Sprint 1

GOAL: Publish incoming requests to the utilization management platform

***TODO

- **** Install Azure CLI ****

- ??? Include App Insights when publishin???

- Do we need to ensure Azure emulator installed?
- Add completed solution to sprint 2

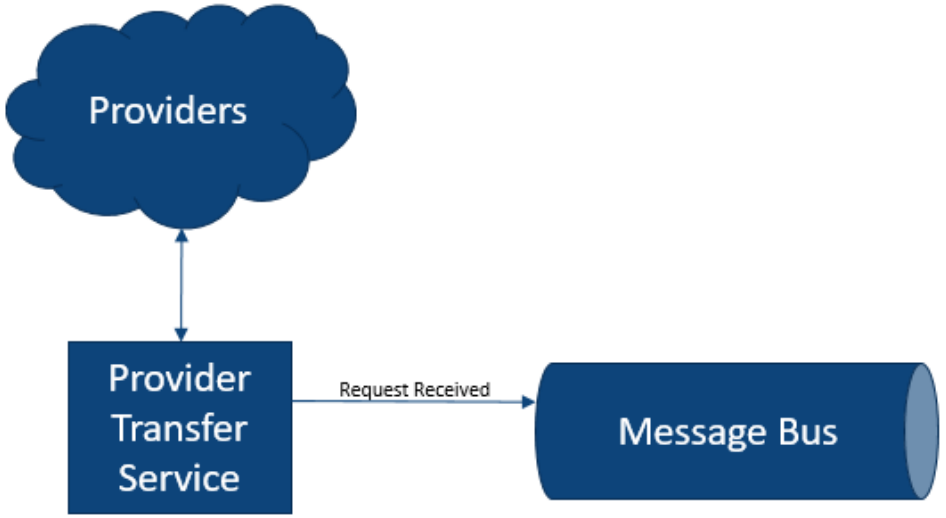
User Story 1-1

As a TAH developer, I need all incoming requests for service to be available to all current and future components of the utilization management platform so that I can implement the features provided by TAH.

Acceptance Criteria

1. Requests are available to current and future services within 10 seconds of arriving on the network
2. Consumers must be able to receive the requests in the order they were received
3. Consumers should be able to
   a. receive requests asynchronously
   b. replay previously received requests
4. System must be able to receive up to 10,000 incoming requests per minute

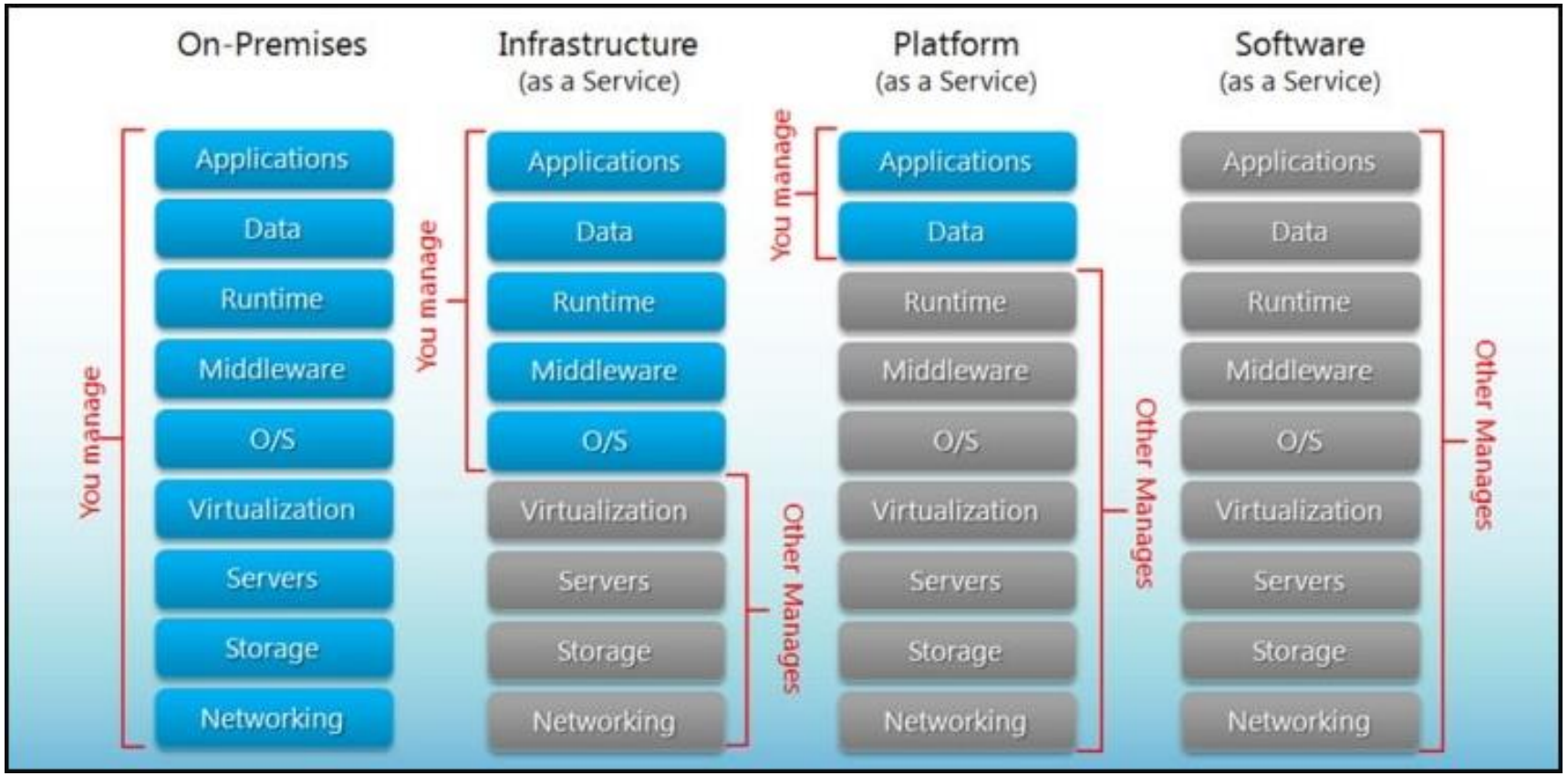
User Story 1-1: Design



- We'll simulate incoming requests for service by generating random requests in the Provider Transfer Service

- What Azure resources to use for message bus and service?
  - Generally prefer PaaS (Platform as a Service) over IaaS (Infrastructure as a Service)

Cloud Maturity Model



https://www.hostingadvice.com/how-to/iaas-vs-paas-vs-saas/?_ga=2.28238708.972024003.1631988165-2010794804.1631988165

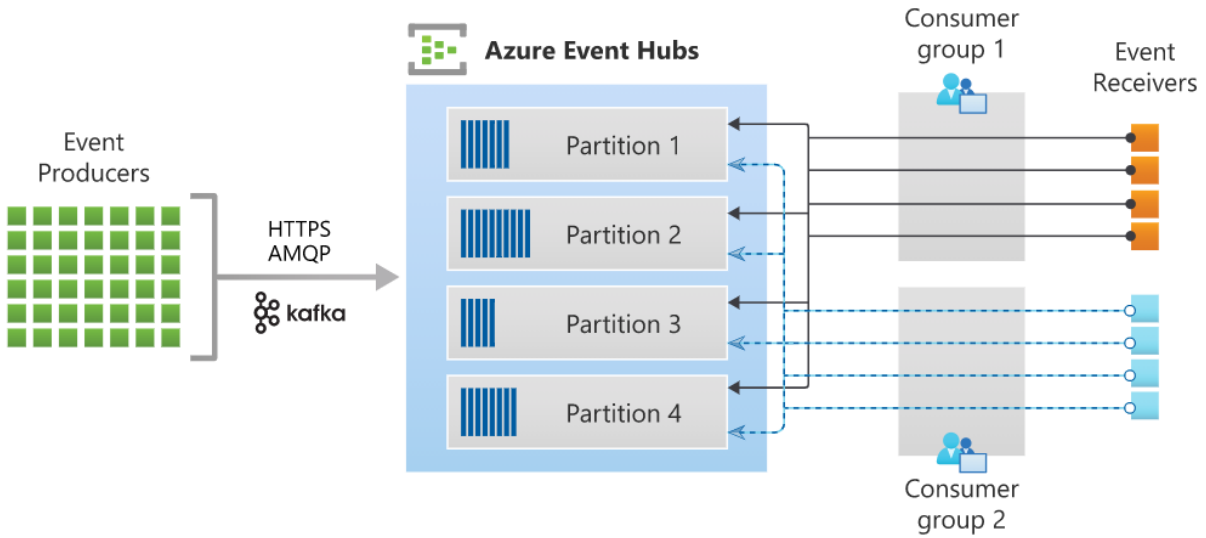
Azure Messaging Options

	Service Bus	Event Hubs	Event Grid
Purpose	High-value enterprise data	Big data pipeline	Reactive programming
Unit of work	Message	Event (streams)	Event (discrete)
Throughput	Thousands of messages per second	Millions of messages per second	Thousands of messages per second
Cost	$0.0135/hour + $0.80 per million operations (after 13M operations)	$0.03/hour + $0.028 per million events	$0.60 per million operations
Consume previously published messages / replay	No	Yes	No

User Story 1-1: Message Bus Design

- Choosing Event Hub
  - Can handle huge volumes – millions of messages per second
  - Can add consumers after messages published
  - Can replay messages (until removed)

- Key Terms
  - Namespace (i.e. cluster)
  - Hub (i.e. topic)
  - Publisher (i.e. producer)
  - Consumer (i.e. subscriber)



User Story 1-1: Provider Transfer Service Design

- Choosing Azure Function
  - Azure's event-driven, serverless compute option
  - Automatically scales – up to 200 parallel instances
  - We'll discuss other hosting options in later sprint

- Possible triggers
  - HTTP call
  - Timer
  - Message arrived / event occurred
  - Database record created, updated, or deleted
  - BLOB created, updated, or deleted

- Most often used in consumption mode – only charged for use
  - Based on number of executions, execution time, and memory used
  - Generally inexpensive for lightweight, infrequently used functionality

Task 1-1: Create Azure Event Hub

- Develop in Azure Portal
- Create Azure Hub Namespace
  - Name must globally unique (so we can't all use the same name)
  - Pricing tier: Standard so we can have multiple consumer groups
- Create Azure Hub for received requests
- Create Shared Access Policy for publisher

Task 1-2: Create Provider Transfer Service

- Simulate incoming requests by generating and sending randomly requests
- Start with ThreeAmigosHealth.sln in Sprint1
- Ensure logged into Azure (via CLI)
  - i.e. az login
- Add Azure Functions / c# with Timer Trigger
  - Timer will send a request each time it fires
  - Use EventHubProducerClient from Azure.Messaging.EventHubs nugget package
    - Requires *Shared Access Policy* connection string from *receivedRequests* hub
  - Business logic in BusinessLogic project in solution

Task 1-3: Deploy Provider Transfer Service

- Publish from Visual Studio to Azure

- Create Publish Profile
  - ***App Service name must globally unique***
  - Requires Azure credentials
  - Azure Windows / Linux
  - Double check Account, Subscription, Resource Group, and Location
  - New Function App
  - New Storage Account

- Publish

- Ensure messages making it to *receivedRequests* hub

Sprint 1: Retrospective

- We have the beginning of our utilization management platform
  - Incoming requests are available to any platform components
- We are taking advantage of asynchronous messaging
  - Publishing requests with little concern for who will consume them or how many consumers there will be
- We are using PaaS options to minimize development and maintenance work
- Everything about the platform is scalable