AnyAirline

A Case Study for APDevL2 and potentially APAIntSols  
Version of 2019-02-05

[Introduction](#_nbnkhdrh33re)

[Requirements](#_l7jd2fo768tr)

[Functional Requirements](#_mm76p62iepff)

[Terminology](#_1abtb6tg2mcu)

[Actors](#_hgn1e18avgu4)

[User Stories](#_8y43grzc4jop)

[US1: Mobile Check-In](#_30ytc8k3vil0)

[US2: Flight Cancellation Mobile Notifications](#_8phzcttbrji4)

[US3: Offline Check-In Submissions](#_nr9eiwunmr7g)

[Non-Functional Requirements](#_4yxd61uqbxhz)

[Solution Architecture](#_6uwosu9qt9ub)

[High-Level Architecture](#_fk5ndadqxyr1)

[User Story Realizations](#_swc426egkm4p)

[US1: Mobile Check-In](#_4wesfunuy5un)

[Overview](#_x7w0d7czj930)

[PayPal Payments](#_1gw36umyk2w0)

[Detailed Component Interactions](#_297gkigpl1y9)

[US2: Flight Cancellation Mobile Notifications](#_9aq2awenj5mh)

[US3: Offline Check-In Submissions](#_7w3sxx61kxsb)

[Deployment](#_2xvqin4hzax0)

[Infrastructure Required For This Case Study](#_123a4xctgj0i)

[Document Version History](#_hj5y8r3gx1cu)

# Introduction

AnyAirline is a regional airline and an existing MuleSoft customer. They use the MuleSoft-hosted Anypoint Platform control plane, and a mixture of CloudHub and customer-hosted Mule runtimes to support a Mobile app and a few other, ad-hoc integrations.

This case study focuses on a small selected number of use cases within this system landscape.

# Requirements

Preexistent systems with which some integration is required, out-of-scope for this phase of solution design and implementation:

* Mobile app
  + AnyAirline’s native mobile app, the main strategic driver for this project.
* PayPal
  + For customer/passenger payments.
* Flights Management system
  + Accessible via SOAP web services over mutually authenticated HTTPS.
  + Deployed on-premises in the AnyAirline data center.
* Passenger Data system
  + In-house legacy PostgreSQL database to be accessed directly.
  + Deployed on-premises in the AnyAirline data center.
* Salesforce CRM
  + Recently introduced.

## Functional Requirements

### Terminology

* **Record Locator**: A 6- or 7-digit [alphanumeric code that identifies a data record in an airline reservation system](https://en.wikipedia.org/wiki/Record_locator), e.g., RW4TAB or KZVGX5. The term “Record Locator” is usually used to refer to a PNR. In this case study we do not make use of the term “Record Locator” itself.
* **PNR**: [“Passenger Name Record”](https://en.wikipedia.org/wiki/Passenger_name_record), contains personal information about a passenger, as well as their itinerary (i.e., flights). For simplicity, in this case study, itineraries consist of only one flight, so that each PNR **identifies exactly one passenger and their flight**. Furthermore, in this case study, we do not deal with the Passenger Name Records themselves, just with Record Locators identifying those Passenger Name Records. Therefore, in this case study, the term PNR is used to always mean **“Record Locator referring to a Passenger Name Record”**.

### Actors

* Passenger
  + A customer of AnyAirline, in possession of an AnyAirline ticket.
* 3rd party ground handling check-in partner company
  + An external company that partners with AnyAirline to provide 3rd party ground handling check-in for AnyAirline passengers.

### User Stories

#### US1: Mobile Check-In

As a passenger, I want to be able to check-in to an AnyAirline flight using the mobile app, by providing my PNR and last name, paying for any baggage using Paypal.

Preconditions:

* Passenger holds an AnyAirline ticket and knows its PNR.
* Passenger has a PayPal account.
* Passenger has mobile app installed.

#### US2: Flight Cancellation Mobile Notifications

As a passenger, I want to be notified through the mobile app if a flight to which I've checked-in is cancelled.

Preconditions:

* Passenger has checked-in to the flight using the mobile app.

#### US3: Offline Check-In Submissions

As a 3rd party ground handling check-in partner company, I want to be able to submit all check-in data that was accumulated while being offline.

Context:

* Ground handling check-in must continue even if ground handling systems are offline due to an outage, so that normal on-line check-in using AnyAirline’s on-line systems can not be performed.
* After coming online again, ground handling partner companies must submit to AnyAirline data about all check-ins performed during the offline period.

Preconditions:

* Company is a known 3rd party ground handling check-in partner of AnyAirline.
* Company has suffered an outage so that they couldn't use AnyAirline online check-in services.
* Company has continued checking-in passengers while offline.
* Company has restored full connectivity.
* Company now wants to send all check-in data accumulated during the offline period to AnyAirline.

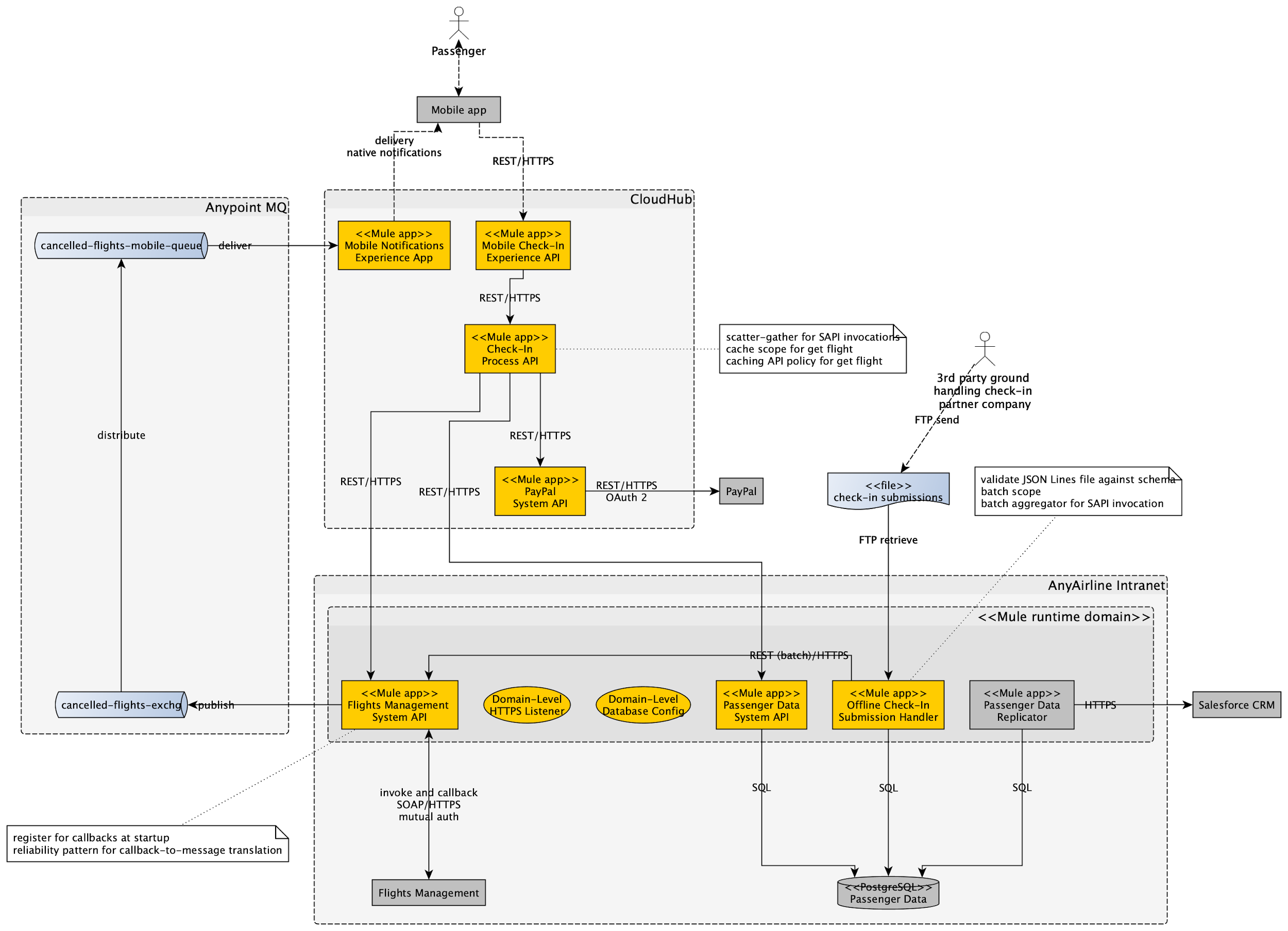
## Non-Functional Requirements

* NFR1: Application components interacting directly with on-premises systems (Flights Management, Passenger Data) must be deployed on-premises in the AnyAirline data center. This applies, for instance, to System APIs.
* NFR2: All application components interacting only with cloud-hosted systems (PayPal, Salesforce) or APIs must be deployed to CloudHub.
* NFR3: All data must be encrypted in flight (e.g., using HTTPS).
* NFR4: API-led connectivity should be followed unless performance considerations suggest otherwise.
* NFR5: (Optional) All application components must write audit log entries following the common JSON audit log format.

# Solution Architecture

## High-Level Architecture

The following free-form diagram gives an overview of the components under design and development (shown in yellow and blue) and the preexisting systems to integrate with (or otherwise out-of-scope in this project phase; shown in grey). Comments allude to implementation features of some of the application components, some of which will be addressed in more detail later.



## User Story Realizations

### US1: Mobile Check-In

#### Overview

#### PayPal Payments

Integration with PayPal is

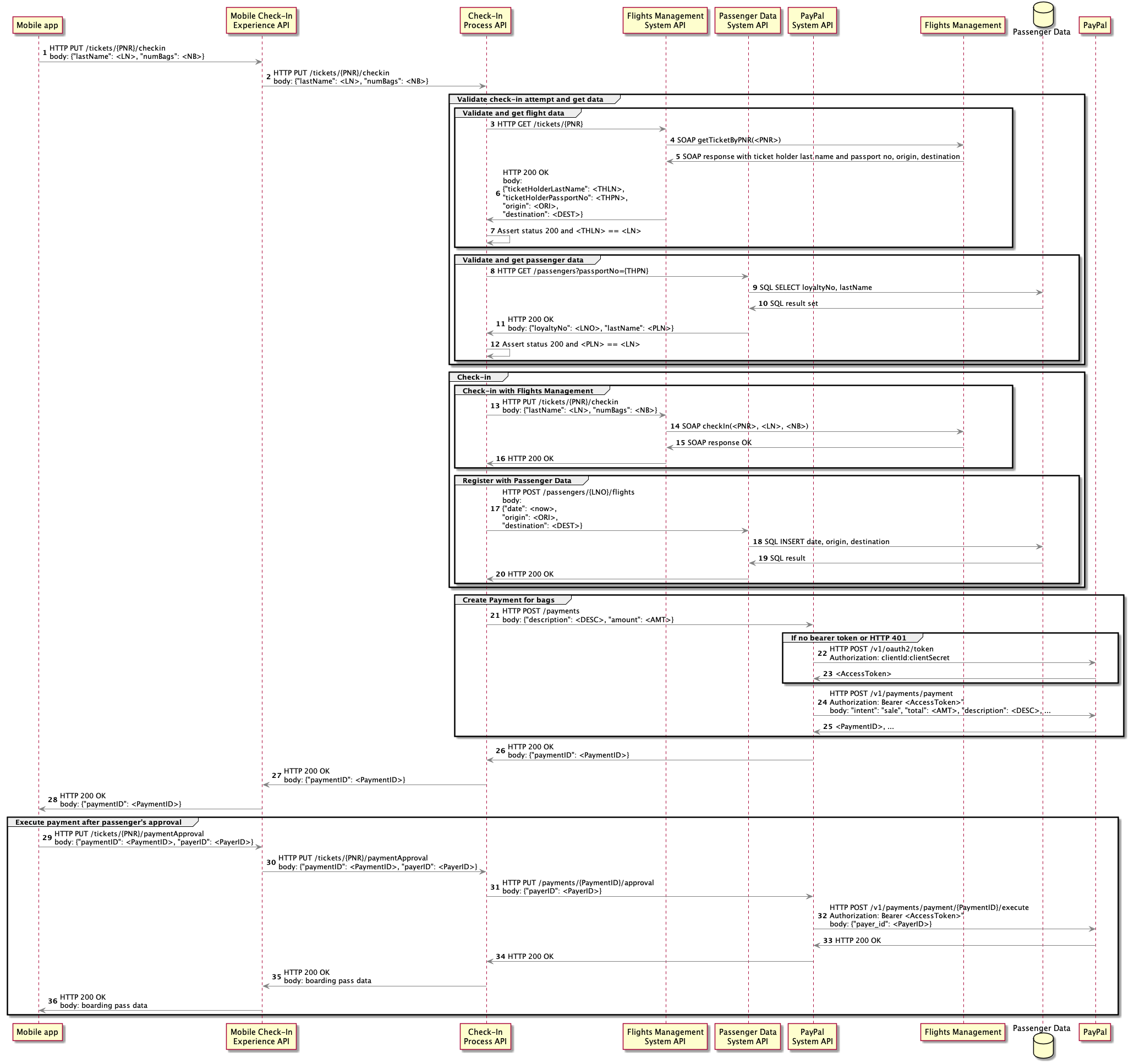
* Via PayPal REST APIs from the “PayPal System API” to [*create*](https://developer.paypal.com/docs/integration/direct/payments/paypal-payments/#create-paypal-payment) and [*execute*](https://developer.paypal.com/docs/integration/direct/payments/paypal-payments/#execute-payment) a PayPal Payment
* Via the PayPal web-based [“Check-Out Button”](https://developer.paypal.com/docs/checkout/how-to/server-integration/#1-set-up-your-client-to-call-your-server) from the Mobile app to [*approve*](https://developer.paypal.com/docs/integration/direct/payments/paypal-payments/#get-payment-approval) a PayPal Payment

A PayPal Payment must first be *created* by the “PayPal System API” via a REST API invocation. The Payment ID created by PayPal must then be used in the Mobile app to present the passenger with a UI to *approve* that payment. PayPal thereby identifies the payer and hands its Payer ID to the Mobile app. To *execute* a Payment, the “PayPal System API” must pass both the Payment ID and the Payer ID to PayPal, again in a REST API invocation.

Invocation of the PayPal REST API requires the creation/registration of a PayPal app for the “PayPal System API”, which means that PayPal generates a pair of client ID and Secret for that System API for both the Sandbox and Production environments:

* Client ID and Secret must be [exchanged](https://developer.paypal.com/docs/api/overview/#get-an-access-token) for a temporary bearer access token.
* All other, subsequent PayPal REST API invocations must [present that bearer access token](https://developer.paypal.com/docs/api/overview/#make-rest-api-calls) in the HTTP Authorization header.

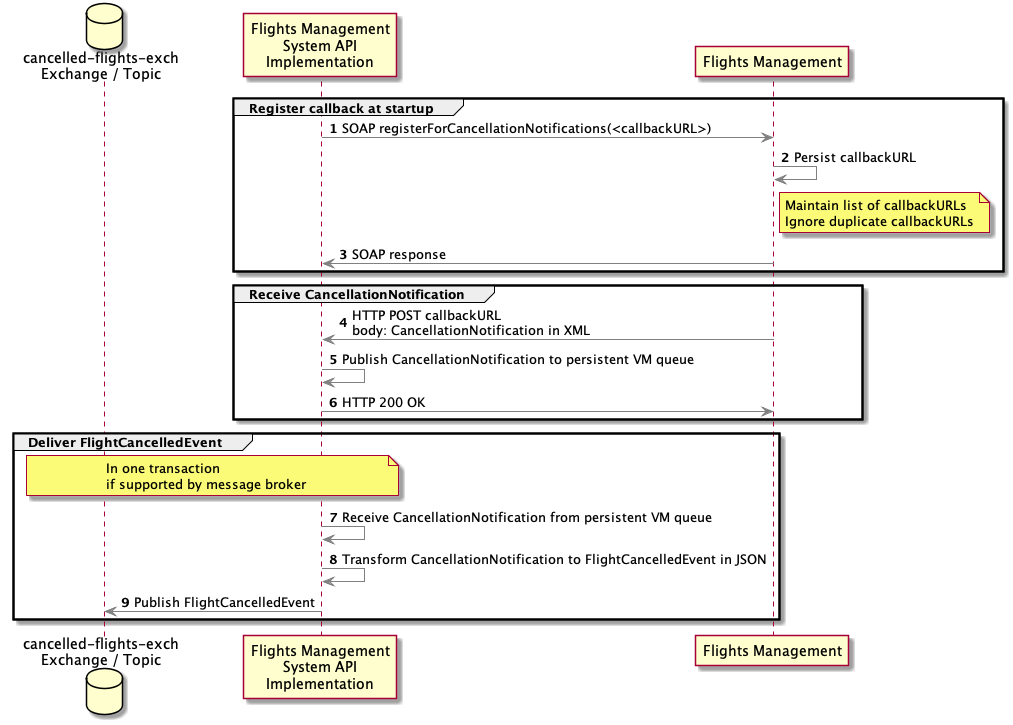
#### Detailed Component Interactions



### US2: Flight Cancellation Mobile Notifications

1. From AnyAirline’s perspective, flight cancellations appear first in the Flights Management system.
2. SOAP clients to the Flights Management system, such as the relevant System API, can register a webhook (HTTP callback), which will then be invoked by the Flights Management system when a flight cancellation occurs. That registration should occur at startup of the API implementation (duplicate registrations are ignored).
3. The webhook is used to deliver a notification from Flights Management to the System API in that the former sends a HTTP POST to the latter. One notification for each previously successful check-in, now affected by the cancellation, is sent. Each notification contains PNR and last name of the passenger:  
   <CancellationNotification>  
    <PNR>RW4TAB</PNR>  
    <PassengerLastName>Smith</PassengerLastName>  
   </CancellationNotification>
4. The System API uses the reliability pattern to accept that notification and immediately publish it onto a persistent VM queue. Then, asynchronously, in an XA transaction, the System API unqueues the notification, transforms it to a backend-neutral FlightCancelledEvent and publishes that event to a JMS topic (or Anypoint MQ Exchange, without transaction) to deliver it to interested parties. A FlightCancelledEvent is JSON-formatted:  
   {  
    "pnr": "RW4TAB",  
    "lastNameOfPassenger": "Smith"  
   }
5. Events are then delivered via publish-subscribe messaging (using Anypoint MQ) to an Experience-layer app, which delivers them to the mobile app via native mobile notifications (such as Apple’s APNs).

The essential parts of this interaction are shown in the following sequence diagram.



### US3: Offline Check-In Submissions

* Files are submitted per FTP and are in [JSON Lines format](http://jsonlines.org):
  + Each submissions file contains 0 to arbitrary many (typically: tens of thousands) records, one record on each line.
  + Each record contains data about a passenger’s check-in.
  + Each record is a stand-alone JSON document that is formatted as one single line.
  + Each record is independent of each other record. There is no need to deal with records that pertain to duplicate check-ins of the same passenger to the same flight.

## Deployment

Deployment of Mule apps is to a mix of CloudHub and customer-hosted Mule runtimes, as alluded to in [High-Level Architecture](#_fk5ndadqxyr1).

On-premises, customer-hosted Mule runtimes are clustered for HA and have domain-deployed shared resources:

* HTTPS listeners for exposing API endpoints and
* Database configurations (incl. JDBC connection pools) for accessing the Passenger Data system.

CloudHub deployments are to the public worker cloud in US-West-1 under the control of the MuleSoft-hosted control plane in the US.

# Infrastructure Required For This Case Study

Stepping outside the case study scenario, the following is needed to support this case study:

* Passenger Data system, as a PostgreSQL database with suitable table(s).
* Flights Management system, as a publicly accessible Mule application (deployed to CloudHub) exposing a suitable SOAP/HTTPS API and enforcing TLS mutual auth.
* PayPal mock, an approximate mock implementation of the relevant PayPal API endpoints, as a publicly accessible Mule application (deployed to CloudHub), exposing a suitable REST/HTTPS API and enforcing OAuth 2.
* Message broker, such as Active MQ or Anypoint MQ, publicly accessible to all students.

# Document Version History

|  |  |  |
| --- | --- | --- |
| **Date / Version** | **Author** | **Changes** |
| 2019-02-05 | Gerald Loeffler | Clarified distinction between “PNR” and “Record Locator” and that in this case study “PNR” always means “Record Locator for a Passenger Name Record” (prompted by Roger’s feedback); elaborated on CancellationNotification vs FlightCancelledEvent |
| 2019-01-31 | Gerald Loeffler | FIxed notifications registration (via SOAP not REST) and callback (XML not JSON, one notification per check-in) |
| 2018-12-17 | Gerald Loeffler | Clarified use of reliability pattern to implement US2; specified payment via PayPal |
| 2018-12-11 | Gerald Loeffler | First rough version |
| 2018-12-10 | Mark Nguyen, Ethan Port, Gerald Loeffler | Agreement on basic case study scenario, components and use cases |