

# Demo- Azure Functions & .NET Core

8 minutes

## Value Props

The **three things developers** should take away from this demo:

**Key Message – Build, debug and deploy high performance serverless apps across the globe all with the productivity and power of Visual Studio, .NET and Azure**

### 1. Azure Functions – integrate powerful serverless functions in your apps

- Azure Functions is a serverless compute experience that scales on demand and you only pay for the resources you consume. Never worry about infrastructure, just your code.
- Using Azure Functions, .NET developers can add functionality in small, serverless chunks and you only pay for the resources you consume when the function is running. They can be written and debugged locally in the environment you're familiar with - Visual Studio.

### 2. Use the power of Visual Studio 2017 to build, debug, deploy and manage the lifecycle of your functions.

- Visual Studio 2017 fully supports developing Azure Functions in C# as well as local debugging and diagnostics. Use the highly productive IDE you are used to for building serverless functions in the cloud.

### 3. .NET Core and Azure App Service give developers unrivaled efficiency and production runtime visibility.

- .NET Core is our cross-platform and open source runtime that gives us great performance and is perfect for cloud-native apps. App Service is perfect for websites build with ASP.NET Core. It provides a fully managed platform where we can scale and monitor our apps effortlessly.

## Install Pre-Requisites

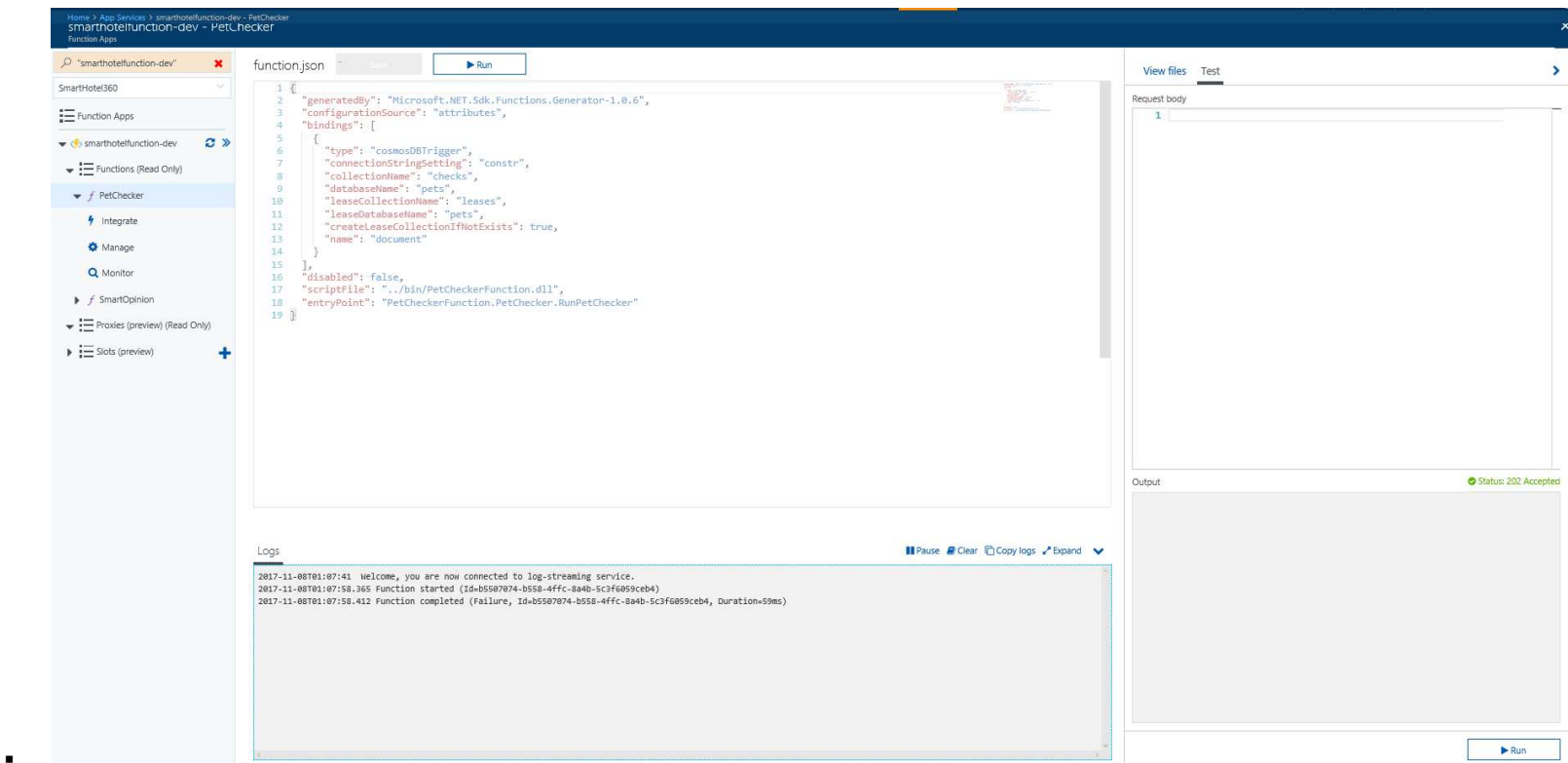
- **Visual Studio 2017 15.5 or higher – Azure & .NET Core workloads selected**
- **Clone repo:** <https://github.com/Microsoft/SmartHotel360-public-web>
- Install Node 8.9.1
- Install Python 2

From an admin prompt, navigate to SmartHotel360.PublicWeb:

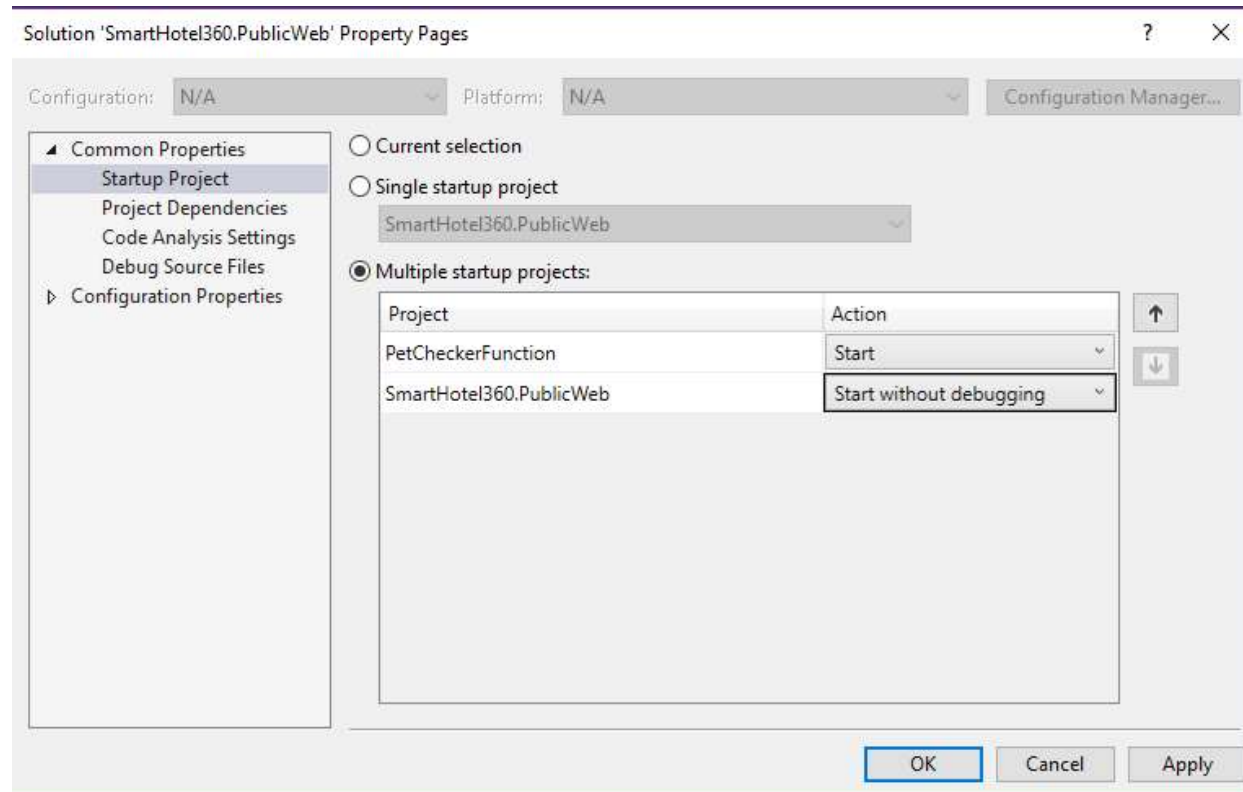
- `npm install -g windows-build-tools`
- `Npm install`
- `Npm rebuild node-sass -force`
- `Npm run dev`

## Demo Pre-Setup

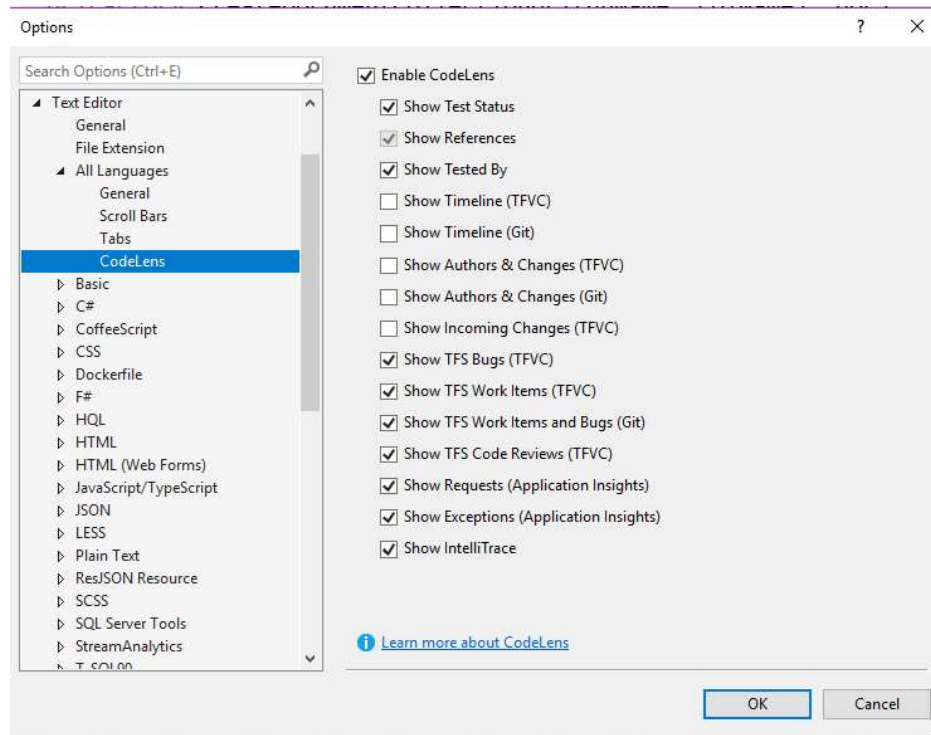
- Follow the instructions for setting up the Azure services related to this demo: <https://github.com/Microsoft/SmartHotel360-public-web/blob/master/doc/pet-demo.md>
- Open browser tabs to:
  - o PublicWeb deployed to Azure app service, either your own or the publicly available one (i.e. mysmarthotel360app.azurewebsites.net or <http://smarthotel360public.azurewebsites.net/>)
  - o Open your Azure portal to the PetChecker Azure Function. Make sure the Logs section is showing and expand test section.



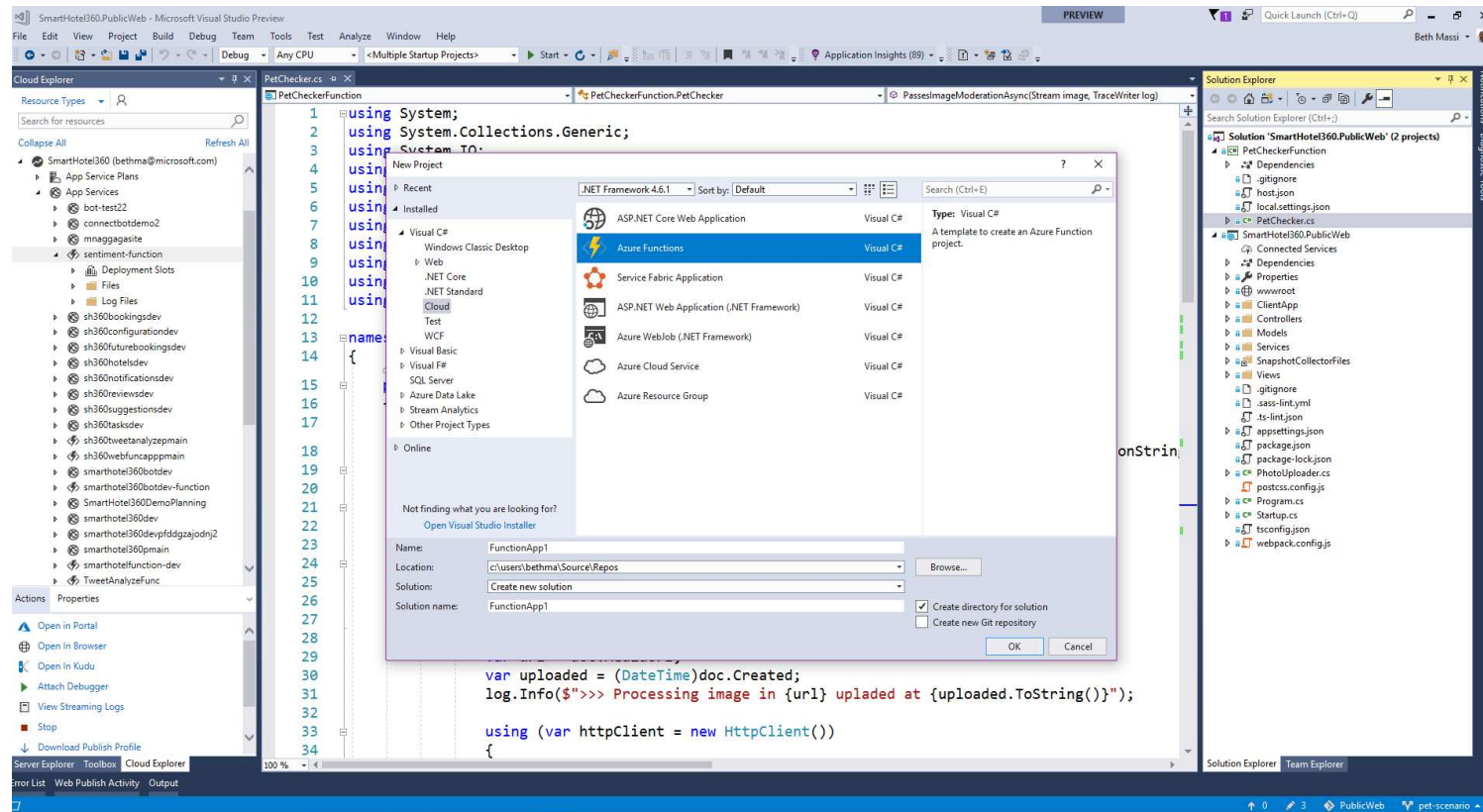
- Warm up the AzureFunction by hitting the Test -> run
- Open an instance of Visual Studio to SmartHotel360.PublicWeb.sln. This contains the PublicWeb and the PetCheckerFunction project.
  - o Right-click on the solution and verify Multiple Startup projects are set to start up on debug. Set website to start without debugging.



- 
- Set Code Lens options



- Set Fonts & Colors (For Text Editor, Data Tips, Watch windows) to 11pt
- Open the Solution Explorer and open the PetChecker.cs
- Open the Cloud Explorer to the PetCheckerFunction.
- Open the New project dialog and highlight the new Azure Function template.

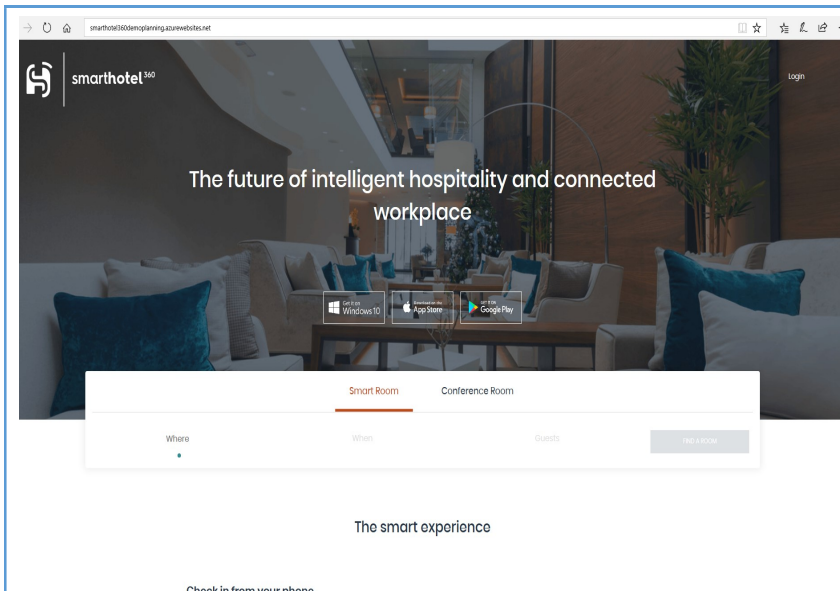


- Grab a couple pictures of animals. You'll need one large dog photo (to test) and one small animal like a pet pig or small cat.

## Overall Demo walk-through

Story – Debug and integrate Azure function into a public ASP.NET Core site.

Screenshot	Actions	Talk track
------------	---------	------------



Open the public website

Quickly walk through features

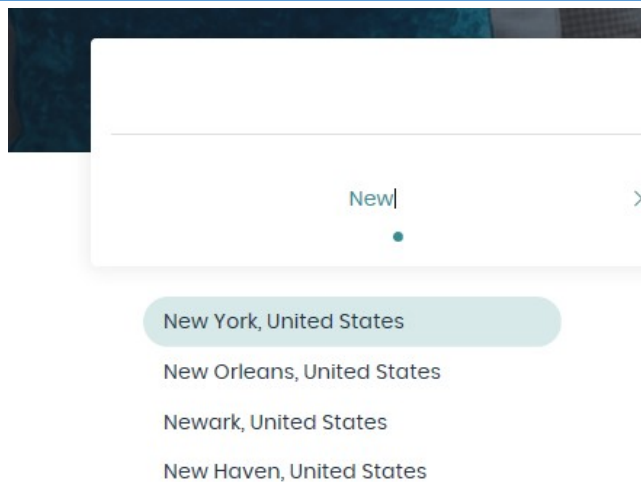
Search for a city: New York

Click "Find a room"

Show a room detail

Let's take a look at the SmartHotel360 public website. This website was **built in C#** and runs on **.NET Core** which is our **open source, cross-platform** runtime that gives us great performance and is **perfect for cloud-native apps**.

This website is deployed to **Azure App Service** which is **perfect for websites** like this. It provides a fully managed platform where we can **scale and monitor our apps effortlessly**.



Walk through booking a room.

As you can see this is where people get their first impression of our Smart Hotel and make reservations.

November 2017

Su

Mo

Tu

We

Th

Fr

Sa

Su

Mo

Tu

We

29

30

31

1

2

3

4

26

27

28

29

5

6

7

8

9

10

11

3

4

5

6

12

13

14

15

16

17

18

10

11

12

13

19

20

21

22

23

24

25

17

18

19

20

26

27

28

29

30

1

2

24

25

26

27

31

1


2

3

Reset

Apply

Select a date range and click Apply



Rooms

-

1

+

Rooms

Are you bringing a pet?

No

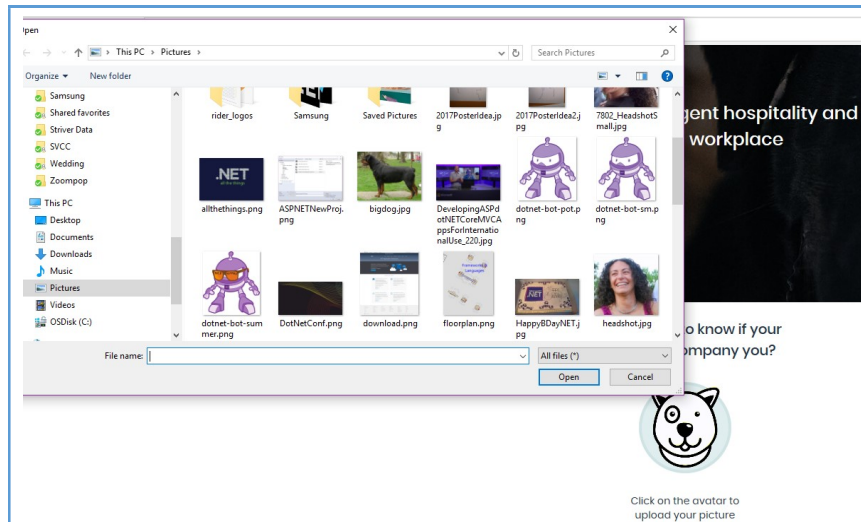
Yes

Check it

Click "Are you bringing a pet?"

Click "Check it"

Guests can also indicate whether they are bringing a pet.



Click the Dog image and select a picture of a small pig.

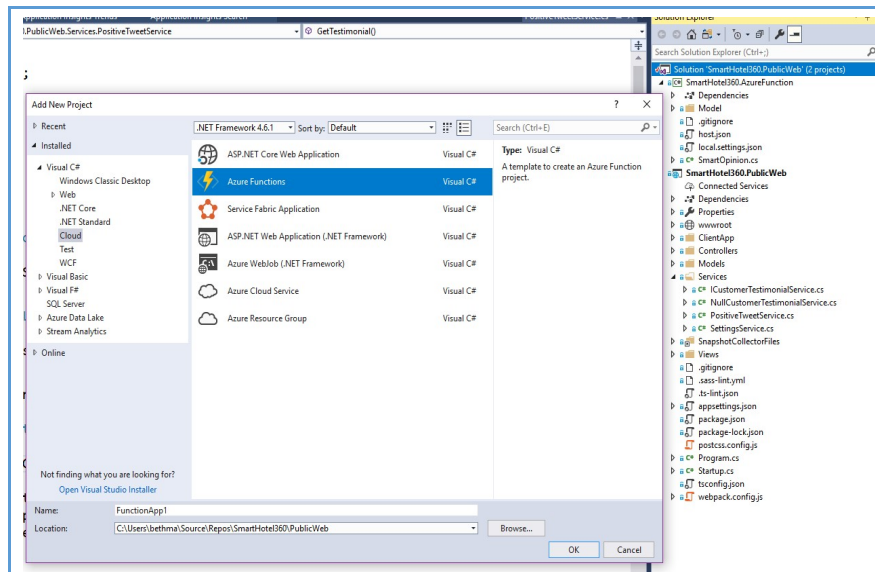
Here we request the guest upload a picture of their animal. We're only allowing dogs at the hotel.

Let's try this photo of my pet pig and see what happens.

While you wait, tell the audience what's happening....

We analyze the picture to make sure it's a dog using Cognitive Services Vision API. This is implemented using Azure Functions.





Flip Visual Studio.

Point out new template.

Click Cancel

**Azure Functions is a is an event-based serverless compute experience to accelerate your development. Azure Functions can scale on demand and you pay only for the resources you consume.**

But what's really cool is that Visual Studio supports building, debugging, and managing Azure Functions. This gives you all the benefits of Visual Studio including intellisense, debugging and source control. This is needed particularly as our functions get more complex.

We used Visual Studio to build this one. Notice the new Azure Function App template. A function app lets you group functions as a logical unit for easier management, deployment, and sharing of resources.

```

1 using System;
2 using System.Collections.Generic;
3 using System.IO;
4 using System.Linq;
5 using System.Net.Http;
6 using System.Threading.Tasks;
7 using Microsoft.Azure.Documents;
8 using Microsoft.Azure.Documents.Client;
9 using Microsoft.Azure.WebJobs;
10 using Microsoft.Azure.WebJobs.Host;
11 using Microsoft.ProjectOxford.Vision;
12
13 namespace PetCheckerFunction
14 {
15     public static class PetChecker
16     {
17         [FunctionName("PetChecker")]
18         public static async Task RunPetChecker([CosmosDBTrigger("pets", "checks", ConnectionStri
19             TraceWriter log)
20         {
21             foreach (dynamic doc in document)
22             {
23                 var isProcessed = doc.IsApproved != null;
24                 if (isProcessed)
25                 {
26                     continue;
27                 }
28
29                 var url = doc.MediaUrl;
30                 var uploaded = (DateTime)doc.Created;
31                 log.Info($">>> Processing image in {url} uploaded at {uploaded.ToString()}");
32
33                 using (var httpClient = new HttpClient())

```

Show PetChecker.cs Code

Highlight CosmosDBTrigger (line 18)

Here's the code for our function.  
Notice we're using Azure Cosmos DB here.

**Cosmos DB is Microsoft's globally distributed, multi-model database that can elastically and independently scale across any number of Azure's geographic regions.**

From the website, we're uploading the picture to Cosmos DB which triggers the function to perform the analysis and stores the result.

```

65 public static async Task<(bool, string)> PassesImageModerationAsync(Stream image, TraceWri
66 {
67     log.Info("--- Creating VisionApi client and analyzing image");
68     var key = Environment.GetEnvironmentVariable("MicrosoftVisionApiKey");
69     var endpoint = Environment.GetEnvironmentVariable("MicrosoftVisionApiEndpoint");
70     var client = new VisionServiceClient(key, endpoint);
71     var features = new VisualFeature[] { VisualFeature.Description };
72     var result = await client.AnalyzeImageAsync(image, features);
73     log.Info($"--- Image analyzed with tags: {String.Join(", ", result.Description.Tags)}");
74     if (int.TryParse(Environment.GetEnvironmentVariable("MicrosoftVisionNumTags"), out var
75     }

```

Highlight VisionAPI code (line 70)

Here's where we are calling the Cognitive Services VisionAPI. **Cognitive Services provide intelligent algorithms to see, hear, speak, understand and interpret your user needs through natural methods of communication. The VisionAPI provides image-processing algorithms to smartly identify, caption, and moderate pictures.**

Once the service analyzes our picture, it returns a set of tags that it sees and a message describing our picture.

```

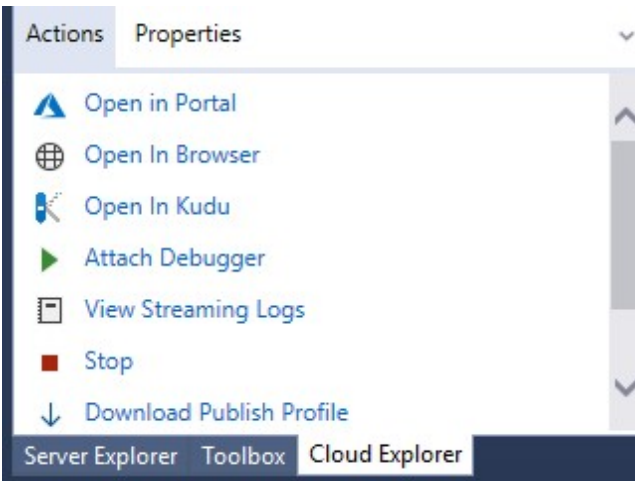
76         tagstorecn = >;
77     }
78     bool isAllowed = result.Description.Tags.Take(tagsToFetch).Contains("small");
79     string message = result?.Description?.Captions.FirstOrDefault()?.Text;
80     return (isAllowed, message);
81 }
82

```

Add code to check for small animals:

Line 78: change "dog" to "small"

Here we're looking for the tag "dog". However, we're finding that our hotel isn't really set up to accommodate large animals. I also don't think it's fair to not allow my pet pig. So instead of approving only dogs, we'll modify this code to determine the size of the animal instead.



Point out the "Attach Debugger" in Cloud Explorer (don't click it).

Now we need to debug this to make sure it works.

Notice we can debug our functions in production using the cloud explorer.

```

78     bool isAllowed = result.Description.
79     string message = result?.Description
80     return (isAllowed, message);
81 }
82

```

Set breakpoint on line 80

Hit F5

However, I want to locally debug this change I made before deploying it. Let's set a breakpoint here.



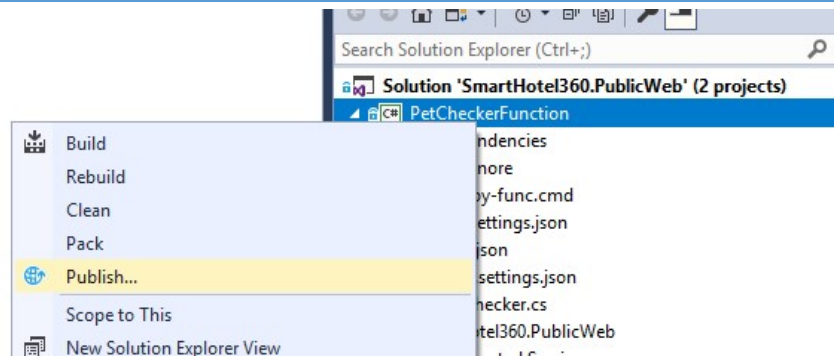
```

1 reference | 0 exceptions
public static async Task<(bool, string)> Pas
{
    log.Info("--- Creating VisionApi client");
    var key = Environment.GetEnvironmentVari
    var endpoint = Environment.GetEnvironmen
    var client = new VisionServiceClient(key
    var features = new VisualFeature[] { Vis
    var result = await client.AnalyzeImageAs
    log.Info($"--- Image analyzed with tags:
    if (!int.TryParse(Environment.GetEnviron
    {
        tagsToFetch = 5;
    }
    bool isAllowed = result.Description.Tags
    string message = result?.Description?.C
    return (isAllowed, message);
}

```

Inspect the Tags by hovering over  
Description.Tags on line 78

Notice the tags that come back  
that describe our picture. The  
VisionAPI provides us rich  
information about our photo.

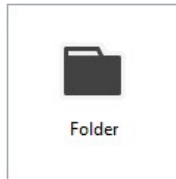
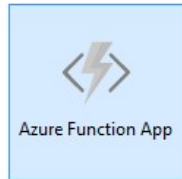


Right click on function  
Select Publish

I can also easily publish this back  
into production right here.

## Publish

Publish your app to Azure or another host. [Learn more](#)



- ☒ Create New  
☐ Select Existing



Don't do anything just say this ->

**Azure functions and Visual Studio make building, debugging and managing serverless microservices super simple and easy. No other cloud gives you the power and productivity like this!**

Thank you!