DUO – <CustServ Replatform>

High level Architecture Blueprint Document

Version 0.1

Arturo Velez, Solutions Architect

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 4/29/2016 | 0.1 | Initial draft | A Velez |
|  |  |  |  |

Table of Contents

1. Introduction 4

Purpose 4

Scope 4

Definitions, Acronyms, and Abbreviations 5

References 5

Assumptions 5

2. Logical View 6

Logical view diagram 6

Overview 6

3. Design Considerations 7

Logical Overall Flow 7

Mule MuleSoft 7

Invoice Reconciliation Process 7

Contact Reconciliation Process 8

4. Reusability considerations 9

5. Deployment View 12

Maven 13

Jenkins 14

6. Logging for Alerting and Analytics 14

Logging framework design 14

Splunk Configuration 14

7. Security Architecture 15

HTTPS/TLS Support 17

**One-way TLS** 18

**2-way TLS (aka Mutual Authentication)** 20

**Roadmap (Subject to Change)** 20

# Introduction

DUO has requested MuleSoft to assist during the planning of their MuleSoft environment and the deployment for their MuleSoft iPasS solutions as well as DUO’s first two sprints which is anticipated to integrate Recurly, SalesForce and TrustedPath. The primary use case is replacing today’s DUO specific Python integrations, with standard MuleSoft’s services.

## Purpose

This document provides a comprehensive architectural overview of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

## Scope

* This project is part of a larger initiative, which in the future will make changes to some of the assumptions made here, for example, the use of events (e.g. webhooks) etc.
* Creating shell or other non-MuleSoft scripts which are directly or indirectly invoked by MuleSoft is out of scope. MuleSoft team will work with appropriate teams to create a pattern for accessing and executing these scripts from Mule.
* The document doesn’t delve into detailed network architecture. The DUO network team will provide specific details and recommendations on network specific decisions, such as firewall configuration, ports, etc.
* oAuth or other federated security/authorization mechanisms are not in scope for this project

## Definitions, Acronyms, and Abbreviations

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Recurly | Recurly (https://recurly.com/) is a company dedicated to. Among other things, billing management, metered billing, subscription management, etc. |
| TrustedPath | TBD. |
| SalesForce | SalesForce () is a very well known CRM company, which DUO uses for Account, Opportunity, Contact, etc. management |

## 

## 

## References

|  |  |  |
| --- | --- | --- |
| **Author** | **Document Name** | **Description** |
| Ron Martin | Python code | Code written in Python to do current integrations |
| Todd Masura | App Ecosystem | Current logical diagram of apps |
|  |  |  |

## Assumptions

The following assumptions were made in architecting the solution

* MuleSoft services will reside in Cloudhub.
* Other <insert here>.

# Logical View

## Logical view diagram – DUO ecosystem



## Overview

The diagram represents the logical flow of data among the DUO systems involved in the project, with the following use cases:

# Design Considerations

## Logical Overall Flow

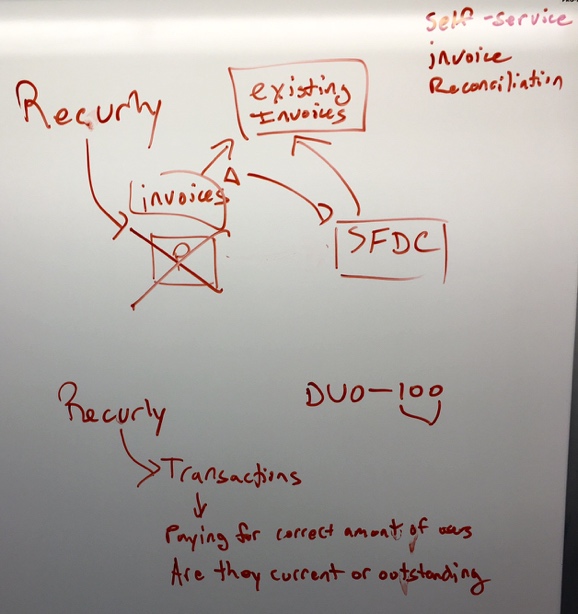
A description of the overall inbound flow, here.

## Mule MuleSoft

MuleSoft is the enterprise service bus tying all components together through a hub-and-spoke architecture. MuleSoft exposes services offered by every component in a platform, protocol and language neutral way thus enabling universal connectivity. Also this avoids what is known as spaghetti code

## Invoice Reconciliation Process

<copy studio flow picture here>



Deployment Scope

CloudHub

Signature

Input -

Output -

Transformation considerations

<describe flow here>.

Exception handling

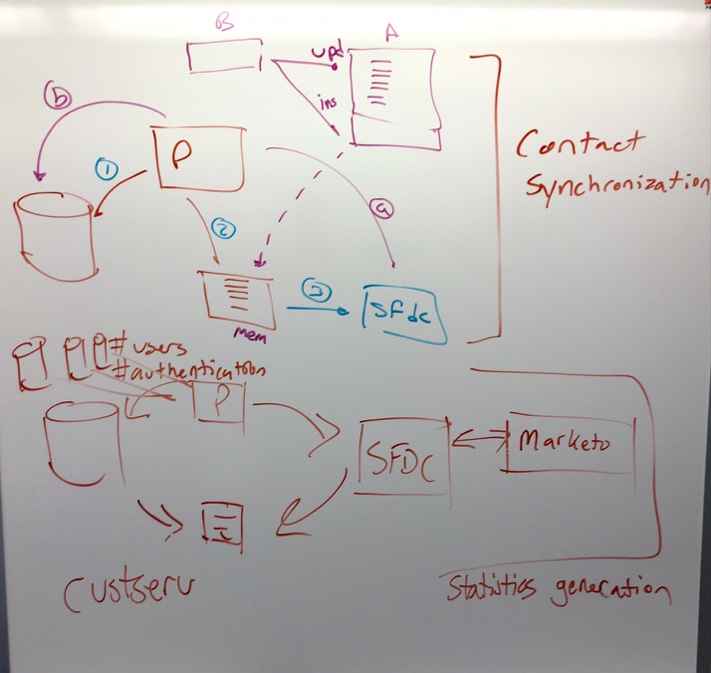
<describe error handling here>.

Flow Logic

1. <pseudocode goes here>

## Contact Reconciliation Process

<copy studio flow picture here>



Deployment Scope

CloudHub

Signature

Input -

Output -

Transformation considerations

<describe flow here>.

Exception handling

<describe error handling here>.

Flow Logic

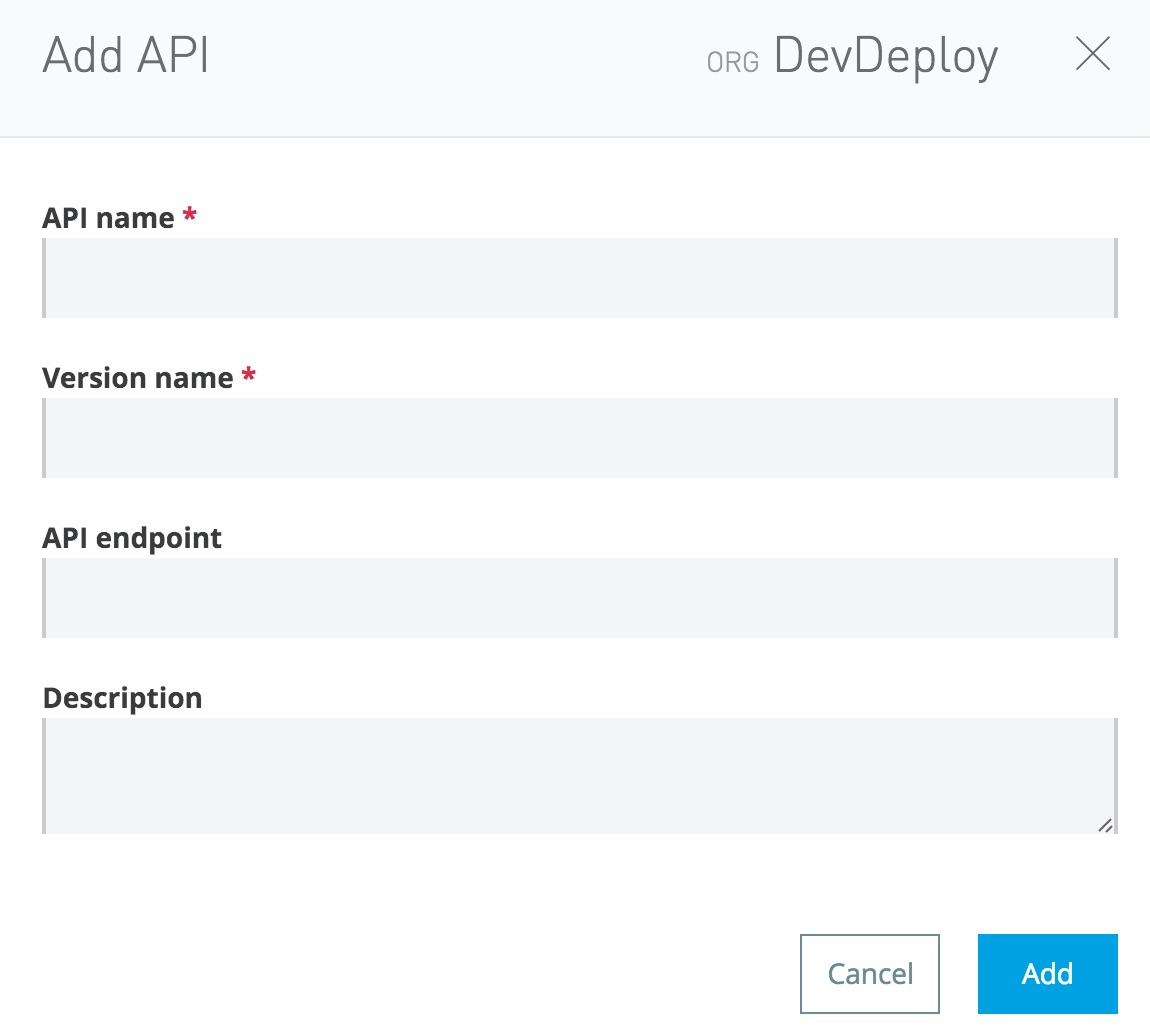
1. <pseudocode goes here>

# Reusability considerations

MuleSoft envisions an API-led Connectivity approach to allow for an agile, composable architecture. This is important because rather than taking a shortcut to write directly to a DB or having to support a legacy SOAP service, the core of MuleSoft integration can be designed by creating modern APIs that can be measured and monitored, in addition to governance through policies, among other things.

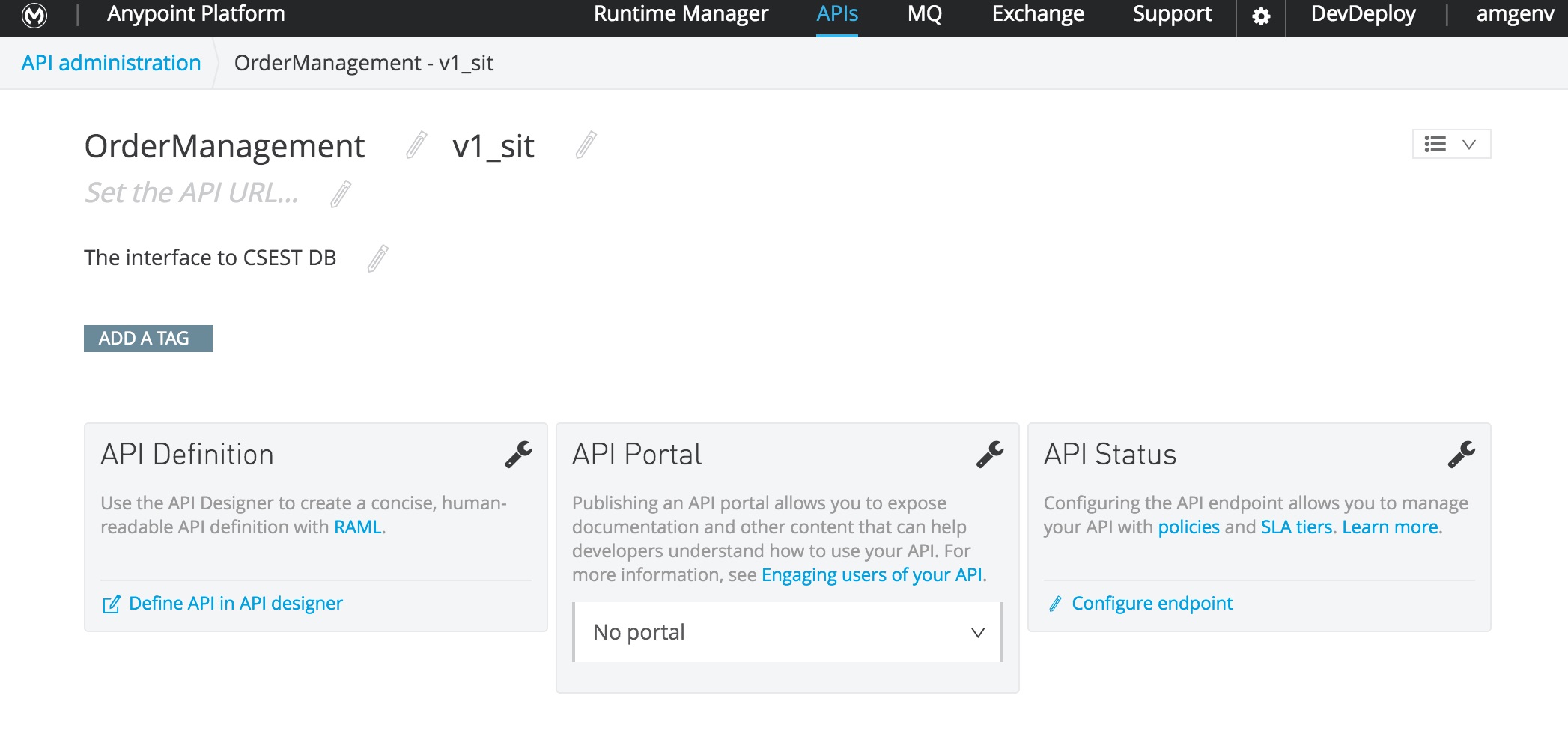
API design begins in the Anypoint Platform with the API Designer, which you use to create a RAML definition. RAML (RESTful API Modeling Language) defines the API contract detailing how the API should be consumed. It defines the API resources exposed, the request and response message details, expected success and error return codes, and security traits. More details on RAML can be found on <http://raml.org>.

The very first step is to “add” an API from the main Anypoint API administration page by selecting the Add API button. This step allocates the API definition from which the RAML definition, API Portal, and API endpoint definition is created or configured.

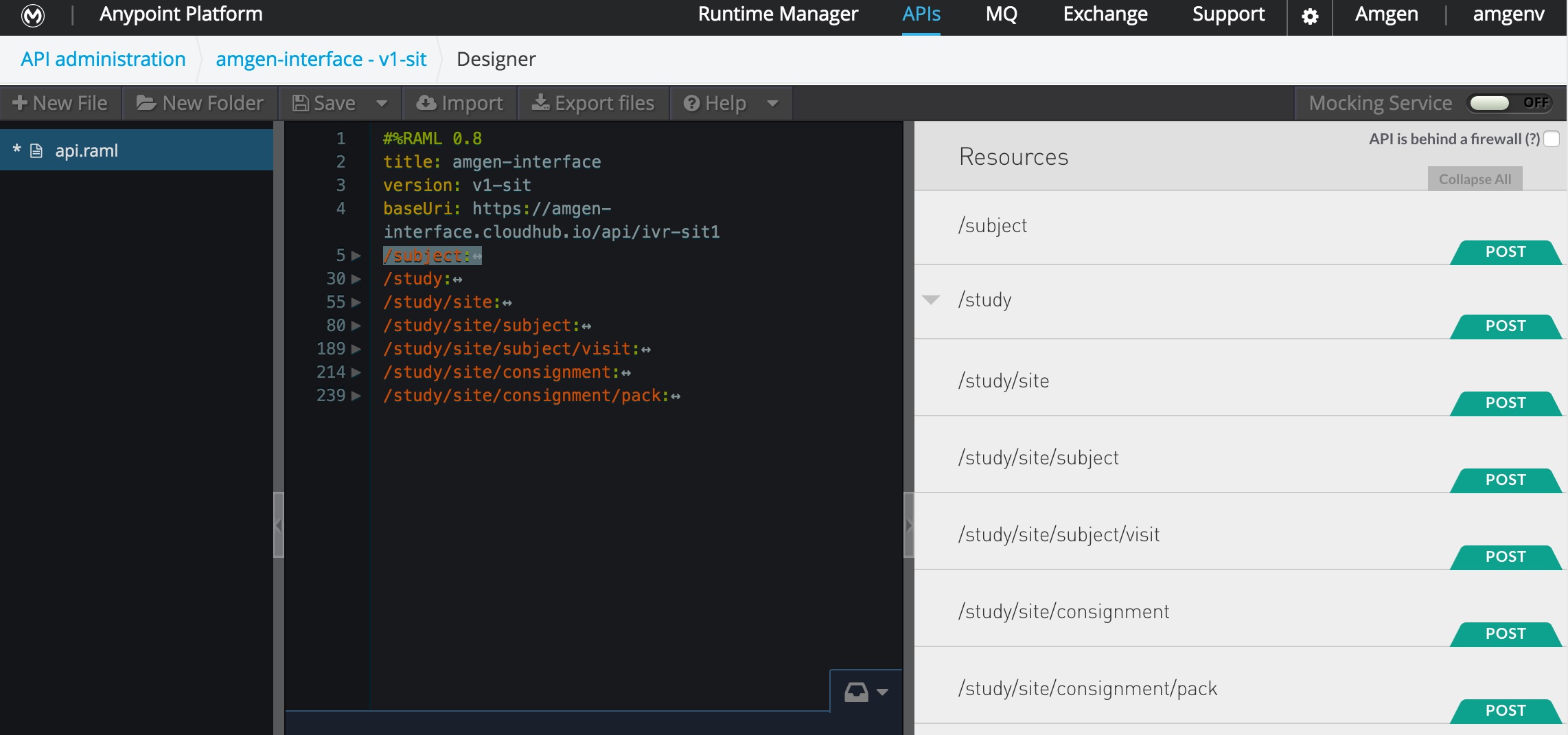


Note that the current practice is to use the “Version name” field to differentiate API definitions that correspond to different environments.

Once the API is added, then you can start editing the corresponding RAML, like this:



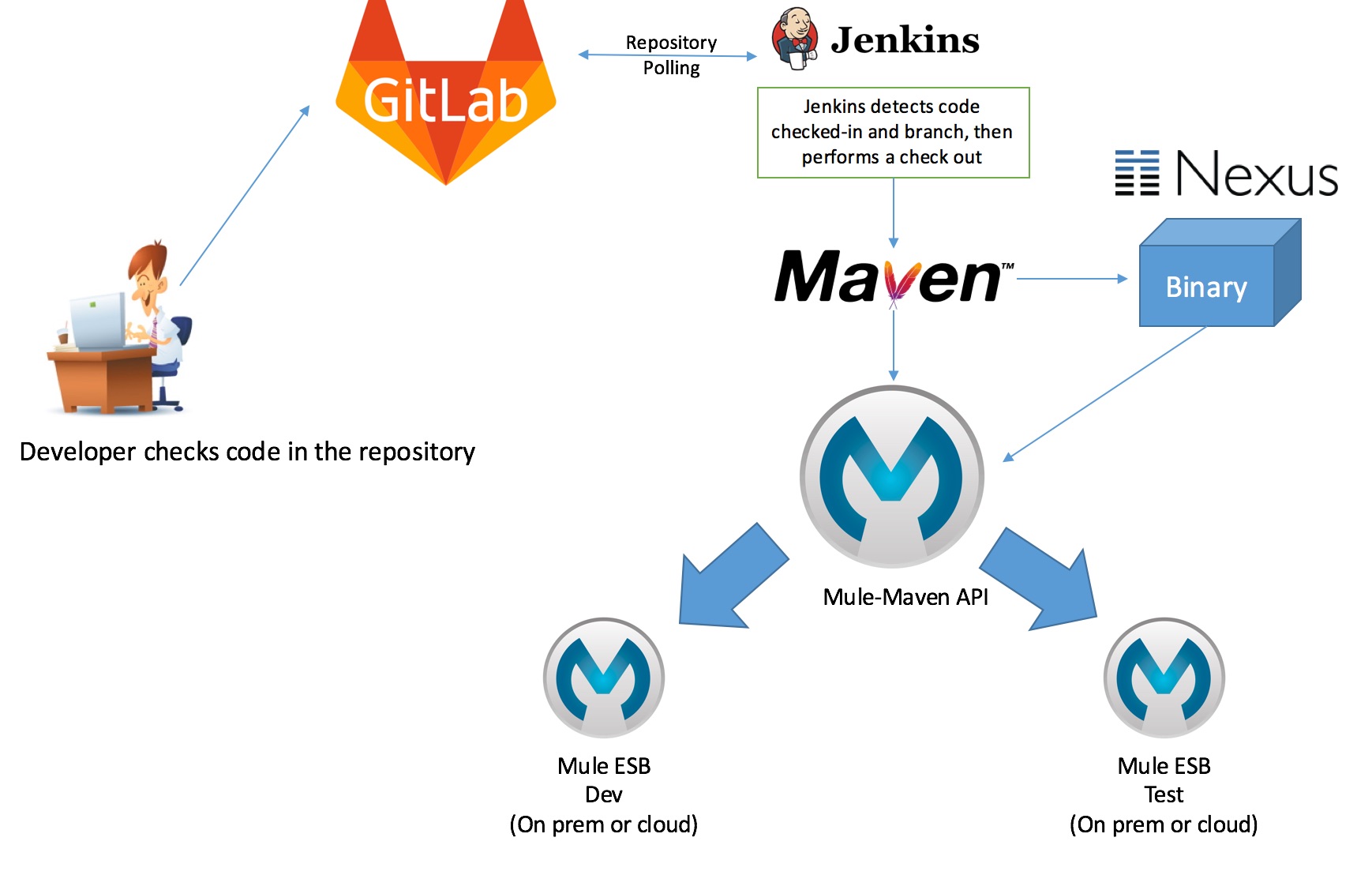
We strongly recommend that you use RAML to model your services, use the API portal to share it and import it into Studio to create the projects for later coding. A sample of RAML we used is shown here:



The inputs and structure of the APIs will be defined later in the project

# Deployment View

The diagram below represents the deployment view of the MuleSoft platform in development & build environment



1. Developer will check in code into Github
2. Either automatically (polling) or manually (via manual deployment) Jenkins will pull code from Github, call Maven to make a build, push it to Nexus
3. Jenkins will use the Mule-Maven API to push the build to cloudhub

## 

For more information about Deploying Mule, please read: <https://docs.mulesoft.com/mule-user-guide/v/3.7/mule-maven-plugin>

## Maven

Maven will be used for both building of the binaries, along with deployment to CloudHub and Mule on-premise. Deployment to either of these environments requires an entry in the application project POM file, providing environment details.

To deploy to CloudHub, include the following entries into the plugins section of the POM:

<plugin>

<groupId>org.mule.tools.maven</groupId>

<artifactId>mule-maven-plugin</artifactId>

<version>2.1</version>

<configuration>

<workerType>Micro</workerType>

<workers>1</workers>

<businessGroup>DUO\DevDeploy</businessGroup> <!--configure as necessary-->

<deploymentType>cloudhub</deploymentType>

<muleVersion>3.7.3</muleVersion>

<username>devdeploy</username> <!--configure as necessary-->

<password>password</password> <!--configure as necessary-->

<redeploy>true</redeploy>

<environment>SIT</environment> <!--configure as necessary-->

</configuration>

<executions>

<execution>

<id>deploy</id>

<phase>deploy</phase>

<goals>

<goal>deploy</goal>

</goals>

</execution>

</executions>

</plugin>

To deploy to on-premise, include the following entries into the plugins section of the POM:

<plugin>

<groupId>org.mule.tools.maven</groupId>

<artifactId>mule-maven-plugin</artifactId>

<version>2.1</version>

<configuration>

<businessGroup>DUO\DevDeploy</businessGroup> <!--configure as necessary-->

<target>mule1</target> <!--configure as necessary-->

<deploymentType>arm</deploymentType>

<targetType>server</targetType>

<muleVersion>3.7.3</muleVersion>

<username>devdeploy</username> <!--configure as necessary-->

<password>password</password> <!--configure as necessary-->

<redeploy>true</redeploy>

<environment>SIT</environment> <!--configure as necessary-->

</configuration>

<executions>

<execution>

<id>deploy</id>

<phase>deploy</phase>

<goals>

<goal>deploy</goal>

</goals>

</execution>

</executions>

</plugin>

## Jenkins

With the above entries in the POM file, Jenkins can be used to deploy to either CloudHub or on-premise using Maven commands as shown below:



# Logging for Alerting and Analytics

## Logging framework design

DUO has <insert logging framework here>:

Audit Login

MuleSoft provides an API to access the Audit Logs for tracking purposes. The Information can be found under this [link](https://docs.mulesoft.com/anypoint-platform-administration/audit-logging). The [API Reference guide](https://anypoint.mulesoft.com/apiplatform/anypoint-platform/#/portals/organizations/68ef9520-24e9-4cf2-b2f5-620025690913/apis/24562/versions/26089/pages/39847) provides full documentation on the Auditing API.

### Splunk Configuration

During the engagement, we configured Splunk for <insert here>:

# Security Architecture

This section of the document covers some of the main security considerations when deploying Mule applications to the DUO CloudHub landscape. It is not meant to cover all the security aspects of Mule and CloudHub, but mainly a few of the main security topics that are relevant for DUO including security and authentication schemes, details on enabling the [External OAuth 2.0 Token Validation Policy](https://docs.mulesoft.com/anypoint-platform-for-apis/external-oauth-2.0-token-validation-policy) and Mule’s support for one-way and 2-way (mutual authentication) TLS.

Since its first release, the MuleSoft Anypoint platform has provided two ways of protecting APIs with OAuth 2.0: Either by using an external OAuth provider (PingFederate and OpenAM) or by using MuleSoft’s own OAuth provider available on the Anypoint platform. For MuleSoft’s Anypoint OAuth provider, a policy-based model has been used for the configuration of OAuth servers. With the release of API gateway 2.0, MuleSoft is deprecating the policy-based model in favor of a manual configuration model to overcome some limitations of the policy-based model:

● Generation and validation of tokens over HTTPS is not supported.

● Customized login screens (e.g. for authorization code grant type) is not supported.

● Configuration of the underlying security manager with non­-LDAP based user repositories (such as for example an RDBMS based user repository) is not supported.

The new model overcomes the limitations of the policy-based model and also allows for a single OAuth provider to be configured for multiple APIs.

More details on the new model are provided on the MuleSoft documentation portal: [External OAuth 2.0 Token Validation Policy](https://developer.mulesoft.com/docs/display/current/External+OAuth+2.0+Token+Validation+Policy).

To enable the new model, MuleSoft provides a ready-to-go OAuth 2.0 provider application template that customers can download from [Anypoint Exchange](https://anypoint.mulesoft.com/exchange/#!/api-gateway-external-oauth2-provider?orgId=1) and deploy onto their API gateway after a few configuration changes.

Once the template is downloaded and imported into Anypoint Studio, the provided mule.dev.properties (or whichever mule.${mule.env}.properties files you choose to use) must be edited. Full documentation for this process is available from the [Building an External OAuth 2.0 Provider Application](https://developer.mulesoft.com/docs/display/current/Building+an+External+OAuth+2.0+Provider+Application) documentation page.

The main OAuth 2.0 grant type being recommended for use during the initial phase of the MuleSoft implementation is Client Credentials. This grant type is the most suitable for machine-to-machine or application-to-application authentication from trusted clients.

Note that LDAP based authentication (if relevant) requires access to the DUO LDAP server from the DUO CloudHub VPC to be configured as well.

A sample mule.dev.properties file that would be used for DUO might have the following property values:

# Properties to be used on the development environment

key.store.password=mule123

key.store.key.password=mule123

key.store.path=keystore.jks

admin.name=admin

admin.password=admin

validate.endpoint.path=validate

authorization.endpoint.path=authorize

access.token.endpoint.path=access\_token

#Defaults for scopes are "READ WRITE"

scopes=

supported.grant.types=AUTHORIZATION\_CODE RESOURCE\_OWNER\_PASSWORD\_CREDENTIALS CLIENT\_CREDENTIALS IMPLICIT

The key entries correspond to the keystore that is also provided to go along with the template so the values as shown above must be used. The download for the keystore is available from the documentation page. This keystore.jks file must be added to the src/main/resources directory of the template Mule project.

The admin.name and admin.password credential properties can be set to your own values. They are used for the underlying, simple security provider provided in the application.

The three endpoint.path properties specify the URI paths that must be called for the indicated OAuth provider related function:

● validate.endpoint.path: Used by the validate OAuth token passed and enforce the policy configured for an API.

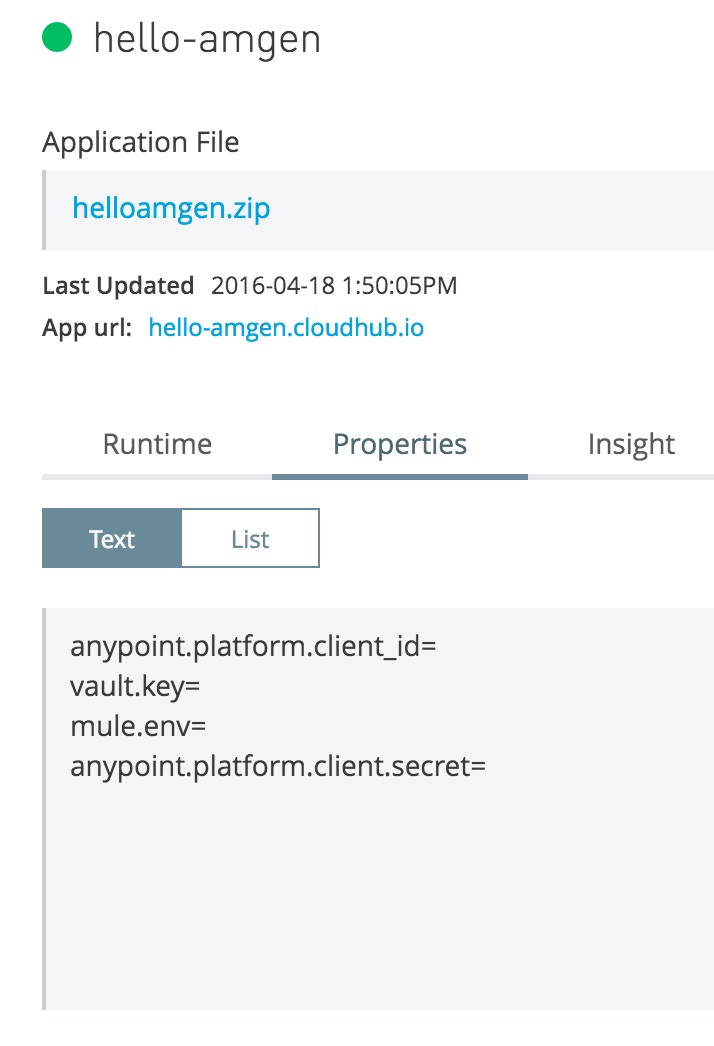
● authorization.endpoint.path: Used specifically for user approval based grant types such as Authorization Code to get the authorization from a user.

● access.token.endpoint.path: Used to retrieve the access token required when accessing the targeted resource – in our case the targeted API endpoint.

The scopes property specifies the supported scopes of this OAuth 2.0 provider.

The supported.grant.types, as implied, specifies all the OAuth grant types that this OAuth 2.0 provider will support.

Enter the correct parameters like application id, secret, environment, etc. corresponding to DUO’s parameters, for example:



Notice that the anypoint.platform.client\_id and anypoint.platform.client\_secret parameters correspond to DUO’s organization ID and secret

## HTTPS/TLS Support

Mule supports encrypted communication at the HTTP transport layer using TLS (Transport Layer Security). At a minimum, one-way HTTPS/TLS is enabled for any HTTP API or service endpoint exposed on the Mule CloudHub runtime requires HTTPS communication. This ensures that the host server exposing the HTTP endpoints are Mule CloudHub hosts and also enables TLS to encrypt the HTTP traffic between clients and the Mule servers.

2-way TLS or mutual authentication, where identity of the server and client is verified, is currently not fully supported on CloudHub. A workaround solution based on connections directly to the CloudHub worker instances instead of the CloudHub load balancer is possible to enable mutual authentication, but with the caveat that full load balancing and scale out based on the CloudHub load balancer is not enabled. However, support for full support of 2-way TLS is on the roadmap and a preview for that approach will be described.

### **One-way TLS**

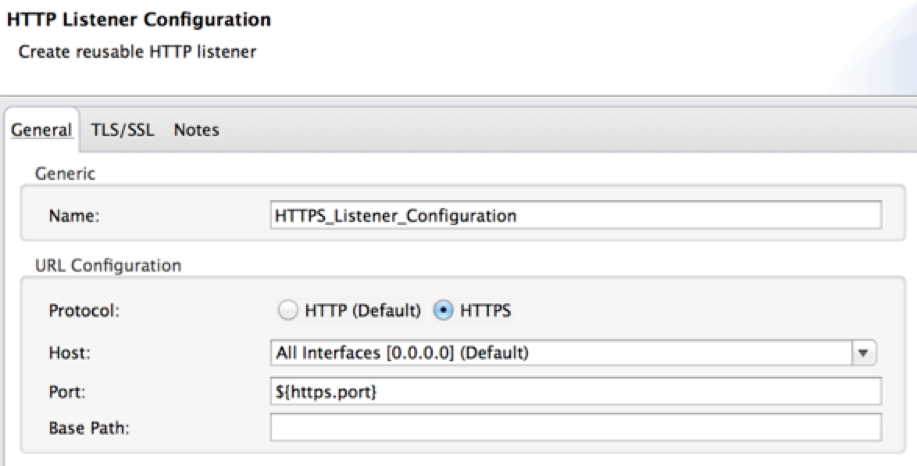
Enabling one-way TLS is straightforward for a Mule application, whether deployed to Mule on-premise or CloudHub. Both require a keystore that holds the public/private certificates to be used. The main difference is that for CloudHub, when the application is deployed, CloudHub will not use the certificates in the application keystore, but its own since applications deployed to CloudHub will be exposed using a “cloudhub.io” domain.

If you don’t have a keystore, you can simply generate one using the Java keytool command. An example is shown below.

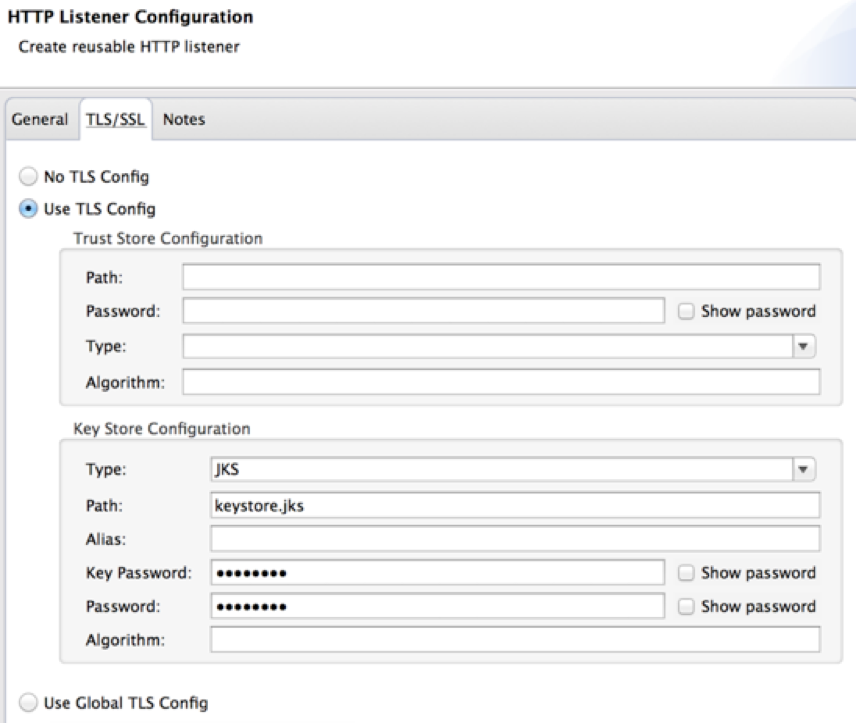
|  |
| --- |
| keytool -genkey -alias mulekey -keyalg RSA -sigalg SHA256withRSA -keystore DUO\_mule\_keystore.jks -keysize 2048 -dname "CN=DUO.com, OU=SAMPLE, O=DUO, L=Thousand Oaks, ST=CA, C=US" -keypass abcd1234 -storepass abcd1234 -validity 9999 |

Once generated, the keystore (e.g. “DUO\_mule\_keystore.jks”) should be copied to the Mule application folder “src/main/resources”. From there it can be referenced from the HTTP(S) connector configuration.

The HTTP Listener configuration should have “HTTPS” selected for the Protocol with a port that will be used for HTTPS communication specified. Note that this port is only valid for local testing on Studio. After deployment on CloudHub, the standard HTTPS port 443 is used when making the call.



The reference to the keystore, along with the key and store password must be configured in the TLS/SSL tab.



Once these settings are configured, after the application is deployed to CloudHub, the exposed endpoint will only be accessible via HTTPS and the traffic will be encrypted.

This applies whether the Mule application being deployed is a managed API (proxy or regular API) or a non-managed API (implementation API).

To prevent unauthorized access to these exposed endpoints, authentication mechanisms must be put in place for these APIs.

### **2-way TLS (aka Mutual Authentication)**

A standard deployment of a Mule application to CloudHub exposes the API or service endpoint on the CloudHub load balancer. One limitation with such a deployment is the support for 2-way TLS where a client also presents a certificate to verify its identity. The issue is that TLS termination takes place at the CloudHub load balancer and it does not verify client certificates.

There is a workaround for CloudHub to support 2-way TLS today, however, it requires the certificates to be bundled into the application and directly hit the worker on the public worker DNS address which should be a CNAME for **mule-worker-appname.cloudhub.io**. Moreover, this means that the TLS connection needs to be terminated directly at the worker and not at the CloudHub load balancer. It also means that the HA and worker scale out features of the platform will not be fully supported, since the load balancing performed by the CH load balancer will be bypassed.

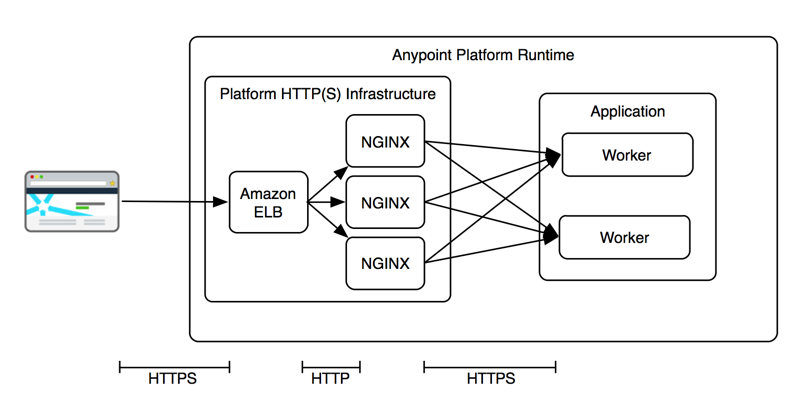
### **Roadmap (Subject to Change)**

MuleSoft is actively planning to address some of the current CloudHub limitations including 2-way TLS, customer-specific domains, and load balancing for applications deployed only for internal access from their designated VPC. The key piece in addressing these limitations is the use of a custom load balancer that would be provided by the CloudHub platform.

For full support of 2-way TLS, TLS termination needs to happen at the boundary of the customer application network - in the case of applications running on CloudHub, this would be the customer’s VPC perimeter.

Some customers may require having encryption all the way to the worker running the application. For this requirement, an option should be provided to either terminate the 2-way TLS connection at the VPC boundary and provide another encrypted connection between the LB and the worker, or have the 2-way TLS connection terminated at the worker instance.

Enforcing communication between any client and CH - normally, this could be achieved with IP whitelisting implemented in the MuleSoft app using IP filtering (<https://docs.mulesoft.com/mule-user-guide/v/3.7/anypoint-filter-processor>). But there are limitations in CH because the requests would actually be coming from Nginx servers - see below and not from client directly:



This page was left intentionally blank

# Appendix A – Recurly connectivity details

Recurly (<https://recurly.com/)> offers REST API on top of the various language APIs such as C#, PHP, Ruby, Python, among others. Python was the language of choice for the current DUO implementation in production. However, for the MuleSoft integration, we recommend using the simple REST API. This section will describe how to configure it within Anypoint Studio.

Recurly has good documentation regarding the REST API, which can be found here: https://dev.recurly.com/v2.1/docs .

Here is a sample configuration of the http connector within Anypoint Studio that retrieves a list of all Accounts:



The following parameters are defined in the mule-app.properties located under src/main/app:

http.port=8081

recurly.port=443

recurly.host=insert\_recurly\_user.recurly.com

recurly.APIkey=06e2c6dd2cbdfgjhfe37dc8b0f8

recurly.Accept=application/xml

recurly.Content-Type=application/xml

recurly.X-Api-Version=2.1

recurly.password=

The important notes here are:

* ${recurly.port} is not required since the URL typically would go to 443
* ${recurly.host} has embedded the *recurly user* plus *recurly.com*. Use the assigned user
* ${recurly.password} must be an empty string in order to use the API key.

The Recurly API Key can be found under <https://recurly_user.recurly.com/developer/api_keys>



where recurly\_user is your Recurly assigned user. There are two API keys, use the Private Key.

Host, port and Base Path are set in the global http connector, like this:



We chose the Base Path to be /v2 because the rest of the URI could be parametrized to include accounts, adjustments, invoices, subscriptions, plans, etc. Mule will construct the entire URL and assign headers to make the call. The resulting URL would look something like:

https://subdomain.recurly.com/v2/accounts

where subdomain, again, us the user assigned for your project.

Information about the account object, and any other object within recurly, is easily found under the corresponding documentation page, e.g. <https://dev.recurly.com/docs/account-object>.

Finally, the last piece of configuration is Basic Authentication, also part of the Global Connector as shown in here:



Like we mentioned before, the user name is the assigned recurly private Api key and the password must be a null string.