Exercise 2

Pro/Con of each approach

Stream analytics approach

Pro

PAAS - operational overhead should be minimal
Opportunity to learn new technology
Easier to troubleshoot - less pieces

Con

Cost - PAAS systems
Learning curve - new technology to learn
Stream analytics technology didn’t feel like a good fit at this time
- Has a focus on IOT technology we wouldn’t be using for this solution
- Output lists mostly other Azure resources - want to share data with other AA groups

No control when things go wrong - if there are platform issues it is out of our control

Service approach

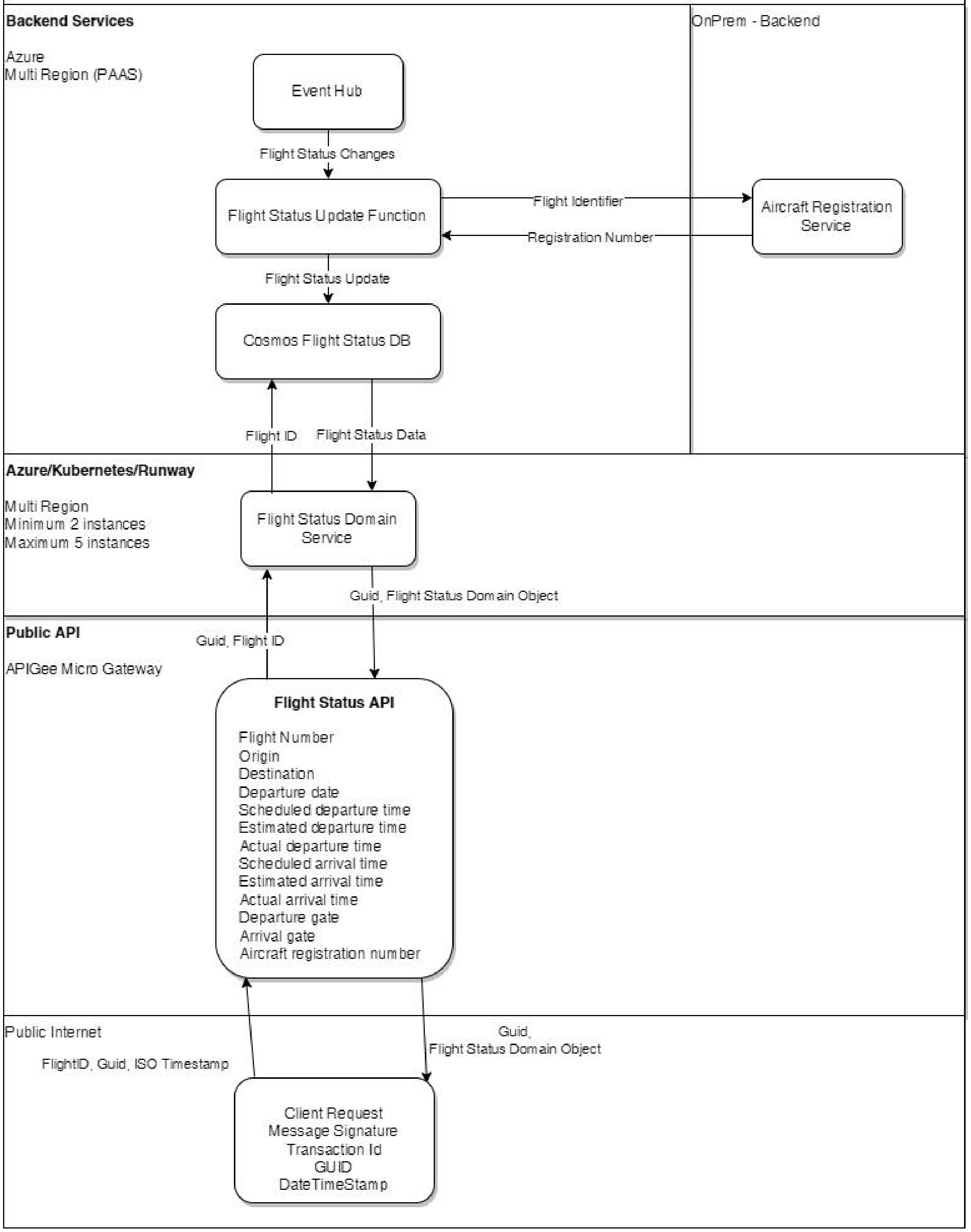
Pro

Familiar architecture / fits with existing support model
Lower Cost
Flexible public facing interface
Can use Runway

Con

More operational support and setup required
Possibly some latency due to web calls and layers

Exercise 2



Backend Services
Azure
Multi Region (PAAS)
OnPrem - Backend
Event Hub
Flight Status Changes
Flight Status Update Function
Flight Identifier
Registration Number
Aircraft Registration Service
Flight Status Update
Cosmos Flight Status DB
Flight ID
Flight Status Data
Azure/Kubernetes/Runway
Multi Region
Minimum 2 instances
Maximum 5 instances
Flight Status Domain Service
Guid, Flight Status Domain Object
Public API
APIGee Micro Gateway
Guid, Flight ID
Flight Status API
Flight Number
Origin
Destination
Departure date
Scheduled departure time
Estimated departure time
Actual departure time
Scheduled arrival time
Estimated arrival time
Actual arrival time
Departure gate
Arrival gate
Aircraft registration number
Public Internet
FlightID, Guid, ISO Timestamp
Guid,
Flight Status Domain Object
Client Request
Message Signature
Transaction Id
GUID
DateTimeStamp

Explanation of design:

Took the best parts of both proposed designs. Backend service functionality was isolated and separated using a function app to populate the data in the database. The database serves as the system of record that will be updated all from the back end.

Because there is no user interaction with the application, an API would be created using Spring Boot to provide access to the data from outside systems. API management (who can and will be calling the API) would be handled by APIGee.

Backend service layer:

Function app:

- Kafka connection from the event hub to handle incoming messages
- Hits the on-prem service as needed to get registrations for aircraft - may need to do some APIgee work to get this securely exposed to Azure.
- Scaled abstractly by the platform so any changes in load should be handled automatically

Data layer:

Cosmos Database which can be multi-region and also will dynamically scale to handle changes in load

Front end service layer:

Kubernetes / Runway:

A scalable multi region spring boot app running in Runway would be hosting the front end API.

Public Gateway:

APIGee would be used for API Management

Provides:

- Security
- Monitoring