please be aware that your Function App must activate System Assigned Identity on its own.



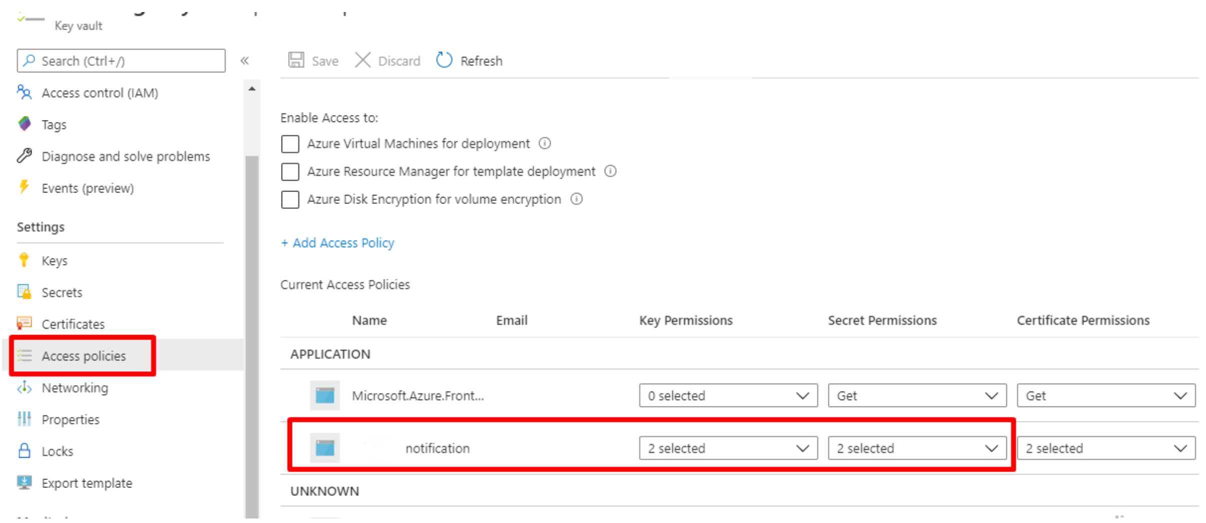
A system assigned managed identity enables Azure resources to authenticate permissions can be granted via Azure role-based-access-control. The lifecycle Virtual Machine) can only have one system assigned managed identity. Learn

Status: Off [✓ On]

Object ID: [redacted]

Permissions

Additionally, you must set up Get and List permissions in Azure Key Vault for the Function App.


Key vault

Search (Ctrl+/)

Save Discard Refresh

Enable Access to:

- [ ] Azure Virtual Machines for deployment
- [ ] Azure Resource Manager for template deployment
- [ ] Azure Disk Encryption for volume encryption

+ Add Access Policy

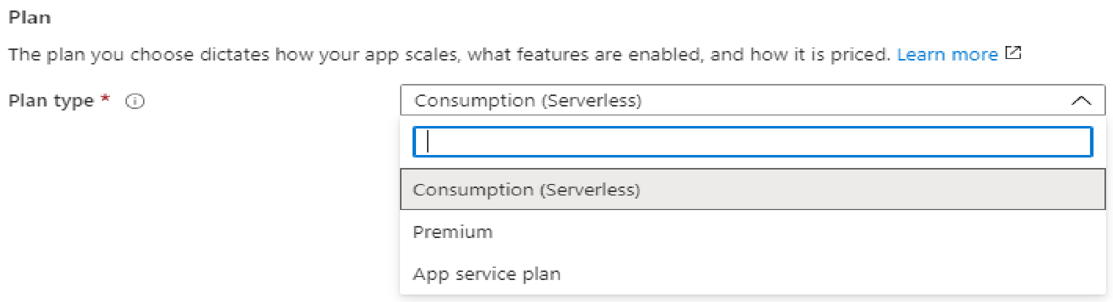
Current Access Policies

Name	Email	Key Permissions	Secret Permissions	Certificate Permissions
APPLICATION				
Microsoft.Azure.Front...		0 selected	Get	Get
notification		2 selected	2 selected	2 selected
UNKNOWN				

With just a few mouse clicks, all of this is possible.

Consumption Strategy

This is most likely the most exciting aspect of the entire Function. Consumption Plan is an option when creating an Azure Function. The Plan only charges for the function's execution time. The billing for my blog notification system is less than $5 per month, assuming that the API is only called a few times per day. The previous Standard S1 App Service Plan cost $69.35 per month, so the Consumption Plan saved me over $60.


Plan

The plan you choose dictates how your app scales, what features are enabled, and how it is priced. Learn more

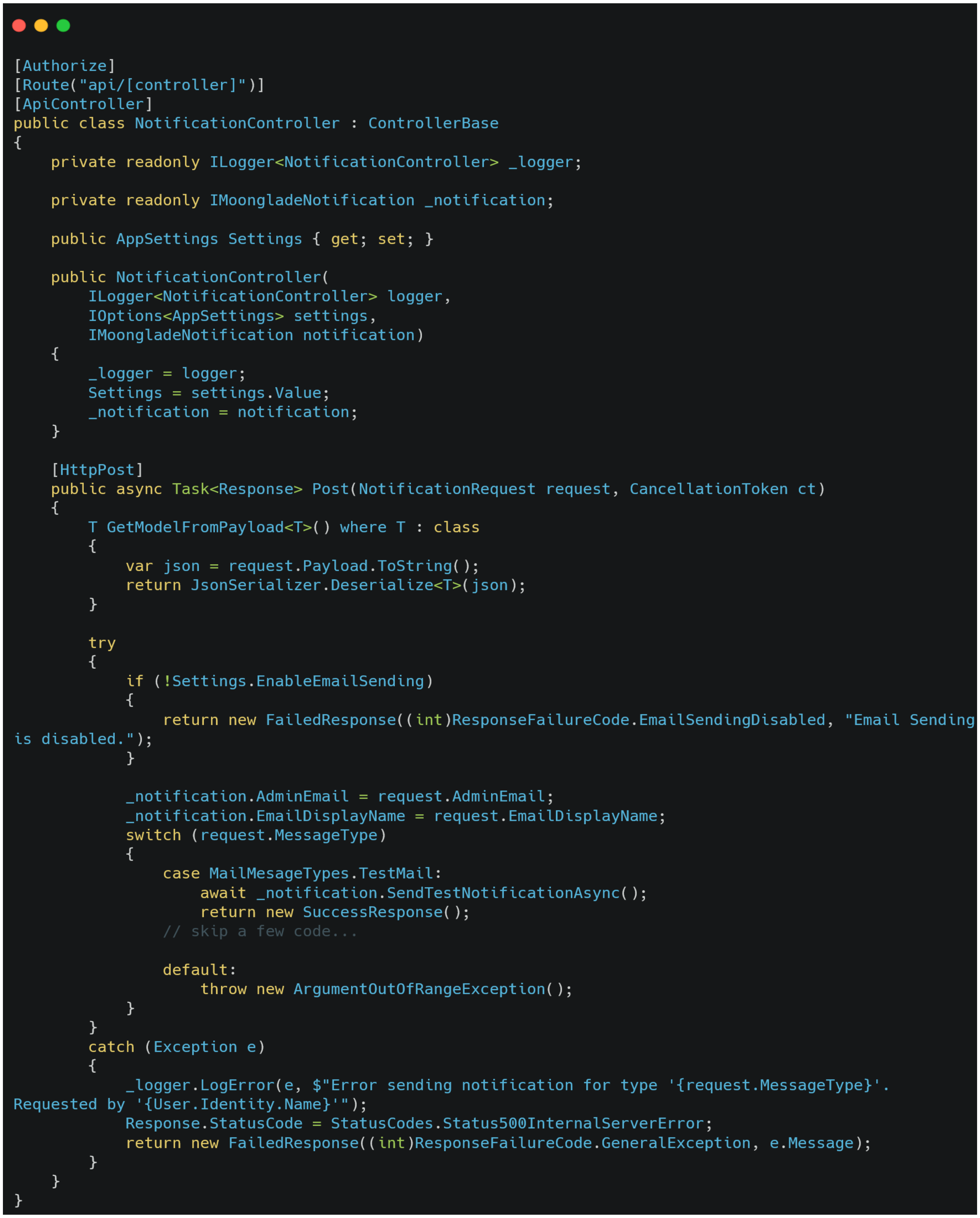
Plan type * : Consumption (Serverless)

The Code

What distinguishes the Function code from the original Web API code, which is the one thing that.NET programmers care about the most?

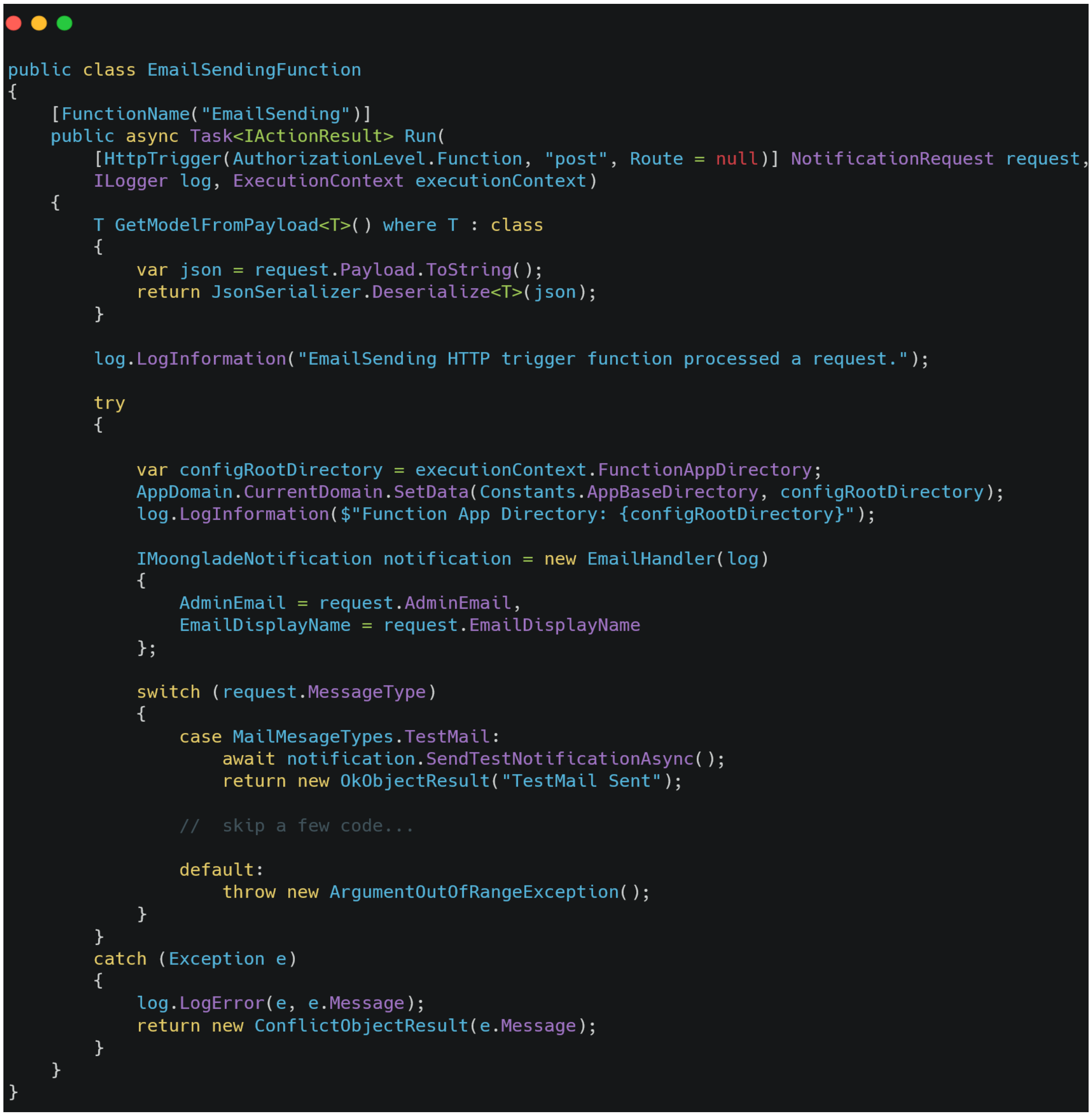
First of all, only your business code is still required; Program.cs, Startup.cs, Controller, and even appsettings.json are no longer required. Now that there is only one class in my application tier, the logic is remarkably identical to that of the original Web API Controller.

Original API Controller source code:



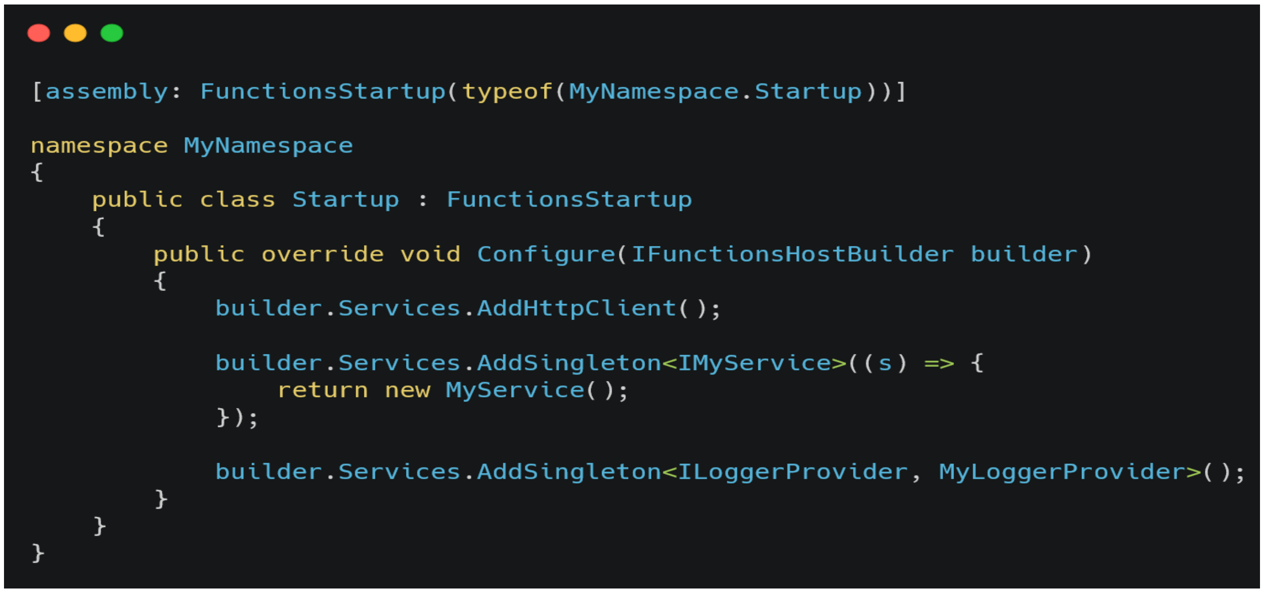
```
[Authorize]
[Route("api/[controller]")]
[ApiController]
public class NotificationController : ControllerBase
{
    private readonly ILogger<NotificationController> _logger;

    private readonly IMoongladeNotification _notification;

    public AppSettings Settings { get; set; }

    public NotificationController(
        ILogger<NotificationController> logger,
        IOptions<AppSettings> settings,
        IMoongladeNotification notification)
    {
        _logger = logger;
        Settings = settings.Value;
        _notification = notification;
    }

    [HttpPost]
    public async Task<Response> Post(NotificationRequest request, CancellationToken ct)
    {
        T GetModelFromPayload<T>() where T : class
        {
            var json = request.Payload.ToString();
            return JsonSerializer.Deserialize<T>(json);
        }

        try
        {
            if (!Settings.EnableEmailSending)
            {
                return new FailedResponse((int)ResponseFailureCode.EmailSendingDisabled, "Email Sending
is disabled.");
            }

            _notification.AdminEmail = request.AdminEmail;
            _notification.EmailDisplayName = request.EmailDisplayName;
            switch (request.MessageType)
            {
                case MailMesageTypes.TestMail:
                    await _notification.SendTestNotificationAsync();
                    return new SuccessResponse();
                // skip a few code...

                default:
                    throw new ArgumentOutOfRangeException();
            }
        }
        catch (Exception e)
        {
            _logger.LogError(e, $"Error sending notification for type '{request.MessageType}'.
Requested by '{User.Identity.Name}'");
            Response.StatusCode = StatusCodes.Status500InternalServerError;
            return new FailedResponse((int)ResponseFailureCode.GeneralException, e.Message);
        }
    }
}
```

The following is the Azure Function code:



```
public class EmailSendingFunction
{
    [FunctionName("EmailSending")]
    public async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "post", Route = null)] NotificationRequest request,
        ILogger log, ExecutionContext executionContext)
    {
        T GetModelFromPayload<T>() where T : class
        {
            var json = request.Payload.ToString();
            return JsonSerializer.Deserialize<T>(json);
        }

        log.LogInformation("EmailSending HTTP trigger function processed a request.");

        try
        {

            var configRootDirectory = executionContext.FunctionAppDirectory;
            AppDomain.CurrentDomain.SetData(Constants.AppBaseDirectory, configRootDirectory);
            log.LogInformation($"Function App Directory: {configRootDirectory}");

            IMoongladeNotification notification = new EmailHandler(log)
            {
                AdminEmail = request.AdminEmail,
                EmailDisplayName = request.EmailDisplayName
            };

            switch (request.MessageType)
            {
                case MailMesageTypes.TestMail:
                    await notification.SendTestNotificationAsync();
                    return new OkObjectResult("TestMail Sent");

                //  skip a few code...

                default:
                    throw new ArgumentOutOfRangeException();
            }
        }
        catch (Exception e)
        {
            log.LogError(e, e.Message);
            return new ConflictObjectResult(e.Message);
        }
    }
}
```

And if you still need to customize the hosting framework, such as by using DI, you can do so:



```csharp
[assembly: FunctionsStartup(typeof(MyNamespace.Startup))]

namespace MyNamespace
{
    public class Startup : FunctionsStartup
    {
        public override void Configure(IFunctionsHostBuilder builder)
        {
            builder.Services.AddHttpClient();

            builder.Services.AddSingleton<IMyService>((s) => {
                return new MyService();
            });

            builder.Services.AddSingleton<ILoggerProvider, MyLoggerProvider>();
        }
    }
}
```

Conclusion

A straightforward Web API with little business demand might be moved to Azure Function to dramatically reduce development and maintenance costs. We may concentrate on the business logic itself rather than the infrastructure code by employing a serverless platform. We can keep using the programming language that we are accustomed to while still keeping some flexibility and security.

Thank you