Serverless Azure

Michał Jankowski

about me

Michał Jankowski

software developer / architect / team leader / traveller / photographer

Nokia Siemens Networks

tieto

Objectivity
Bespoke Software Specialists

aim.

Learn how we can use Serverless in Azure in our solutions to improve our productiveness.

way
of working

we should have fun

A bit of theory, then a lot of demos and practice.

I would encourage you to work together and exchange your knowledge.

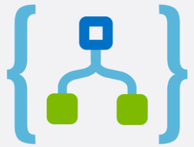
Great things in business are never done by one person. They're done by a team of people.

Steve Jobs, cofounder of Apple

we will
be working with

Azure Functions

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



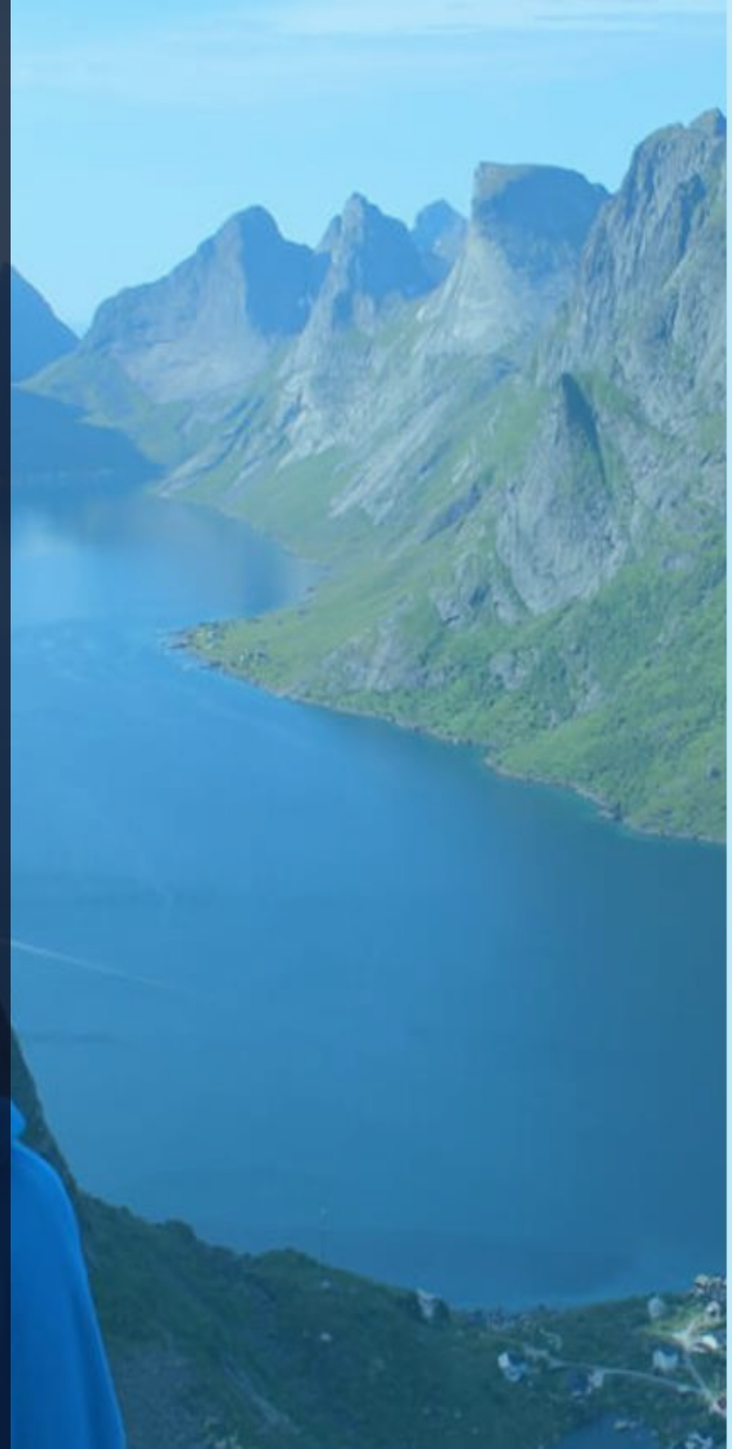
Logic Apps

Provide a way to simplify and implement scalable integrations and workflows in the cloud. It provides a visual designer to model and automate your process as a series of steps known as a workflow.

todo.

We will provide Azure Function backend for registration form that will:

- validate data
- store customer data
- send thank you message
- send SMS to us each time new customer will register



Get The Best Ticket Today!

You can add unlimited fields directly from HTML

Your name

Your surname

Your county

Your e-mail

Your birth year

Send Information

presentation
agenda

theory

Serverless in Azure environment

azure functions

Main part of presentation. You will learn how to develop them correctly and in effective way.

presentation
agenda

logic apps

Later you will see that you can achieve the same effect with less code.

flow / event grid

Next I will present you the last two items from Azure Serverless ecosystem. I would treat them as an extension.

theory.

types of approaches

01 Do I really know how hardware works? What hardware specification should be delivered?

Am I a good system administrator? When I should update servers' OS? **02**

03 Finally I can focus on features development.

On-Premises	IaaS	PaaS	Serverless
Applications	Applications	Applications	Applications
Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware
O/S	O/S	O/S	O/S
Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

Managed by us

Managed by vendor

serverless characteristics

server abstraction

There is no server managing tasks.

event driven

Function does not work when there is no event triggering it. It can also instantly scale up.

microbilling

Pay only when there are events. But think about DDOS on your wallet.

productivity

Reduce tasks related to infrastructure. You can focus on development activities.

focus on features

And then you are able to focus on business logic of your app.

faster time to market

All items mentioned together allow you to reduce time to market.

serverless
in Azure

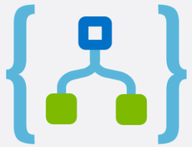
Azure Functions

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

Event Grid

A single service for managing routing of all events from any source to any destination. Designed for high availability, consistent performance and dynamic scale. Event Grid lets you focus on your app logic rather than infrastructure.

serverless
in Azure



Logic Apps

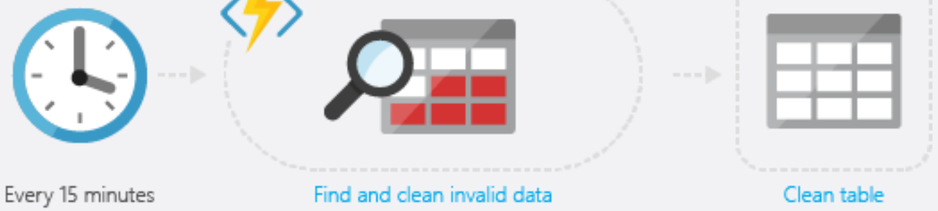
Provide a way to simplify and implement scalable integrations and workflows in the cloud. It provides a visual designer to model and automate your process as a series of steps known as a workflow.

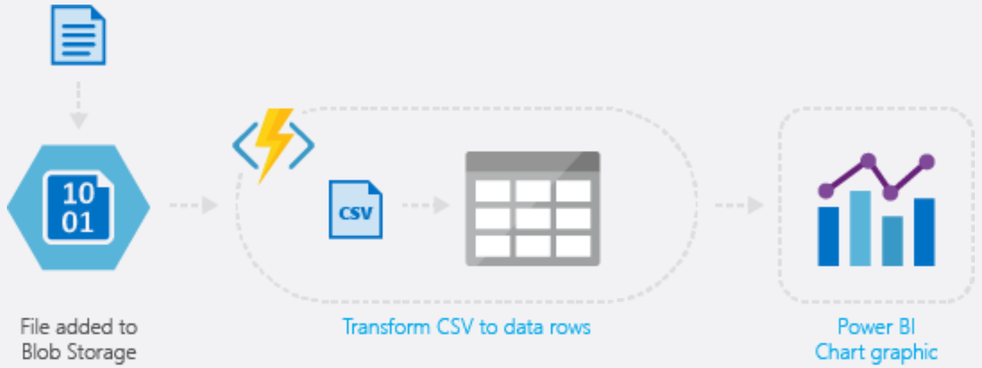
Flow

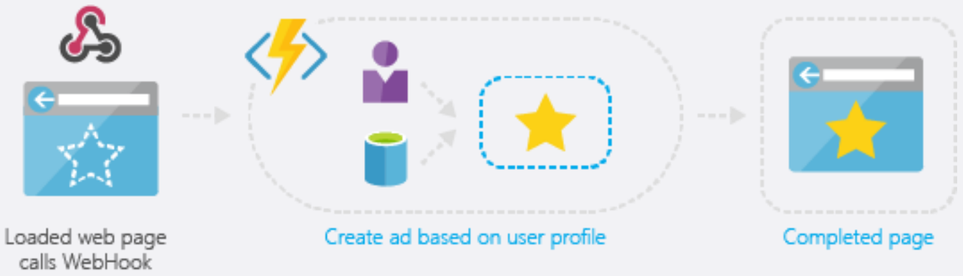
Is a service that allows you to create automated workflows between your favourite applications and services to synchronize files, get notifications, collect data, and more.

azure
functions.

common scenarios







key features

choice of languages

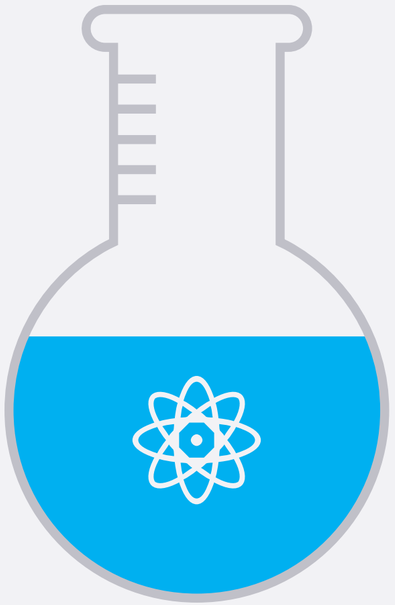
C#, F#, Node.js, Python, PHP, batch, bash, or any executable

pay-per-use pricing model

Consumption plan vs *App Service* plan

bring your own dependencies

Functions supports *NuGet* and *NPM*



open-source

The Functions runtime is open-source and available on GitHub

integrated security

Protect HTTP-triggered functions with OAuth providers such as AAD, Facebook, Google, Twitter, and Microsoft Account

simplified integration

Defined services can trigger your function or can serve as input and output for your code.

flexible development

Set up continuous integration and deploy your code through GitHub, Visual Studio Team Services, and other supported development tools.

time of starting

We will need additional time for our function start. Normal application are always ready for response.

think about state

Functions are stateless. You should save somewhere state if you need.

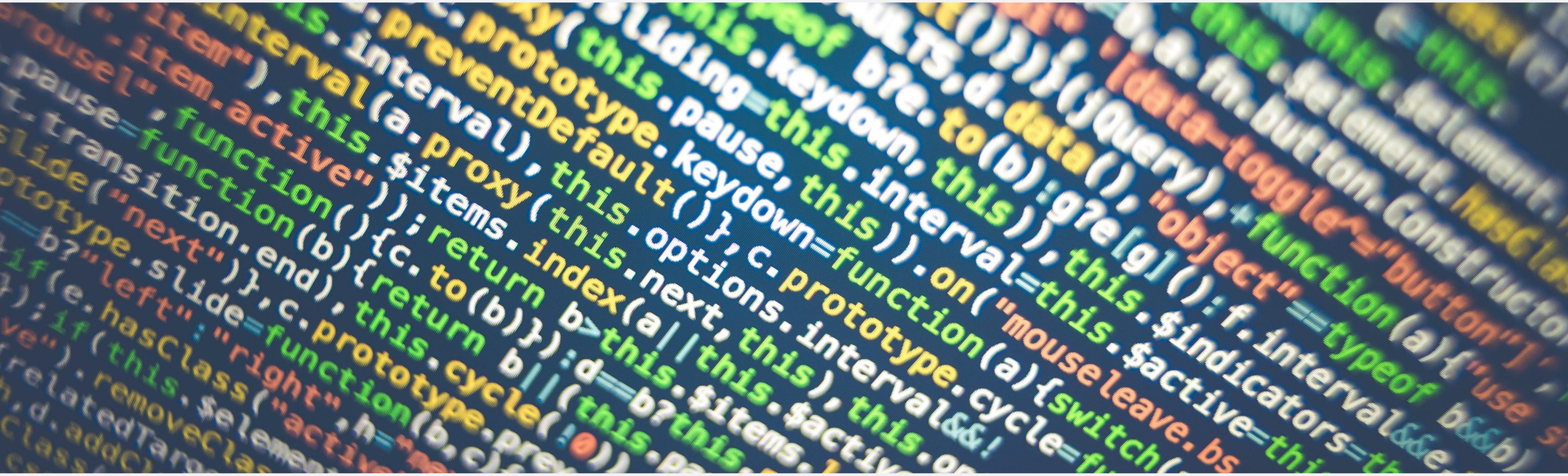
local environment

It is not so easy to start your function locally and it can be run only under Windows.

vendor locking

It will be hard to change your vendor in latter stage of your application life.

maybe
we should start coding



demo 1.

is your
environment ready

applications needed

- Azure account
- Visual Studio Code
- Visual Studio 2017 version 15.3.x with Azure Functions Tools for Visual Studio extension installed
- azure-functions-core-tools
- Azure Storage Explorer
- Postman

links to installers

https://goo.gl/Pr3NBK

this is time
for you

1. Create your first function in Azure Portal:
    a. Check compilation and errors
    b. Execute your function
2. Create function that will execute every 30 seconds and log time
    a. Read entries from logs
    b. Add possibility to change time format without changing the function code
3. Create simple calculator API that will support addition, subtraction, multiplication and division
    a. Check Azure Functions Proxies
4. Create Swagger documentation for calculator API

https://goo.gl/rZx11C

triggers & bindings.

triggers & bindings

type	service	trigger	Input	output
Schedule	Azure Functions	+	-	-
HTTP (REST or webhook)	Azure Functions	+	-	+
Blob Storage	Azure Storage	+	+	+
Events	Azure Event Hubs	+	-	+
Queues	Azure Storage	+	-	+
Queues and topics	Azure Service Bus	+	-	+
Storage tables	Azure Storage	-	+	+
SQL tables	Azure Mobile Apps	-	-	+
No-SQL DB	Azure DocumentDB	-	-	+
Push Notifications	Azure Notification Hubs	-	-	+
Twilio SMS Text	Twilio	-	-	+
SendGrid email	SendGrid	-	-	+

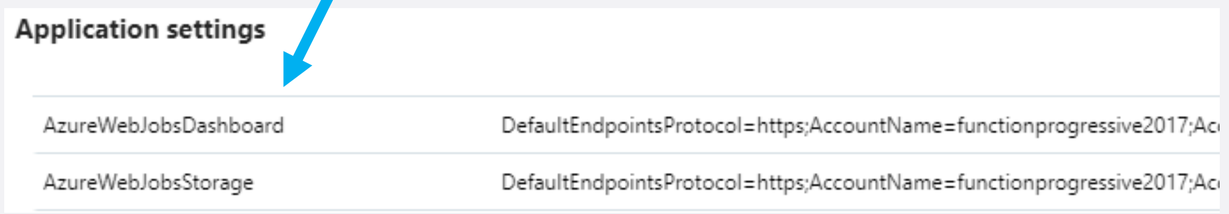
classic approach

run.csx:

```
public static void Run(string myQueueItem, TraceWriter log)
{
  log.Info($"C# Queue trigger function processed: {myQueueItem}");
}
```

function.json:

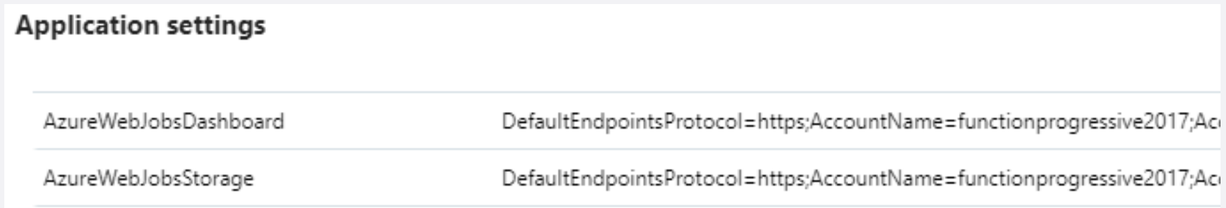
```
{
  "bindings": [
    {
      "name": "myQueueItem",
      "type": "queueTrigger",
      "direction": "in",
      "queueName": "myqueue-items",
      "connection": "AzureWebJobsDashboard"
    }
  ],
  "disabled": false
}
```



01

new approach

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



02

tools.

there are
some options

VS 2017 v15.3.x

Native support for Azure Functions. Possibility of pre-compilation, deployment and debugging

VS Code

Light code editor. You need to use other tools to be able to do everything that VS 2017 can do.

azure-function-core-tools

CLI that helps you working with Azure Functions locally

cold start.

After some time without any action your function will be turned off / terminated.

Then next request will take longer – function will need to recover environment, reload dependencies and compile it again.

Solution for that is **pre-compilation** of your function.

pre-compilation.

1. We can use full features on Visual Studio, including IntelliSense.
2. We can easily write unit test codes.
3. We can easily attach function codes to existing CI/CD pipelines.
4. We can easily migrate the existing codebase with barely modifying them.
5. We don’t need project.json for NuGet package management.
6. We can reduce the total amount of cold start time by removing compiling on-the-fly when requests hit to the Functions.

demo

2.

this is time for you

We will provide Azure Function backend for registration form that will:

1. Validate data
2. Store customer data
3. Send thank you message
4. Send SMS to you each time new customer will register

https://goo.gl/q2gChq

good
practices.

good practices

avoid long running functions

Large, long-running functions can cause unexpected timeout issues. A function can become large due to many dependencies. Importing dependencies can also cause increased load times that result in unexpected timeouts. Whenever possible, refactor large functions into smaller function sets that work together and return responses fast.

cross function communication

When integrating multiple functions, it is generally a best practice to use storage queues for cross function communication. The main reason is that storage queues are cheaper and much easier to provision. Service Bus topics are useful if you require message filtering before processing. Event hubs are useful to support high volume communications.

write function to be stateless

Functions should be stateless and idempotent if possible. Associate any required state information with your data. For example, an order being processed would likely have an associated state member. A function could process an order based on that state while the function itself remains stateless.

good practices

write defensive functions

Assume your function could encounter an exception at any time. Design your functions with the ability to continue from a previous fail point during the next execution. Depending on how complex your system is, you may have: involved downstream services behaving badly, networking outages, or quota limits reached, etc. All of these can affect your function at any time. You need to design your functions to be prepared for it.

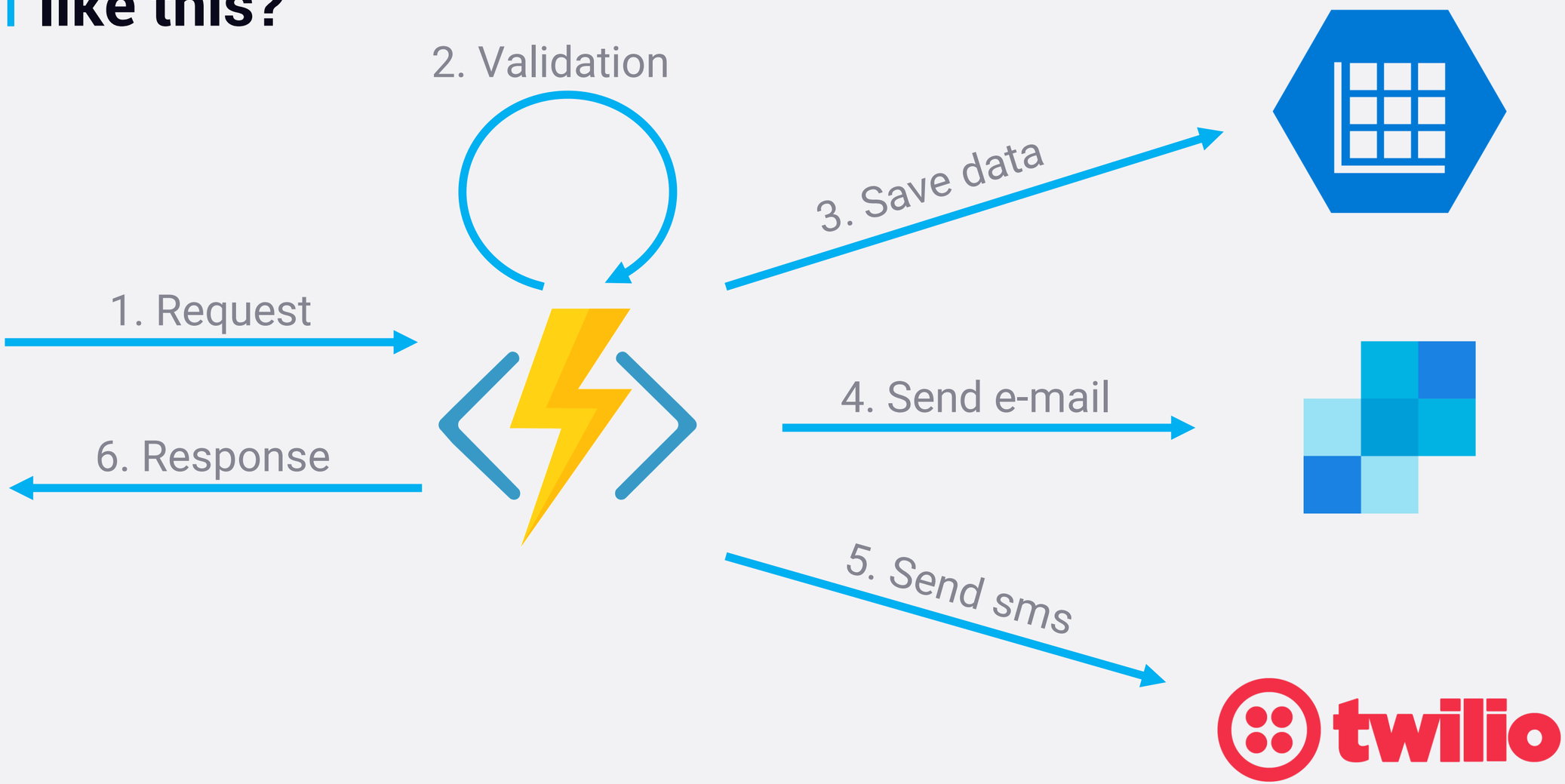
don't mix test and production code in the same function app

Functions within a function app share resources. For example, memory is shared. If you're using a function app in production, don't add test-related functions and resources to it. It can cause unexpected overhead during production code execution. Be careful what you load in your production function apps.

use async code but avoid blocking calls

Asynchronous programming is a recommended best practice. However, always avoid referencing the *Result* property or calling *Wait* method on a *Task* instance. This approach can lead to thread exhaustion.

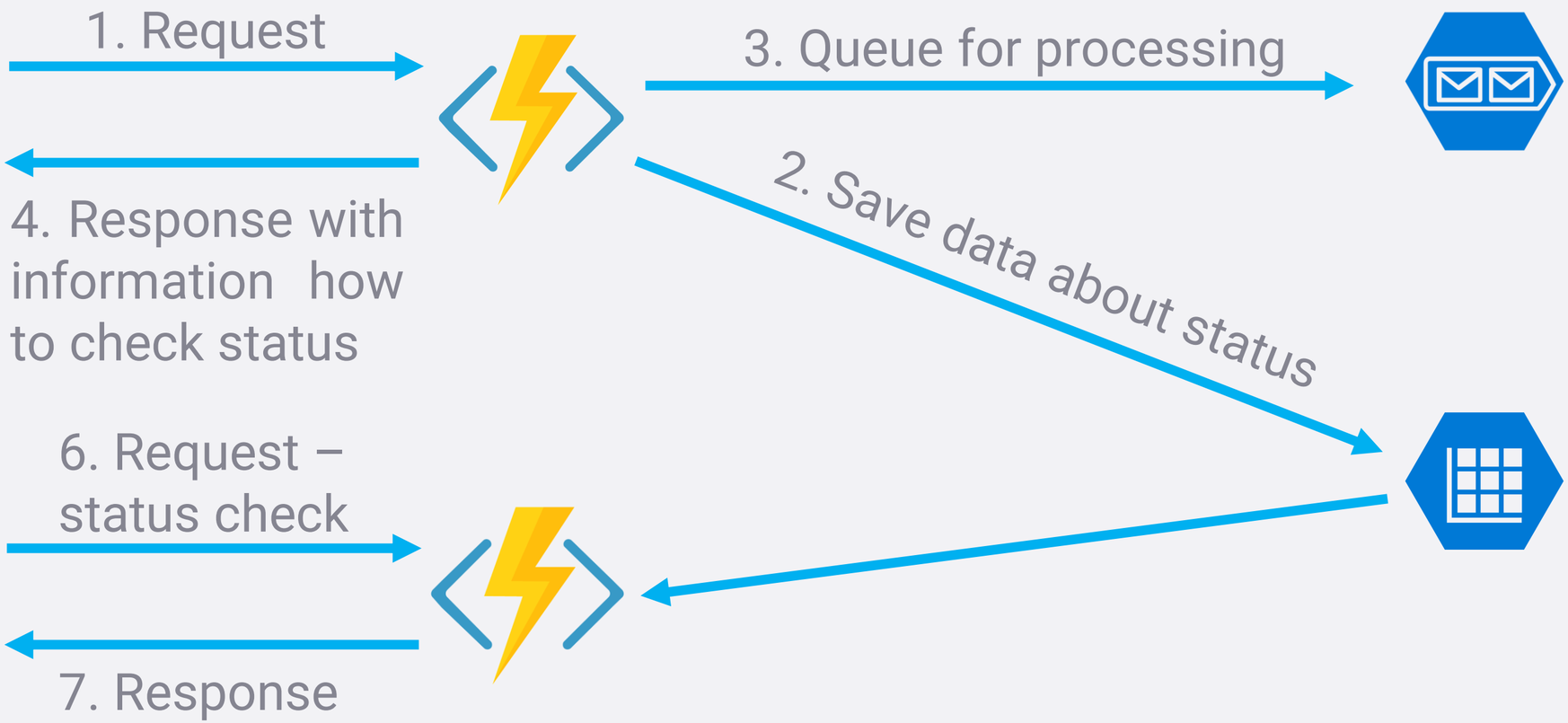
is your design like this?



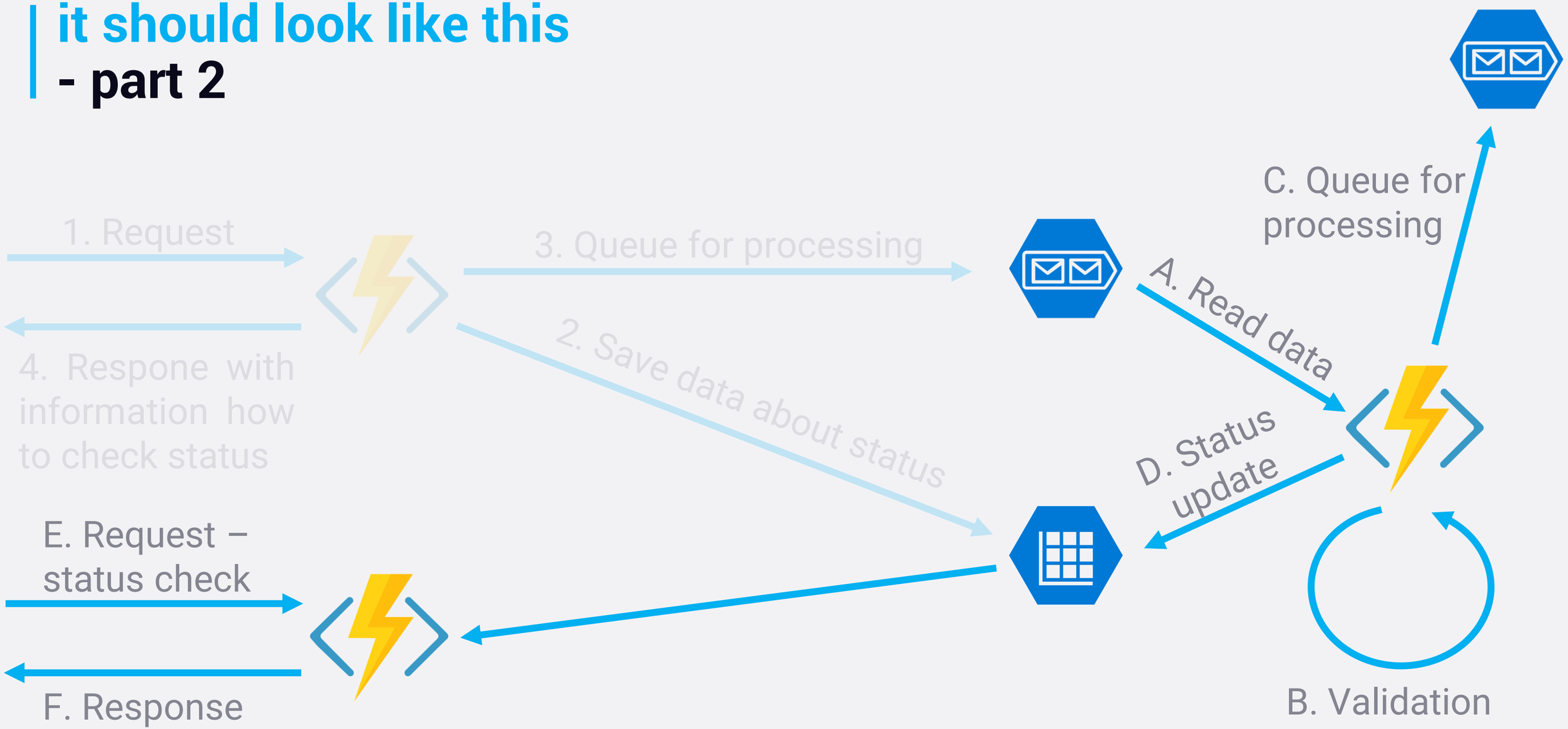
demo

3.

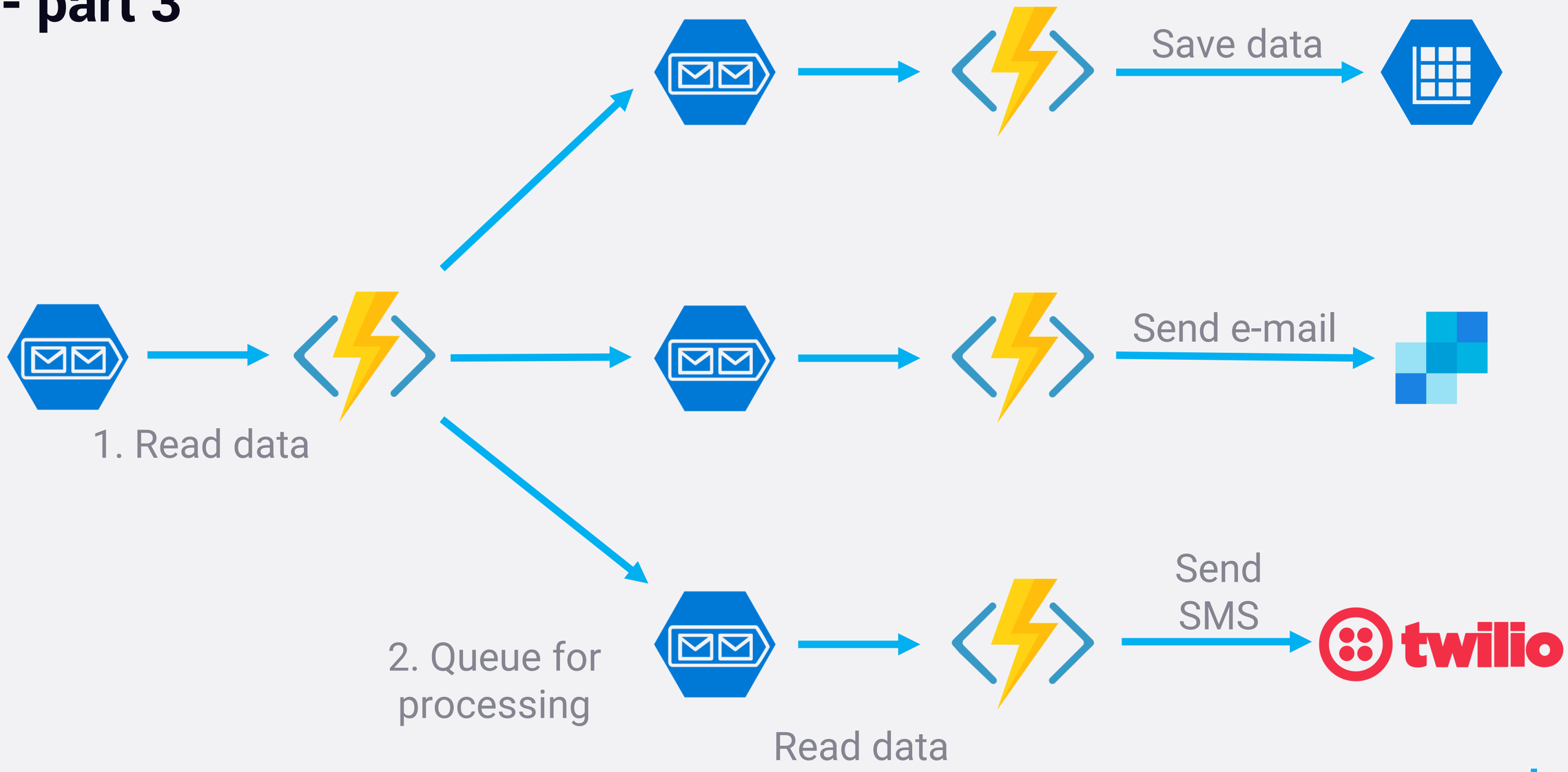
it should look like this
- part 1



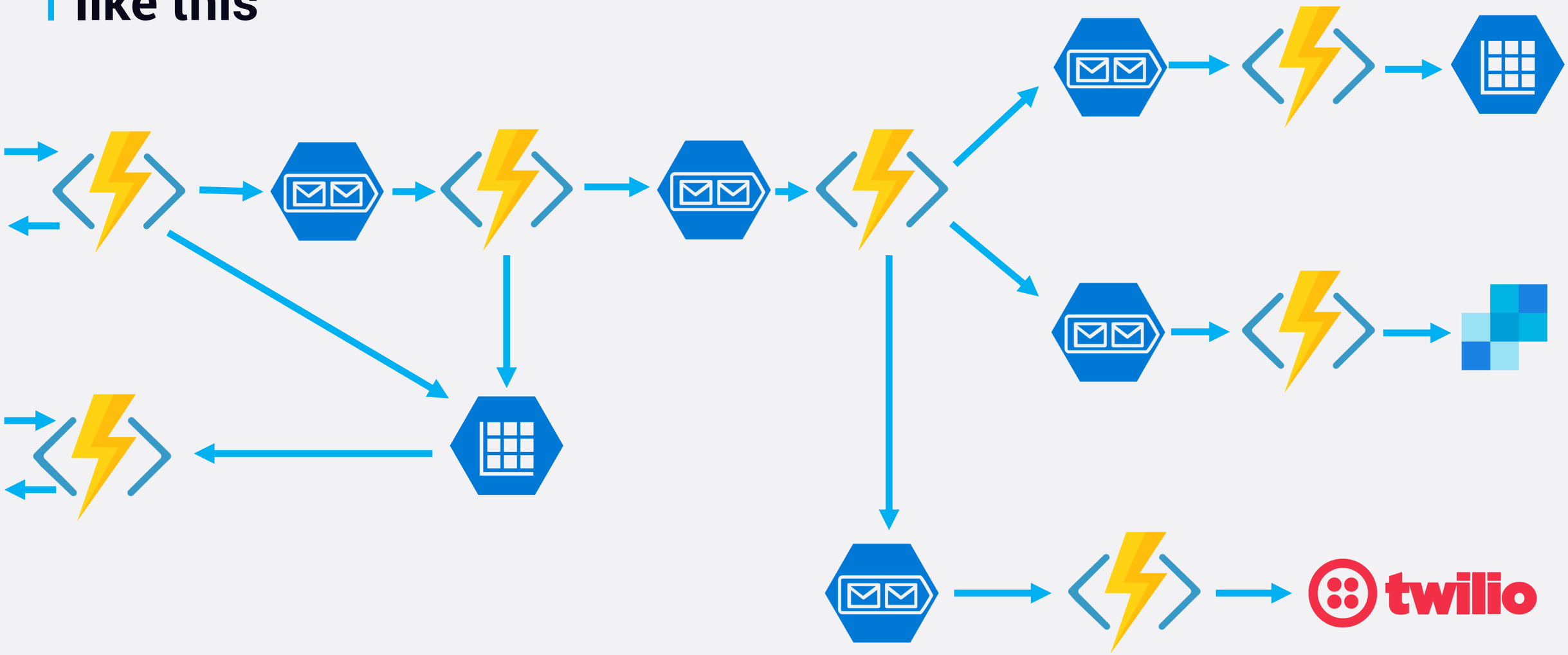
it should look like this - part 2



it should look like this
- part 3



it should look like this



demo
4.

azure functions
deployment

by simple file copy

- Integration with different repositories (Bitbucket, Dropbox, local Git, GitHub, OneDrive, …)
- Your deployment source must have correct source configuration and structure – the code for all of the functions in a given function app lives in a root folder that contains a host configuration file and one or more subfolders, each of them contain the code for a separate function
- Your functions can be affected by cold start
- Sometimes Azure Portal does not detect changes in files after deployment
- Very cheap for configuration
- Good for POC – solutions
- Not able to detect compilation errors

by CI (VSTS, Jenkins, TeamCity)

- Professional approach for larger projects
- You will be able to also configure environment provisioning
- Your code and functions will be compiled before deployment
- You will be able to check code quality, run unit tests and detects compilation errors
- You can have your own code structure
- Higher cost of configuration
- More options for different customisations and integrations

logic
apps.

Logic Apps provide a way to simplify and implement scalable integrations and workflows in the cloud. It provides a visual designer to model and automate your process as a series of steps known as a workflow. There are many connectors across the cloud and on-premises to quickly integrate across services and protocols. A logic app begins with a trigger (like 'When an account is added to Dynamics CRM') and after firing can begin many combinations of actions, conversions, and condition logic.

connectors

Most powerful element of this solution. Right now you are able to connect to 166 *systems*.

Basically, connectors are web APIs that use REST for pluggable interfaces, Swagger metadata format for documentation, and JSON as their data exchange format.

actions
& tiggers

actions

Actions are changes directed by a user. For example, you would use an action to look up, write, update, or delete data in a SQL database. All actions directly map to operations defined in the Swagger.

triggers

Triggers can notify your app when specific events occur. For example, the FTP connector has the *OnUpdatedFile* trigger.
There are two types of trigger:

- Polling Triggers: These triggers call your service at a specified frequency to check for new data. When new data is available, it causes a new run of your workflow instance with the data as input.
- Push Triggers: These triggers listen for data on an endpoint, that is, they wait for an event to occur. The occurrence of this event causes a new run of your workflow instance.

sample connector
twitter

limitations

- Maximum number of connections per user: 2
- API call rate limit for POST operation: 12 per hour
- API call rate limit for other operations: 600 per hour
- Frequency of trigger polls: 60 seconds
- Maximum size of image upload: 5 MB
- Maximum size of video upload: 15 MB
- Maximum number of search results: 100
- Mentioning a @user while posting a tweet is not supported

actions

- Get followers
- Get following
- Get home timeline
- Get my followers
- Get my following
- Get user
- Get user timeline
- Post a tweet
- Search tweets

triggers

- When a new tweet is posted

demo
5.

this is time for you

Play with Logic Apps

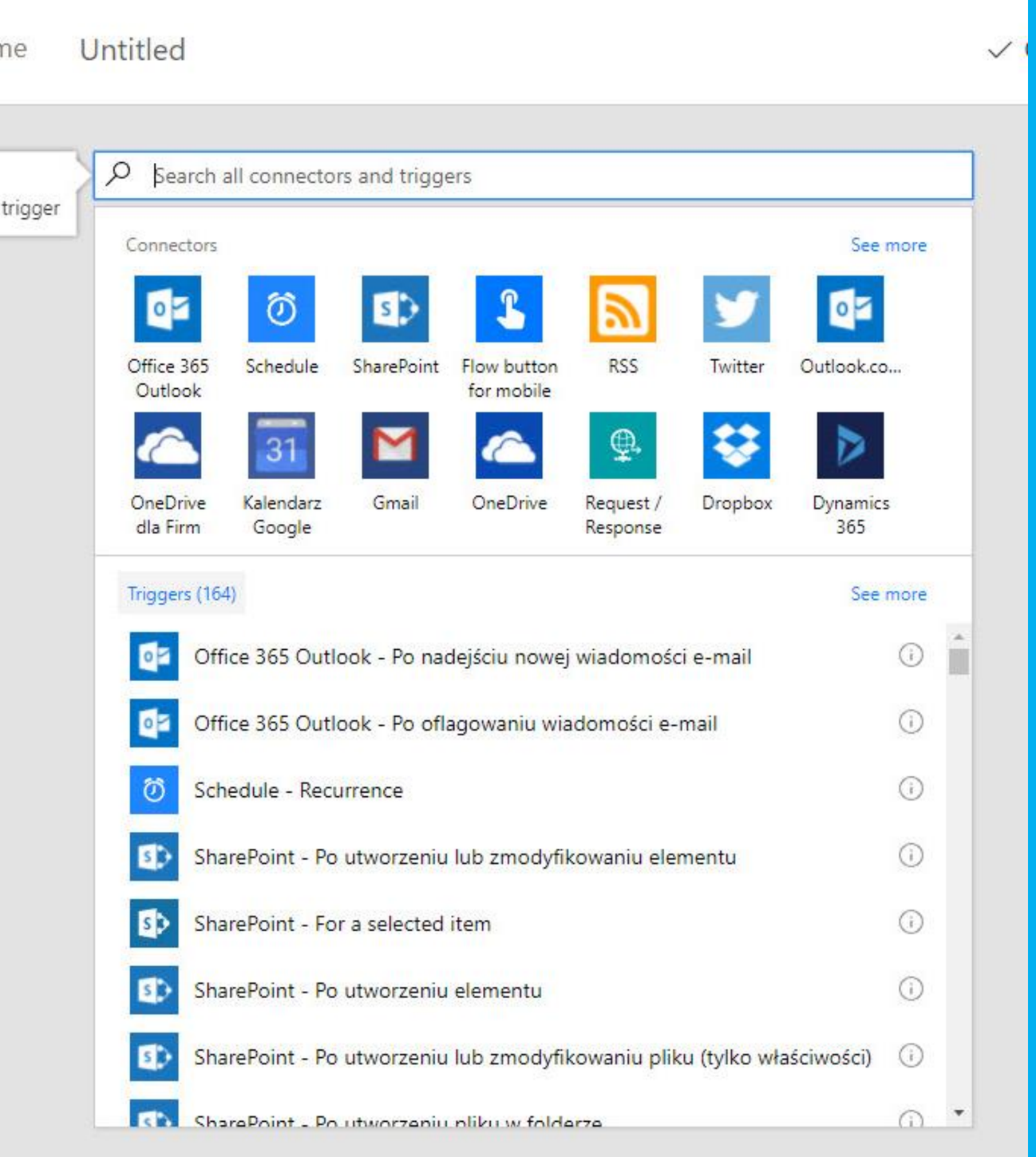
Try to implement form back-end with Logic Apps.

or

Please refactor your solution.

flow.

Create automated workflows between your favourite apps and services to get notifications, synchronize files, collect data, and more…

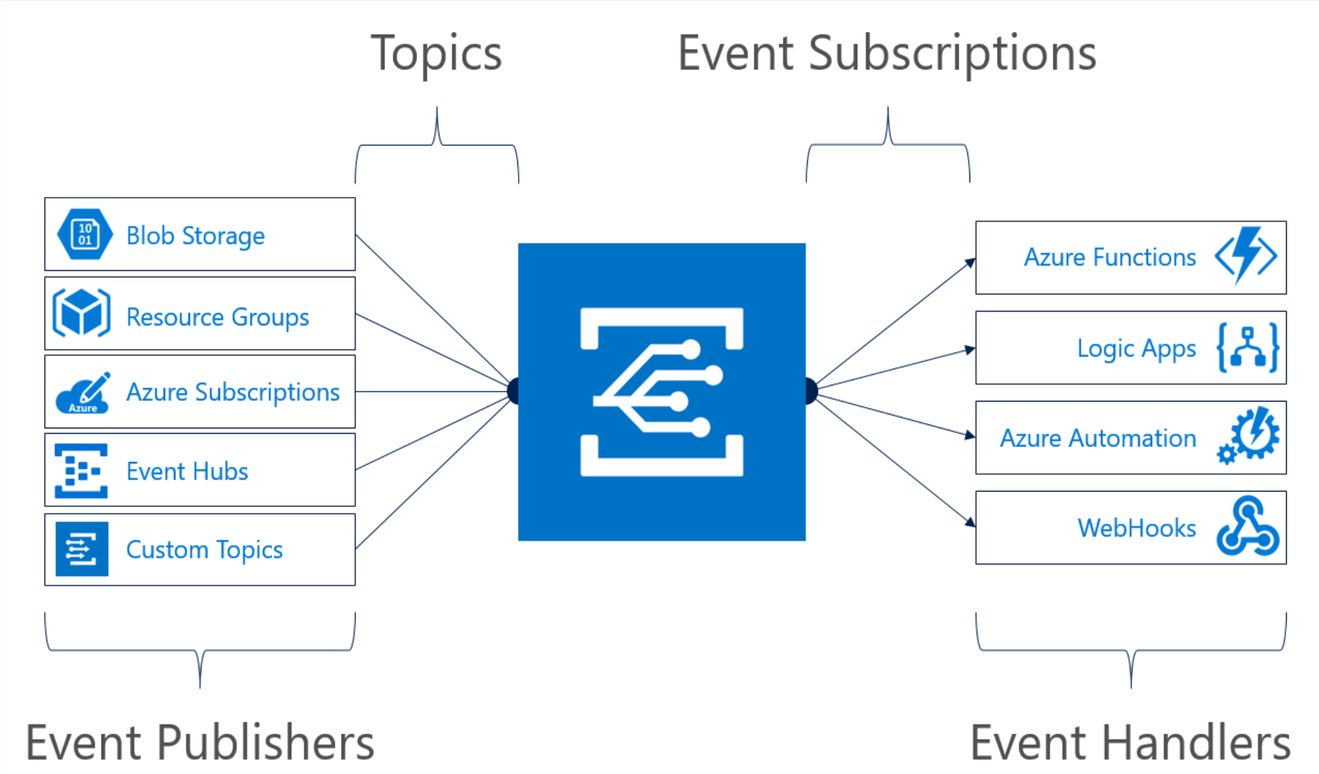


- For simple business optimization
- You do not need to have Azure subscription
- Build on top of Logic Apps

event
grid.

event grid

- a single service for managing routing of all events from any source to any destination
- advanced filtering - filter on event type or event publish path to ensure event handlers only receive relevant events
- reliability – utilize 24-hour retry with exponential backoff to ensure events are delivered and ensure that message will be delivered once and only once
- high throughput – build high-volume workloads on Event Grid with support for millions of events per second
- supports only subset of apps
- is not usable from the UI or CLI



demo
6.

summary.

We have learnt possibilities of Azure Serverless environment

You should know how to develop and deploy your solution to cloud

You can do it in effective way

do you have any questions?

www.jankowskimichal.pl

@JankowskiMichal

github.com/MichalJankowskii

sli.do #ProgNET | goo.gl/83iLsr

thank
you

www.jankowskimichal.pl

@JankowskiMichal

github.com/MichalJankowskii

sli.do #ProgNET | goo.gl/83iLsr