

Azure Functions

Lecture 14

Deep Azure@McKesson

Zoran B. Djordjević

Need for serverless computing

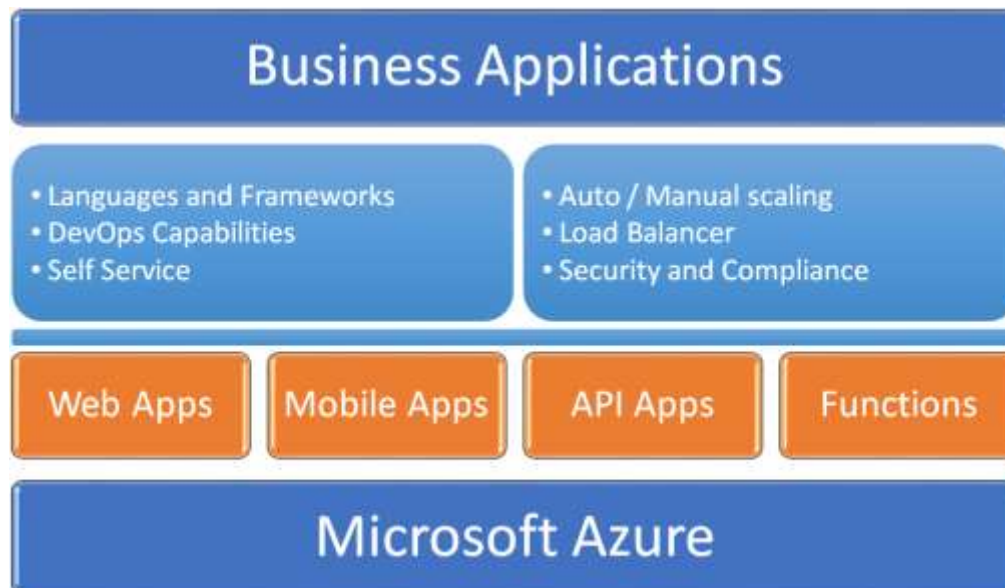
- Cloud has eliminated many if not most of system and database administration tasks and replaced them with highly automated procedures. It is only natural that one looks to an environment that is completely free from any administration.
- Azure Functions and AWS Lambda Functions offer such environments.
- Serverless is not actually serverless. It means that users only need to manage code/application and not servers. The server will be managed by the service provider. We as a user only pay when our code or function is executed in the serverless or in the server that is not managed by us.
- Scaling is based on the request and pricing differs based on the service provider. AWS Lambda and Azure Functions are two examples of serverless computing or **Function as a Service (FaaS)**.
- AWS provides a pay-as-you-go billing model, while Microsoft Azure provides a consumption plan as well as an App Service plan for Azure Functions.

Benefits

- The following are some of the benefits of serverless computing:
 - Faster time to market as you can write code in the functions editor in the Azure portal and click on **Run** for execution
 - No need to worry about the infrastructure and provisioning resources
 - Easy bindings to services and external services
 - Create functions in multiple languages as supported by the cloud service provider
 - Pay only for what you use
 - More cost-effective than IaaS and PaaS
 - No configuration is required to set up scaling in and scaling out policies

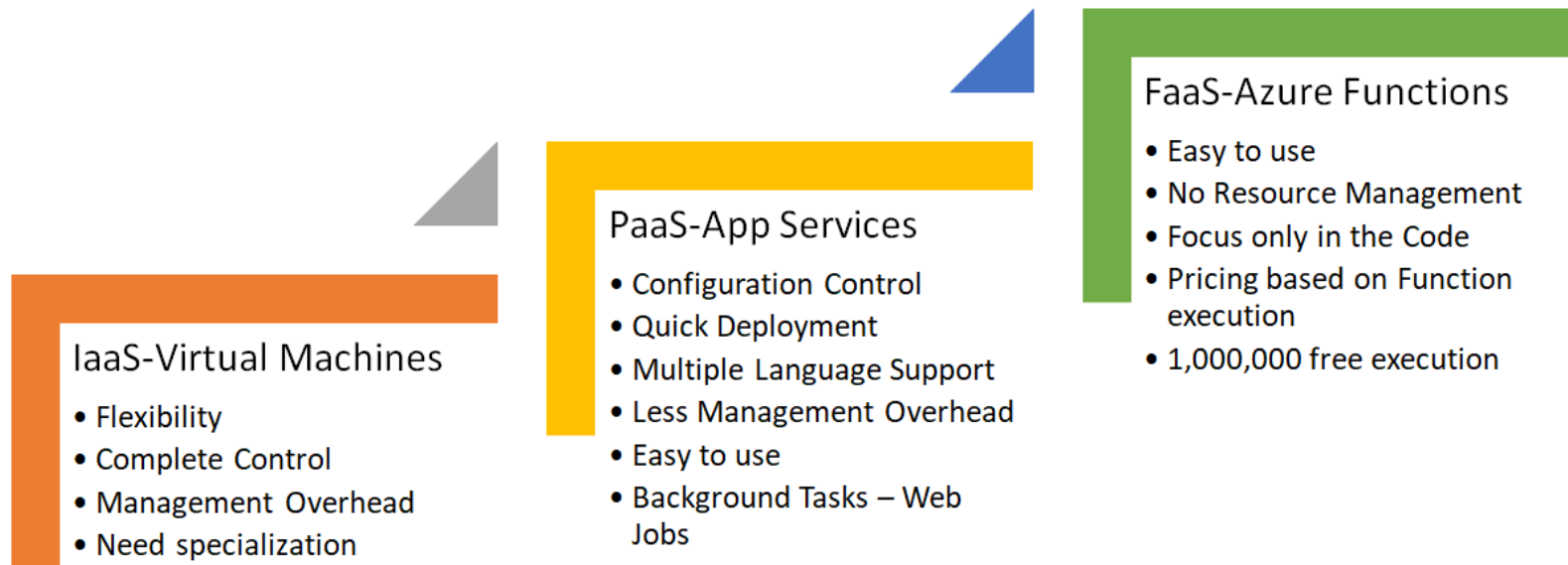
Azure Function

- Azure Function is a service running a function in a cloud environment.
- We use Azure Functions in scenarios where we know the problem, we know the code that can fix the problem, and we don't want to worry about resources that execute this code.
- Azure Function is the easiest way to focus on logic and business and enhance the our productivity.
- We only need to pay for what we use. There are no maintenance of system support costs. There are development costs, though.



Comparison to Other Types of Service

- Azure Functions are similar to Azure Web Jobs with some differences such as scaling policies, trigger events, and language support. We don't need to worry about infrastructure for execution of the piece of code or function.
- We can execute Azure functions in response to events as well.
- The languages that are supported are C#, F#, Node.js, Python or PHP, batch, bash, or PowerShell.



Pricing

There are two types of pricing plans available in the Azure Functions:

- **Consumption plan:** When we execute functions, Microsoft Azure provides all the resources. We only pay for the time that our functions are executed. The consumption plan pricing includes a monthly free grant of 1 million requests and 400,000 GBs of resource consumption per month. Free grants apply to paid consumption subscriptions only.
- **App Service plan:** This executes functions just the way we execute Azure App Services. We can utilize the same App Service plan created for any application and execute Azure Functions on it without any extra cost.
- Having App Service plan as the host for Azure Functions provides lots of benefits that are available with Azure App Services. We can utilize remote debugging, deployment slots, continuous deployment, vertical scaling and horizontal scaling, auto-scaling, and so on.
- If we use the Azure App Service, then it is a multitenant scenario. If we want to utilize a dedicated environment, then we can utilize the App Service Environment that is a dedicated service from Microsoft Azure, where we can host a function in a virtual network and configure **network security groups (NSGs)** for an enhanced level of security.

Triggers & Bindings

- Triggers and bindings are the core of Azure Functions.
- Triggers allow us to write a function to respond to events that occur in the Azure or other services. As the name suggests, trigger indicates how the function should be invoked. AzureFunctions can have only one trigger associated with it
- Bindings specify how is a particular function related to data. Bindings are the connection to the data from within the code available in the function. Unlike triggers, functions can have one or more input and output bindings.
- The table on the following slide shows which triggers and bindings that are supported with Azure Functions.
- All triggers have associated input data.
- The HTTP output binding requires an HTTP trigger.

Triggers & Bindings

Type	Service	Trigger	Input	Output
Schedule	Azure Function	Yes		
HTTP	Azure Functions	Yes		Yes**
Blob Storage	Azure Storage	Yes	Yes	Yes
Events	Azure Event Hubs	Yes		Yes
Queues	Azure Storage	Yes		Yes
Queues and Topics	Azure Service Bus	Yes		Yes
Storage tables	Azure Storage		Yes	Yes
SQL tables	Azure Mobile Apps		Yes	Yes
No-SQL DB	Azure DocumentDB		Yes	Yes
Push Notifications	Azure Notification Hubs			Yes
Twilio SMS Text	Twilio			Yes
SendGrid email	SendGrid			Yes

App Service Plan

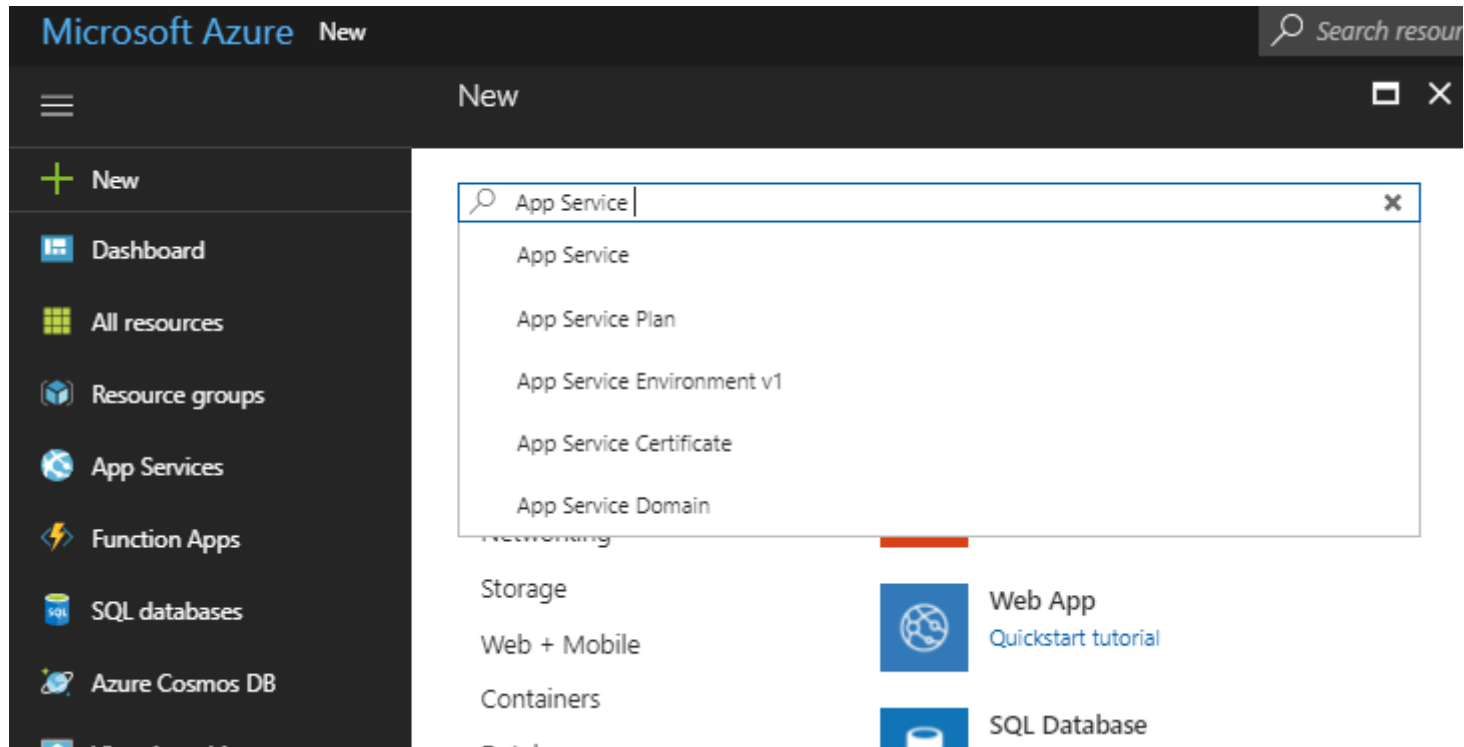
- An **App Service plan (ASP)** is a combination of capacities (instance size and instance count on which the application is hosted) and features.
- Capacity is directly related to cost and hence it is similar to choosing a pricing tier. There are different capabilities and limits available in pricing plans.
- Each ASP can be used for different purposes and they provide different features too.
- There are five pricing tiers as follows:
 - **Free:** no scaling
 - **Shared:** no scaling
 - **Basic:** SLA - 99.95%; maximum instances for scaling - 3
 - **Standard:** SLA - 99.95%; autoscale, 5 deployment slots; Geo-distributed deployment, VPN hybrid connectivity, deployment slots, and automated backups; maximum instances for scaling - 10
 - **Premium:** SLA - 99.95%; 20 deployment slots; autoscale, geo-distributed deployment, VPN hybrid connectivity, deployment slots, and automated backups; maximum instances for scaling - 50

Features of App Service Plan

- An App Service plan can be shared by multiple applications.
- Deployment slots are usually deployed on the same App Service plan.
- Azure Web Apps configured with an App Service plan are changed, and then these changes affect all the applications hosted on the App Service plan.
- By default, ASP comes with a single instance. If we increase the instance count, then the applications hosted on a single instance will be hosted on other instances too.
- The number of instances in ASP is directly associated with the price of Azure Web Apps

Create App Service Plan

- To create an App service plan, open Azure portal, hit + New and type App Service plan in the search field of New pane that pops-up.



Select App Service Plan

- On the next (Everything) pane that shows up, select App Service Plan

The screenshot shows the Azure Marketplace interface. On the left is a sidebar with navigation links: New, Dashboard, All resources, Resource groups, App Services, Function Apps, SQL databases, and Azure Cosmos DB. The main area is titled 'Marketplace' and 'Everything'. A search bar contains 'App Service Plan'. Below the search bar, a list of results is shown under the heading 'NAME'. The results are: App Service Plan (highlighted), App Service Environment, and Soha Cloud - VPN Plan. To the right of the search results, a detailed view for 'App Service Plan' by Microsoft is displayed. This view includes a description: 'App Service plans give you the flexibility to allocate specific apps to further optimize your Azure resource utilization. This way, if you want an environment you can share a plan across multiple apps.' Below the description are social media icons for Twitter, Facebook, LinkedIn, YouTube, Google+, and Email. Further down, the 'PUBLISHER' is listed as 'Microsoft' and 'USEFUL LINKS' include 'Documentation', 'Service Overview', and 'Pricing details'. At the bottom of this panel is a blue 'Create' button.

NAME
App Service Plan
App Service Environment
Soha Cloud - VPN Plan

App Service Plan
Microsoft

App Service plans give you the flexibility to allocate specific apps to further optimize your Azure resource utilization. This way, if you want an environment you can share a plan across multiple apps.

[Twitter](#) [Facebook](#) [LinkedIn](#) [YouTube](#) [Google+](#) [Email](#)

PUBLISHER: Microsoft

USEFUL LINKS: [Documentation](#), [Service Overview](#), [Pricing details](#)

[Create](#)

- On the following pane hit [Create](#)

Parameters of App Service Plan

- Populate details, name, resource group, location, Pricing tier. Hit [Select](#) and [Create](#)

- After a short while, App service Plan will be deployed and will be visible in my resource group.

New App Service Plan

Create a plan for the web app

* App Service plan

zjdjrdjeappserviceplan

* Subscription

Pay-As-You-Go

* Resource Group

Create new Use existing

zjdjrdjergp

* Operating System

Windows

* Location

Central US

* Pricing tier

S1 Standard

☐ Pin to dashboard

Create

Automation options

Choose your pricing tier

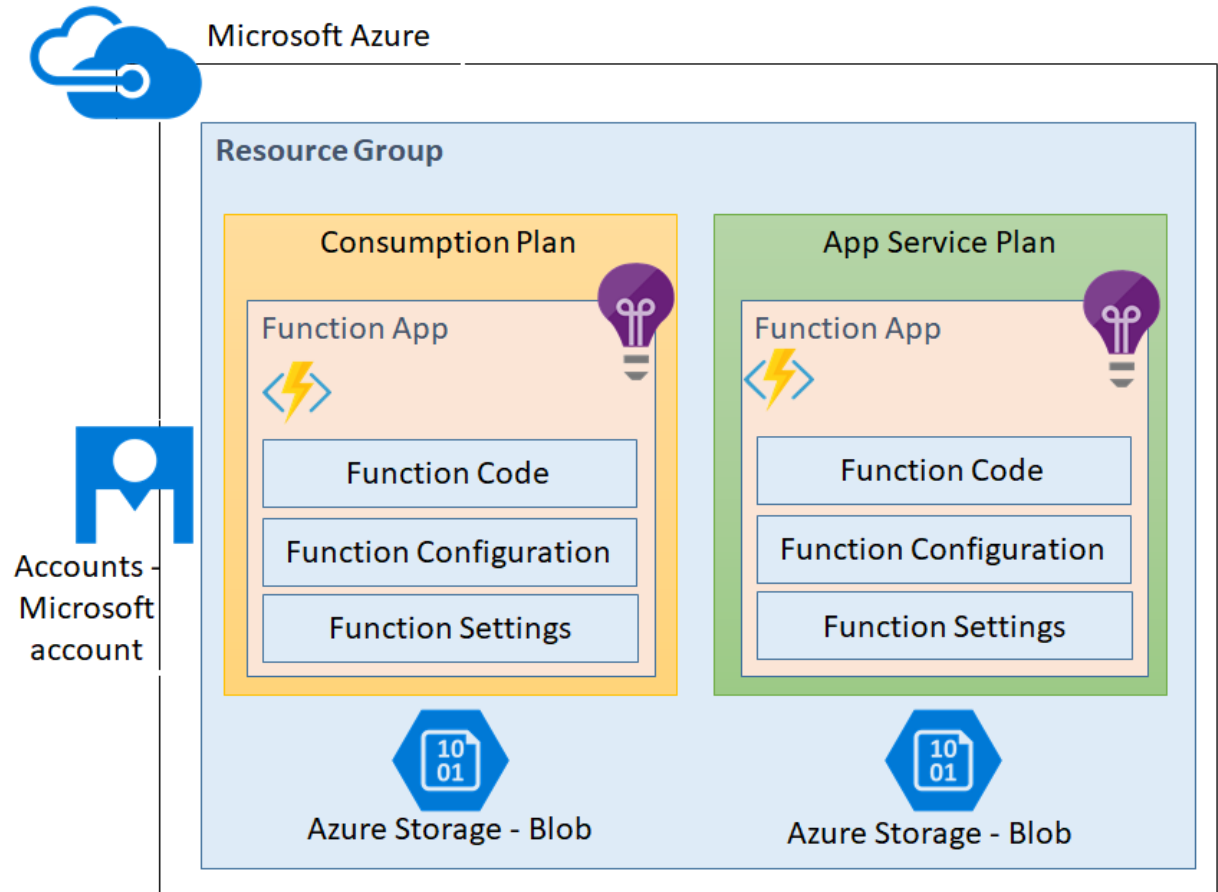
Browse the available plans and their features

1	Core	2	Core
1.75	GB RAM	3.5	GB RAM
	10 GB Storage		10 GB Storage
	Custom domains		Custom domains
	SSL Support SNI SSL Included		SSL Support SNI SSL Included
	Up to 3 instance(s) Manual scale		Up to 3 instance(s) Manual scale
55.80 USD/MONTH (ESTIMATED)		111.60 USD/MONTH (ESTIMATED)	
F1 Free		D1 Shared*	
- Shared infrastructure		- Shared infrastructure	
	1 GB Storage		1 GB Storage
0.00 USD/MONTH (ESTIMATED)		9.67 USD/MONTH (ESTIMATED, *PER APP)	

Select

Create an Azure Function

- We will create an Azure Function App and then we will create an Azure Function with triggers and output bindings.
- One of the most important things in Azure App Services or Azure Functions is the Kudu editor which we could use to troubleshoot any issues in our functions
- We will go through in detail about the Kudu editor, folder structure, and files available in the Azure Functions.
- Azure function can exist in two environments: App Service Plan and Consumption Plan.



Azure Function App, Function Code

- A function app is a collection of one or more functions that are managed together. All the functions in a Function App share the same pricing plan and it can be a consumption plan or an App Service plan.
- When we utilize Visual Studio Team Services for Continuous Integration and Continuous Delivery using build and release definitions, then the Function app is also shared.
- The way we manage different resources in Azure with the Azure Resource Group is similar to how we can manage multiple functions with the Function App.
- In the following, we will consider a scenario where photographers need to upload photographs to the portal. The moment a photograph is uploaded, a thumbnail should be created immediately.
- As an illustration we will use a JavaScript code, presented on the following slide.

Function Code

```
var Jimp = require("jimp");
// JavaScript function must export a single function via module.exports // To find the function and execute it
module.exports = (context, myBlob) => {
  // context is a must have parameter and first parameter always, // context is used to pass data to and from the function
  //context name is not fixed; it can be anything
  // Read Photograph with Jimp
  Jimp.read(myBlob).then((image) => {
    // Manipulate Photograph, resize the Photograph. Jimp.AUTO can be passed as one of the values.
    image
      .resize(200, 200).quality(40).getBuffer(Jimp.MIME_JPEG, (error, stream) => {
        // Check for errors while processing the Photograph.
        if (error) {
          // To print the message on log console
          context.log('There was an error processing the Photograph.');
```

```
          // To communicate with the runtime that function is finished to avoid timeout
          context.done(error);
        }
        else {
          // To print the message on log console
          context.log('Successfully processed the Photograph');
```

```
          // To communicate with the runtime that function is finished to avoid timeout
          // Bind the stream to the output binding to create a new blob
          context.done(null, stream);
        }
      });
  });
};
```


Function Configuration

- Function configuration defines the function bindings and other configuration settings. It contains configurations such as the type of trigger, paths for blob containers, and so on:

```
{
  "bindings": [
    {
      "name": "myBlob",
      "type": "blobTrigger",
      "direction": "in",
      "path": "photographs/{name}",
      "connection": "origphotography2018_STORAGE",
      "dataType": "binary"
    },
    {
      "type": "blob",
      "name": "$return",
      "path": "thumbnails/{name}",
      "connection": "origphotography2018_STORAGE",
      "direction": "out"
    }
  ],
  "disabled": false
}
```

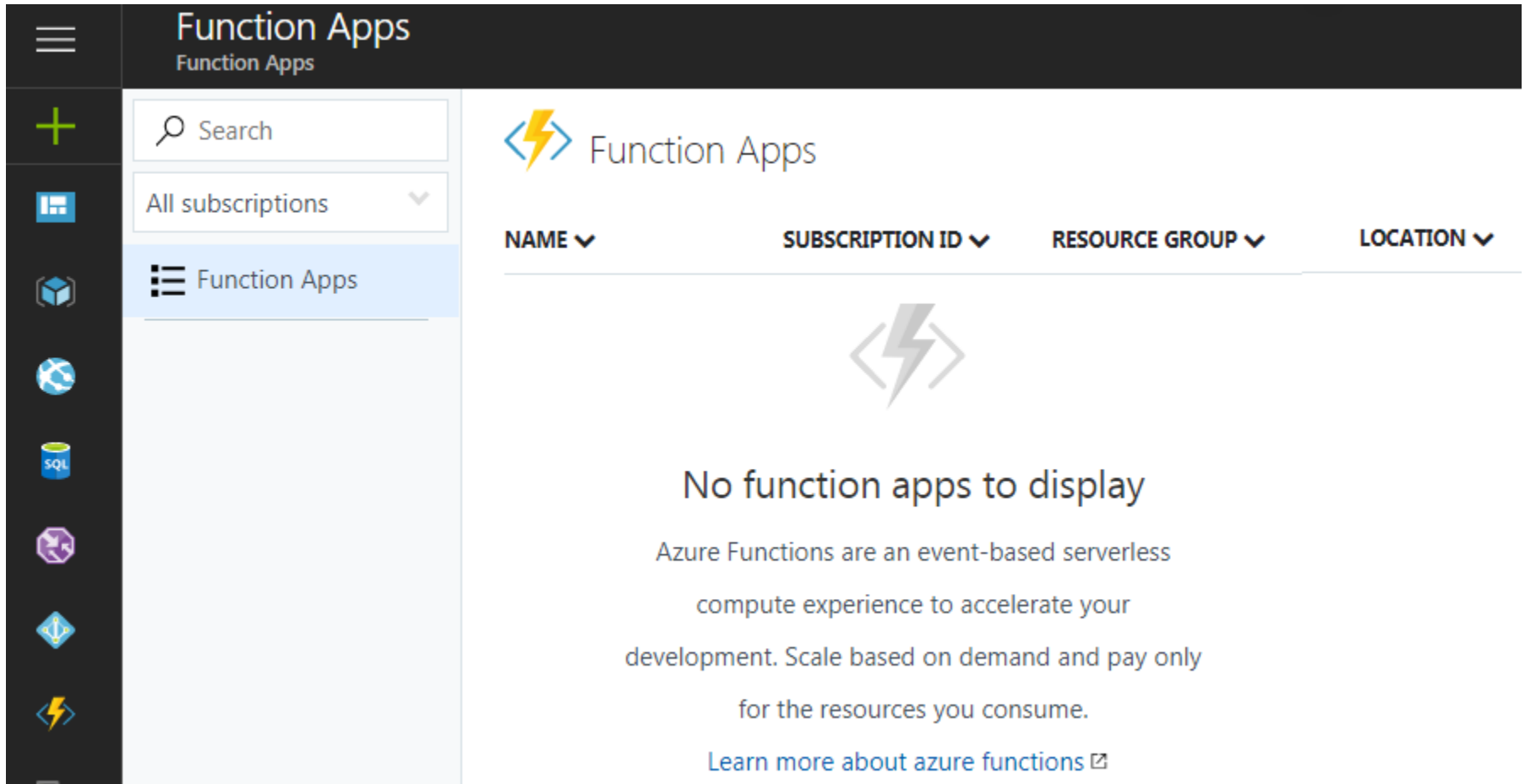
- The function runtime uses this configuration file to decide which events to monitor and how to pass data to and from the function execution.

Function Settings & Runtime

- We can limit the daily usage quota and application settings. We can enable Azure Function proxies and change the edit mode of our function app.
- The application settings in the Function App are similar to the application settings in Azure App Services.
- We can configure .NET Framework v4.6, Java version, Platform, ARR Affinity, remote debugging, remote Visual Studio version, app settings, and connection strings.
- The runtime is responsible for executing function code on the underlying WebJobs SDK host.

Setting up an Azure Function

- In Azure portal we select + New, Function Apps. Initially we have no function apps.



The screenshot shows the Azure portal interface for 'Function Apps'. On the left, a sidebar contains a search bar, a subscription dropdown set to 'All subscriptions', and a list of resource types with 'Function Apps' selected. The main content area has a header with the 'Function Apps' icon and title. Below this is a table with columns: NAME, SUBSCRIPTION ID, RESOURCE GROUP, and LOCATION. The table is empty, and a large message states 'No function apps to display'. The message explains that Azure Functions are an event-based serverless compute experience and provides a link to learn more.

NAME	SUBSCRIPTION ID	RESOURCE GROUP	LOCATION
No function apps to display			

Azure Functions are an event-based serverless compute experience to accelerate your development. Scale based on demand and pay only for the resources you consume.

[Learn more about azure functions](#)

Create Function App

- On Function App pane, select [Create](#)

The screenshot shows the Microsoft Azure portal's 'New' page. The top navigation bar includes a hamburger menu, the word 'New', window controls, and a selected tab for 'Function App' with the Microsoft logo. On the left, a vertical sidebar lists various service categories: Compute, Networking, Storage, Web + Mobile, Databases, Data + Analytics, AI + Cognitive Services, and Internet of Things. A search bar at the top of this sidebar contains the text 'Function App'. Below the search bar, the 'MARKETPLACE' section is visible with a 'See all' link. The main content area on the right features a descriptive paragraph about creating functions, a row of social media icons (Twitter, Facebook, LinkedIn, YouTube, Google+, and Email), and a section titled 'PUBLISHER' showing 'Microsoft'. Below this, 'USEFUL LINKS' are listed: 'Documentation', 'Solution Overview', and 'Pricing Details'. At the bottom of the main content area, there is a prominent blue 'Create' button.

Provide App name, other details

- Provide the **App name**, **Subscription** details, and existing **Resource Group**.
- Select **Consumption Plan** or **App Service Plan** in **Hosting Plan**. Then select **Location**:

Function App
Create

* App name
AZFnTest ✓
.azurewebsites.net

* Subscription
Visual Studio Enterprise with MSDN ▼

* Resource Group ⓘ
☐ Create new ☒ Use existing
AzureFunctions ▼

* Hosting Plan ⓘ
Consumption Plan ▼

* Location
South Central US ▼

☐ Pin to dashboard
Create Automation options

App Service plan
Select a plan for the web app

An App Service plan is the container for your app. The App Service plan settings will determine the location, features, cost and compute resources associated with your app.

+ Create New

ServicePlane98ac6b5-bef8(S1) (New)
South Central US New Plan

Create New Storage

- Select **Create New** in **Storage** and click on **Create**:

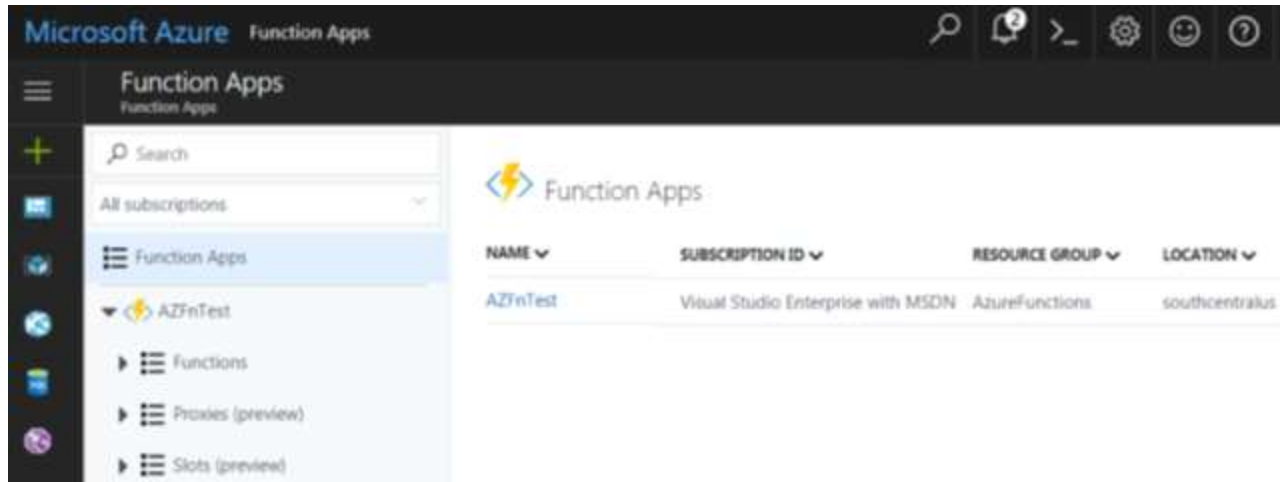
The screenshot displays the 'Function App Create' wizard in the Azure portal. The left-hand navigation pane includes icons for various services, with the 'Storage' icon (a purple cube) currently selected. The main content area is titled 'Function App Create' and contains the following configuration steps:

- Resource Group**: A dropdown menu showing 'AzureFunctions'.
- Hosting Plan**: A dropdown menu showing 'Consumption Plan'.
- Location**: A dropdown menu showing 'South Central US'.
- Storage**: Radio buttons for 'Create New' (selected) and 'Select Existing'. Below this, a text box contains the name 'azfntest9cd0'.
- Application Insights**: A toggle switch currently set to 'Off'.

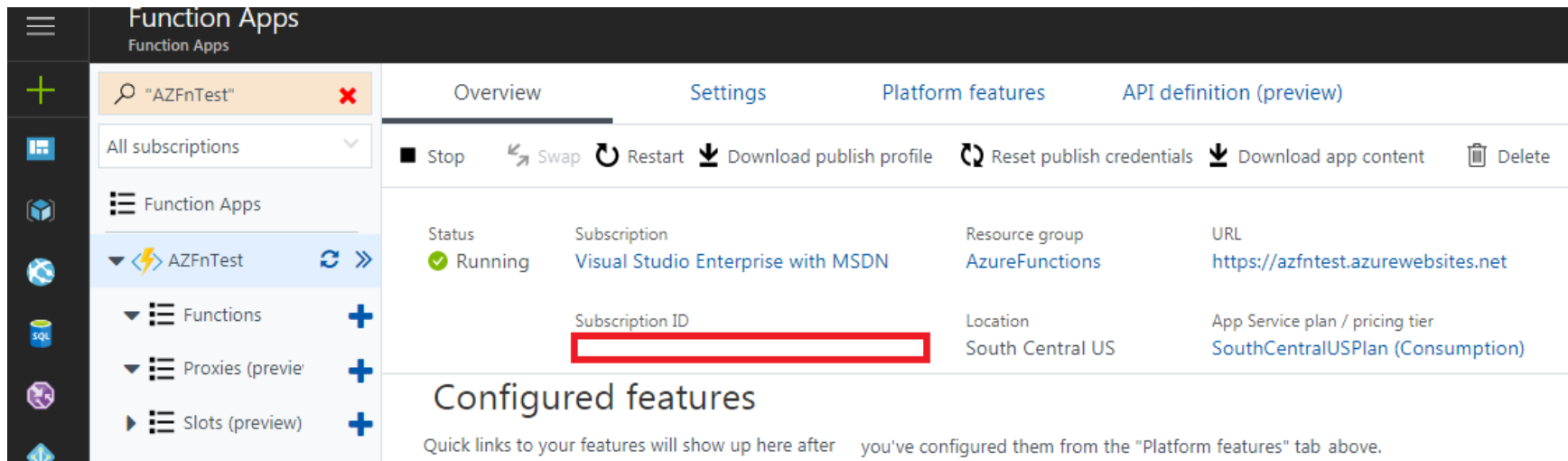
At the bottom of the wizard, there is a checkbox for 'Pin to dashboard' and a prominent blue 'Create' button. To the right of the 'Create' button is a link labeled 'Automation options'.

Verify Function App Exist

- Now, let's go to **Function Apps** in the left sidebar and verify whether the recently created Function App is available in the list or not:



- Click on the Function App and we see the details: Subscription, Resource group, URL, Location, App Service Plan / pricing tier. We can Stop/Start function app



Settings

- The **Settings** tab provides details on the **Runtime version**, **Application settings**, and the limit on daily usage:

The screenshot shows the Azure portal interface for a Function App named 'AZFnTest'. The left sidebar contains a search bar with 'AZFnTest' and a list of subscriptions. Below this, the 'Function Apps' section is expanded, showing 'AZFnTest' with a refresh icon and a double arrow icon. Under 'AZFnTest', there are three items: 'Functions', 'Proxies (preview)', and 'Slots (preview)', each with a plus icon. The main content area has four tabs: 'Overview', 'Settings' (which is selected), 'Platform features', and 'API definition (preview)'. The 'Settings' tab contains several sections: 'Daily Usage Quota (GB-Sec)' with a value of '1' and a 'Set quota' button; 'Application settings' with a link to 'Manage application settings'; 'Runtime version' showing '1.0.11002.0 (~1)'; 'Proxies (preview)' with a toggle switch set to 'Off'; 'Function app edit mode' with a toggle switch set to 'Read\Write'; and 'Slots (preview)' with a toggle switch set to 'Off'. A detailed description for 'Slots (preview)' explains that it is a one-time opt-in that cannot be disabled and will reset pre-existing secrets, with a note to 'Manage' the node for each function.

Function Apps
Function Apps

Search "AZFnTest" ✕

All subscriptions ▼

Function Apps

▼ ⚡ AZFnTest ↻ »

▼ ☰ Functions +

▼ ☰ Proxies (preview) +

▶ ☰ Slots (preview) +

Overview Settings Platform features API definition (preview)

Daily Usage Quota (GB-Sec) ⓘ

1 Set quota

Application settings
Manage application settings

Runtime version
Runtime version: 1.0.11002.0 (~1)

Proxies (preview)
Enable Azure Functions Proxies (preview)

Off On

Function app edit mode
Change the edit mode of your function app

Read\Write Read Only

Slots (preview)
Enable deployment slots (preview). This is a one-time opt-in on the Function app that cannot be disabled and will reset any pre-existing secrets. After the update, the secrets may be copied from under the **'Manage' node for each function**.

Off On

Settings

- Settings also allows us to keep the Function App in **Read/Write** or **Read Only** mode. We can also enable deployment slots, a well-known feature of Azure App Services:

The screenshot shows the 'Function Apps' settings page in the Azure Portal. The left sidebar contains a search bar with 'AZFnTest' and a list of subscriptions. Below this, the 'Function Apps' section is expanded, showing 'AZFnTest' with a refresh icon and a right arrow. Under 'AZFnTest', there are links for 'Functions', 'Proxies (preview)', and 'Slots (preview)', each with a plus icon. The main content area displays the following settings:

- Runtime version:** Runtime version: 1.0.
- Proxies (preview):** Enable Azure Functions Proxies (preview). A toggle switch is currently set to 'Off'.
- Function app edit mode:** Change the edit mode of your function app. A toggle switch is currently set to 'Read\Write'.
- Slots (preview):** Enable deployment slots (preview). This is a one-time opt-in on the Function app that cannot be disabled and will reset any pre-existing secrets. After the update, the secrets may be copied from under the 'Manage' node for each function. A toggle switch is currently set to 'Off'.
- Host Keys (All functions):** A table listing host keys with columns for NAME, VALUE, and ACTIONS.

NAME	VALUE	ACTIONS
_master	Click to show	Copy Renew
default	Click to show	Copy Renew Revoke

Below the table is a button labeled 'Add new host key'.

Platform Features Tab

- In the **Platform features** tab as shown below, we get different kinds of options to enable the Function App with **MONITORING, NETWORKING, DEPLOYMENT TOOLS**, and so on.

The screenshot displays the Azure Portal interface for a Function App named 'AZFnTest'. The left-hand navigation pane shows the 'Function Apps' section expanded, with 'AZFnTest' selected. The main content area is divided into four tabs: 'Overview', 'Settings', 'Platform features' (which is the active tab), and 'API definition (preview)'. The 'Platform features' tab contains a search bar and several categorized feature groups:

- GENERAL SETTINGS**: Application settings, Properties, Backups, All settings.
- CODE DEPLOYMENT**: Deployment options, Deployment credentials.
- DEVELOPMENT TOOLS**: Console, Advanced tools (Kudu).
- NETWORKING**: Networking, SSL, Custom domains, Authentication / Authorization, Push notifications.
- MONITORING**: Diagnostic logs, Log streaming, Process explorer, Security scanning.
- API**: CORS, API definition.
- APP SERVICE PLAN**: App Service plan, Quotas.
- RESOURCE MANAGEMENT**: Activity log, Access control (IAM), Tags, Locks.

Properties

- Click on **Properties**. Verify the different details that are available:

The screenshot shows the Microsoft Azure portal interface for a Function App named 'azfntest'. The 'Properties' tab is selected, displaying various configuration options. The left sidebar contains a search bar and a list of settings categories: GENERAL SETTINGS, CODE DEPLOYMENT, and DEVELOPMENT TOOLS. The main content area is divided into three columns: GENERAL SETTINGS, NETWORKING, and API. The right-hand pane shows the 'Properties' for the function app, including STATUS (Running), URL (azfntest.azurewebsites.net), VIRTUAL IP ADDRESS (No IP-based SSL binding is configured), MODE (Consumption), OUTBOUND IP ADDRESSES (a redacted IP address), DEPLOYMENT TRIGGER URL (https://\$AZFnTest:2PnT0ibKu7YtrQi1gc...), and FTP/DEPLOYMENT USER.

Category	Property	Value
STATUS	Running	
URL	azfntest.azurewebsites.net	
VIRTUAL IP ADDRESS	No IP-based SSL binding is configured	
MODE	Consumption	
OUTBOUND IP ADDRESSES	[Redacted] .89.24,104.215.89...	
DEPLOYMENT TRIGGER URL	https://\$AZFnTest:2PnT0ibKu7YtrQi1gc...	
FTP/DEPLOYMENT USER		

- There is a property named **OUTBOUND IP ADDRESSES** which is useful if we need the IP addresses of the Function App for whitelisting.

App Service Plan

- Click on **App Service plan** and it will open a consumption plan in the pane:

The screenshot displays the Azure portal interface for an App Service plan named 'SouthCentralUSPlan'. The left-hand navigation pane includes a search bar and a list of options: Overview (selected), Activity log, Access control (IAM), Tags, Diagnose and solve problems, SETTINGS (Apps, Properties, Locks, Automation script), and OBSERVE. The main content area is divided into two sections. The top section, titled 'Delete', contains a 'Delete' button and a list of 'Essentials' details: Resource group (change) 'AzureFunctions', Status 'Ready', Location 'South Central US', Subscription name (change) 'Visual Studio Enterprise with MSDN', and Subscription ID (highlighted with a red box). The bottom section, titled 'Monitoring', shows 'CPU Percentage and Memory Percentage' with an 'Edit' button.

SouthCentralUSPlan App Service plan	
<input type="text" value="Search (Ctrl+/)"/>	
Overview	
Activity log	
Access control (IAM)	
Tags	
Diagnose and solve problems	
SETTINGS	
Apps	
Properties	
Locks	
Automation script	
OBSERVE	
Delete	
Essentials	
Resource group (change)	Pricing Tier
AzureFunctions	Consumption
Status	Apps / Slots
Ready	1 / 0
Location	
South Central US	
Subscription name (change)	
Visual Studio Enterprise with MSDN	
Subscription ID	
Monitoring	
CPU Percentage and Memory Percentage	
Edit	

Storage

- On the left sidebar in the Azure Portal, go to **Storage** services and verify the storage accounts that are available:

Storage accounts

+ Add Columns Refresh

Storage accounts and Storage accounts (classic) can now be managed together in the combined list below.

Subscriptions: Visual Studio Enterprise with MSDN – Don't see a subscription? [Switch directories](#)

Filter by name... All types All locations N

1 items

NAME	TYPE	KIND	RESOURCE GR...	LOCATION
azfntest9cd0	Storage account	Storage	AzureFunctions	South Central US

- What we want to achieve is that when we upload an image in a specific blob container, the function should be available immediately in the Function App and should be executed and create a thumbnail in another blob container.

Create 2nd Storage Account

Storage accounts (Default Directory)

Add

Columns

Refresh

i

Storage accounts and Storage accounts (classic) can now be managed together in the combined list below.

Subscriptions: Visual Studio Enterprise with MSDN – Don't see a subscription? [Switch directories](#)

Filter by name...

1 items

NAME

azfntest9cd0

Create storage account

usage and the options you choose below.
[Learn more](#)

*

Name

origphotography2018

✓

.core.windows.net

Deployment model

Resource manager

Classic

Account kind

General purpose

Performance

Standard

Premium

Replication

Read-access geo-redundant storage (RA...

☐

Pin to dashboard

Create

Automation options

@Nishava, Inc - Zoran B. Djordjević

30

Create Container for Photos

- We need containers in both storage accounts. Click on **+ Container** and fill in the **Name** and **Access type** and click on **OK**:

Search (Ctrl+/)

BLOB SERVICE

- Containers
- CORS
- Custom domain
- Encryption
- Azure CDN
- Add Azure Search
- Metrics
- Usage

FILE SERVICE

+ Container Refresh

New container

* Name ✓

Access type ⓘ ▼

OK Cancel

Search containers by prefix

NAME	LAST MODIFIED	ACCESS TYPE	LEASE STATE
You don't have any containers yet. Click '+ Container' to get started.			

Create Containers for Thumbnails

- Similarly create a container for thumbnails

The screenshot displays the Azure Storage Explorer application. On the left, the 'Storage accounts and Storage accounts (classic) can now be managed together in the combined list below.' message is visible. Below it, a 'Subscriptions' section shows 'Visual Studio Enterprise with MSDN - Don't see a subscription? [Switch directories](#)'. A search bar labeled 'Filter by name...' is present. The main pane shows a list of storage accounts: 'azfntest9cd0', 'azurefunctionteab18', and 'origphotography2018'. The 'origphotography2018' account is selected. On the right, the 'BLOB SERVICE' pane is open, showing a list of options: 'Containers', 'CORS', 'Custom domain', and 'Encryption'. The 'Containers' option is highlighted. A 'New container' dialog box is open on the right, showing the 'Name' field with the value 'thumbnails' and a green checkmark. The 'OK' button is highlighted.

Storage Explorer Interface:

- Top Bar:** + Add, Columns, Refresh
- Left Pane:** Storage accounts and Storage accounts (classic) can now be managed together in the combined list below.
- Subscriptions:** Visual Studio Enterprise with MSDN – Don't see a subscription? [Switch directories](#)
- Search:** Filter by name...
- Storage Accounts:** 3 items. NAME ▾
 - azfntest9cd0
 - azurefunctionteab18
 - origphotography2018
- Right Pane (BLOB SERVICE):**
 - Containers
 - CORS
 - Custom domain
 - Encryption

New container Dialog:

- Container:** Refresh
- Name:** thumbnails (with a green checkmark)
- Buttons:** OK, Cancel

Verify Containers in the Storage Accounts

+

Add

≡

Columns

↺

Refresh

i

Storage accounts and Storage accounts (classic) can now be managed together in the combined list below.

Subscriptions:

Visual Studio Enterprise with MSDN – Don't see a subscription? [Switch directories](#)

Filter by name...

3 items

NAME

▼

azfntest9cd0

...

azurefunctionteab18

...

origphotography2018

...

Search (Ctrl+/)

Access keys

Configuration

Shared access signature

Properties

Locks

Automation script

BLOB SERVICE

Containers

CORS

Custom domain

Encryption

+

Container

↺

Refresh

Essentials

▼

Search containers by prefix

NAME

LAST MODIFIED

photographs

10/22/2018, 10:22:03 PM

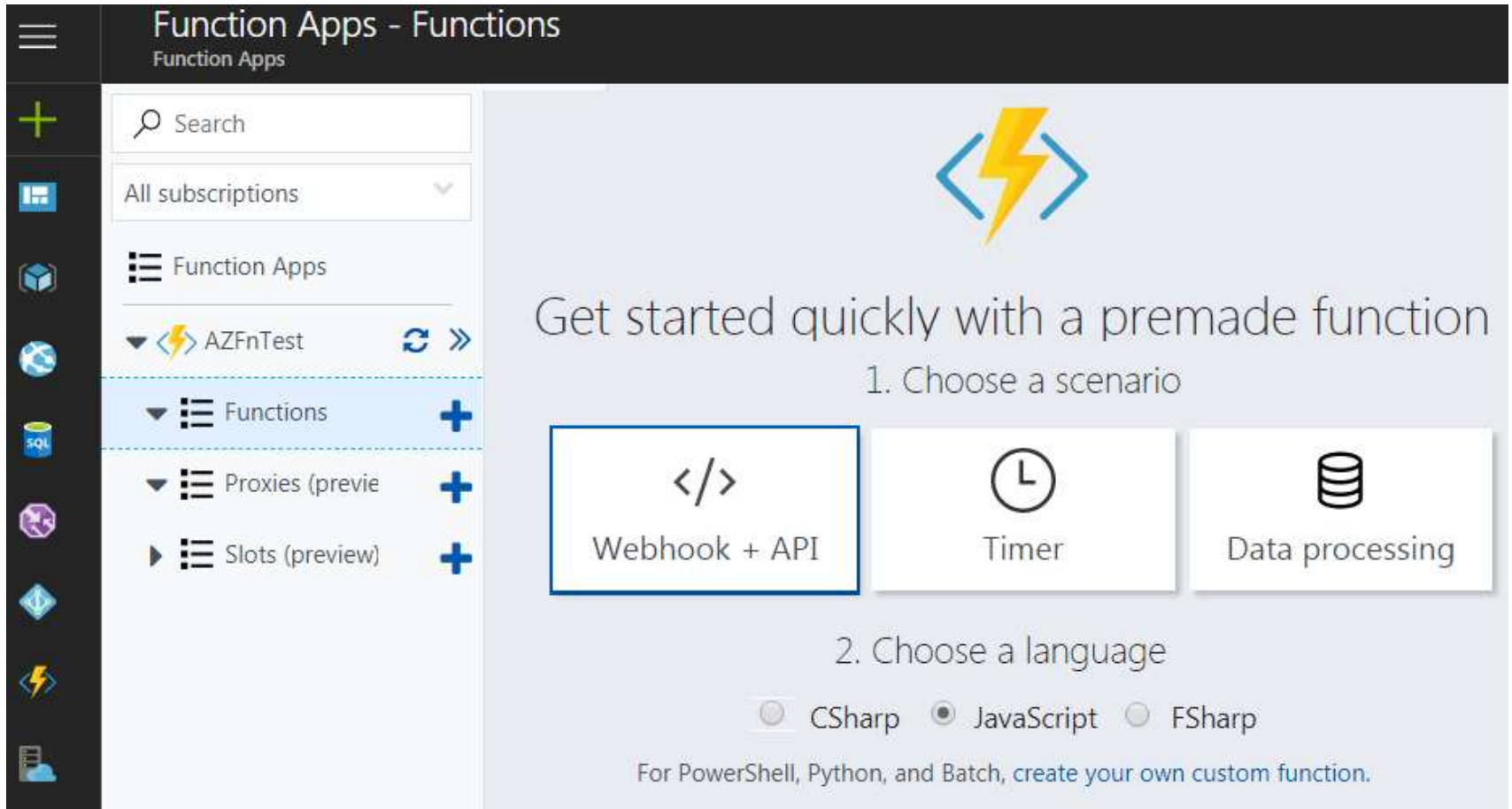
thumbnails

10/22/2018, 11:10:08 PM

- We have all the components ready to achieve our main objective of creating a function that creates thumbnails of photographs, and we can start creating a function.

Create Function in Function App

- Click on the **Functions** section in the Function App.
- Select Webhook + API and JavaScript as the language



The screenshot shows the Azure Functions portal interface. On the left, a sidebar contains a search bar, a subscription dropdown set to 'All subscriptions', and a list of function apps. The 'AZFnTest' app is selected, and its 'Functions' section is expanded, showing a '+' icon. The main area displays the 'Get started quickly with a premade function' wizard. It features a large yellow lightning bolt icon and the text 'Get started quickly with a premade function'. Below this, the first step is '1. Choose a scenario', with three options: 'Webhook + API' (highlighted with a blue border), 'Timer', and 'Data processing'. The second step is '2. Choose a language', with three radio button options: 'CSharp', 'JavaScript' (selected), and 'FSharp'. At the bottom, a link states 'For PowerShell, Python, and Batch, create your own custom function.'

Function Apps - Functions
Function Apps

Search

All subscriptions

Function Apps

AZFnTest

Functions

Proxies (previe)

Slots (preview)

Get started quickly with a premade function

1. Choose a scenario

Webhook + API

Timer

Data processing

2. Choose a language

☐ CSharp ☒ JavaScript ☐ FSharp

For PowerShell, Python, and Batch, [create your own custom function.](#)

Create Custom Function

- Click on **Custom function** so that we can utilize the already available templates:

The screenshot shows the Azure Functions portal interface. On the left is a sidebar with a search bar, a subscription dropdown set to 'All subscriptions', and a list of resources: 'Function Apps', 'AZFnTest' (with a refresh and expand icon), 'Functions' (highlighted with a plus icon), 'Proxies (preview)' (with a plus icon), and 'Slots (preview)' (with a plus icon). The main area is titled 'Function Apps - Functions' and contains two steps: '1. Choose a scenario' and '2. Choose a language'. Under the first step, three templates are shown: 'Webhook + API' (with a code icon and highlighted by a blue border), 'Timer' (with a clock icon), and 'Data processing' (with a database icon). Under the second step, three language options are shown: 'CSharp' (unselected), 'JavaScript' (selected with a radio button), and 'FSharp' (unselected). Below these is a link: 'For PowerShell, Python, and Batch, create your own custom function.' A large blue button labeled 'Create this function' is centered. Below this button is a dark blue section with the text 'Get started on your own' and two links: 'Custom function' and 'Start from source control'.

Function Apps - Functions
Function Apps

Search

All subscriptions

Function Apps

AZFnTest

Functions

Proxies (preview)

Slots (preview)

1. Choose a scenario

Webhook + API

Timer

Data processing

2. Choose a language

☐ CSharp ☒ JavaScript ☐ FSharp

For PowerShell, Python, and Batch, create your own custom function.

Create this function

or

Get started on your own

Custom function

Start from source control

Select a Template

- Select JavaScript

Function Apps - Functions
Function Apps

Search

All subscriptions

Function Apps

AZFnTest

Functions

Proxies (preview)

Slots (preview)

Choose a template below or go to the quickstart

Language: All Scenario: Core

HttpTrigger - C#
A C# function that will be run whenever it receives an HTTP request

HttpTrigger - F#
An F# function that will be run whenever it receives an HTTP request

HttpTrigger - JavaScript
A JavaScript function that will be run whenever it receives an HTTP request

TimerTrigger - C#
A C# function that will be run on a specified schedule

TimerTrigger - F#
An F# function that will be run on a specified schedule

TimerTrigger - JavaScript
A JavaScript function that will be run on a specified schedule

QueueTrigger - C#
A C# function that will be run whenever a message is added to a specified Azure Queue Storage

QueueTrigger - F#
An F# function that will be run whenever a message is added to a specified Azure Queue Storage

Note: Webhooks are "user-defined HTTP callbacks". They are usually triggered by some event, such as pushing code to a repository or a comment being posted to a blog. When that event occurs, the source site makes an HTTP request to the URL configured for the webhook. Users can configure them to cause events on one site to invoke behaviour on another

BlobTrigger Template

- Next, among JavaScript template, select BlobTrigger template:

The screenshot shows the 'Function Apps - Functions' page in the Azure portal. On the left, a sidebar contains a search bar, a subscription dropdown set to 'All subscriptions', and a list of function apps. The 'AZFnTest' app is selected, and its 'Functions' sub-item is expanded, showing a plus icon. The main area displays a heading 'Choose a template below or go to the quickstart' and two dropdowns: 'Language' set to 'JavaScript' and 'Scenario' set to 'Core'. Below these are eight template cards arranged in a 2x4 grid. The 'BlobTrigger - JavaScript' card is highlighted in the top row, fourth column. It describes a JavaScript function that runs whenever a blob is added to a specified container. Other templates include HttpTrigger, TimerTrigger, QueueTrigger, EventHubTrigger, ServiceBusQueueTrigger, ServiceBusTopicTrigger, and Generic Webhook.

HttpTrigger - JavaScript	TimerTrigger - JavaScript	QueueTrigger - JavaScript	BlobTrigger - JavaScript
A JavaScript function that will be run whenever it receives an HTTP request	A JavaScript function that will be run on a specified schedule	A JavaScript function that will be run whenever a message is added to a specified Azure Queue Storage	A JavaScript function that will be run whenever a blob is added to a specified container
EventHubTrigger - JavaScript	ServiceBusQueueTrigger - JavaScript	ServiceBusTopicTrigger - JavaScript	Generic Webhook - JavaScript
A JavaScript function that will be run whenever an event hub receives a new event	A JavaScript function that will be run whenever a message is added to a specified Service Bus queue	A JavaScript function that will be run whenever a message is added to the specified Service Bus topic	A JavaScript function that will be run whenever it receives a webhook request

Name your Function

- Provide the name of our function.
- Give the path to the container for the source and select **Storage account connection**:

The screenshot shows the 'Name your function' page in the Azure Functions portal. The left sidebar contains a search bar, 'All subscriptions' dropdown, and a navigation menu with 'Function Apps', 'AZFnTest' (selected), 'Functions' (highlighted with a plus icon), 'Proxies (preview)', and 'Slots (preview)'. The main content area has a header 'Function Apps - Functions' and three function templates: 'A JavaScript function that will be run whenever it receives a GitHub webhook request', 'A JavaScript function that will be run whenever it receives an HTTP request', and 'A JavaScript function that is triggered manually via the portal "Run" button'. Below the templates, the 'Name your function' section includes a text input for the function name 'photoProcessing'. The 'Azure Blob Storage trigger (myBlob)' section has a 'Path' input with the value 'photographs' and a blue 'Create' button. The 'Storage account connection' section features a dropdown menu with the selected value 'AzureWebJobsDashboard' and a 'show value' link. The dropdown list is open, showing three options: 'AzureWebJobsDashboard', 'AzureWebJobsStorage', and 'WEBSITE_CONTENTAZUREFILECONNECTIONSTRING origphotography2018 STORAGE', with the last option highlighted in blue and marked as 'new'.

Function Apps - Functions
Function Apps

Search

All subscriptions

Function Apps

AZFnTest

Functions +

Proxies (preview) +

Slots (preview) +

A JavaScript function that will be run whenever it receives a GitHub webhook request

A JavaScript function that will be run whenever it receives an HTTP request

A JavaScript function that is triggered manually via the portal "Run" button

Name your function

photoProcessing

Azure Blob Storage trigger (myBlob)

Path ⓘ

photographs

Create

Storage account connection ⓘ show value

AzureWebJobsDashboard new

AzureWebJobsDashboard

AzureWebJobsStorage

WEBSITE_CONTENTAZUREFILECONNECTIONSTRING origphotography2018 STORAGE

Code Editor

- Look at the function and code available in the code editor in the Microsoft Azure Portal:

The screenshot shows the Microsoft Azure Portal interface for editing a function. The top bar is dark grey with the text "Function Apps - Functions" and "Function Apps" below it. On the left, there is a sidebar with a search bar, "All subscriptions" dropdown, "Function Apps" section, and a tree view showing "AZFnTest" > "Functions" > "photoProcessing" (selected). Below the tree are links for "Integrate", "Manage", "Monitor", "Proxies (preview)", and "Slots (preview)". The main area is titled "index.js" and contains a "Save" button and a "Run" button. The code editor shows the following JavaScript code:

```
1 module.exports = function (context, myBlob) {  
2   context.log("JavaScript blob trigger function processed blob \n Name:", context.bindingData.name,  
3   context.done());  
4 };
```

Below the code editor is a "Logs" section.

Triggers and Outputs

- Before we write the actual code in the function, let's configure the triggers and outputs. Select **Blob** parameter name, **Storage account connection**, and **Path**.

The screenshot shows the 'Function Apps - Functions - Integrate' interface. On the left is a sidebar with a search bar and a list of subscriptions and function apps. The 'AZFnTest' function app is selected, and the 'Integrate' tab is active. The main area is titled 'Triggers' and shows the 'Azure Blob Storage (myBlob)' trigger selected. Below this, the configuration fields are visible: 'Blob parameter name' is 'myBlob', 'Path' is 'photographs', and 'Storage account connection' is 'origphotography2017_STORAGE'. There are also buttons for '+ New Input' and '+ New Output' under the 'Inputs' and 'Outputs' tabs respectively. A 'delete' link is next to the trigger name. At the bottom, there is a '+ Documentation' link.

Function Apps - Functions - Integrate
Function Apps

Search

All subscriptions

Function Apps

AZFnTest

Functions

photoProcess

Integrate

Manage

Monitor

Proxies (preview)

Slots (preview)

Triggers

Azure Blob Storage (myBlob)

Azure Blob Storage trigger (myBlob) delete

Blob parameter name

myBlob

Path

photographs

Storage account connection

origphotography2017_STORAGE

show value

new

+ Documentation

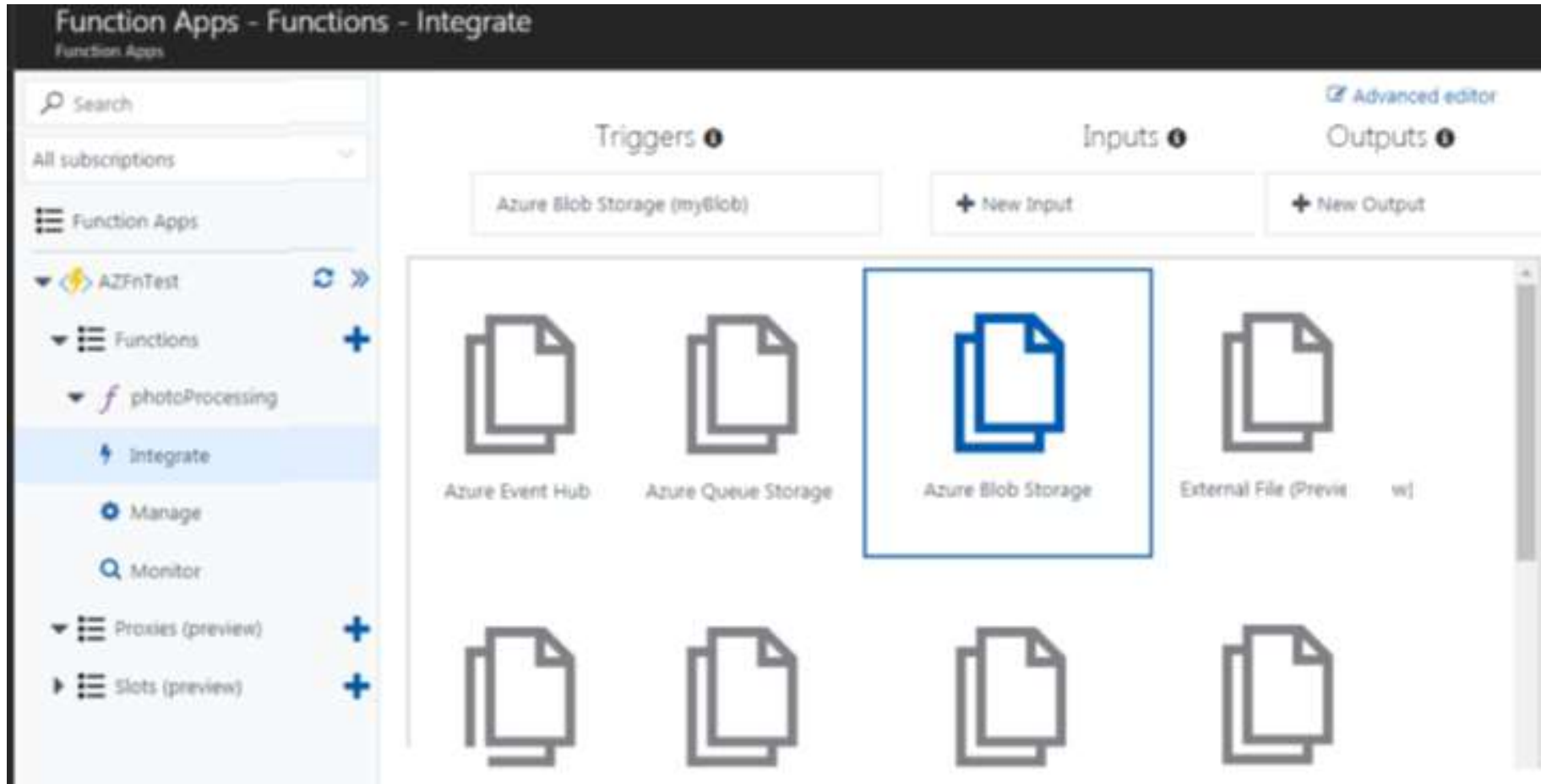
+ New Input

+ New Output

Advanced editor

Outputs

- Click on **New Output**. Select **Azure Blob Storage**:



Blob Parameters

- Select **Blob** parameter name, **Storage account connection**, and **Path**. Click on **Save**:

The screenshot shows the 'Function Apps - Functions - Integrate' interface. On the left, the 'Integrate' option is selected in the sidebar. The main area is titled 'Azure Blob Storage output (outputBlob)'. It contains the following fields:

- Blob parameter name**: A text box containing 'outputBlob'.
- Path**: A text box containing 'thumbnails/{name}'.
- Use function return value**: An unchecked checkbox.
- Storage account connection**: A dropdown menu showing 'origphotography2018_STORAGE' with a 'show value' link and a 'new' button.

At the bottom, there are 'Save' and 'Cancel' buttons. The top of the interface shows the function name 'AZFnTest' and a search bar.

Review Output Bindings

Function Apps - Functions - Integrate

Function Apps

🔍 "AZFnTest" ❌

All subscriptions ▾

☰ Function Apps

▼ ⚡ AZFnTest ↻ >>

▼ ☰ Functions +

▼ *f*

⚡ Integrate

⚙️ Manage

🔍 Monitor

▶ *f* HttpTriggerForLoginAPI

▼ ☰ Proxies (preview) +

▶ ☰ Slots (preview) +

Triggers ⓘ

Azure Blob Storage (myBlob)

+ New Input

Inputs ⓘ

Outputs ⓘ

Azure Blob Storage (\$return)

+ New Output

Advanced edito

Azure Blob Storage output (\$return) delete

Blob parameter name ⓘ

\$return

☒ Use function return value

Storage account connection ⓘ show value

origphotography2018_STORAGE ▼ new

Path ⓘ

thumbnails/{name}

@Nishava, Inc - Zoran B. Djordjević

43

Review Function in Advance Editor

- Click on the advanced editor link to review **function.json**:

Function Apps - Functions - Integrate

Function Apps

All subscriptions

Function Apps

AZFnTest

Functions

Integrate

Manage

Monitor

HttpTriggerForLoginAPI

Proxies (preview)

Slots (preview)

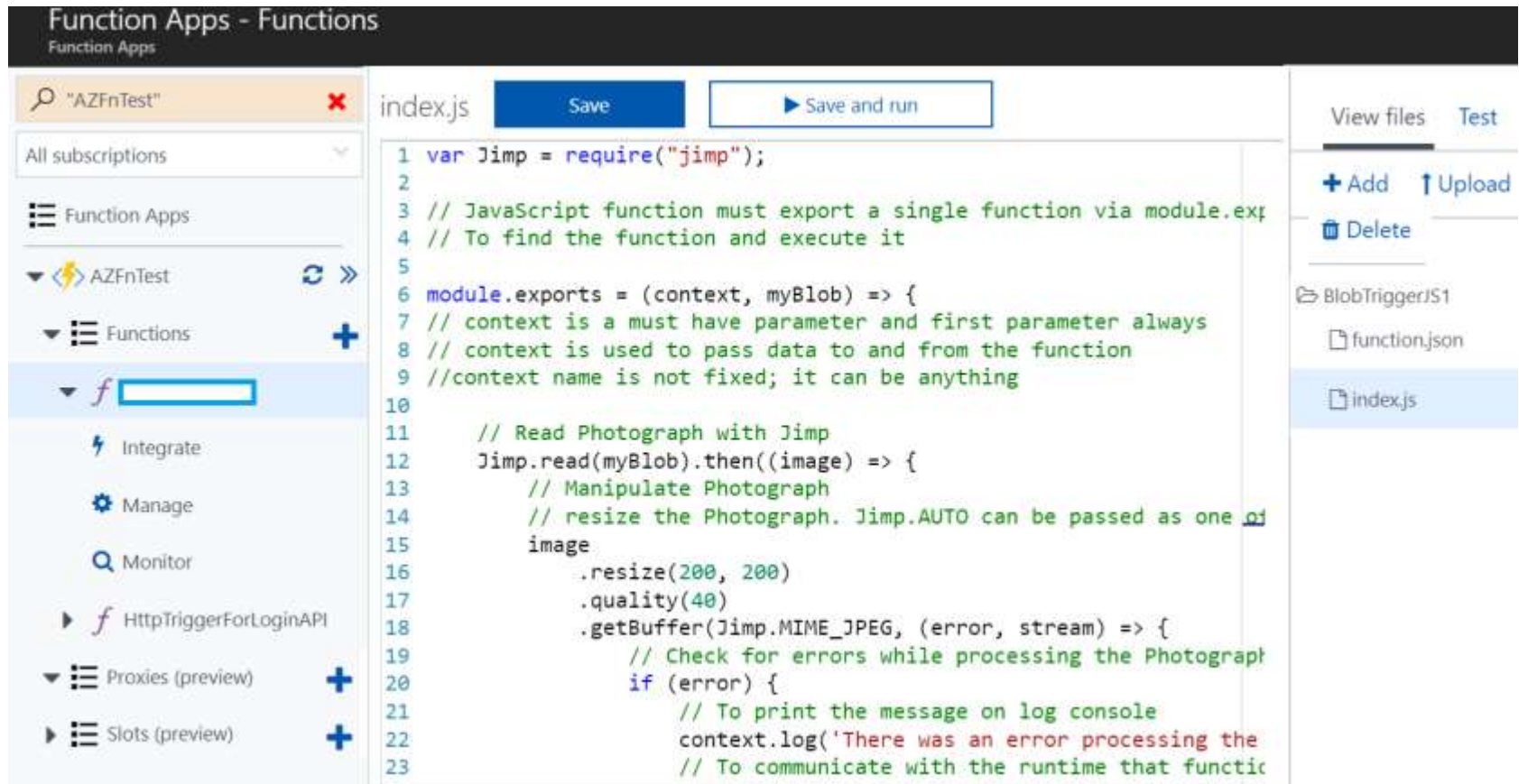
Your app is currently in read\write mode because you've set the edit mode to read\write despite having source control enabled. Any changes you make may get overwritten with your next deployment. To change edit mode visit [function app settings](#).

function.json [Standard edi](#)

```
1 {
2   "bindings": [
3     {
4       "name": "myBlob",
5       "type": "blobTrigger",
6       "direction": "in",
7       "path": "photographs/{name}",
8       "connection": "origphotography2017_STORAGE",
9       "dataType": "binary"
10    },
11    {
12      "type": "blob",
13      "name": "$return",
14      "path": "thumbnails/{name}",
15      "connection": "origphotography2017_STORAGE",
16      "direction": "out"
17    }
18  ],
```

Paste your Code into the Editor

- Now, paste the function code for creating a thumbnail into the **Functions** code editor:



- Now, everything is set and configured, let's upload a photograph in the photographs blob container:

Upload Photograph

- Click on the container and click on **Upload**. Select a photograph to be uploaded to the container.

The screenshot displays the Microsoft Azure Storage Explorer interface. The breadcrumb navigation at the top shows the path: **Microsoft Azure** > Storage accounts > origphotography2018 - Containers > photographs > Upload blob. The interface is divided into three main panes. The left pane, titled 'Containers', shows a list of containers with 'photographs' selected. The middle pane, titled 'photographs Container', shows the 'Upload' button and a list of files: DSC01525.JPG, DSC01585.JPG, DSC09771-001.JPG, and IMG_4854.JPG. The right pane, titled 'Upload blob', shows the 'Files' section with the filename 'DSC00591.JPG' entered, and the 'Blob type' set to 'Block blob' with a 'Block size' of '100 MB'.

Microsoft Azure Storage accounts > origphotography2018 - Containers > photographs > Upload blob

Containers ✦ ✕ photographs Container

+ Container Refresh Upload Refresh Delete container

Essentials ▾

Search containers by prefix

NAME

photographs ...

thumbnails ...

Search blobs by prefix (case-sensitive)

NAME

DSC01525.JPG

DSC01585.JPG

DSC09771-001.JPG

IMG_4854.JPG

Files ⓘ

"DSC00591.JPG"

Blob type ⓘ

Block blob

Block size ⓘ

100 MB

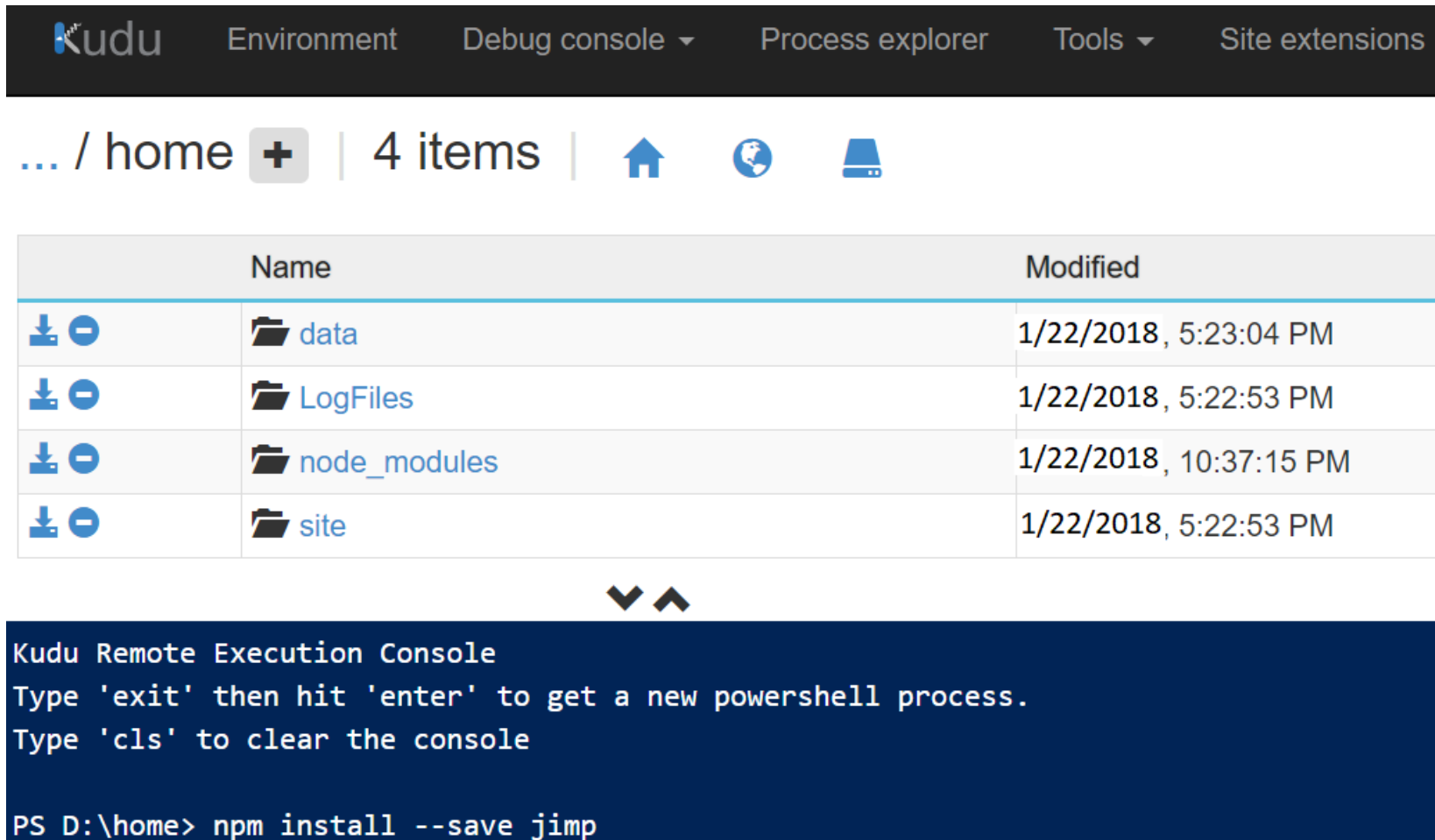
Check Logs in Function Apps

- Go to **Function Apps** and check the logs.
- We may get an Error: **Cannot find module 'jimp'**:









```
2017-06-30T16:55:01.309 Function started  
(Id=411e4d84-5ef0-4ca9-b963-ed94c0ba8e84)  
2017-06-30T16:55:01.371 Function completed (Failure,  
Id=411e4d84-5ef0-4ca9-b963-ed94c0ba8e84, Duration=59ms)  
2017-06-30T16:55:01.418 Exception while executing function:  
Functions.photoProcessing. mscorlib: Error: Cannot find module  
'jimp'  
at Function.Module._resolveFilename (module.js:455:15)  
at Function.Module._load (module.js:403:25)  
at Module.require (module.js:483:17)  
at require (internal/module.js:20:19)  
at Object.<anonymous>
```

Troubleshoot Azure Function

- Go to the **Kudu** console of the Azure Function App.
- Click on **Debug Console** and select **Powershell**.
- Execute the `npm install --save jimp` command:



The screenshot displays the Kudu console interface. At the top, there is a navigation bar with links: Kudu, Environment, Debug console (selected), Process explorer, Tools, and Site extensions. Below the navigation bar, the breadcrumb path is "... / home" followed by a plus icon, "4 items", and three icons representing a home, a globe, and a server. A table lists the files and folders in the current directory:

	Name	Modified
	 data	1/22/2018, 5:23:04 PM
	 LogFiles	1/22/2018, 5:22:53 PM
	 node_modules	1/22/2018, 10:37:15 PM
	 site	1/22/2018, 5:22:53 PM

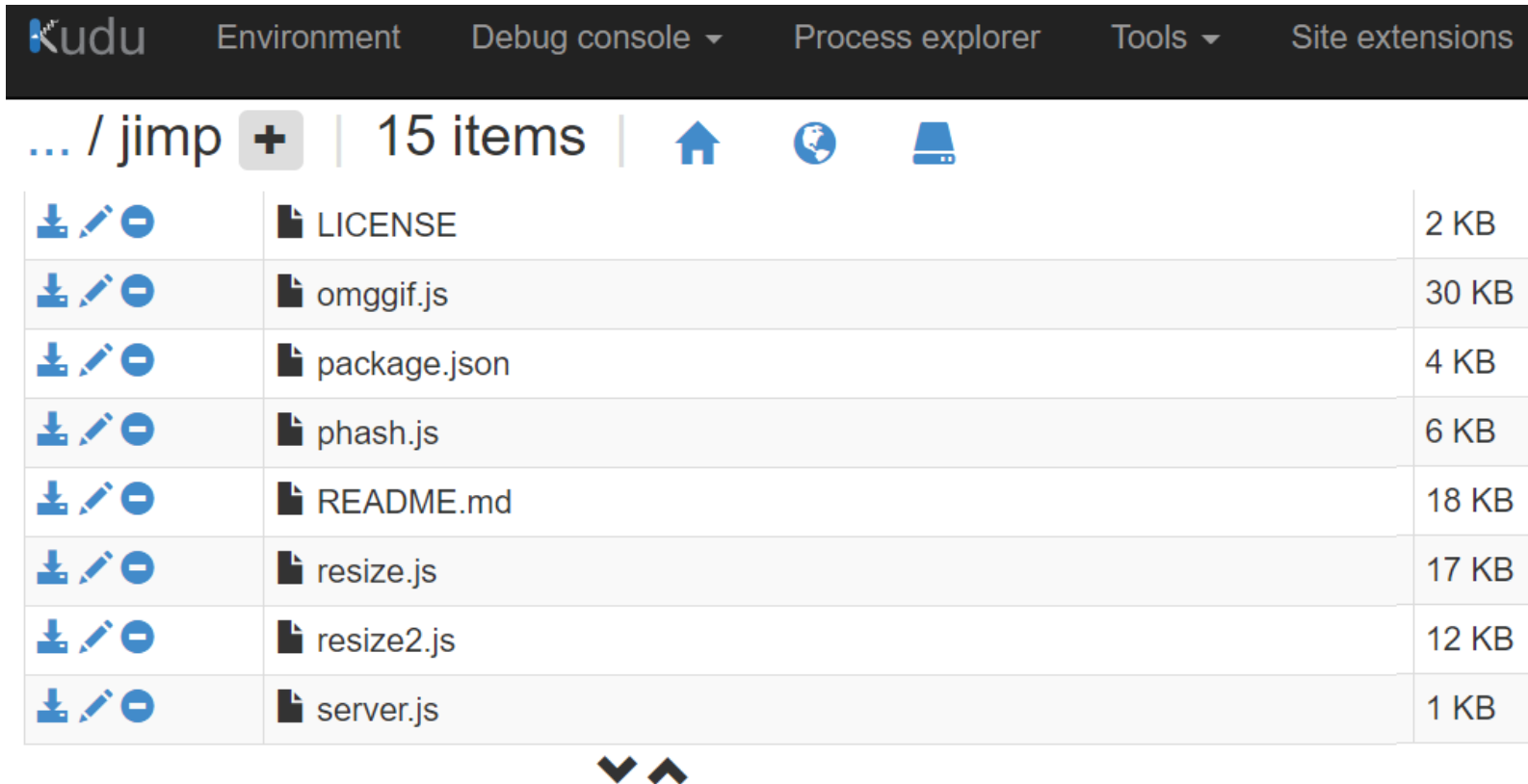
Below the table, there are two small arrows pointing up and down. At the bottom, the Kudu Remote Execution Console is shown with the following text:

```
Kudu Remote Execution Console
Type 'exit' then hit 'enter' to get a new powershell process.
Type 'cls' to clear the console

PS D:\home> npm install --save jimp
```


jimp Module

- Once the command execution has completed successfully, go to the node_modules directory and review the jimp module in this directory:



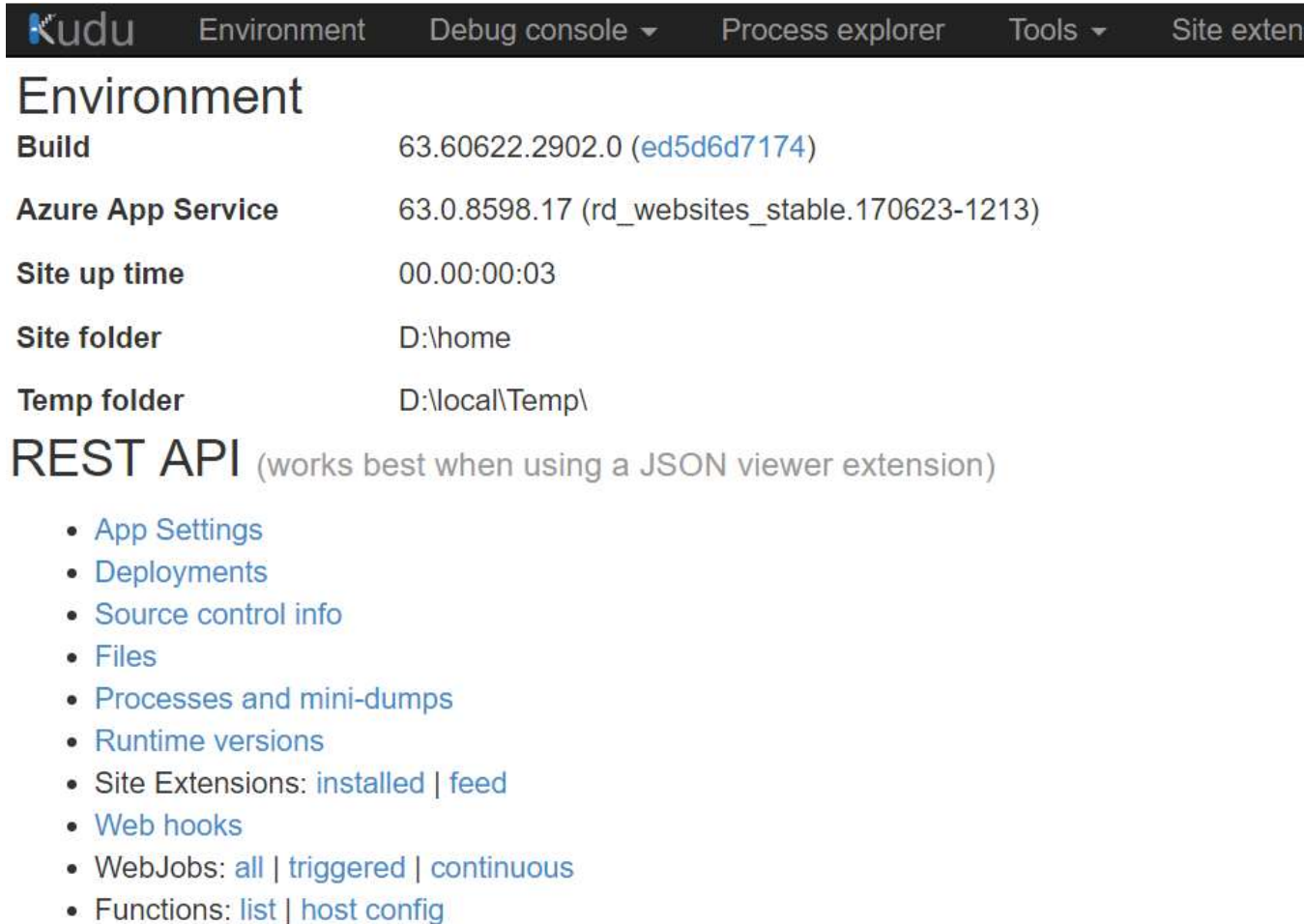
The screenshot shows the Kudu web interface for a remote environment. The top navigation bar includes 'Kudu', 'Environment', 'Debug console', 'Process explorer', 'Tools', and 'Site extensions'. Below the navigation bar, the breadcrumb path is '... / jimp' with a '+' icon, followed by '15 items'. There are three icons: a home icon, a globe icon, and a server icon. A table lists the files in the directory:

File Name	Size
LICENSE	2 KB
omggif.js	30 KB
package.json	4 KB
phash.js	6 KB
README.md	18 KB
resize.js	17 KB
resize2.js	12 KB
server.js	1 KB

```
Kudu Remote Execution Console
Type 'exit' then hit 'enter' to get a new powershell process.
Type 'cls' to clear the console
PS D:\home>
```

Folder Structure

- Understand the folder structure of a function in the Kudu editor. Click on the **Debug Console** and select **Powershell**:



The screenshot shows the Kudu editor interface. At the top is a dark navigation bar with the Kudu logo and several tabs: Environment, Debug console (selected), Process explorer, Tools, and Site extensions. Below the navigation bar, the 'Environment' tab is active, displaying a list of system and application properties. Below this, the 'REST API' section is visible, featuring a list of links for various API endpoints.

Environment	
Build	63.60622.2902.0 (ed5d6d7174)
Azure App Service	63.0.8598.17 (rd_websites_stable.170623-1213)
Site up time	00.00:00:03
Site folder	D:\home
Temp folder	D:\local\Temp\

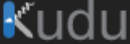
REST API (works best when using a JSON viewer extension)





- [App Settings](#)
- [Deployments](#)
- [Source control info](#)
- [Files](#)
- [Processes and mini-dumps](#)
- [Runtime versions](#)
- Site Extensions: [installed](#) | [feed](#)
- [Web hooks](#)
- WebJobs: [all](#) | [triggered](#) | [continuous](#)
- Functions: [list](#) | [host config](#)













More information about Kudu can be found on the [wiki](#).



node_modules Directory

- Click on the node_modules directory to review all the modules available for usage:

 Environment Debug console ▾ Process explorer Tools ▾ Site extensions

/  | 4 items |   

Name	
 	 data
 	 LogFiles
 	 node_modules
 	 site

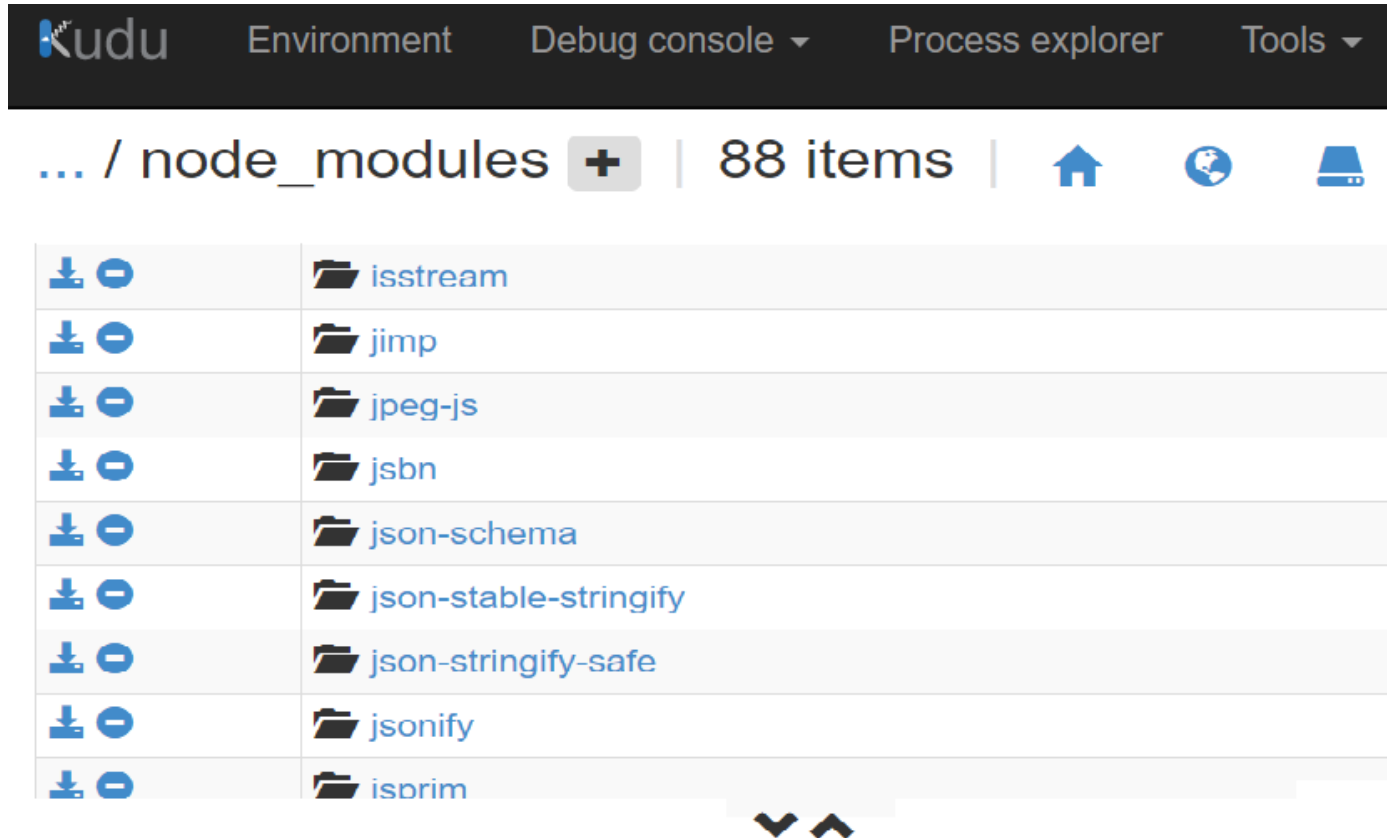
Kudu Remote Execution Console

Type 'exit' then hit 'enter' to get a new powershell process.
Type 'cls' to clear the console



















PS D:\home>

Available Modules

- There are many modules available for the Azure Functions to utilize:



The screenshot shows the Kudu web interface for a remote execution console. The top navigation bar includes 'Kudu', 'Environment', 'Debug console', 'Process explorer', and 'Tools'. Below the navigation bar, the breadcrumb path is '... / node_modules' with a plus icon, followed by '88 items'. A list of modules is displayed in a table-like format, each with a download icon, a minus icon, and the module name. The modules listed are: isstream, jimp, jpeg-js, jsbn, json-schema, json-stable-stringify, json-stringify-safe, jsonify, and isprim. Below the list, there are two small icons: a downward arrow and an upward arrow.

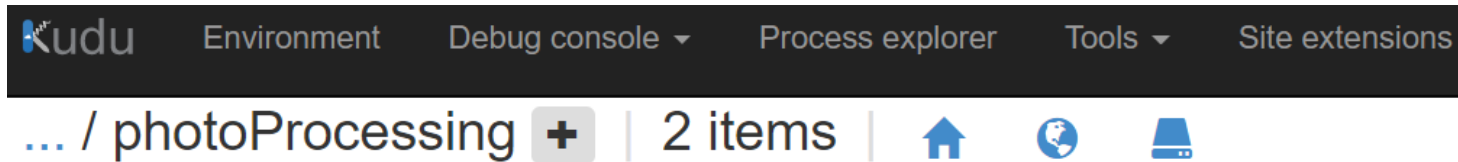
 	isstream
 	jimp
 	jpeg-js
 	jsbn
 	json-schema
 	json-stable-stringify
 	json-stringify-safe
 	jsonify
 	isprim









Kudu Remote Execution Console
Type 'exit' then hit 'enter' to get a new powershell process.
Type 'cls' to clear the console

```
PS D:\home>  
PS D:\home\node_modules>
```

www.root

- Click on the site in the Kudu editor. Go to `www.root` and select the function name to review which files are available in the specific function.
- As we saw in the Azure Portal, we can also see `function.json` and `index.js`:



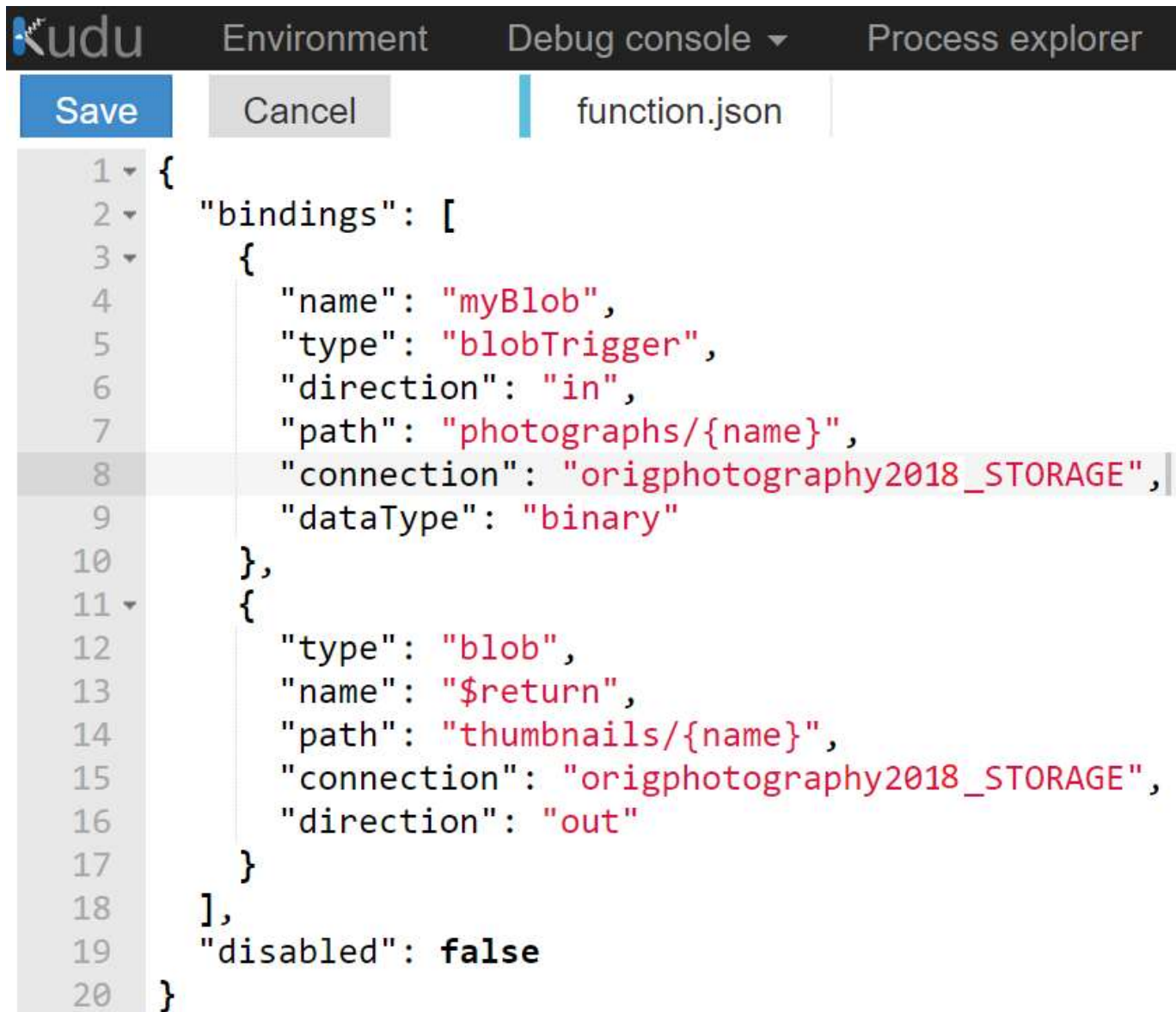
	Name	Size
  	 <code>function.json</code>	1 KB
  	 <code>index.js</code>	1 KB



```
Kudu Remote Execution Console
Type 'exit' then hit 'enter' to get a new powershell process.
Type 'cls' to clear the console

PS D:\home>
PS D:\home\node_modules>
PS D:\home>
PS D:\home\site>
PS D:\home\site\wwwroot>
PS D:\home\site\wwwroot\photoProcessing>
```

Edit function.json



The screenshot shows the Kudu web interface for editing a function.json file. The top navigation bar includes 'Kudu', 'Environment', 'Debug console', and 'Process explorer'. Below this, there are buttons for 'Save' and 'Cancel', and a tab labeled 'function.json'. The main area displays the JSON content of the file, with line numbers 1 through 20 on the left. The JSON is a function definition with bindings for 'myBlob' and '\$return'.

```
1 {  
2   "bindings": [  
3     {  
4       "name": "myBlob",  
5       "type": "blobTrigger",  
6       "direction": "in",  
7       "path": "photographs/{name}",  
8       "connection": "origphotography2018_STORAGE",  
9       "dataType": "binary"  
10    },  
11    {  
12      "type": "blob",  
13      "name": "$return",  
14      "path": "thumbnails/{name}",  
15      "connection": "origphotography2018_STORAGE",  
16      "direction": "out"  
17    }  
18  ],  
19  "disabled": false  
20 }
```

Execute Function

- Next, we execute the Azure Function again by uploading a photograph to a blob container.
- We upload a photo to the photographs blob container that we created earlier.
- Once the photograph is uploaded, go to Function Apps and verify the logs:

```
2018-01-22T11:03:11 Welcome, you are now connected to log-streaming service.
```

```
2018-01-22T11:04:11 No new trace in the past 1 min(s).
```

```
2018-01-22T11:05:11 No new trace in the past 2 min(s).
```

```
2018-01-22T11:05:11.656 Function started (Id=e3f715fa-da5b-4cf6-9ada-410ec8db956a)
```

```
2018-01-22T11:06:53.592 Successfully processed the Photograph
```

```
2018-01-22T11:06:53.686 Function completed (Success, Id=e3f715fada5b-4cf6-9ada-410ec8db956a, Duration=102034ms)
```

Check the Photograph is Uploaded

tainers	photographs	Blob properties
<div>✦ ✕</div> <div>✚ Container ↻ Refresh</div> <div>Essentials ▾</div> <div><input type="text" value="Search containers by prefix"/></div> <div><div>NAME</div><div>photographs ...</div><div>thumbnails ...</div></div>	<div>⬆ Upload ↻ Refresh 🗑 Delete container ≡ Propert</div> <div>Location: photographs</div> <div><input type="text" value="Search blobs by prefix (case-sensitive)"/></div> <div><div>NAME</div><div><div>DSC00591.JPG</div><div>DSC01525.JPG</div><div>DSC01585.JPG</div><div>DSC09771-001.JPG</div><div>IMG_4854.JPG</div></div></div>	<div>⬇ Download 🗑 Delete</div> <div>NAME</div> <div>DSC00591.JPG</div> <div>URL</div> <div>https://origphotography2018.blob.core.v</div> <div>LAST MODIFIED</div> <div>01/22/2018 4:35:06 PM</div> <div>TYPE</div> <div>Block blob</div> <div>SIZE</div> <div>516.59 KiB</div>

Verify the Thumbnail Container

- Check that the photograph has uploaded in the thumbnails container and verify the size of the photograph:

The screenshot displays the Microsoft Azure portal interface for managing storage accounts. The breadcrumb navigation at the top indicates the path: **Storage accounts** > **origphotography2017 - Containers** > **thumbnails** > **Blob properties**.

The interface is divided into three main sections:

- Containers:** Located on the left, it shows a list of containers. The 'thumbnails' container is selected and highlighted in blue. Other containers visible include 'photographs'.
- thumbnails Container:** The central pane shows the 'Location: thumbnails'. It includes a search bar for blobs by prefix and a list of blobs. The blob 'DSC00591.JPG' is selected and highlighted in blue.
- Blob properties:** The right pane displays the properties for the selected blob 'DSC00591.JPG':
 - NAME:** DSC00591.JPG
 - URL:** [https://origphotography2018.blob](https://origphotography2018.blob.core.windows.net/thumbnails/DSC00591.JPG)
 - LAST MODIFIED:** 01/22/2018, 4:35:08 PM
 - TYPE:** Block blob
 - SIZE:** 6.44 KiB

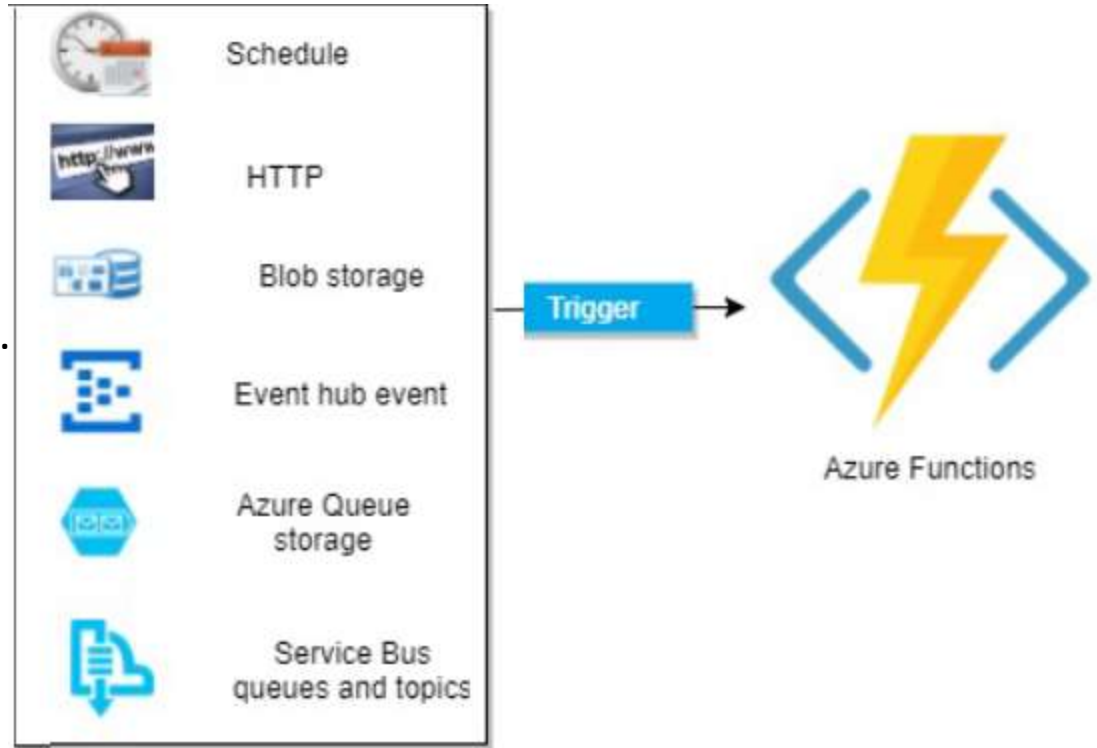
Triggers

- In normal English, a trigger is an event or situation that causes something to start. This something can be some sort of processing of data or some other service that performs some action.
- Triggers are a set of functions that get executed when some event gets fired.
- Most of us are familiar with Database Triggers. Azure triggers are very similar to database triggers, except that they react to a broader set of events.
- In Azure, we have different types of triggers, such as an implicit trigger, and we can also create a manual trigger.
- Azure Functions allow you to write code in response to a trigger.
- As we have seen on previous slides when a photograph is uploaded into a blob container of a storage account, the Azure Function will start execution and resize the photograph and create a thumbnail of it.
- One Azure function must have exactly one trigger. One function can not have multiple triggers.

Events that fire triggers

Events that fire triggers are:

- scheduled events,
- HTTP messages,
- uploads or deletions in Blob Storage,
- Event Hub Events,
- Azure Queue Messages,
- and Service Bus messages delivered to queues or topics.



Types of Triggers

- **TimerTrigger:** This trigger is called on a predefined schedule. We can set the time or execution of the Azure Function using this trigger.
- **BlobTrigger:** This trigger will get fired when a new or updated blob is detected. The blob contents are provided as input to the function.
- **EventHubTrigger:** This trigger is used for the application instrumentation, the user experience, workflow processing, and in the **Internet of Things (IoT)**. This trigger will get fired when any events are delivered to an Azure event hub.
- **HTTPTrigger:** This trigger gets fired when the HTTP request comes.
- **QueueTrigger:** This trigger gets fired when any new messages come in an Azure Storage queue.
- **Generic Webhook:** This trigger gets fired when the Webhook HTTP requests come from any service that supports Webhooks.
- **GitHub Webhook:** This trigger is fired when an event occurs in your GitHub repositories. The GitHub repository supports events such as Branch created, Delete branch, Issue comment, and Commit comment.
- **Service Bus trigger:** This trigger is fired when a new message comes from a service bus queue or topic.

Http Trigger

- The HTTP trigger is normally used to create the API or services, where we request for data using the HTTP protocol and get the response. We can also integrate the HTTP trigger with a Webhook.
- Now we will create the HTTP Login API. We will send the login credential through an HTTP post request and get the response as to whether the user is valid or not.
- Since we have already created a Function app in the previous example, we can now add multiple functions to it.

Http Trigger Java Script

- Click on + |, select **HttpTrigger-JavaScript**, provide the function name, and click on the **Create** button:

The screenshot shows the Azure Functions portal interface. On the left, the 'Functions' tab is selected in the left-hand menu, and a green box highlights the '+' icon next to it. The main area displays three function templates: 'HttpTrigger - C#', 'HttpTrigger - F#', and 'HttpTrigger - JavaScript'. The 'HttpTrigger - JavaScript' template is highlighted with a green box. Below the templates, a form titled 'Name your function' is shown, also highlighted with a green box. This form contains a text input field with the value 'HttpTriggerForLoginAPI', a dropdown menu for 'HTTP trigger (req)' set to 'Function', and an 'Authorization level' dropdown set to 'Function'. At the bottom, a blue 'Create' button is highlighted with a green box.

Language: All Scenario: Core

HttpTrigger - C#
A C# function that will be run whenever it receives an HTTP request

HttpTrigger - F#
An F# function that will be run whenever it receives an HTTP request

HttpTrigger - JavaScript
A JavaScript function that will be run whenever it receives an HTTP request

Name your function

HttpTriggerForLoginAPI

HTTP trigger (req)

Authorization level ⓘ

Function

Create

Default Template

- After we click on the **Create** button, the default template will be available. Now, we can edit and test the function:

The screenshot displays the Azure Functions editor interface. At the top, there's a file explorer showing 'index.js' with 'Save' and 'Run' buttons. The main editor area contains the following JavaScript code:

```
1 module.exports = function (context, req) {
2   context.log('JavaScript HTTP trigger function processed a request.');
```

```
3
4   if (req.query.name || (req.body && req.body.name)) {
5     context.res = {
6       // status: 200, /* Defaults to 200 */
7       body: "Hello " + (req.query.name || req.body.name)
8     };
9   }
10  else {
11    context.res = {
12      status: 400,
13      body: "Please pass a name on the query string or in the request body"
14    };
15  }
16  context.done();
17 };
```

On the right side, there's a 'Test' tab with a dropdown menu set to 'POST'. Below it, the 'Query' section shows 'There are no query parameters' with a '+ Add parameter' link. The 'Headers' section shows 'There are no headers' with a '+ Add header' link. The 'Request body' section shows a JSON object:

```
1 {
2   "name": "Azure"
3 }
```

At the bottom, there's a 'Logs' section with buttons for 'Pause', 'Clear', 'Copy logs', and 'Expand'.

Edit the Code

- Now edit the code as follows:

```
module.exports = function (context, req) {
  context.log('JavaScript HTTP trigger function processed a request.');
```

```
  if (req.body && req.body.username && req.body.password) {
    if (req.body.username !== 'admin' && req.body.password !== '@dm!n1'){
      context.res = {
        body: "Invalid user"
      };
    } else {
      context.res = {
        body: "User" + (req.body.username) + " is valid"
      };
    }
  }
  else {
    context.res = {
      status: 400,
      body: "Please provide a username and password in the request body"
    };
  }
  context.done();
};
```

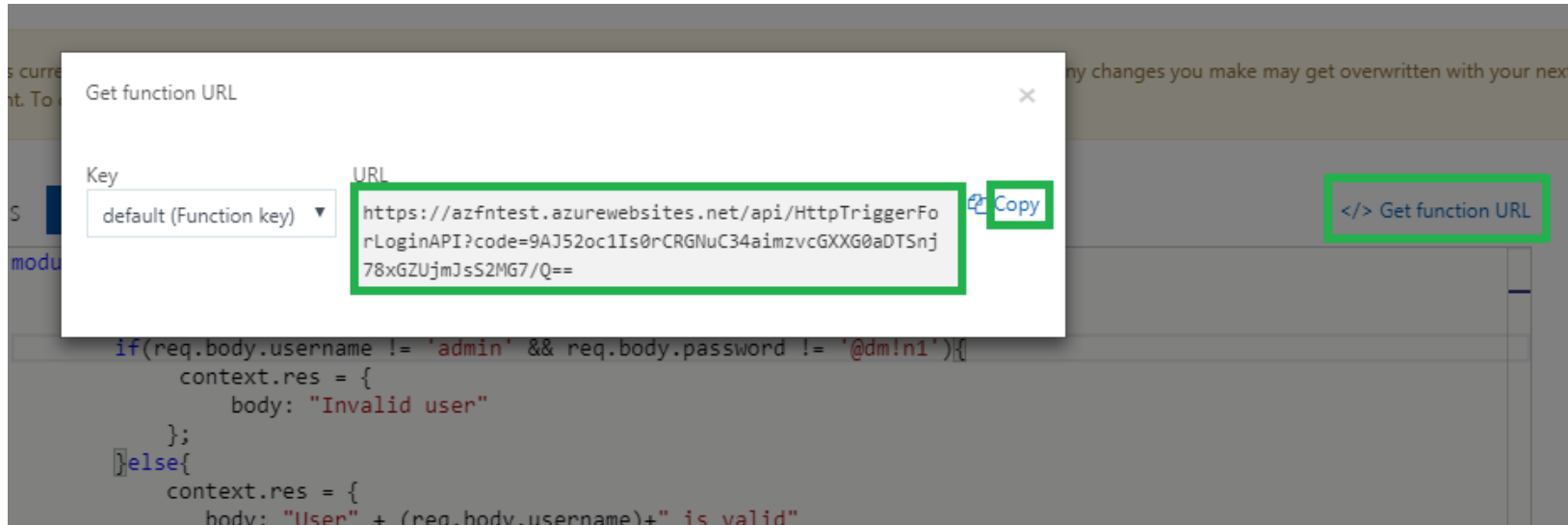

Save and Run

- Save and run the code, as shown in the following screenshot:



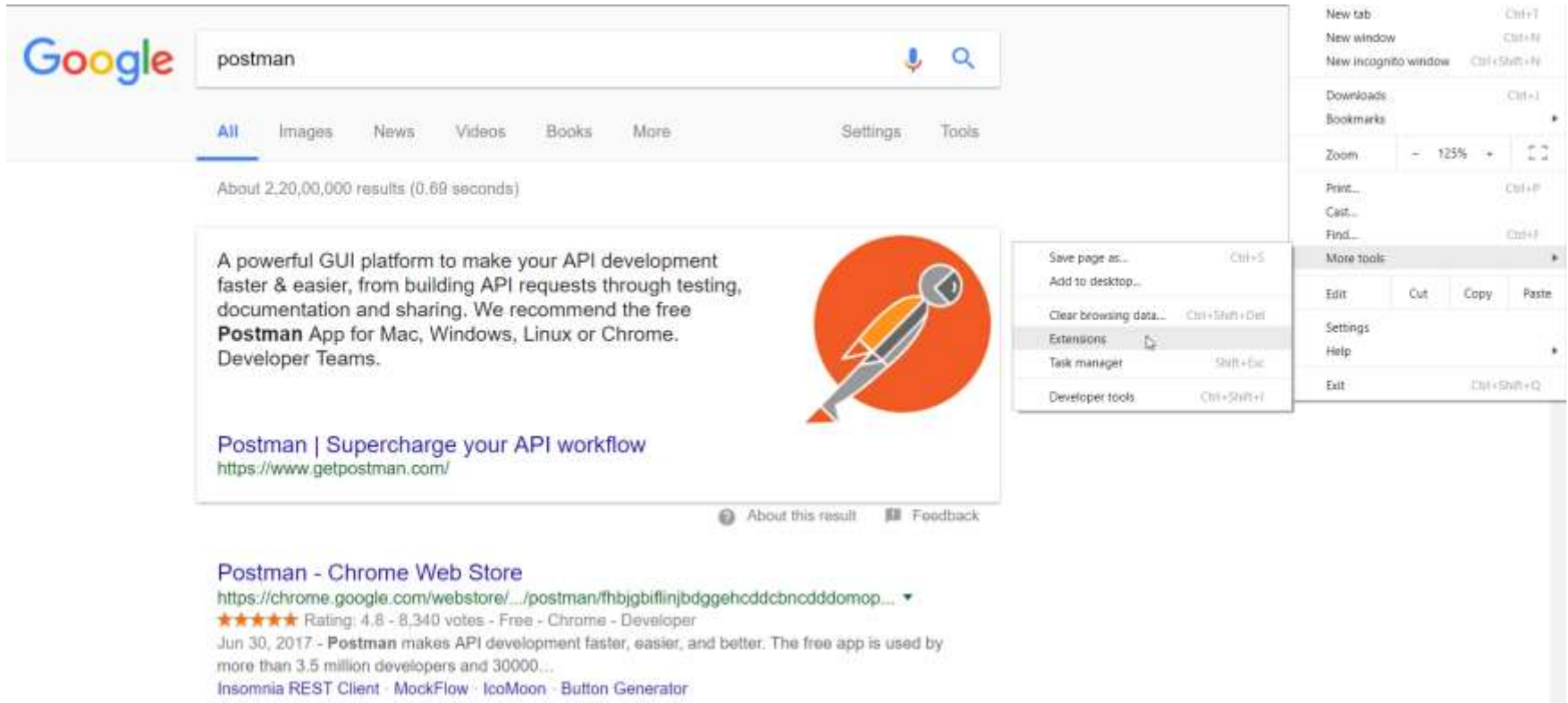
Check the Service in Postman

- Check this service in Postman. To get the **URL** from the function, click on **Get function URL**:



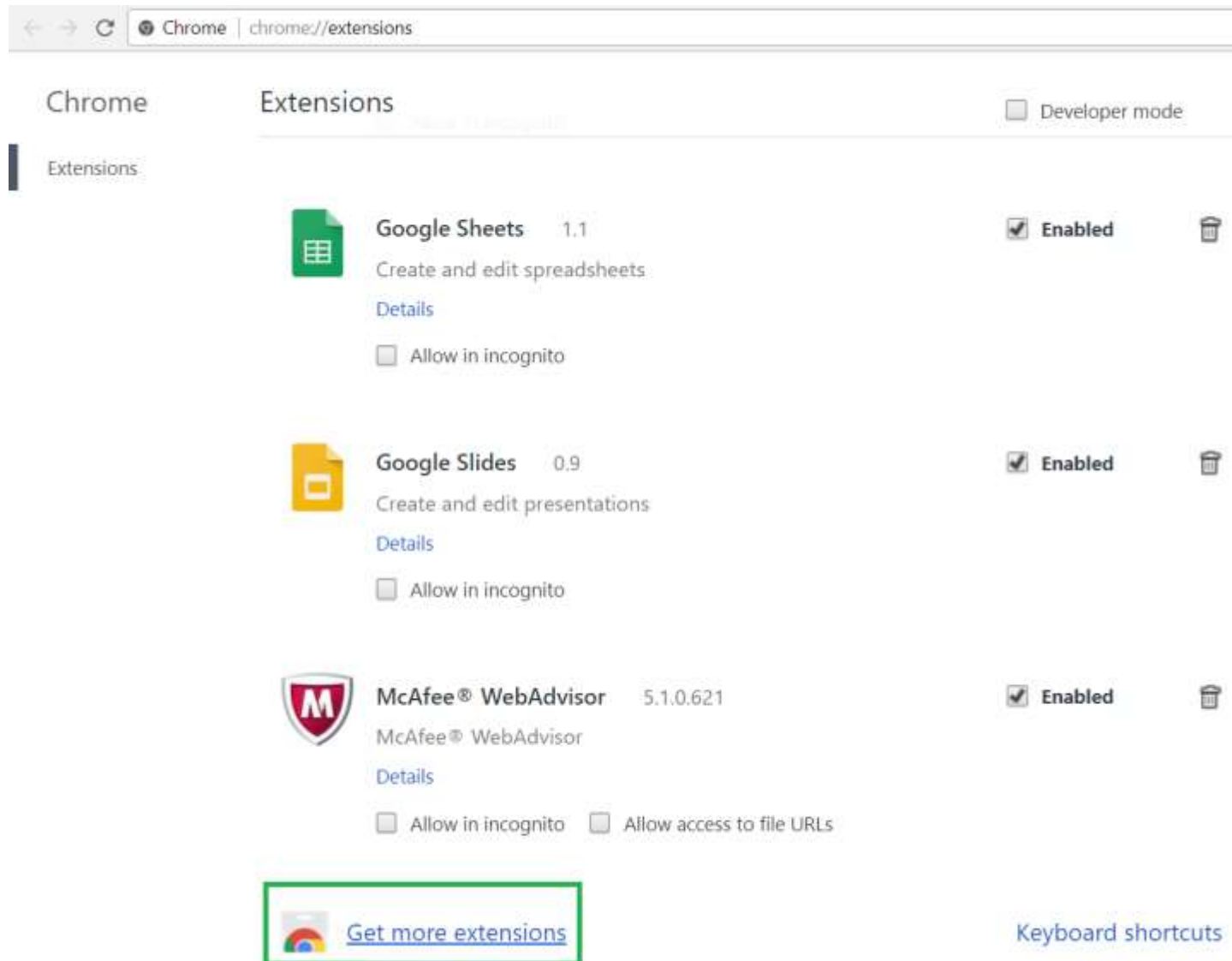
Postman

Postman is a Chrome extension for REST developers to test APIs. To add the Chrome extension, go to **Settings** in Chrome and select **More tools | Extensions**, as shown in the following:



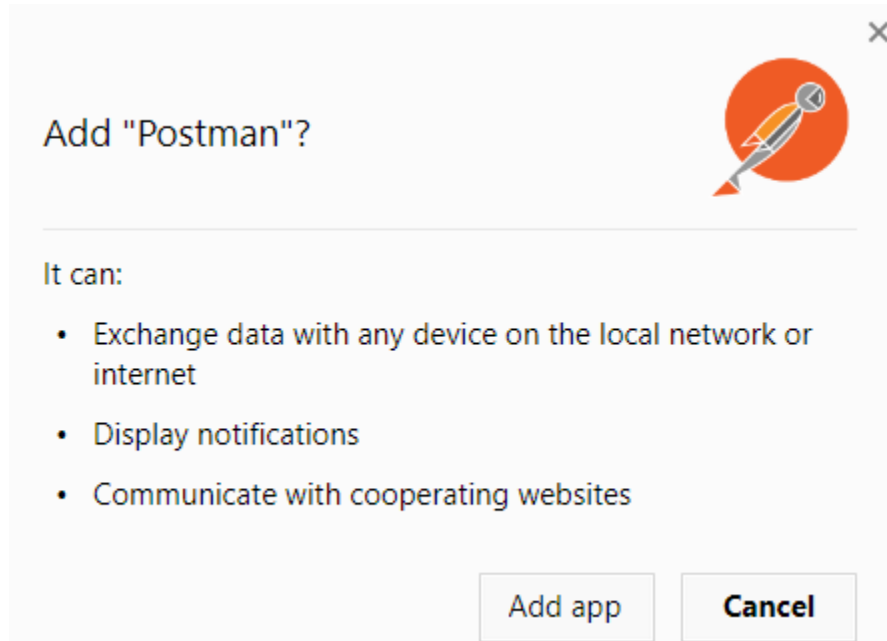
Extensions

- Click on **Get more extensions:**



Add Postman

- Now search for postman and then click on + **ADD TO CHROME**.
- Click on **Add app**.



- Launch the Postman app. Click on **Sign Up with Google**.
- Copy the function URL and paste it in Postman. Select the method type **POST** and provide a request body and click on the **Send** button:

Trigger Response

- If we provide the correct username and password in the request body, we will get the response, **user is valid**; otherwise, the response will be **invalid user**.

The screenshot displays a REST client interface with the following details:

- Method:** POST
- URL:** `https://azfntest.azurewebsites.net/api/HttpTriggerForLoginAPI?code=9AJ52oc1I`
- Params:** (empty)
- Send:** (button)
- Authorization:** (tab)
- Headers (1):** (tab)
- Body:** (selected tab, blue dot)
- Pre-request Script:** (tab)
- Tests:** (tab)
- Form Data:** (radio buttons for form-data, x-www-form-urlencoded, raw, binary)
- JSON (application/json):** (selected format, dropdown arrow)
- Request Body:**

```
1 {  
2   "username": "admin",  
3   "password": "@dm!n1"  
4 }
```
- Body:** (selected tab, orange underline)
- Cookies:** (tab)
- Headers (11):** (tab)
- Tests:** (tab)
- Status:** 200 OK
- Pretty:** (button)
- Raw:** (button)
- Preview:** (button)
- JSON:** (dropdown arrow)
- Response:**

```
1 "Useradmin is valid"
```

Bindings

Functions can output their results in several ways:

- Through available input and output bindings
- Event Bus/Service Bus
- Storage
- In the Azure Function, binding is used to bind other resources of Azure with our Azure Function.
- For example, you want to load an external configuration file from blob storage when your function starts. You need to bind the file with the Azure Function.
- Using binding, you do not need to hardcode anything in your code. You just need to bind the configuration file with the function and take the file from the blob storage when it starts.
- We can bind not only the input of the function but also the output of the function too.
- For example, after executing the function we need to store the result in the Azure Table Storage. For that, we need our function output bind with Azure Table Storage.
- A function can have multiple input and output bindings and bindings are optional.

Types of Input Bindings

There are four types of input bindings:

- **Blob storage:** Blob content is used as input to the Azure Function. For example, in the previous scenario where we wanted to create thumbnails for an image whenever a new image is uploaded to the blob storage. In this case, we will create a blob trigger with input bind and blob storage.
- **Storage tables:** Storage table content is used as input to the Azure Function. For example, instead of hardcoding configuration data in the Azure Function, store all of the configuration in the storage table and bind with the Azure Function. When your function runs, it takes all the input from the storage table.
- **SQL tables:** SQL table data can also be used as input for the Azure Function. For example, we want to check the quantity of a product at the end of every day. We have all the product details stored in the SQL table. We have the Azure Function, which is bound with the SQL table and triggers at the end of every day. Once the Azure Function runs, it takes the input from the SQL table and processes the data.
- **NoSQL DB:** No-SQL data like data, which is stored as a document, can also be used as input to the Azure Function. For example, the Azure Function reads JSON data which is stored in NoSQL DB and processes it.

Types of Output Bindings

There are nine types of output bindings:

- **HTTP (REST or Webhook):** The Azure Function can produce an output as HTTP (REST or Webhook). For example, the output of the Azure Function can be linked to Webhook or can be HTTP REST.
- **Blob storage:** The Azure Function can use blob storage for output binding. For example, we have the Azure Function, which processes the image and compresses the size of image. After the image is compressed, we want it to be stored in blob storage. For this, we need output binding for the Azure Function with blob storage.
- **Events:** An event can also be bound as output from the Azure Function.
- **Queues and topics:** Queues and topics can also be an output from the Azure Function.
- **Storage tables:** Storage tables can also be bound as output to the Azure Function. Storage tables can be used as input or output to the Azure Function. For example, the Azure Function can take input from the storage table, process the data, and after processing again, store data in the storage table.
- **SQL tables:** SQL table can be used as input or output to the Azure Function. The Azure Function can read or take input from the SQL table and after processing again, can store data to the SQL table.
- **NoSQL DB:** No-SQL tables can also be used as input or output to the Azure Function. The Azure Function can read data from NoSQL DB and process it. Once the data is processed, it can be stored in NoSQL DB.
- **Push notifications:** Push notifications can be used as output binding for the Azure Function. For example, we want to send a push notification with a confirmation message to all users who have registered for some event.

