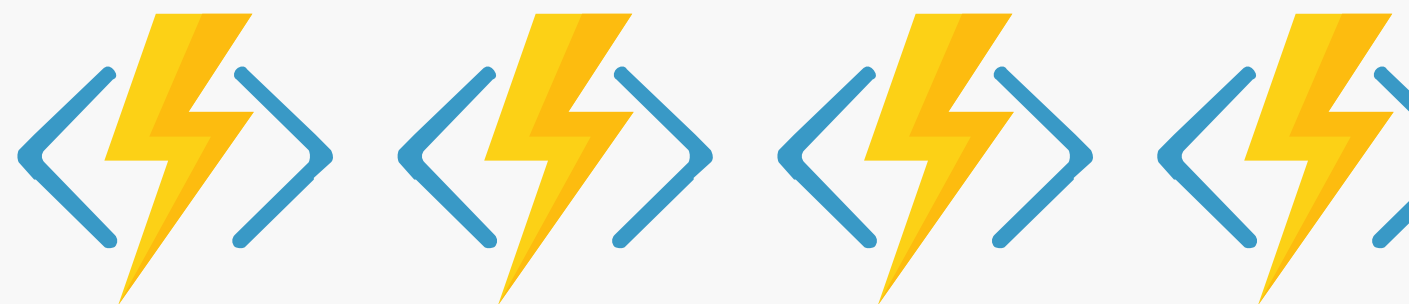
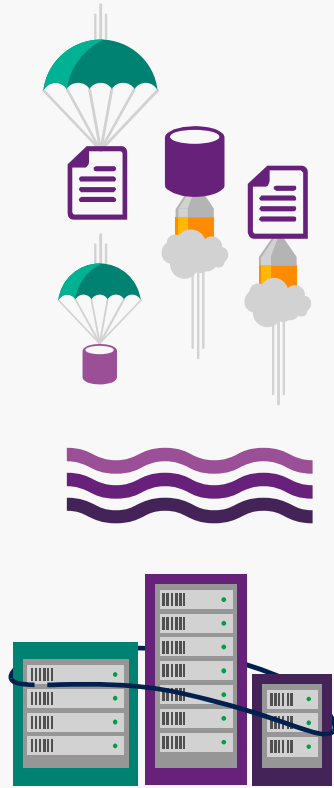




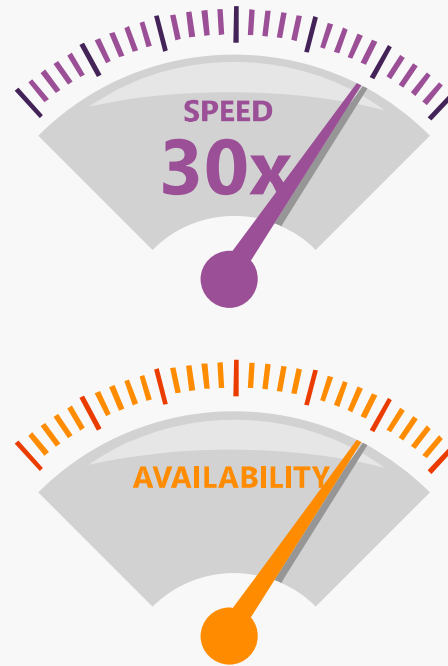
Adding State to
Serverless



What is "serverless"



Abstraction
of servers

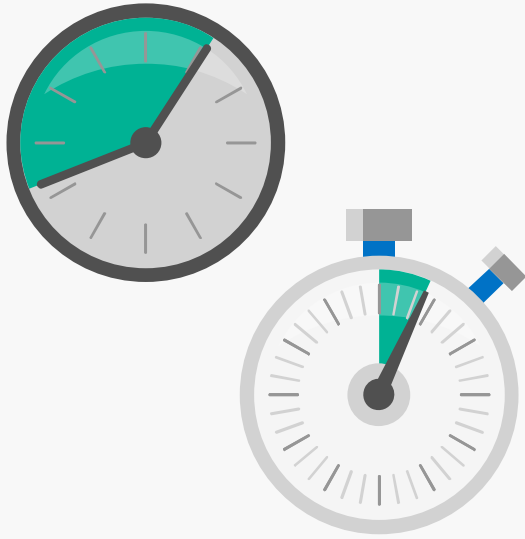


Event-driven
scale

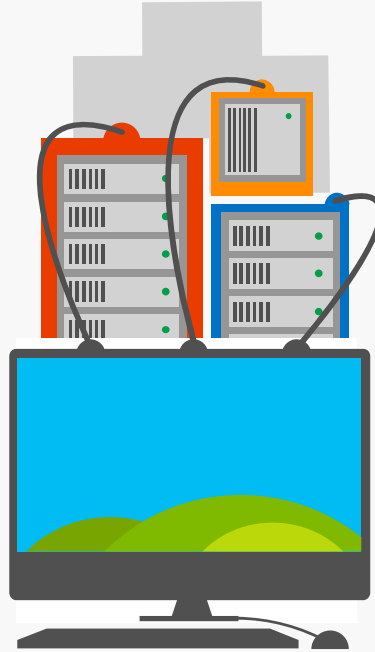


Sub-second
billing

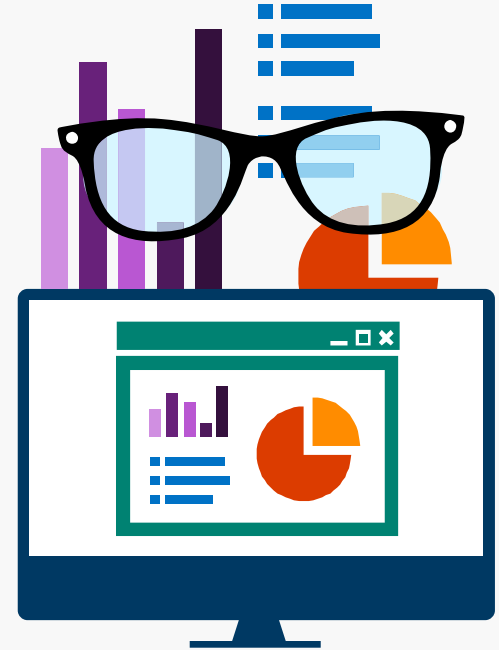
Benefits of "serverless"



Ship faster



Reduced dev
ops



Focus on
business
logic

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”

Contoso Pty Ltd.

1. Employees create several office documents (.docx, .xlsx) 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!*

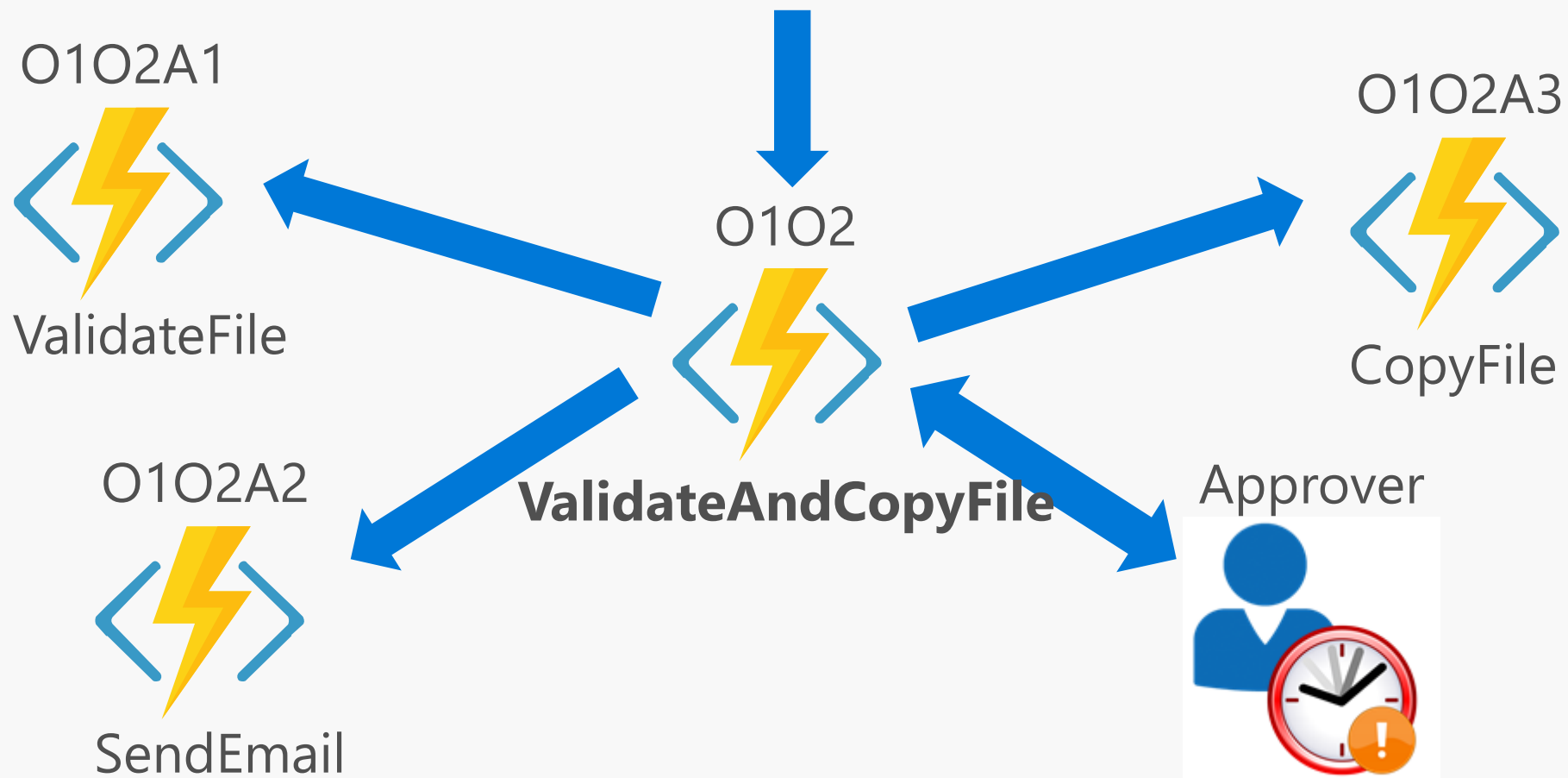
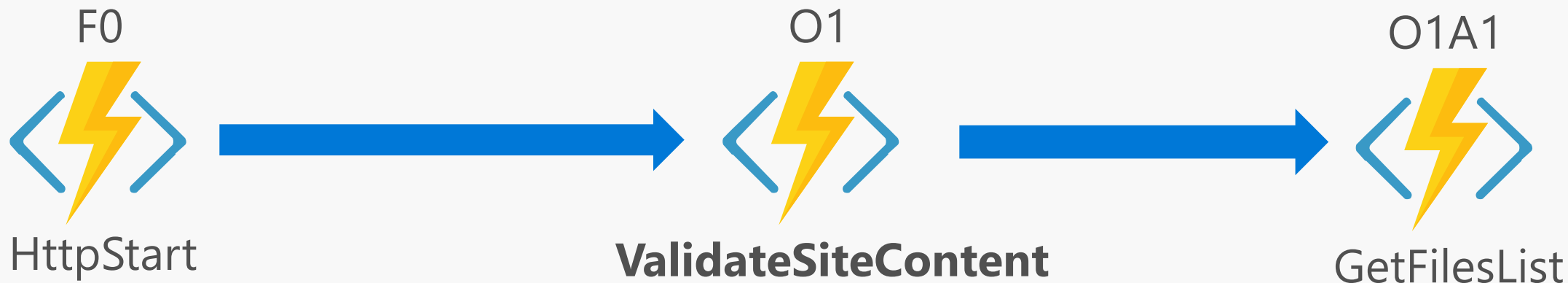
Simplify: Proof of Concept

1. Document Creation:

- a. Given Folder path has .txt files

2. Validation Workflow:

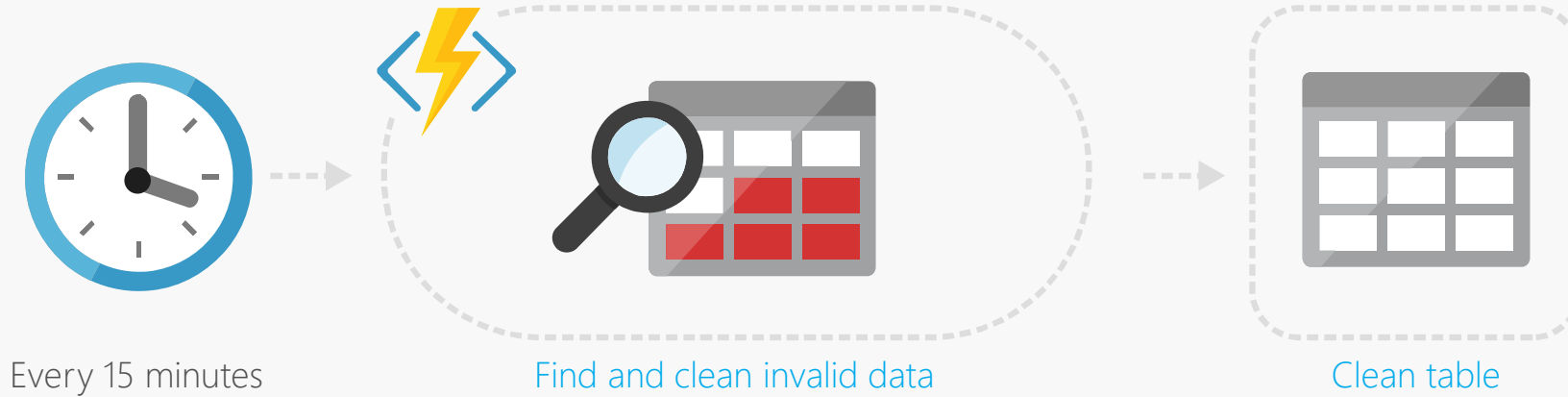
- a. Look for company name "contoso" in each file
- b. If validation succeeds, copy the file to Storage blob.
- c. Send an email to admin if validation fails seeking exception
- d. Admin has 24hrs to approve an exception, failing which file will not be copied.



Demo

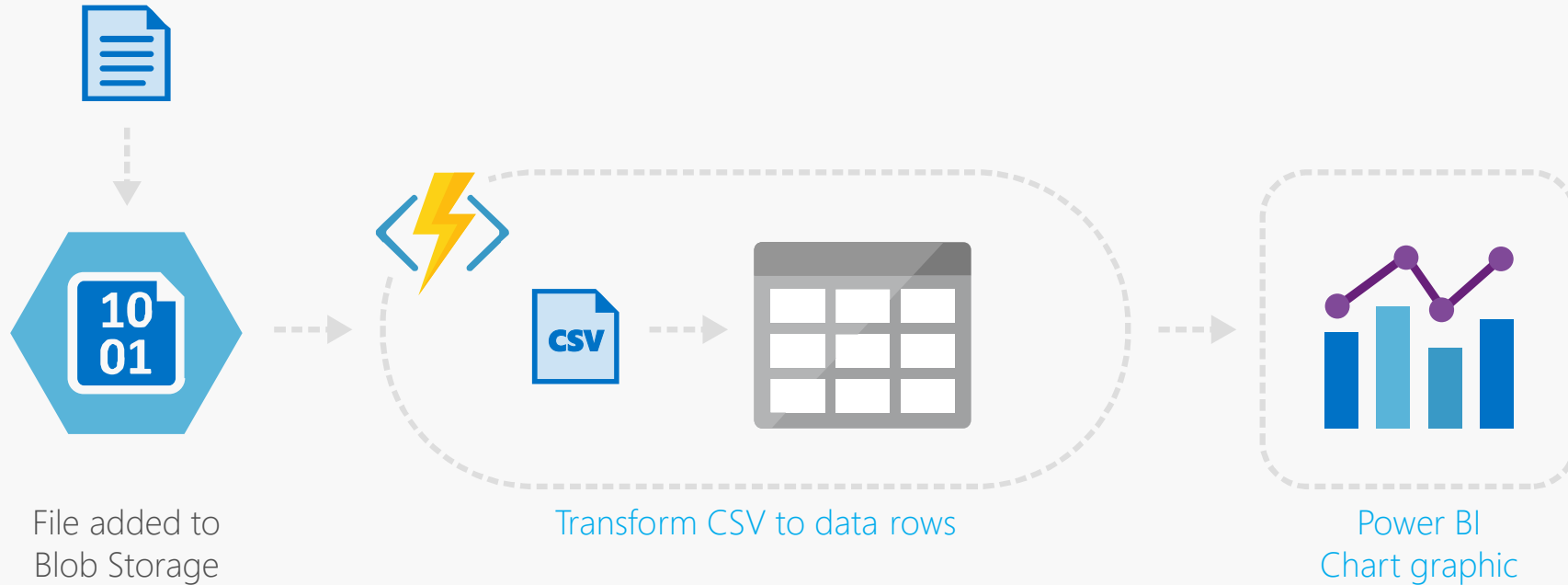
Applications

Example: Timer based processing



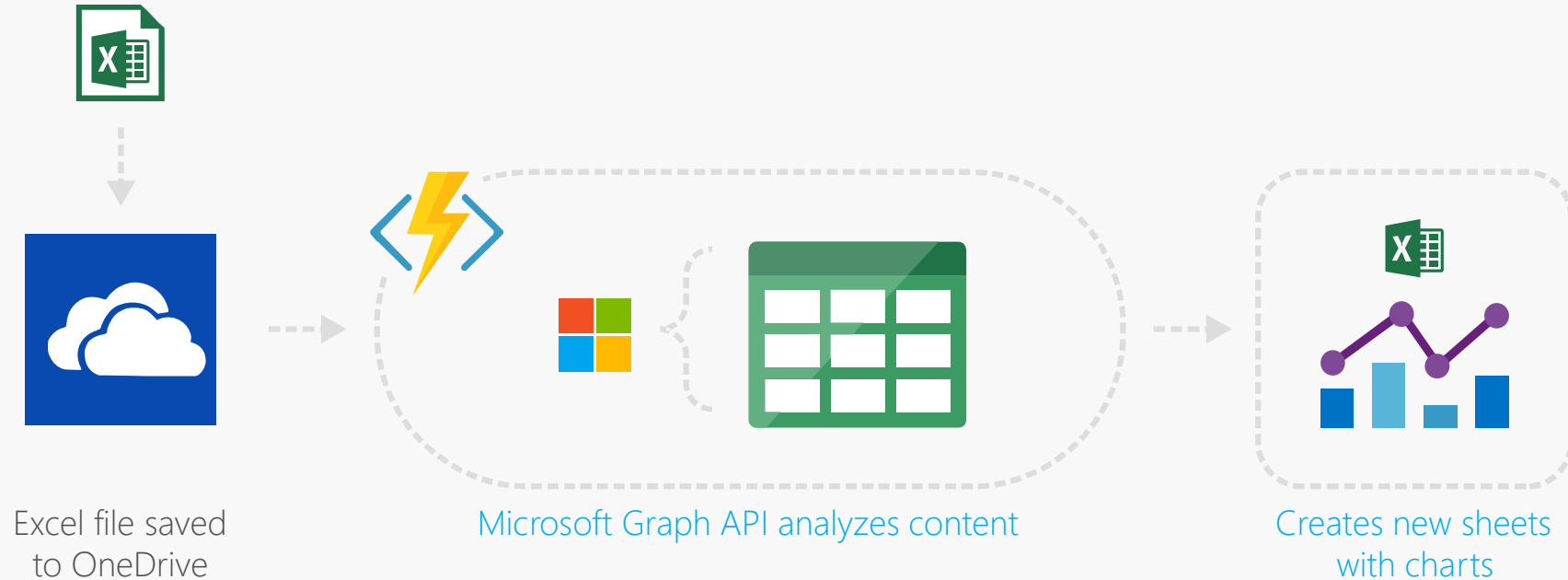
Applications

Example: Azure service event processing



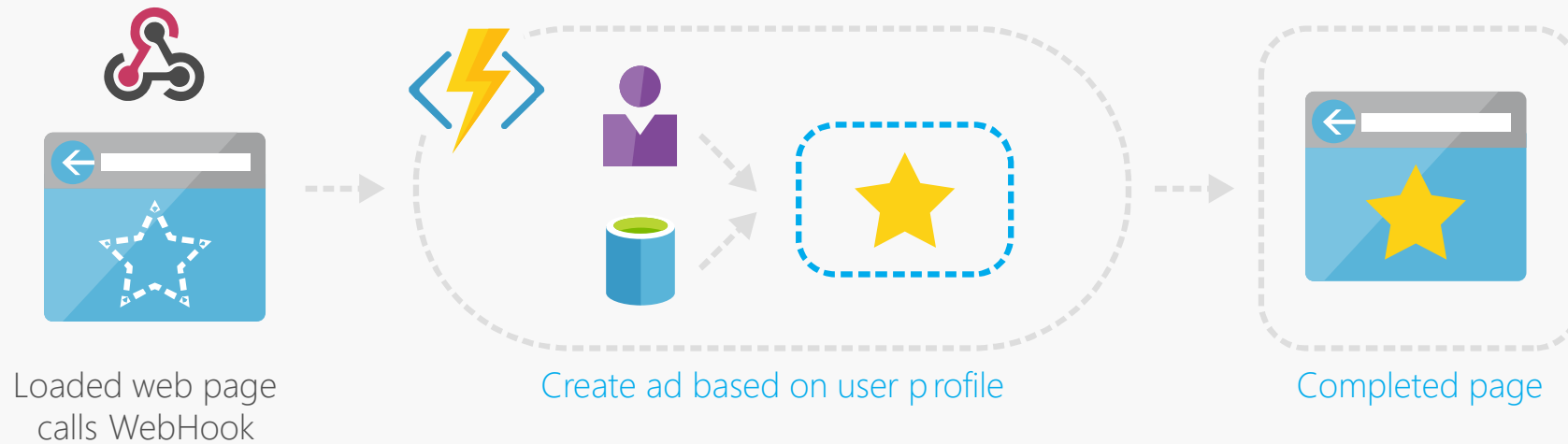
Applications

Example: SaaS event processing



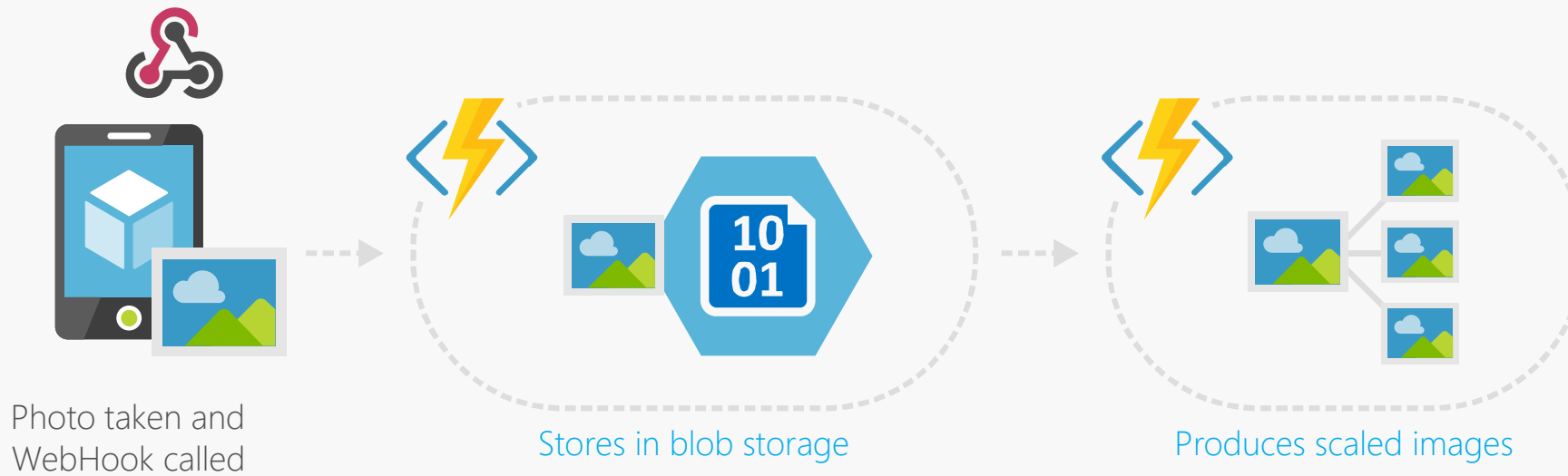
Applications

Example: Serverless Web Applications architectures



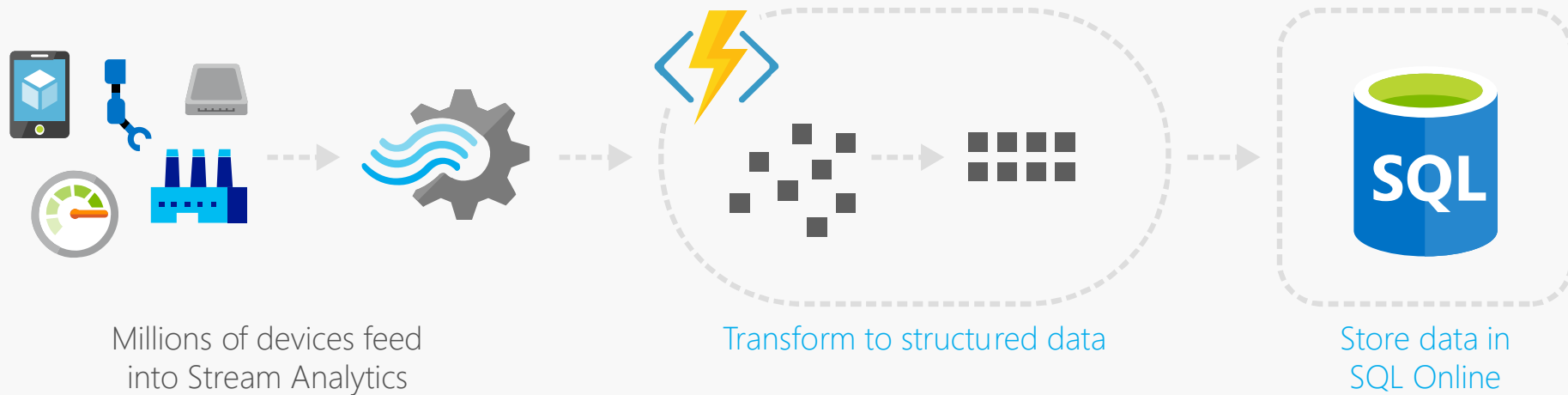
Async background processing

Example: Serverless Mobile back ends



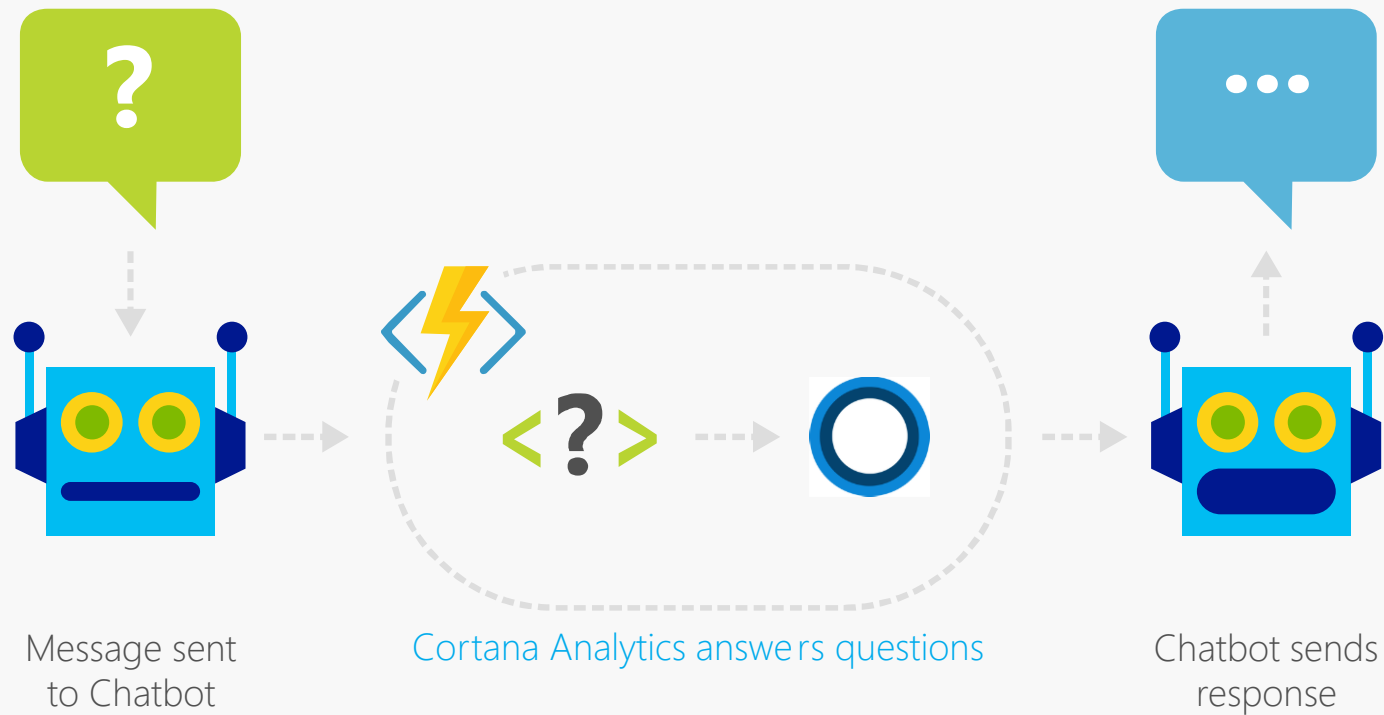
Applications

Example: Real-time stream processing



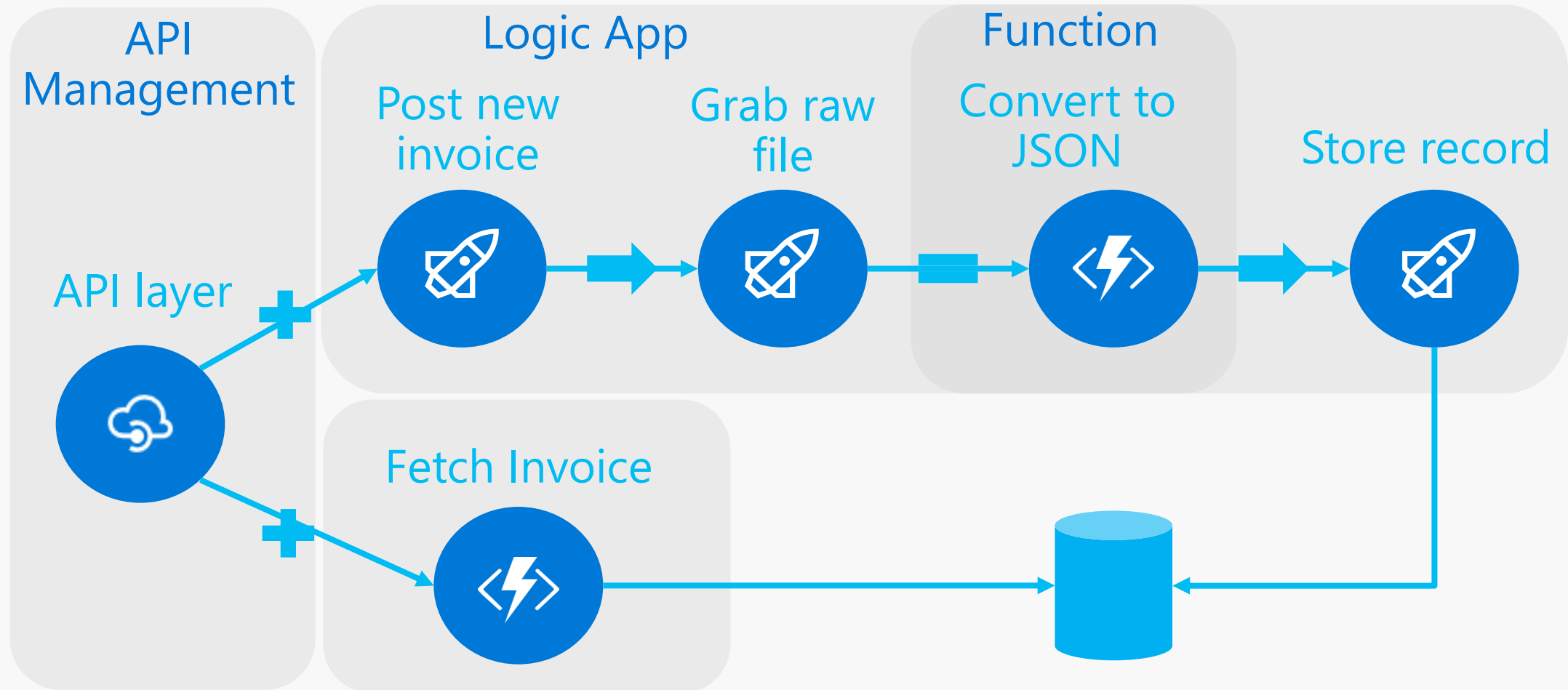
Applications

Example: Real-time bot messaging



Integration

Example: Invoice Processing





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.



Microservice tools and approaches

Microservices in the wild

Implication: Build your own microservices platform

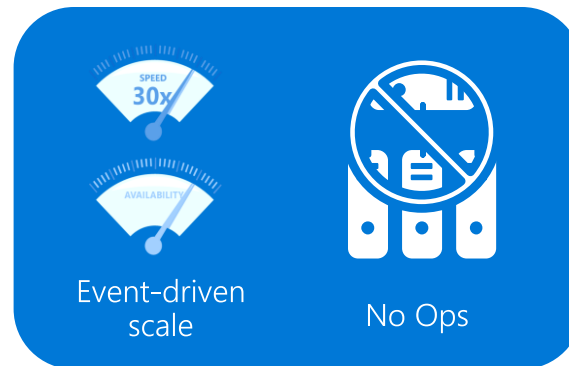
Benefits: Customizable, pick best of breed solutions



Azure Functions

Implication: Serverless microservices

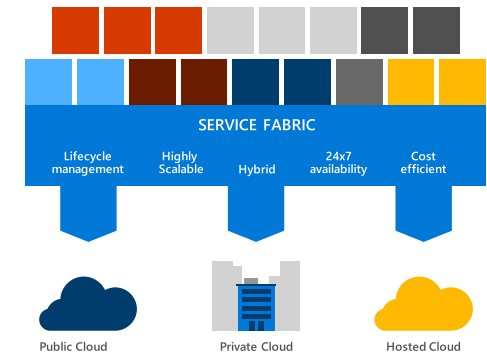
Benefits: Quick ramp up, sub second metering, zero ops



Azure Service Fabric

Implication: Prescriptive microservices platform

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



Dual abstraction

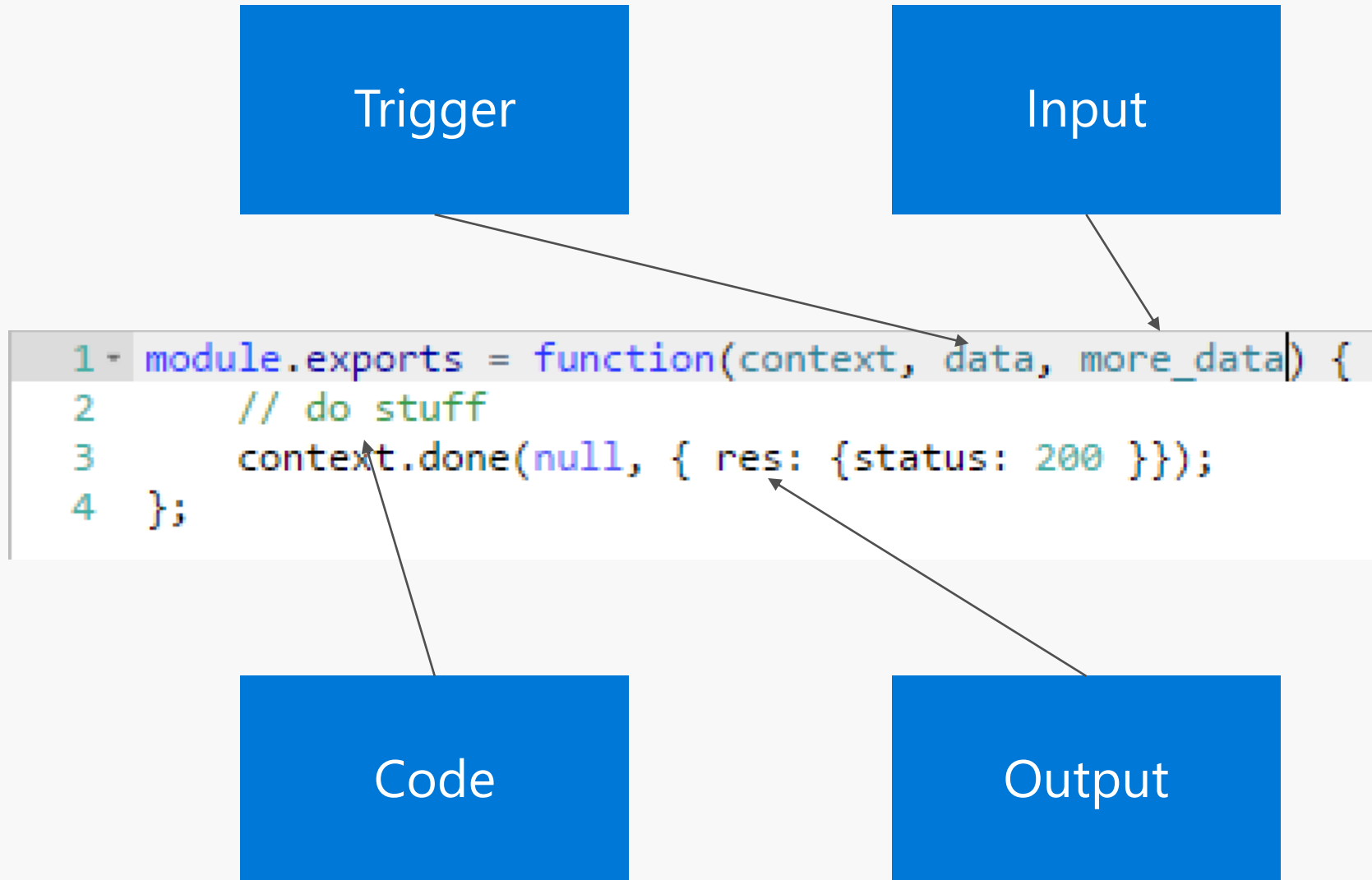
- Serverless compute abstracts away the compute
- Bindings abstract away the services you interact with



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

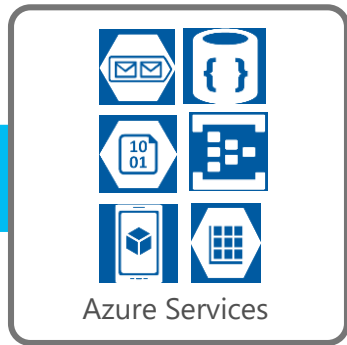
Functions programming concepts



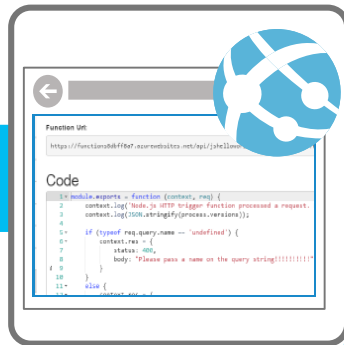
1) Trigger



2) Input Binding



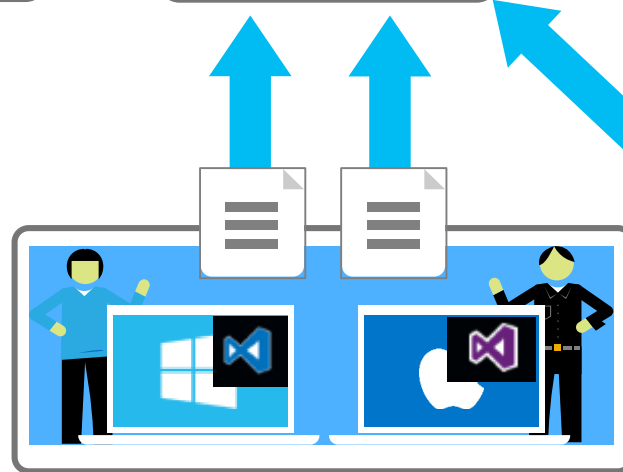
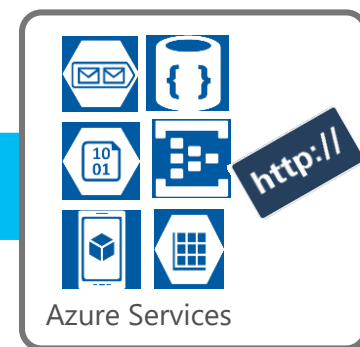
3) Develop



4) Execute



5) Output Binding



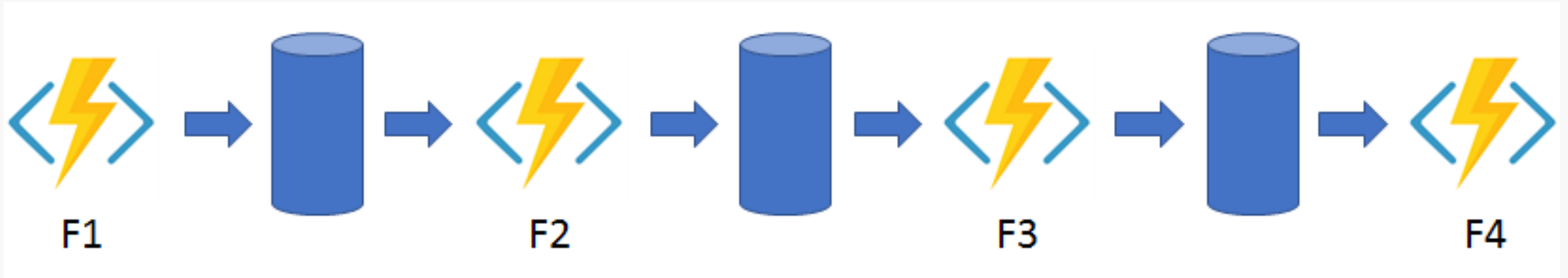
7) Develop Locally



6) Monitor and Improve

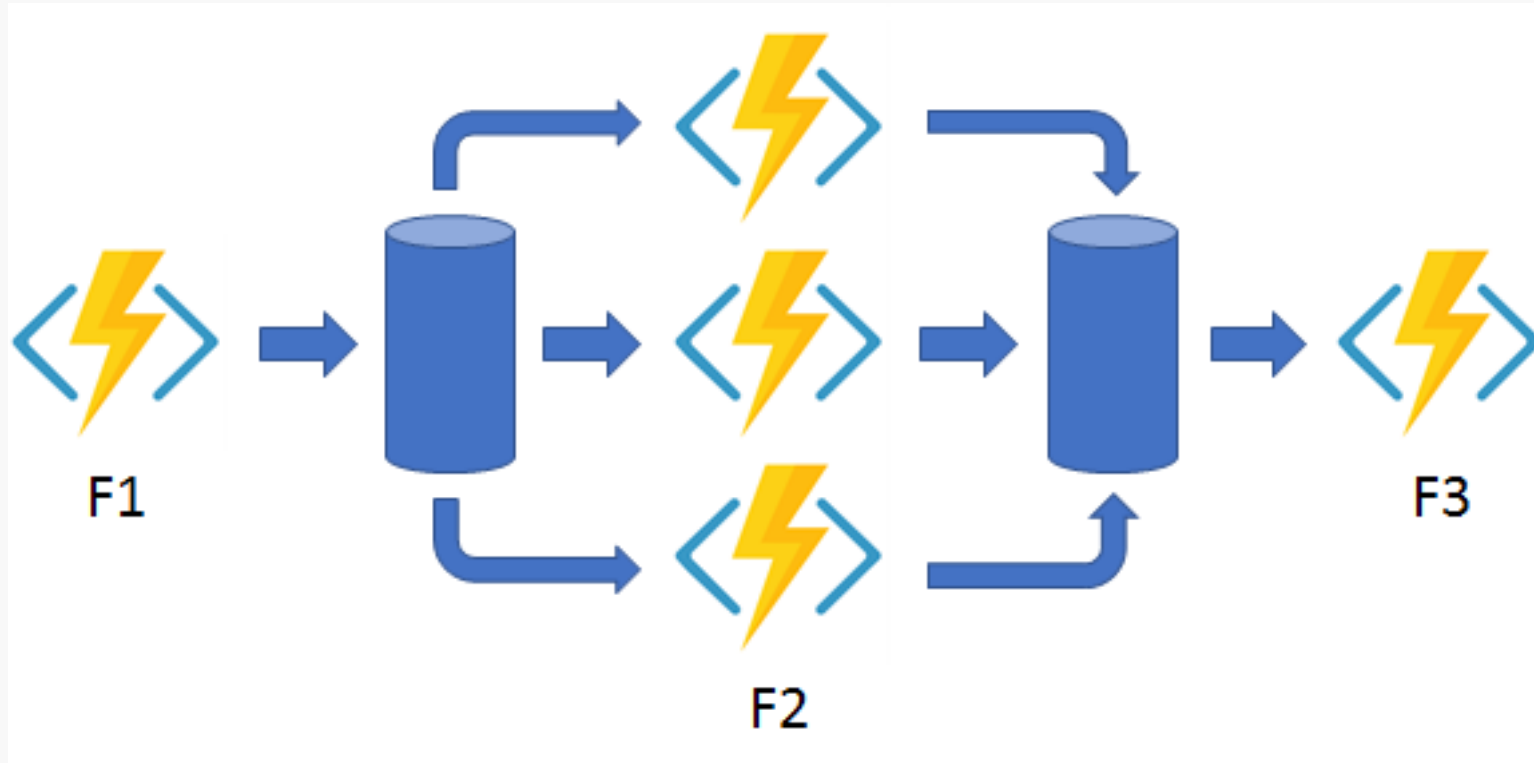
Pattern #1: Function chaining

- 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.



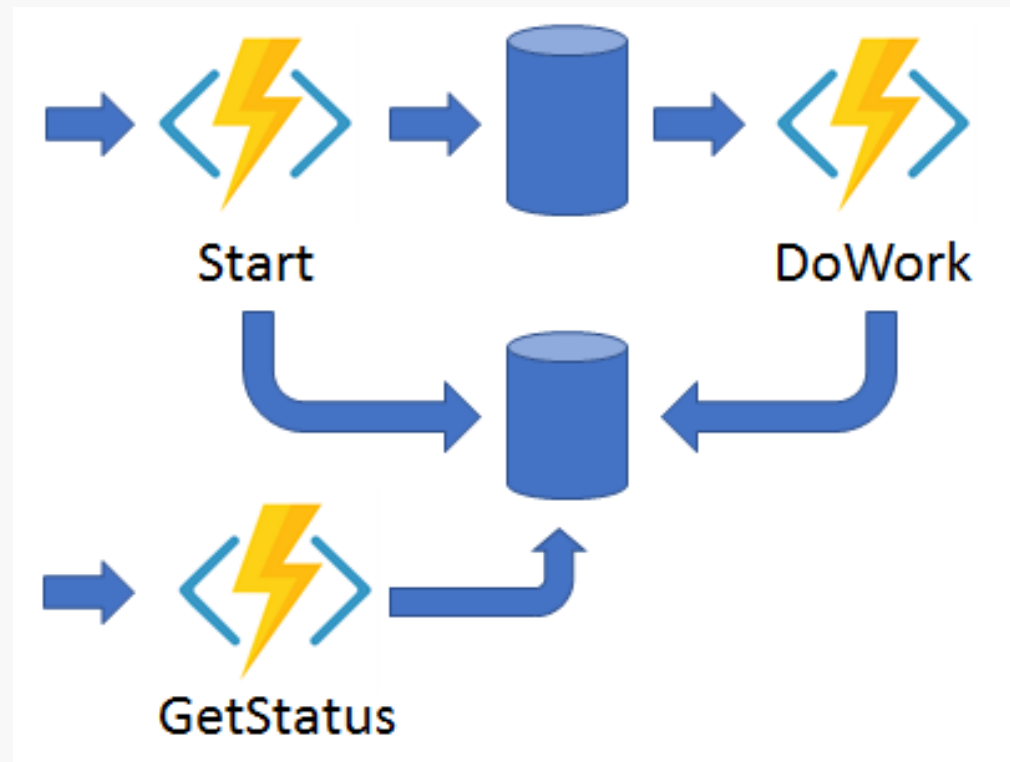
Pattern #2: Fan-out/fan-in

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



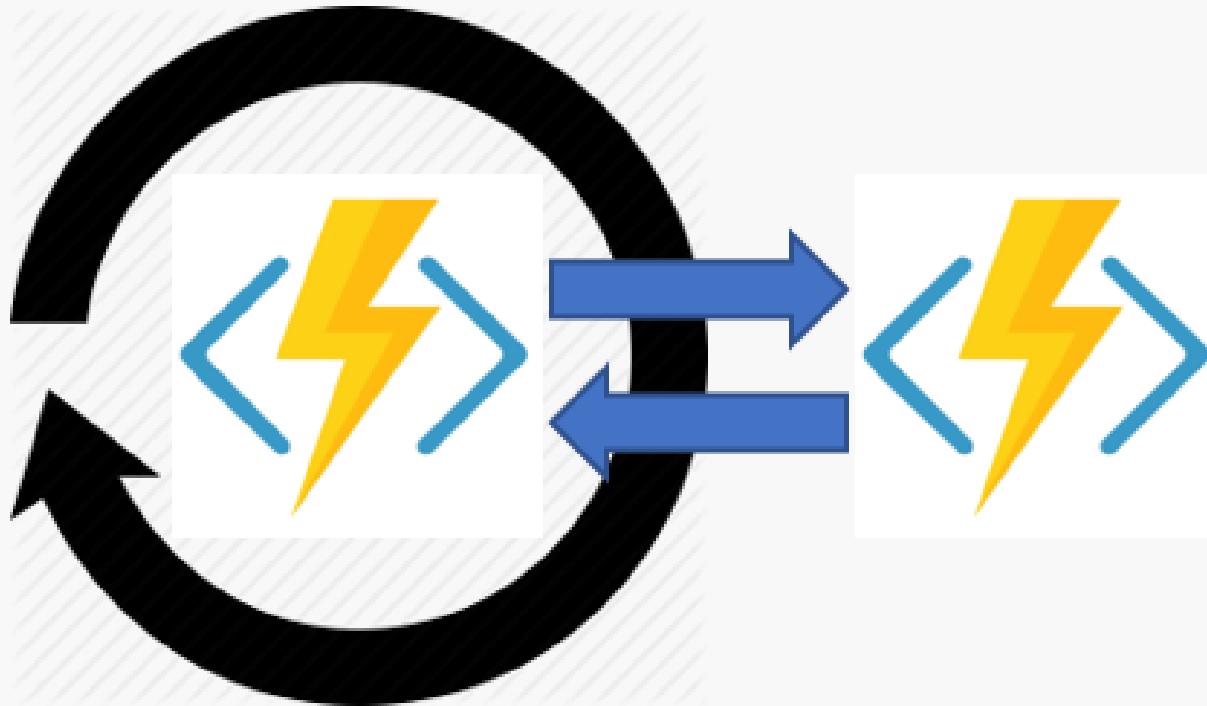
Pattern #3: Async HTTP APIs

- 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

- Reverse the earlier async HTTP API scenario.
- Instead of exposing an endpoint for an external client to monitor a long-running operation, the long-running monitor consumes an external endpoint, waiting for some state change.



Pattern #5: Human interaction

- Involves human interaction in an approval process.



Just a beginning

Versioning

DevOps

Error handling

Sending external events

Sub-orchestrations

Performance

See you at:

<https://www.meetup.com/Azure-Sydney-User-Group/events/249244017/>

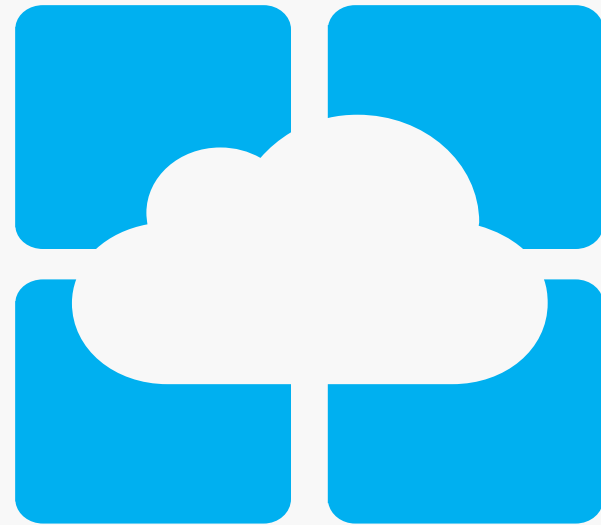


Get started and reach out!

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/