Module 10:

Azure App Service - Azure Functions (Serverless)

What is Azure Functions?

Azure Functions

- Azure Functions is a solution for easily running small pieces of code, or "functions," in the cloud.

- write just the code you need for the problem at hand, without worrying about a whole application or the infrastructure to run it.

- Functions can make development even more productive

What can Azure Functions do?

- Run code based on HTTP requests
- Schedule code to run at predefined times
- Process new and modified:
  - Azure Cosmos DB documents
  - Azure Storage blobs
  - Azure Queue storage messages
- Respond to Azure Event Grid events by using subscriptions and filters
- Respond to high volumes of Azure Event Hubs events
- Respond to Azure Service Bus queue and topic messages

Azure Functions

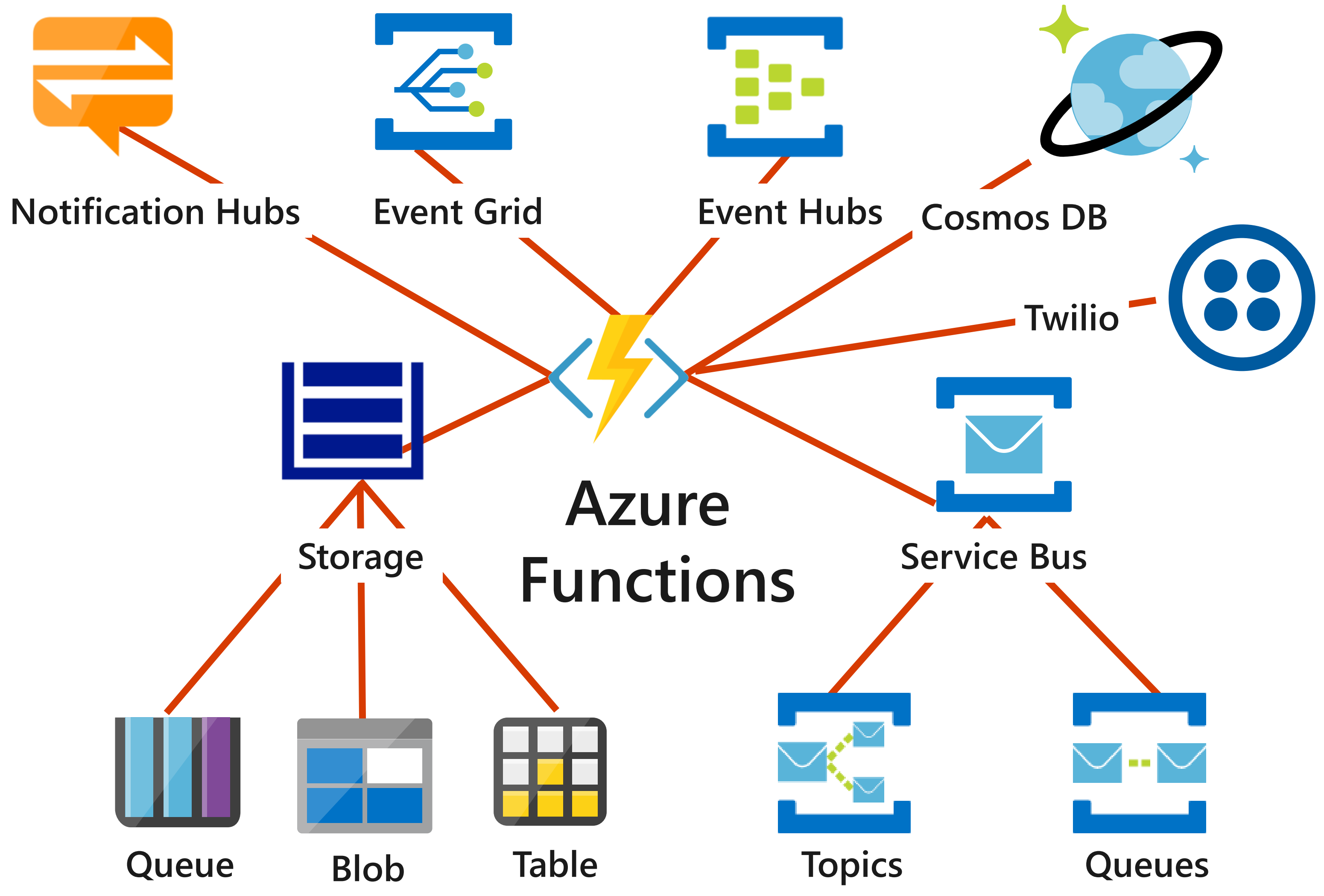
- Solution for running small pieces of code, or "functions," in the cloud:
  - Write only code that is relevant to business logic
  - Removes the necessity to write “plumbing” code to connect or host application components
- Build on open-source WebJobs code
- Supports a wide variety of programming languages, for instance:

C# JAVA PHP PY JS

- Even supports scripting languages, such as:

PS SH

Function integrations



Choosing the best code editor

Editor	Languages	Features	Platform
Azure Portal	C# / F# (script), Node.JS	Browser-based; host compiler; automatic dependencies mgmt	Any
Visual Studio Code	C# / F# (class library), Node.JS, Java	Lightweight; tons of extensions; local debugging	Any
Visual Studio	C# / F# (class library)	Remote debugging; cloud explorer; rich project types	Windows
IntelliJ	Java	Refactoring; Smart Completion; Maven integration	Any
Note taking file editor + Functions Core Tools	C# / F# (class library), Node.JS	Minimalism; traditional-style	Any

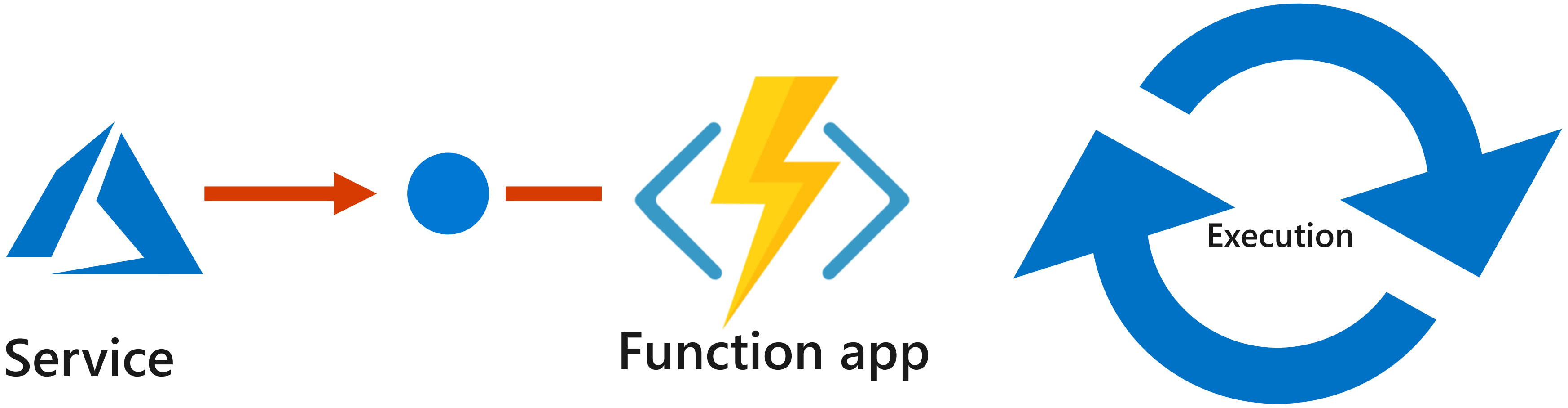
Simplify your code with triggers and bindings

a. **Triggers** – Event source that starts the function. One per function

b. **Bindings**
- **Input** – Data that is pulled **in** at the start of an execution. Can have multiple
- **Output** – Data that is pushed **out** after an execution. Can have multiple

Popular Triggers and Bindings
HTTP
Storage Queues
Cosmos DB
Event Hubs
Blob Storage
Service Bus Queues/Topics
Event Grid
Microsoft Graph
And more…

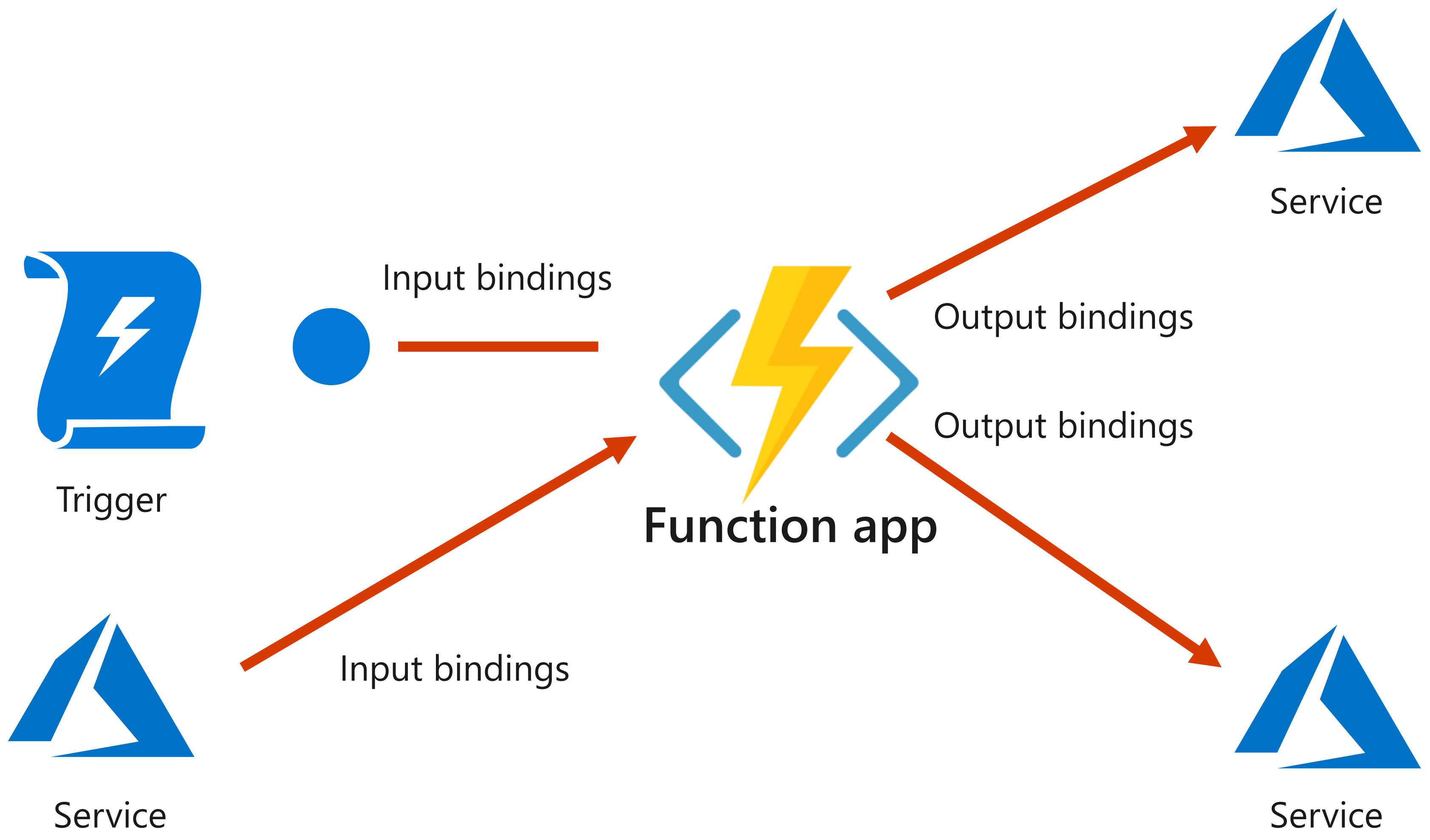
Triggers



Trigger types

- Triggers based on Azure services:
  - Cosmos DB
  - Blob and queues
  - Service Bus
  - Event Hub
- Triggers based on common scenarios:
  - HTTP request
  - Scheduled timer
- Triggers based on third-party services:
  - GitHub
- And more…

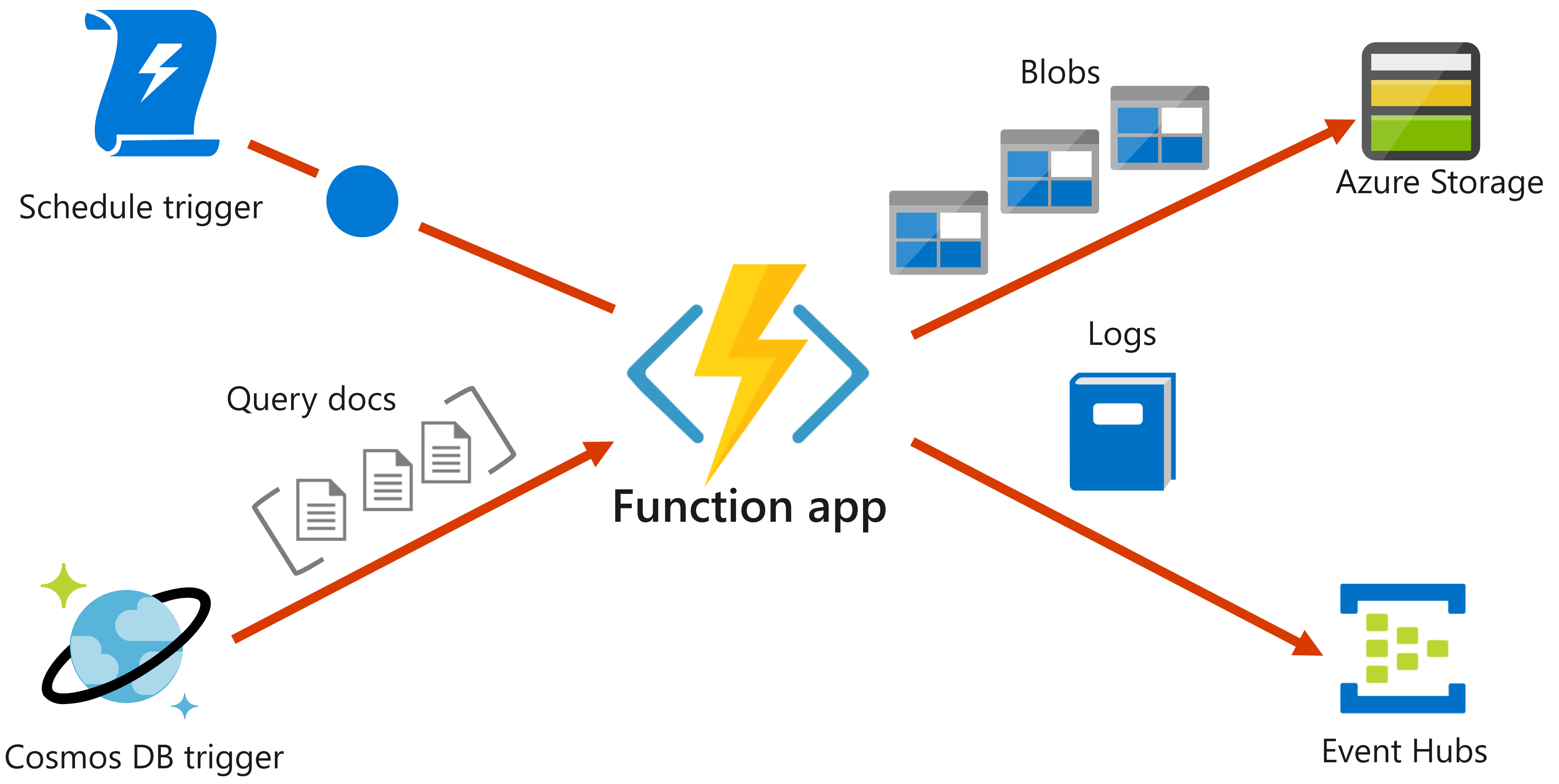
Input and Output Bindings



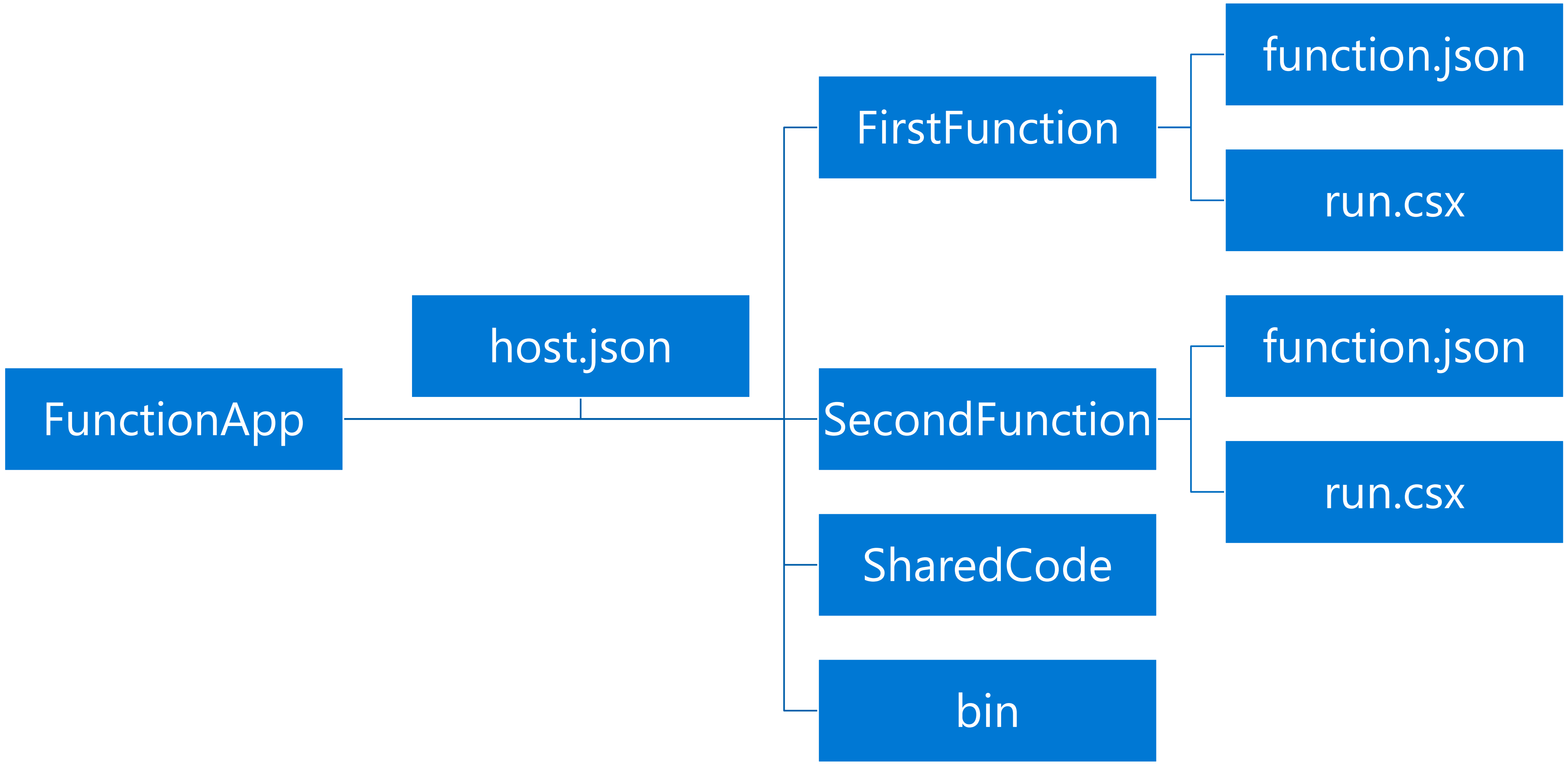
Bindings

- Declarative way to connect to data from your code:
  - Connect to services without writing plumbing code
  - Service credentials are not stored in code
  - Bindings are optional
- Function can have multiple input and output bindings
- Output bindings can send data to Azure services such as:
  - Storage
  - Azure Cosmos DB
  - Service Bus

Trigger and Bindings example



Function folder structure



Function Hosting Plan

App Service Plan

Select amount of resources (App Service Plan size)

Pre-determined pricing

You determine scaling (metric & threshold)

“Traditional” App Service Plan

Consumption Plan

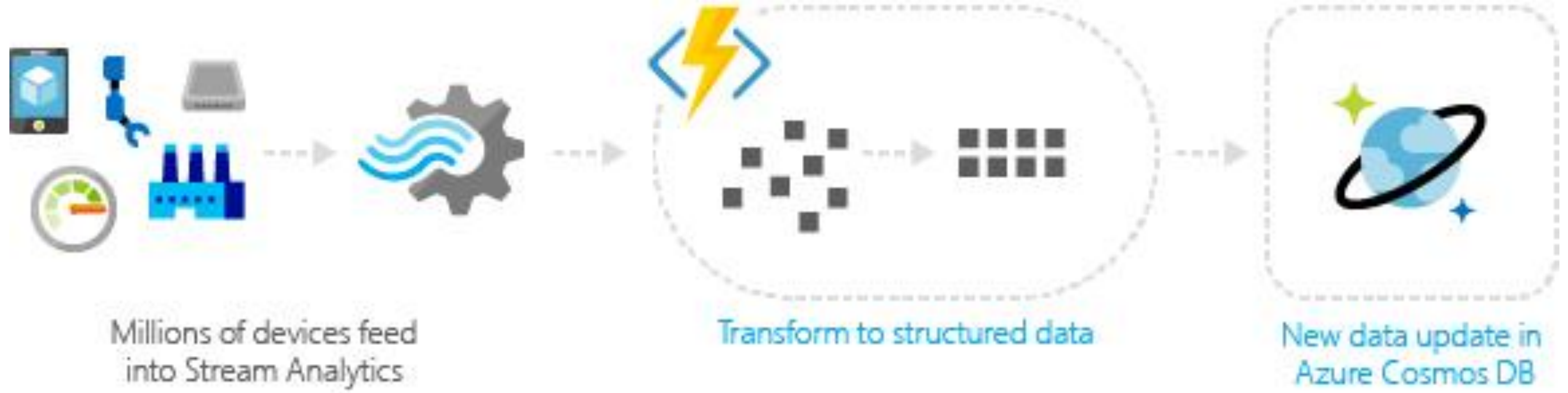
Pay for what you use

Platform determines scaling

“Dynamic” Plan - Serverless

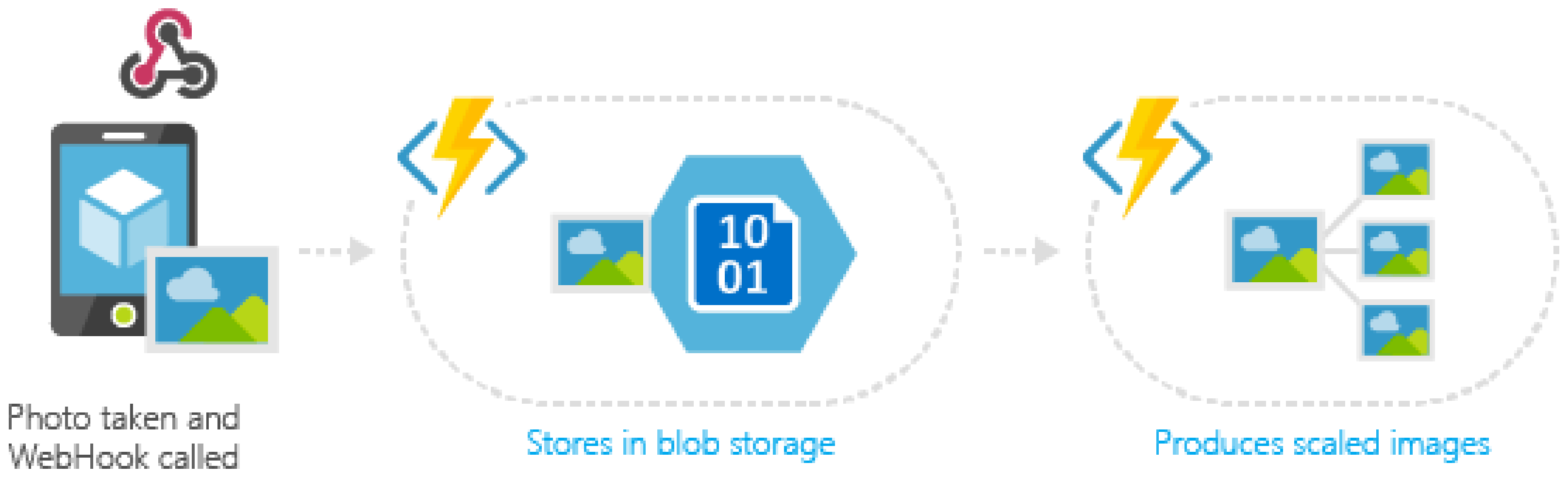
Examples of serverless applications

- IoT back end

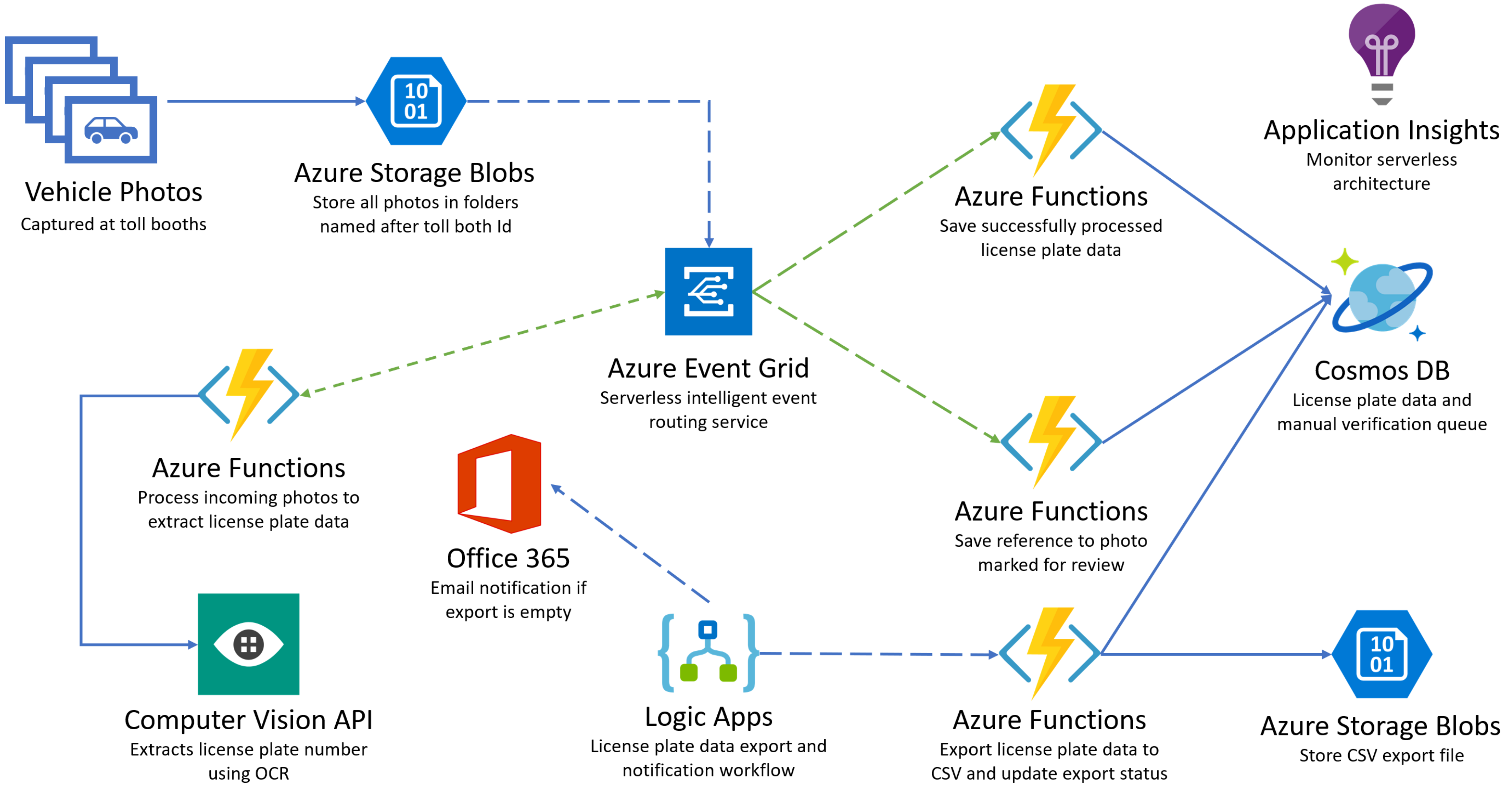

Millions of devices feed into Stream Analytics
Transform to structured data
New data update in Azure Cosmos DB

Examples of serverless applications

- Mobile back end



Preferred solution


Vehicle Photos
Captured at toll booths
Azure Storage Blobs
Store all photos in folders named after toll both Id
Application Insights
Monitor serverless architecture
Azure Functions
Save successfully processed license plate data
Azure Event Grid
Serverless intelligent event routing service
Cosmos DB
License plate data and manual verification queue
Azure Functions
Process incoming photos to extract license plate data
Office 365
Email notification if export is empty
Azure Functions
Save reference to photo marked for review
Computer Vision API
Extracts license plate number using OCR
Logic Apps
License plate data export and notification workflow
Azure Functions
Export license plate data to CSV and update export status
Azure Storage Blobs
Store CSV export file

Azure Functions in Visual Studio Code

- Use the Azure Functions extension for Visual Studio Code to:
  - Build and run functions locally
  - Publish functions to Azure
  - Build C# pre-compiled class libraries
  - Build C# scripts by adjusting the extension settings
- Use the many built-in features and extensions for Visual Studio Code to make development easier

Azure Functions in Visual Studio

- Visual Studio project type:
  - Develop, test, and deploy C# functions to Azure
  - Requires an **Azure development** workload installation
- Use WebJobs attributes to configure functions in C#
- Precompile C# functions:
  - Better cold-start performance

Function code

```
using System;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Host;

namespace FunctionApp1
{
    public static class Function1
    {
        [FunctionName("QueueTriggerCSharp")]
        public static void Run([QueueTrigger("myqueue-items", Connection = 
"QueueStorage")]string myQueueItem, TraceWriter log)
        {
            log.Info($"C# Queue trigger function processed: {myQueueItem}");
        }
    }
}
```

Binding configuration

```
{
    "bindings": [
        {
            "name": "order",
            "type": "queueTrigger",
            "direction": "in",
            "queueName": "myqueue-items",
            "connection": "MY_STORAGE_ACCT_APP_SETTING"
        },
        {
            "name": "$return",
            "type": "table",
            "direction": "out",
            "tableName": "outTable",
            "connection": "MY_TABLE_STORAGE_ACCT_APP_SETTING"
        }
    ]
}
```

Name of input parameter

Binding-based code

```
#r "Newtonsoft.Json"

using Microsoft.Extensions.Logging;
using Newtonsoft.Json.Linq;

public static Person Run(JObject order, ILogger log)
{
    return new Person() {
        PartitionKey = "Orders",
        RowKey = Guid.NewGuid().ToString(),
        Name = order["Name"].ToString(),
        MobileNumber = order["MobileNumber"].ToString()
    };
}
```

Name of input parameter

Lab