Review Checklist

# Design

|  |  |
| --- | --- |
| **Item** | **Details** |
| Does functionality already exist? | Is the functionality already provided in another API or library? If so, use that. |
| Separation of API from Implementation | There is appropriate separation of interface (API) from function (concrete services). |
| Use API-Led Design | Clear separation between experience, process and system APIs |
| Appropriate service decomposition | Service/microservice decomposition is logical and will not impede redeployments unduly |
| Scalability | Scalability has been designed into the solution - prefer stateless architectures that can scale horizontally without any intrinsic limits |
| Security | Security has been considered as an integral part of the design - as a first-class citizen - and using industry-standards |
| No monolithic service | There has been appropriate decomposition of functionality into deployable services - not just one large service that does everything. |
| Group functionality appropriately in services. | Review logical deployment diagrams in context with processing flow to ensure there is not excessive inter-node communication between services to accomplish the various tasks. |
| Create reusable assets | Make sure the developed process and system and system APIs can be reused by future consumers as well. |

# 

# General

|  |  |
| --- | --- |
| **Item** | **Details** |
| Use Naming Standards | * API * App (name in Runtime Manager) * Project * Files * Flows * Variables * Processors * Queues (Anypoint MQ, AMQP, JMS, VM) |
| Project structure conforms to standards | * Each APIkit in separate file. Flow-refs to actual functionality in other files for each endpoint. * Use shared RAML libraries for security, standard errors and health check. * Each functional flow in separate file with supporting flows. * Mule 4: Use a .yaml file for properties. * Properties files are kept in src/main/resources/properties * There is a file for each environment named <environment>.yaml and accessed in the app using ${mule.env}.yaml * Mule 4: Anypoint id/secret and mule vault key are added to secure properties attribute in mule-artifact.json * All configurations in global.xml file * File names conform to standards * Test/example payloads stored under *./src/test/resources* folder * Dataweave transforms stored in external files under *./src/main/resources/dwl* folder |
| Project is mavenized | Project must support maven builds and use the standard pom.xml. |
| Project builds & runs unit tests locally | The project must build and run all unit tests locally successfully. |
| Outbound connections have reconnection strategy | Set reconnection strategy on outbound connections that support it. Reconnections are small number of retries in short intervals, such as 3x1 second. Larger retry strategy as it pertains to message persistence is a different topic.  Mule 3: Use Until-Successful scope around HTTPS outbound requests for reconnection functionality. |
| Exception flows not in-line. | Use Exception References in Error handling section of flows. Exception flows are defined in separate XML files. |
| Exceptions handled at appropriate level. | Only catch exception via nested flow exception handler or try scope, if it is appropriate to handle at that location. Otherwise, let the exception move up the stack.  Always have an exception handler defined on the flow that starts the thread execution, i.e., any flow with an inbound connector. Examples: API Kit main flow or scheduler main flow. |
| Removed any unused flows/sub-flows | Any unused content in Mule configuration files must be removed before committing code. |
| All configurable items externalized in properties | Put configurable items, such as HTTP timeouts, in external properties. These properties go in properties files in *./src/main/resources/properties* per app or in a common location on on-premise runtimes. |
| Code Clarity | * Are the flows easily understood? * Are the flows as modular as possible? * Are there redundant flows? * Are there redundant transforms? |
| Use common patterns | * Error handling/response * HTTP Status * REST/RAML Best Practices |

# 

# Security

|  |  |
| --- | --- |
| **Item** | **Details** |
| Use HTTPS | Always use HTTPS instead of HTTP when connecting to systems and other APIs. TLSv1.1+.  HTTP requester config should **not** have the insecure option set when promoted to production. |
| Use appropriate auth for APIs | The API must be protected by appropriate auth. At the minimum, it needs to support Client ID Enforcement, which is Anypoint’s client credentials over Basic Auth in request header. Auth is specified in RAML using securedBy: [<trait list>].  The preferred auth uses multi-factor authentication mechanism for authentication with the security provider integrated with Anypoint Platform and APIs. The preferred solution for that is [OpenID Connect](https://openid.net/connect/), including [Anypoint Platform access management](https://docs.mulesoft.com/access-management/conf-openid-connect-task), and [API policies](https://docs.mulesoft.com/api-manager/2.x/openam-oauth-token-enforcement-policy). |
| Sensitive properties must be encrypted. No passwords in plain text! | Make sure all sensitive properties, such as user name, password, etc., are encrypted in a properties file, which is imported by Secure Configuration Properties. Encrypt the properties through the Properties Encryption Service. |
| Use Auth for calling systems | Use proper auth when connecting systems. |

# 

# Logging/Monitoring

|  |  |
| --- | --- |
| **Item** | **Details** |
| Only ERROR or WARN log levels in log4j2.xml | Make sure only ERROR or WARN log levels are in an app’s log4j2.xml. INFO and DEBUG are set in Runtime Manager properties tab for an app as needed. |
| Error Logging | Add loggers in error scenarios set to ERROR or WARN, so errors are properly logged. See Error Logging in the Logging Strategy. |
| Event Logging | Add loggers to every flow set to INFO that track the important events as the message is processed. See Event Logging in the Logging Strategy. |
| Debug Logging | Add loggers set to DEBUG wherever you need debug information for development and troubleshooting. These are in addition to the Error and Event loggers. See Debug Logging in the Logging Strategy. |
| No sensitive data is logged | Ensure no sensitive/protected data is logged, even in DEBUG setting. It must be omitted, masked, or encrypted for logging. |
| Use Notification Service for errors that require external action. | Notification Service sends the error to the appropriate location, such as ServiceNow or email. Use this service for when notifications are needed. This is generally in async process APIs and batch processes. Do not use this service in every error handler; use only as required by business case. |

# Documentation

|  |  |
| --- | --- |
| **Item** | **Details** |
| API Summary | Are the components of the API summary documented? |
| Unusual behavior or edge-case scenarios | Document edge-case situations and any hard-to-understand scenarios. |
| Third-party libraries | Any dependencies that should be documented? |
| Incomplete code | Is there incomplete code that should be removed or documented? |
| Implementation Documentation | Before ending development, implementation documentation per integration use-case must be delivered. This may include several APIs in each use-case. |