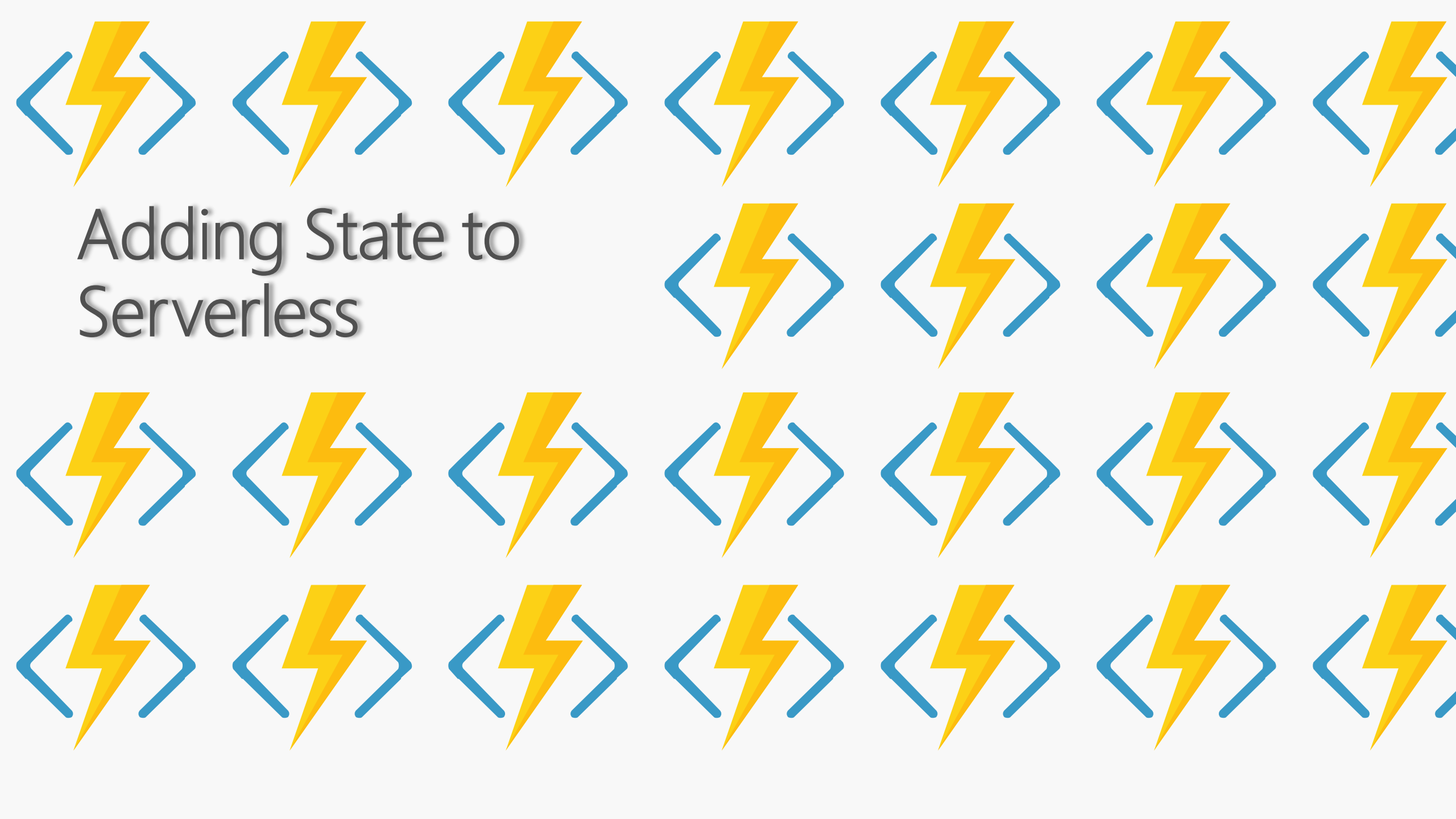
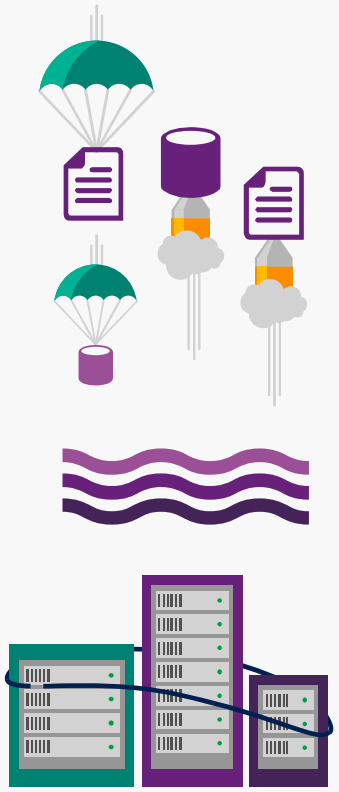
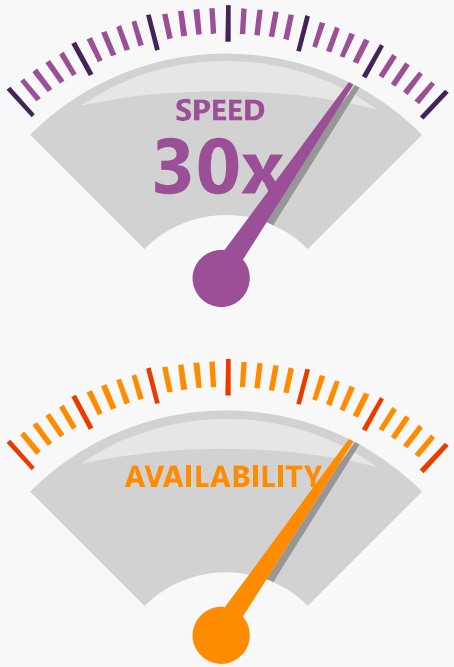


Adding State to Serverless

What is “serverless”



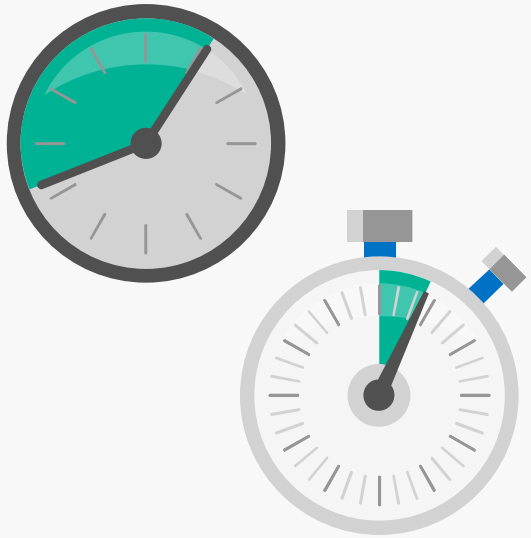
Abstraction of servers



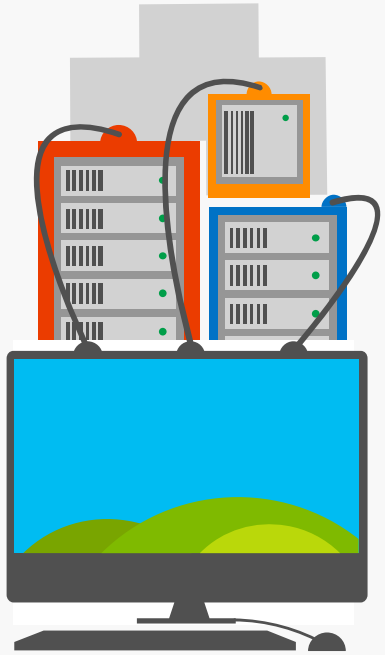
Event-driven scale

Sub-second billing

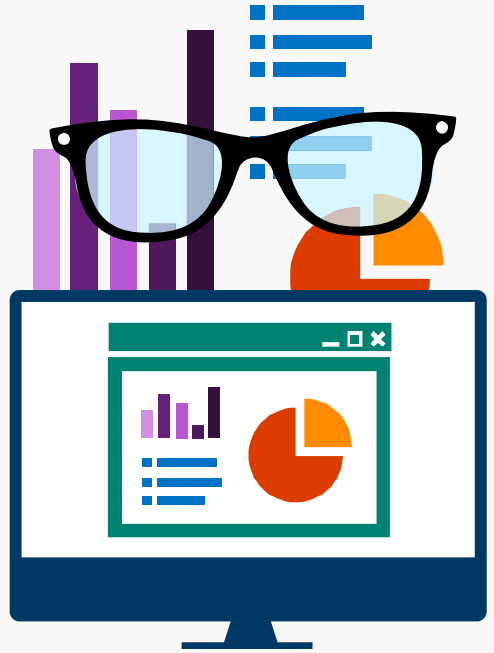
Benefits of “serverless”



Ship faster



Reduced dev ops



Focus on business logic

Microservice tools and approaches

Microservices in the wild

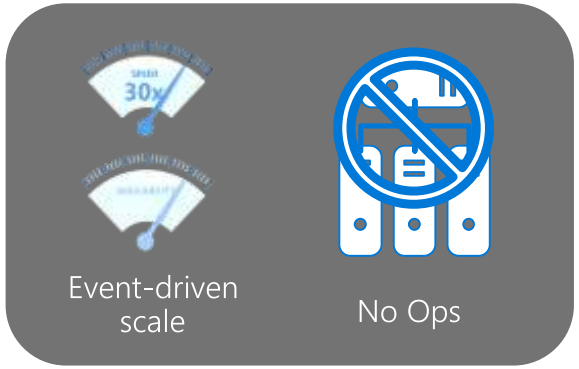
Implication: Build your own microservices platform

Benefits: Customizable, pick best of breed solutions

Azure Functions

Implication: Serverless microservices

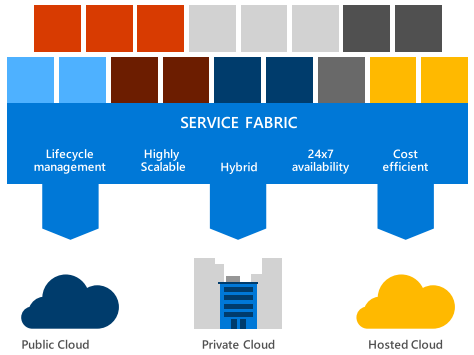
Benefits: Quick ramp up, sub second metering, zero ops



Azure Service Fabric

Implication: Prescriptive microservices platform

Benefits: Easy to build, deploy and manage microservices at scale



Azure Functions

Serverless

Event-driven scale

Reduced Dev Ops

Accelerate development

nodeJS

C#

Develop your way

Local development

Bind into services

Azure Service Bus

Azure Event Hub

Azure Storage

Dropbox

Sendgrid

AzureDocDb

OneDrive

Box

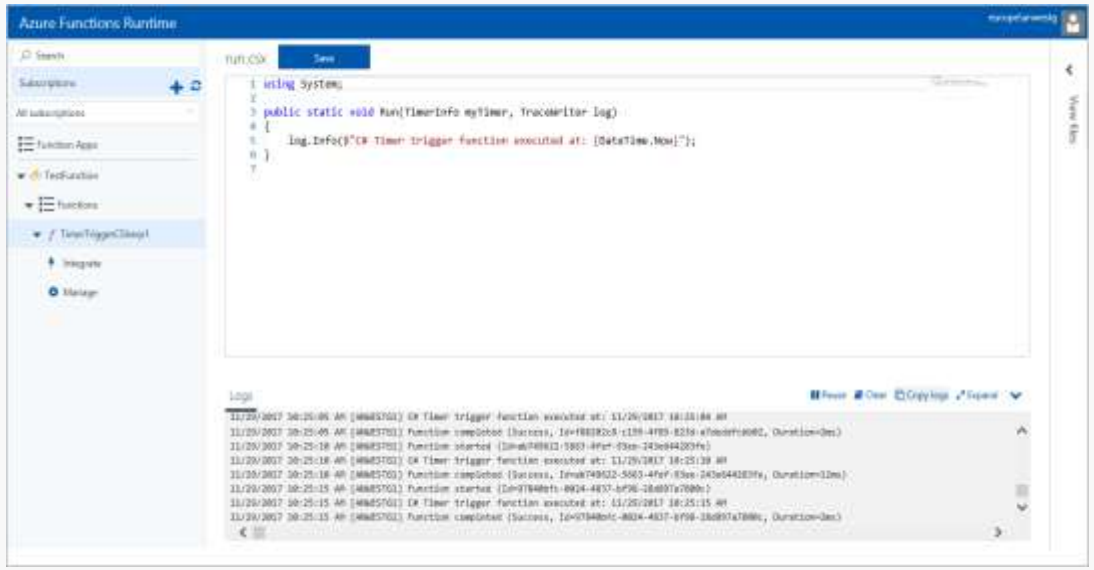
Twilio

Azure Functions architecture

Azure Functions is built around the WebJobs SDK runtime. The WebJobs SDK makes it easy to react to events and work with data in a consistent abstracted fashion.

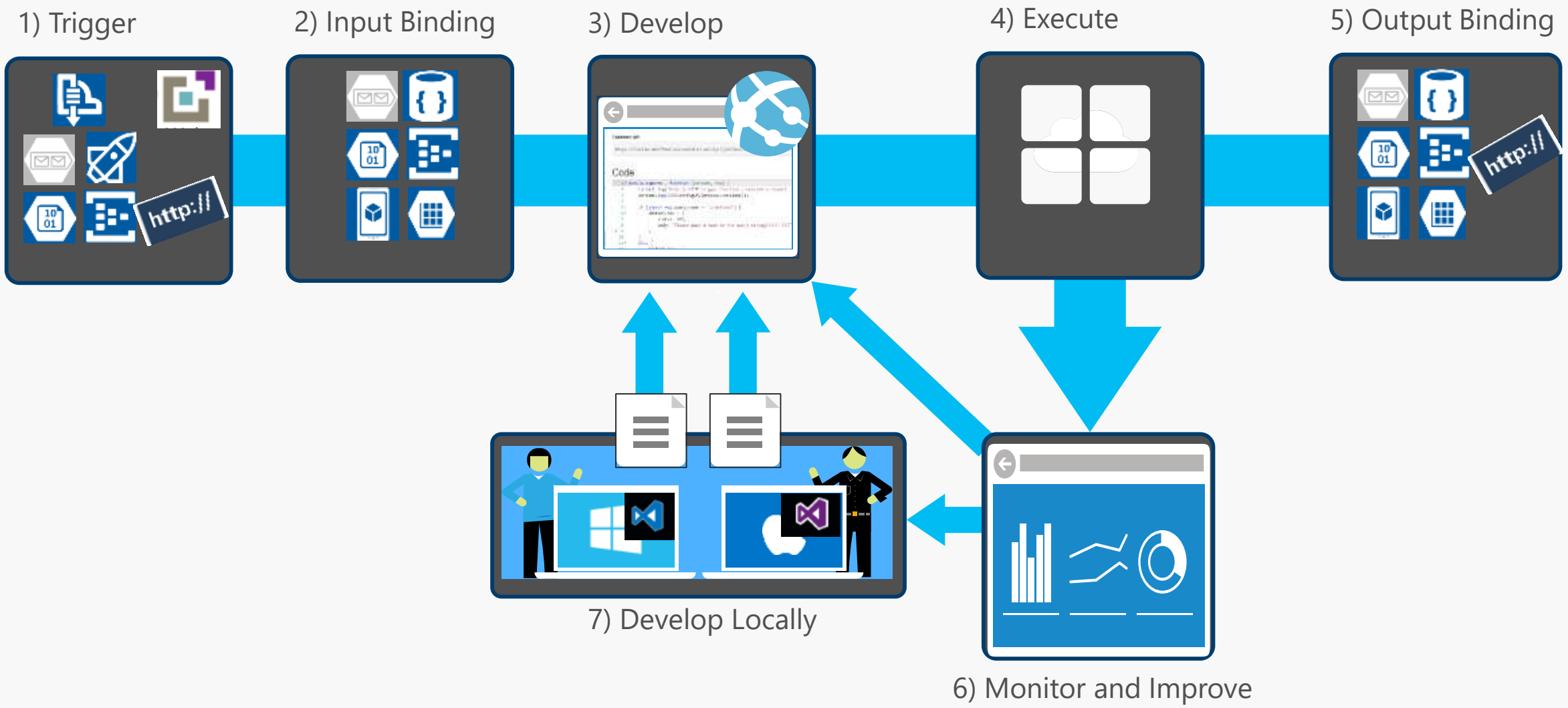
Setup Dev Environment

1. Azure Functions and WebJobs Tools
2. Azure Storage Emulator (For durable)
3. Azure Functions Core (CLI) tools (VS > F5) or npm
4. Azure Functions Runtime (Optional) - Gives ability to run functions on-prem



Platform and scaling

- App Service offers dedicated and dynamic tiers.
- Dedicated is the existing App Service plan tiers
  - Basic, Standard, Premium
  - Pay based on # of reserved VMs
  - You're responsible for scale
- Dynamic
  - Pay on number of executions
  - Platform responsible for scale


1) Trigger
2) Input Binding
3) Develop
4) Execute
5) Output Binding
http://
Code
http://
7) Develop Locally
6) Monitor and Improve

Signs that a serverless pattern might be useful for a given scenario

1. Stateless → Scale → Now Stateful!
2. Not worth deploying a traditional backend
3. Workload is sporadic (very low & high scale)
4. Dev ops favored versus dedicated ops
5. Lots of different services involved that need “glue”

Triggers and Bindings

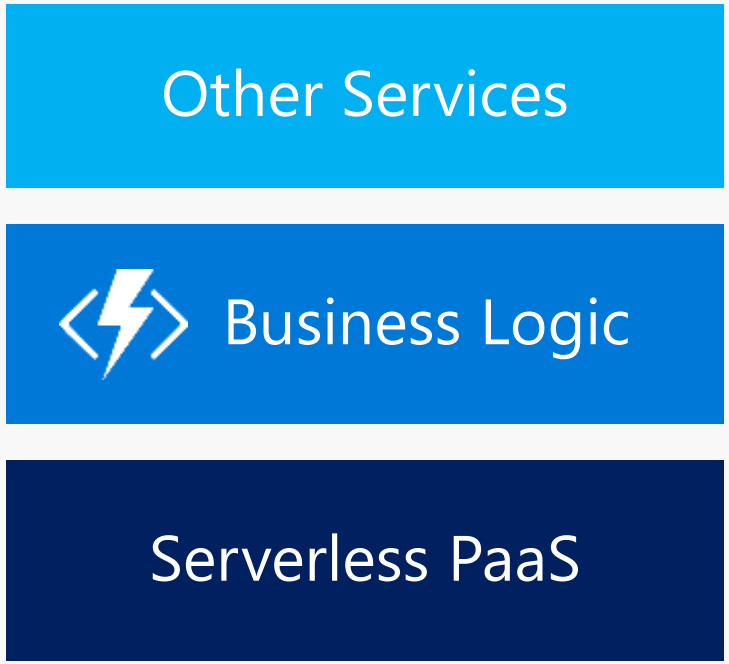
Portal – function.json

Visual Studio – decorate methods and params with attributes

```
[FunctionName("ValidateSiteContent")]
0 references
public static async Task<bool> Run(
    [OrchestrationTrigger] DurableOrchestrationContext context,
    TraceWriter log)
{
```

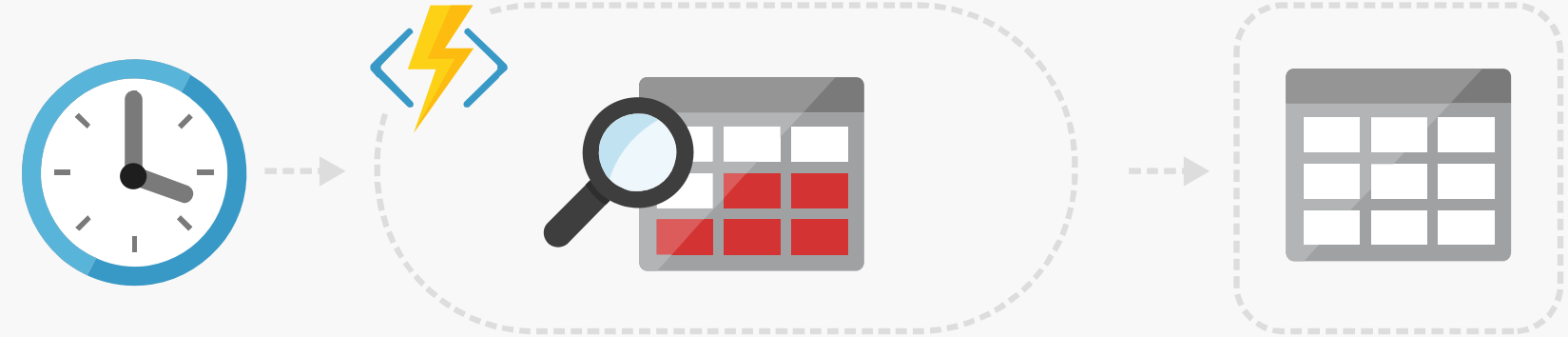
Dual abstraction

- Serverless compute abstracts away the compute
- Bindings abstract away the services you interact with



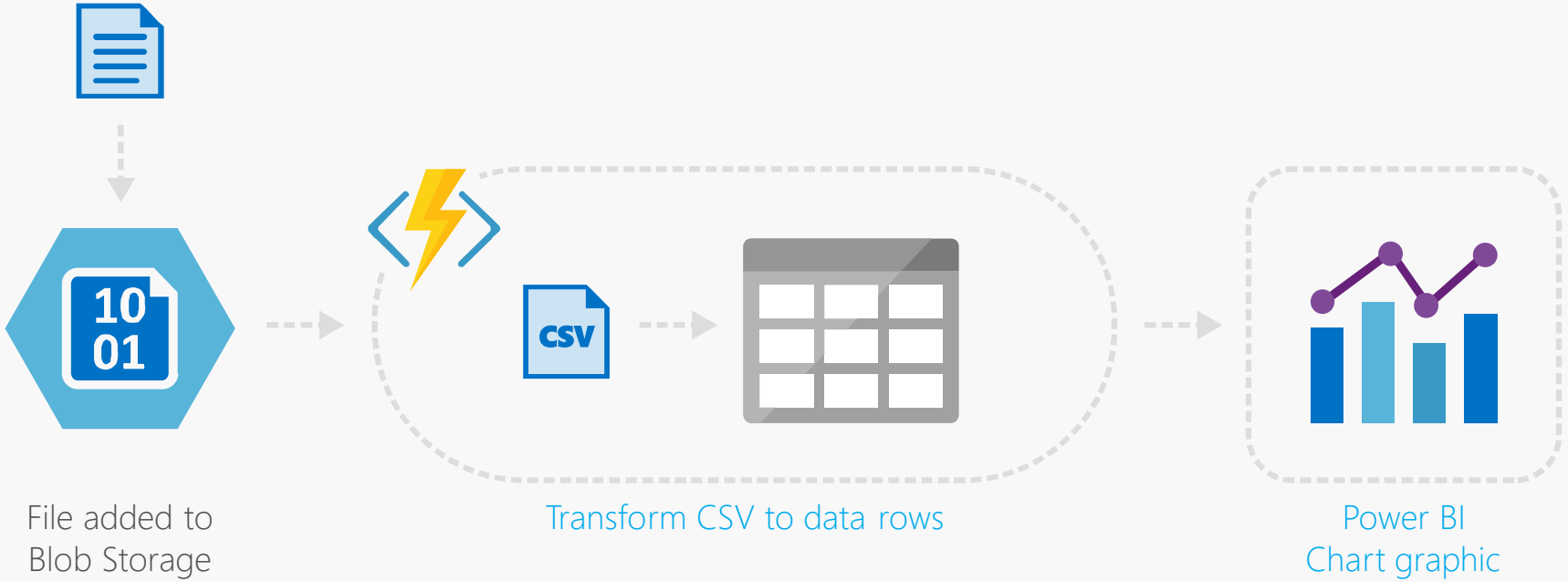
Applications

Example: Timer based processing



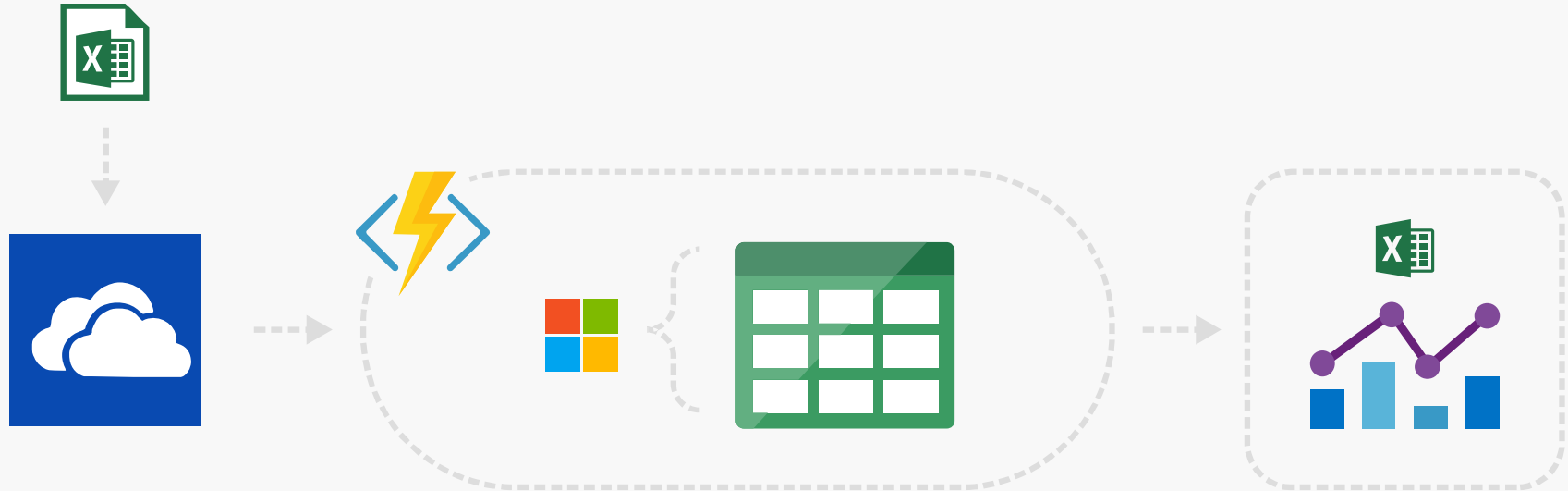
Applications

Example: Azure service event processing



Applications

Example: SaaS event processing



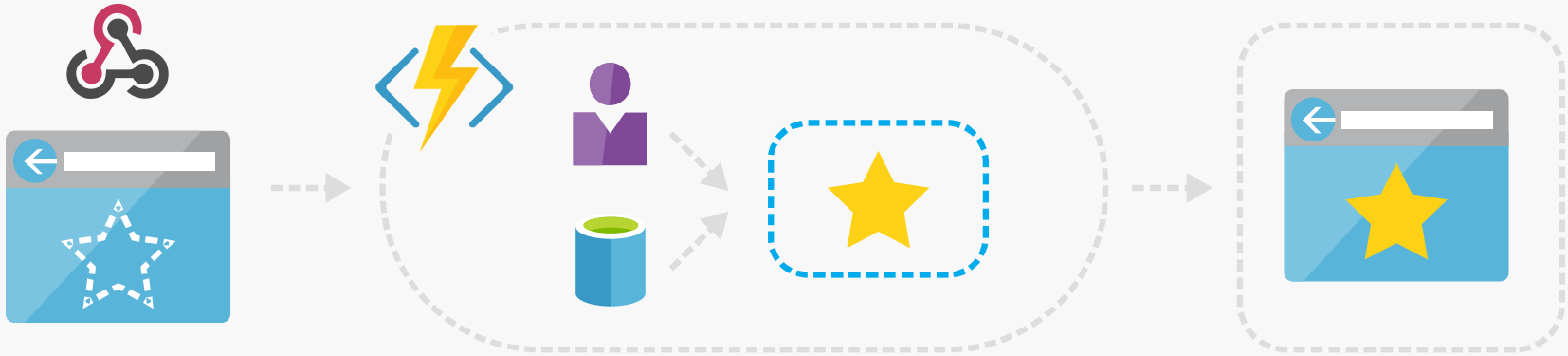
Excel file saved to OneDrive

Microsoft Graph API analyzes content

Creates new sheets with charts

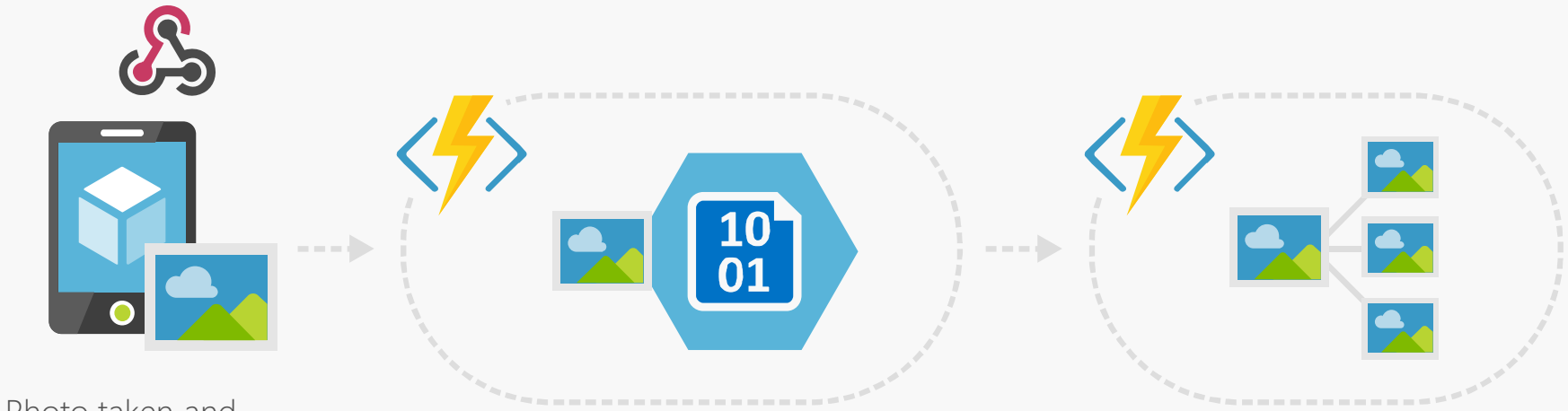
Applications

Example: Serverless Web Applications architectures



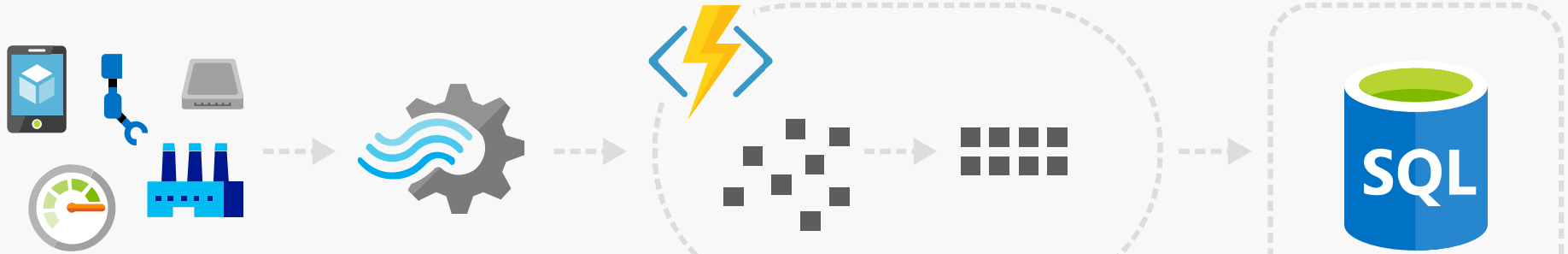
Async background processing

Example: Serverless Mobile back ends



Applications

Example: Real-time stream processing



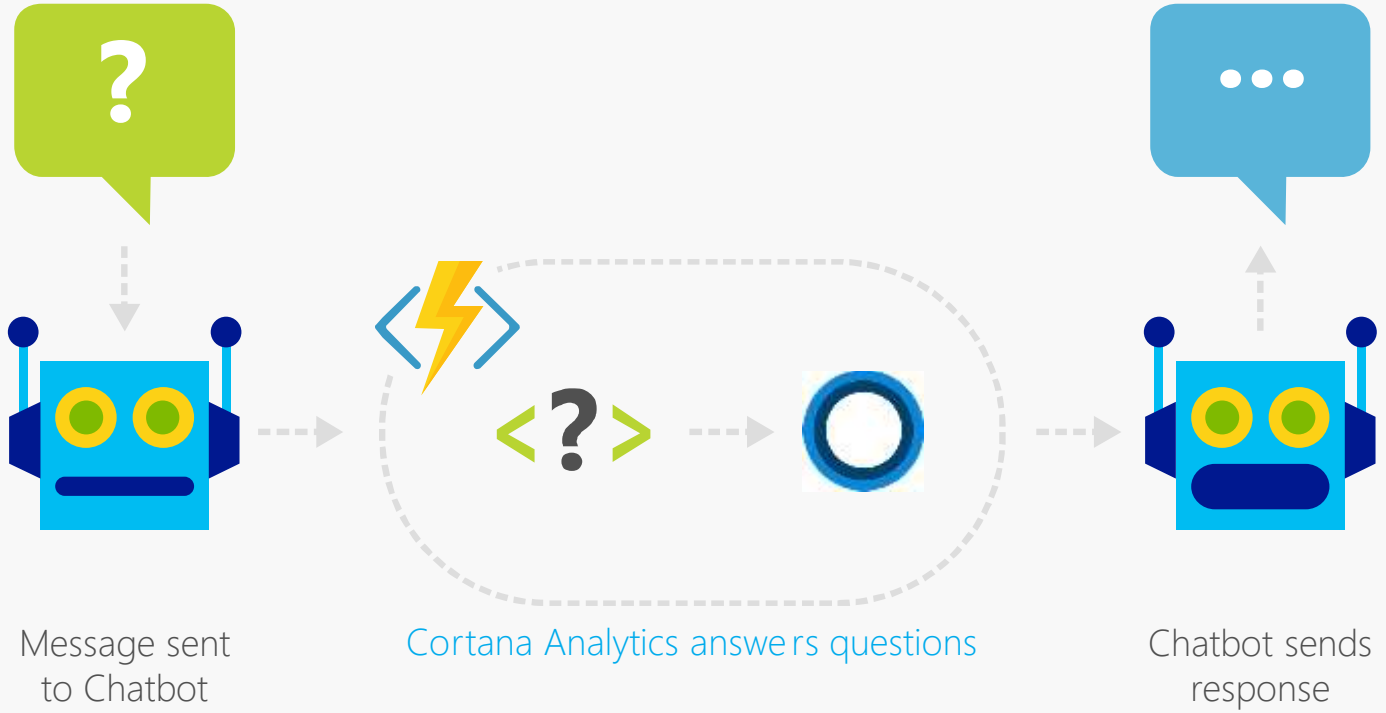
Millions of devices feed into Stream Analytics

Transform to structured data

Store data in SQL Online

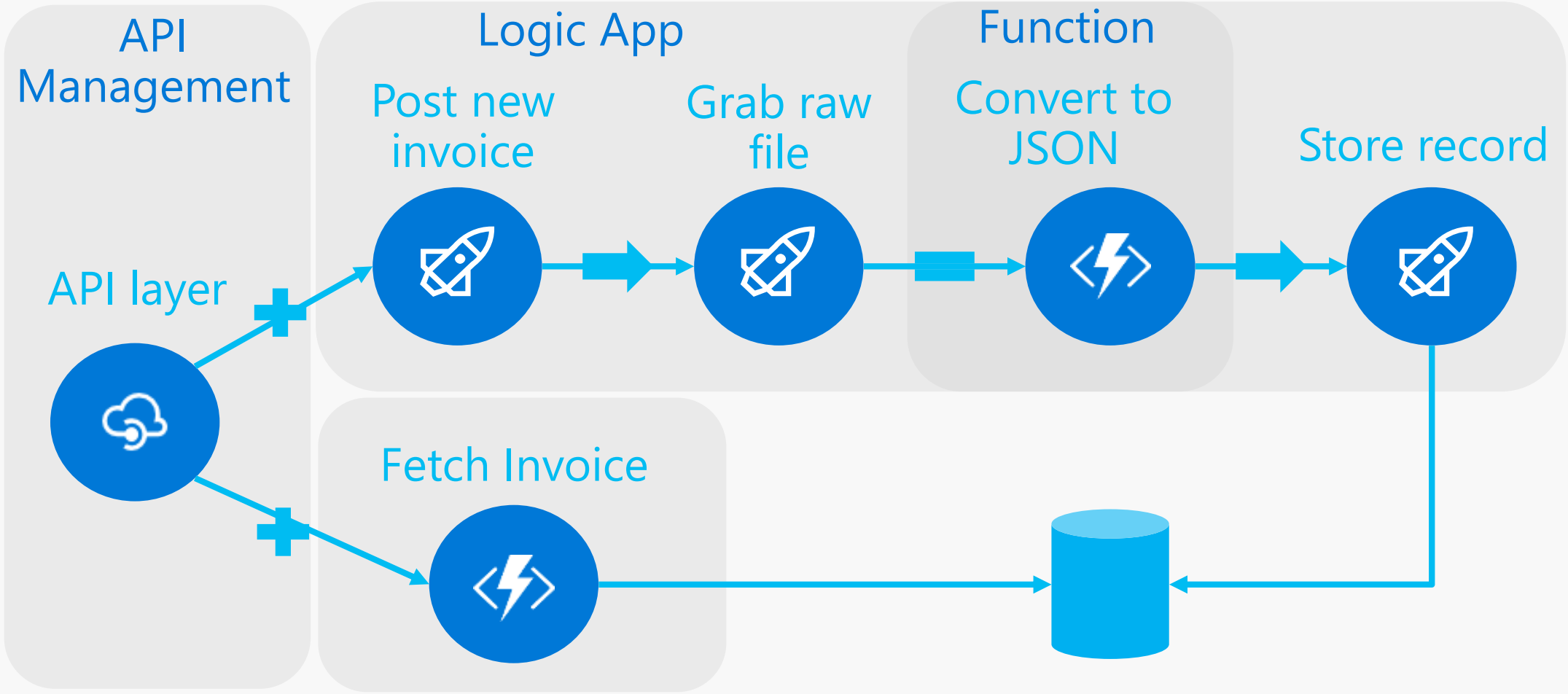
Applications

Example: Real-time bot messaging



Integration

Example: Invoice Processing



Contoso Pty Ltd.

1. Employees create several office documents (.docx, .xslx) etc.
2. Some of these don’t meet required quality criteria.
3. Reject these files unless an exception is given.
4. Few files are created at start of the month, whereas several thousand files get created towards the end of the month
5. These documents must be validated everyday.

Think about the solution

- Scale?
- Fault tolerance?
- Web Jobs?
- Flow/Logic Apps?
- Functions?
- State in functions?

- *& it should be easy to build and easy to manage!*

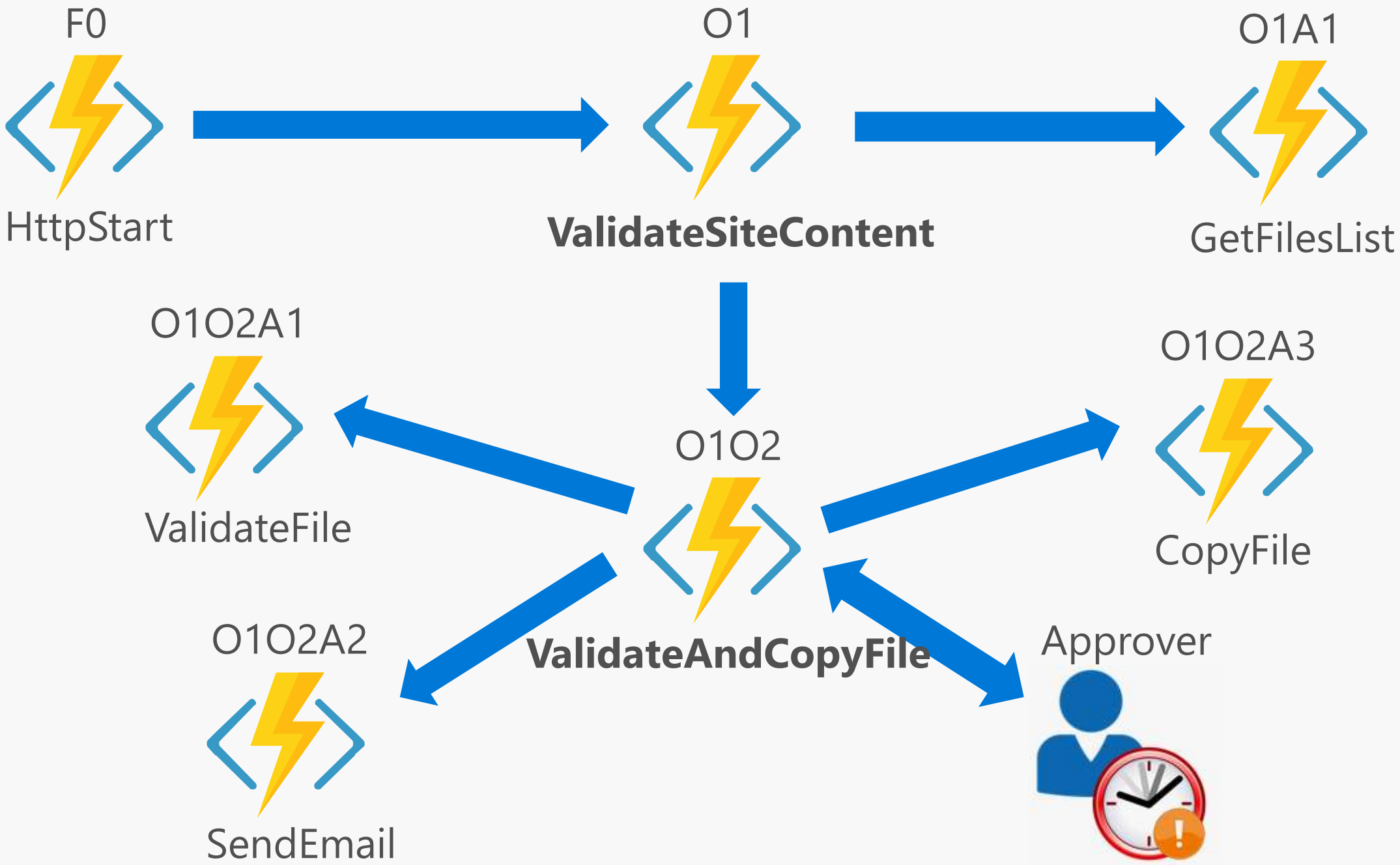
Simplify: Proof of Concept

1. Document Creation:

a. Given Folder path has .txt files

2. Validation Workflow:

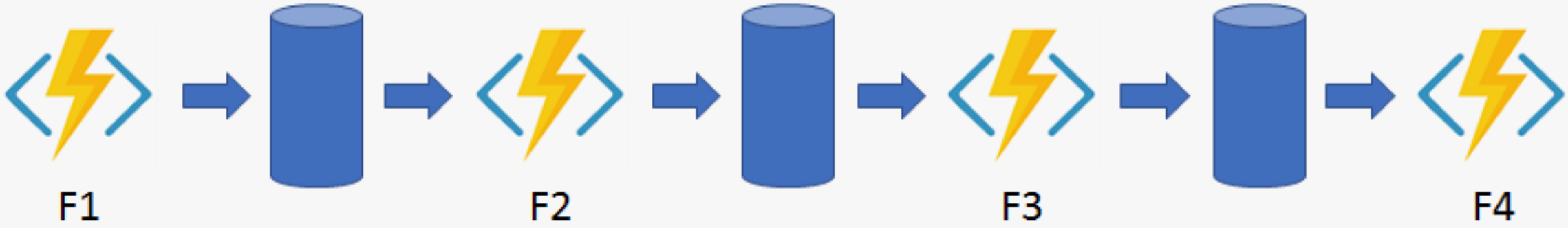
a. Look for company name “contoso” in each file
b. If validation succeeds, copy the file to Storage blob.
c. Send an email to admin if validation fails seeking exception
d. Admin has 24hrs to approve an exception, failing which file will not be copied.


F0
HttpStart
O1
ValidateSiteContent
O1A1
GetFilesList
O1O2A1
ValidateFile
O1O2
ValidateAndCopyFile
O1O2A3
CopyFile
O1O2A2
SendEmail
Approver

Demo

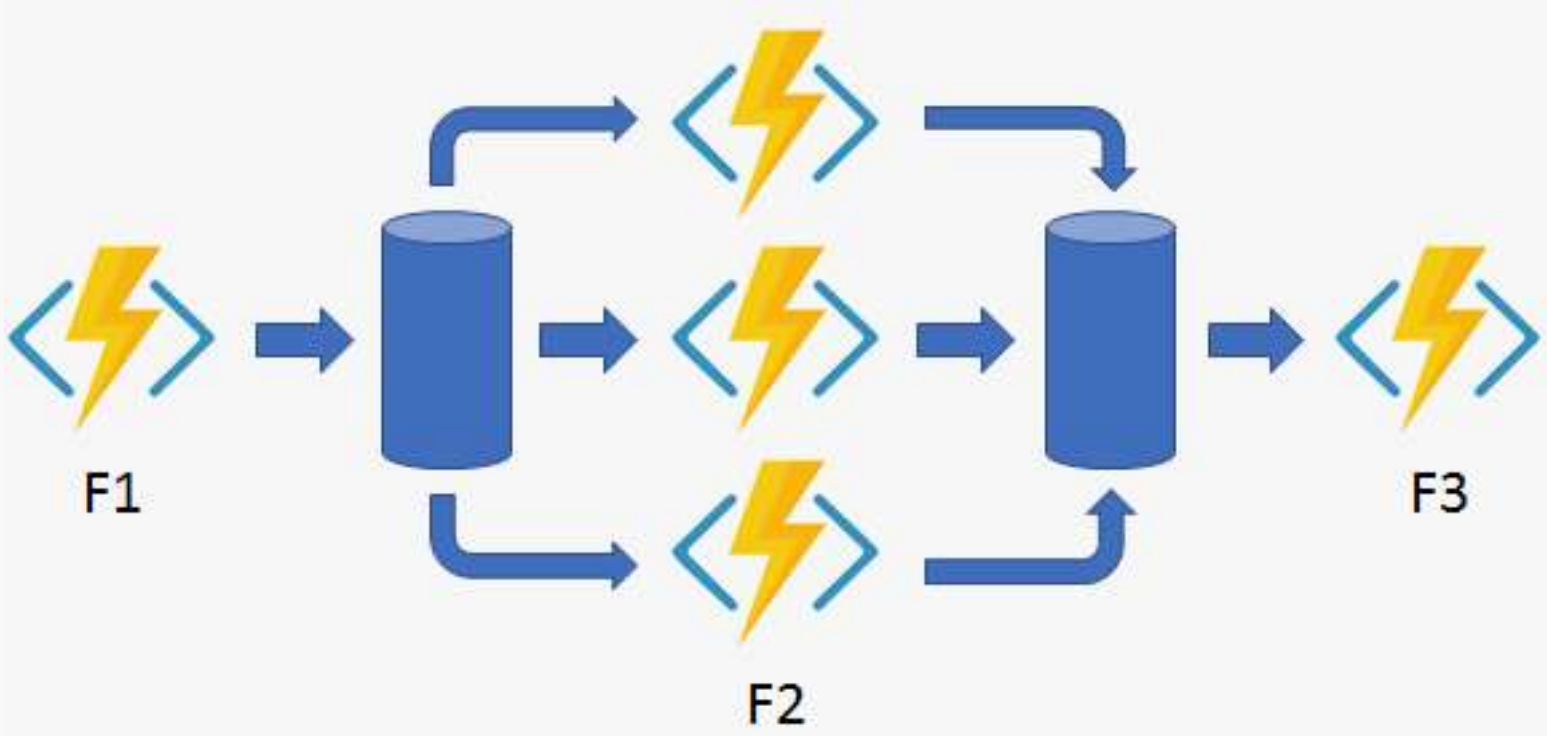
Pattern #1: Function chaining

- Execute a sequence of functions in a particular order.
- Often the output of one function needs to be applied to the input of another function.



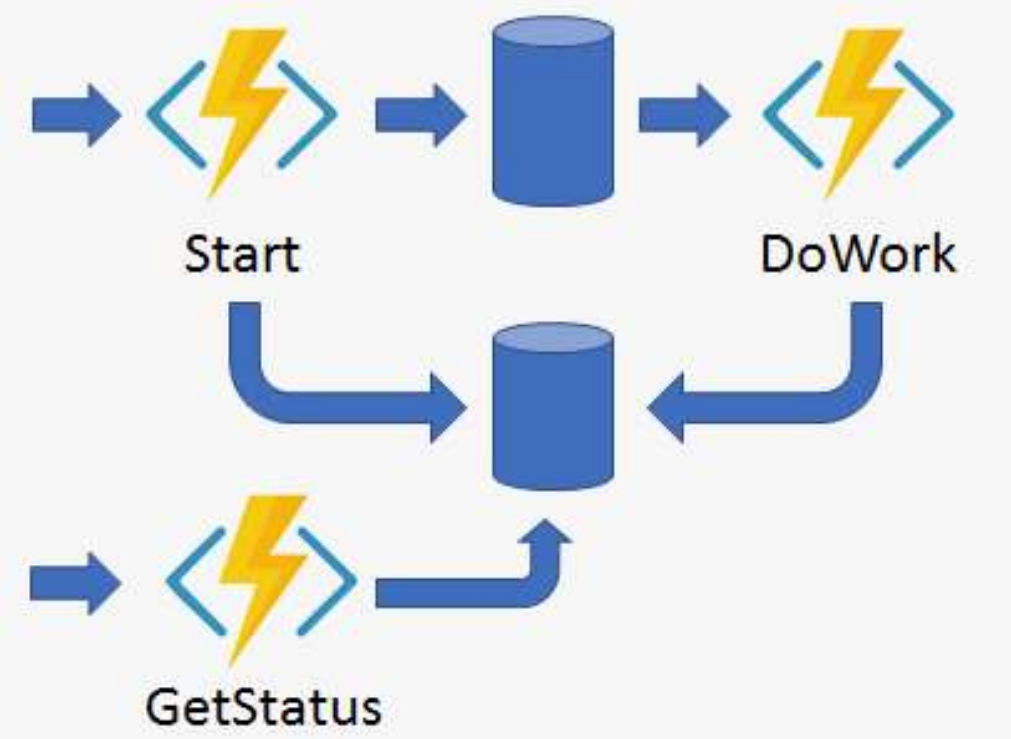
Pattern #2: Fan-out/fan-in

- Execute multiple functions in parallel, and then wait for all to finish.
- Often some aggregation work is done on results returned from the functions.



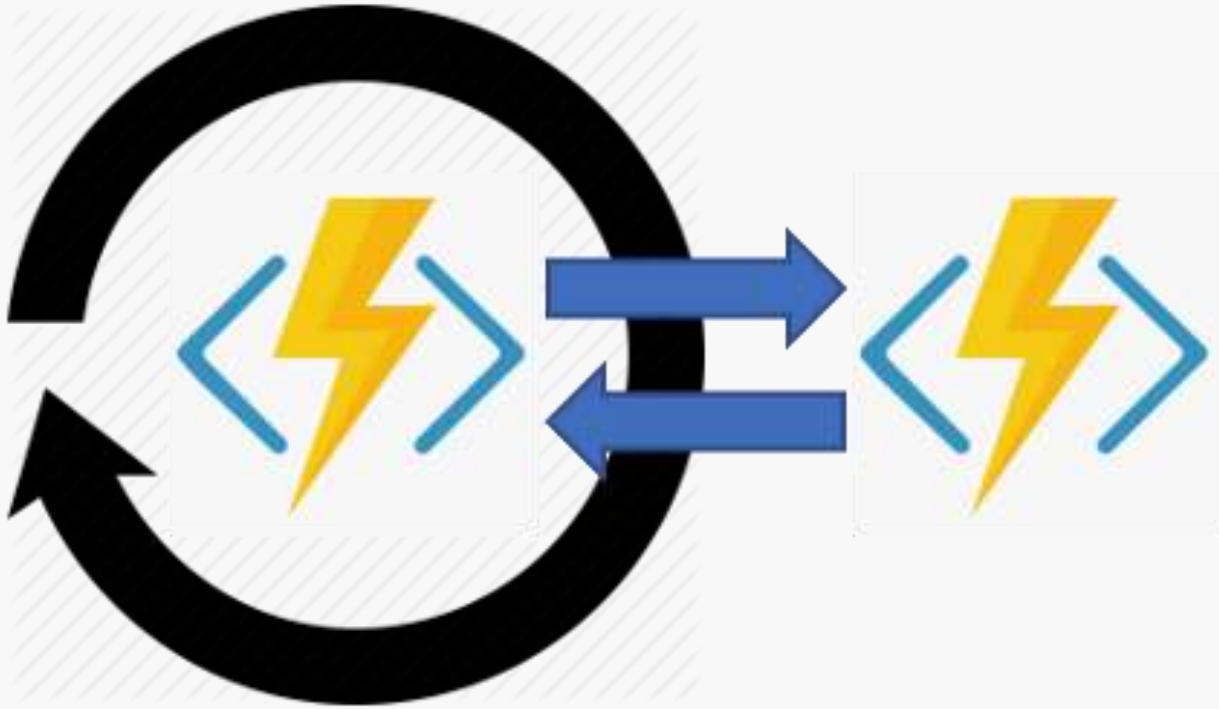
Pattern #3: Async HTTP APIs

- Coordinate the state of long-running operations with external clients
- Have the long-running action triggered by an HTTP call,
- Redirect the client to a status endpoint that they can poll for status



Pattern #4: Monitoring

- Flexible recurring process in a workflow
- Example: Timer trigger can do a periodic clean up job, but interval is static. Using DF, this can be made flexible – or even monitor multiple processes from a single orchestration.



Pattern #5: Human interaction

- Involves human interaction in an approval process.

Triggers

Orchestration client

```
[OrchestrationClient] DurableOrchestrationClient starter
{ return starter.StartNewAsync("FunctionName", input); }
```

Orchestration triggers

```
[OrchestrationTrigger] DurableOrchestrationContext context
{ result = await context.CallActivityAsync<string>("SayHello", name); }
```

Activity triggers

```
[ActivityTrigger] DurableActivityContext helloContext
{ string name = helloContext.GetInput<string>(); }
```

Under the hood

- Reliability
  - though underlying VMs/Network infra may not be 100% reliable
- Durable Task Framework
- Based on Event Sourcing design pattern
- Execution history in Storage table
- Function unloaded <-> Function restarted
- Context.IsReplaying

Tips and Tricks

- Errors in Activity Functions
- Automatic retry on failure (Sub-orchestration/activity)
- Unhandled exceptions in Orchestration Functions
- Durable Timers
- WaitForExternalEvent
- Singleton Orchestrators

Gotchas

Orchestrator code

- Must be deterministic
- Must be non-blocking
- Should not initiate any async operation
- Avoid infinite loops (use ContinueAsNew instead)
- In-flight Versioning
  - Do Nothing
  - Stop all in-flight instances
  - Side by side deployments

Get started and reach out!

Try Functions – https://functions.azure.com/try

Try Durable Functions - https://docs.microsoft.com/en-us/azure/azure-functions/durable-functions-overview

Try App Service – https://tryappservice.azure.com

Questions?

Suhas Rao,
Azure Technology Specialist App Dev, Microsoft
@suhasaraos (twitter)
linkedin.com/in/suhasaraos/