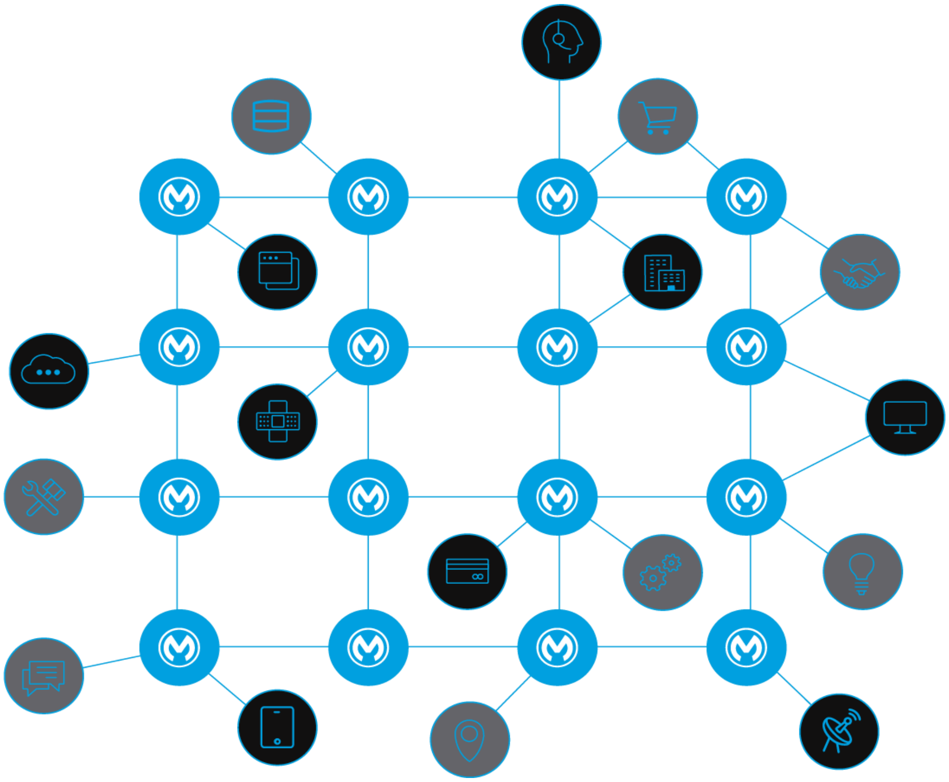
# Neuron

## Release 1 - Solution Design



| Version | Date | Author | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 0.1 | 29-Mar-2023 | Alvaro Acevedo Vahos | Initial version |
| 0.2 | 13-Apr-2023 | Alvaro Acevedo Vahos/Ivan Zarnev | Update Neuron actors, integrate Neuron with multiple insurers |
| 0.3 | 18-Apr-2023 | Alvaro Acevedo Vahos | Introduce updates in the delivery of risk to insurers implementing different API versions. |
| 0.4 | 19-Apr-2023 | Alvaro Acevedo Vahos / Ivan Zarnev | Introduce updates to implement and Idempotent mechanism |
| 0.5 | 21-Apr-2023 | Alvaro Acevedo Vahos | Introduce process to validate risk request before storing to Neuron DB. |
| 0.6 | 12-May-2023 | Alvaro Acevedo Vahos/Ivan Zarnev | Next version includes:<br>- Dup checks out of scope<br>- Integrate with external vault storing Insurers credentials.<br>- Naming conventions alignments.<br>- Add Functional Roles included in the JWT.<br>- Reflect changes to support UI (quotes, static and dynamic reference data)<br>- Expose Insurers<br>- Add Neuron configuration for |

| | | | |
|---|---|---|---|
| | | | Reference Data |
| 0.7 | 18-May-2023 | Alvaro Acevedo Vahos | - Bring back the concept of brokerid i/o brokerUser<br>- Add assumption about multi-tenancy and one database instance for all insurers and Risk input data validations |

# Table of contents

# Introduction

This document defines the technical components required to implement the first use case of the Neuron API/Integration solution.

Neuron is a platform that standardizes the digital interaction within the commercial insurance market (e.g. Brokers and Insurers). Neuron implements a harmonized data model that is exposed to all the brokers participants. The platform also exposes a catalog of digital services, including create risks, retrieve submitted risks and quotes, etc.

## Audience

This document's audience primarily includes architects, consultants, and developers and test engineers engaged in architecting, designing, developing, and testing API-based solutions.

## Glossary

The general terms and acronyms referenced in this document

## References

*Anypoint Platform Architecture*

- *Neuron Anypoint Platform Architecture*

*Others*

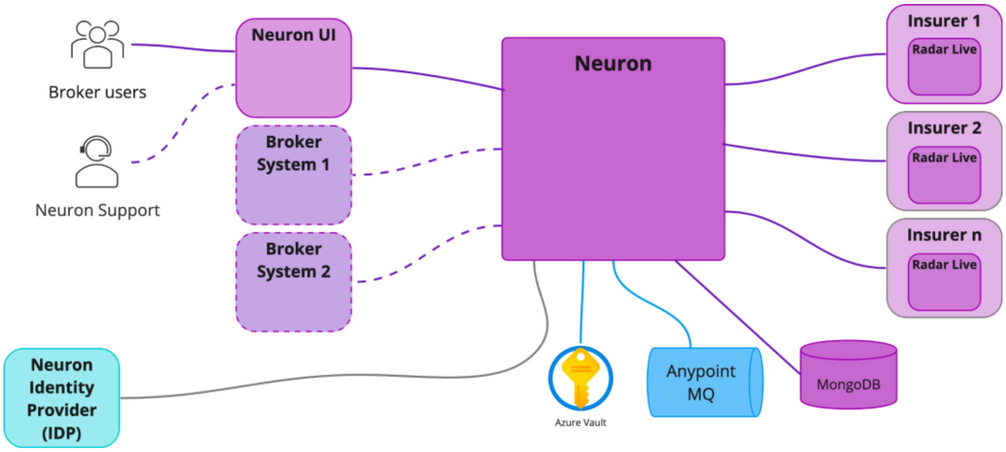
- Experience API Design Considerations
- Process API Design Considerations
- System API Design Considerations

- Mule Development Standards

# Solution Architecture overview

Neuron is a multi-tenant platform, different participants interact with the platform at different stages of the risk lifecycle.

The following diagram illustrates the interactions between the Neuron and other systems and actors of the platform:



Note that this document focuses exclusively on integrations via Neuron UI for brokers and Insurers implementing Radar Live for now, and does not describe the interactions between other actors of the system marked with dotted lines in above diagram, as they will be integrated in a later release.

- **Neuron UI**: This web component exposes a user interface for brokers users and the Neuron support team. After successful authentication, the UI exposes functionality to these users based on their profile.
- **Broker users**: Broker users execute functions through the Neuron UI, for example enter and submit a new risk, review submissions, retrieve and accept quotes.

- **Neuron Support**: Neuron support team is responsible for managing the onboarding process of Insurer and Broker Users and perform maintenance activities on the platform via the Neuron UI. *Note these users are not part of Release 1*
- **Broker Systems**: Similar to broker users, these are applications that communicate with the platform but calling directly the APIs exposed by the Neuron Experience APIs.
- **Insurers**: For this phase of the project, risk submissions are processed by  Radar Live in synchronous real-time fashion (request/response).
- **Anypoint MQ**: Anypoint AQ (AMQ) is a cloud messaging service that enables Neuron to implement advanced integration patterns, such as resiliency, guarantee delivery, circuit breaker,  asynchronous messaging and other patterns.
- **MongoDB (Neuron DB)**: This is the system where Nauron transaction and configuration data will be stored. The implementation of this database is MongoDB Atlas.
- **Azure Vault**: Integrate with external vault to store properties related to Insurers, for example Server URL, subscription keys, credentials, etc.

The initial release of this project exposes a catalog of services to Neuron UI below is an overview of the use cases supported:

- Broker submits a risk,
- Broker retrieves a risk or a collection of risks,
- Neuron dispatch a quote to the Insurers Radar Live system,
- Neuron process response from Insurers Radar Live system.

This solution design document concentrates exclusively on the submission of a risk and handling the responses from Radar-Live.

## API-Led Solution Architecture

The key goal of this approach is to introduce reusable building blocks that can be reused during the initial implementation and future releases of Neuron, resulting in reduced development effort.

API-led connectivity is a methodical way to connect data to applications through a series of reusable and purposeful modern APIs that are each developed to play a specific role to unlock data from systems, compose data into processes, or deliver an experience.

The API building block is a product in itself that consists of functionality and simplicity required for the full lifecycle of APIs.  This lifecycle consists of the ability to compose the data and connect to any other source of data. And it must provide full visibility, security, governance right from design.

### System Layer

System APIs will provide a means of accessing the underlying/core systems (Neuron DB, Radar Live, Insurers etc.) exposing that data in a standardized format, while providing downstream isolation from any interface changes or rationalization of those systems. These APIs will also change more infrequently and will be governed by Neuron IT given the importance of the underlying systems and participants of this solution.

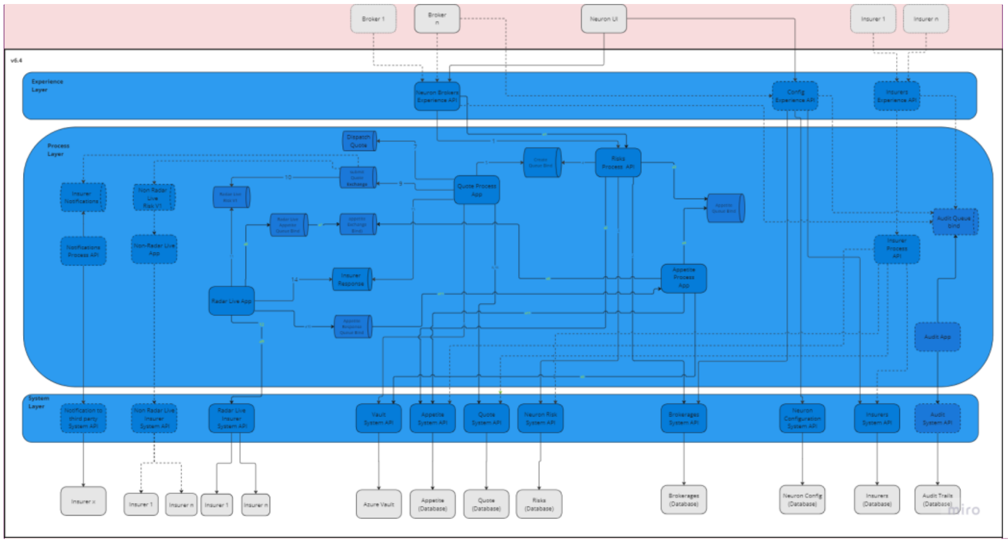
### Process Layer

The underlying business processes that interact and shape this data should be strictly encapsulated independent of the source systems from which that data originates, and the target channels through which that data is to be delivered. These APIs perform specific business processes functions and provide access to non-central data, for example orchestrating the updates in Neuron DB for responses sent by Insurers.

### Experience Layer

Data is consumed across a broad set of channels, each of which want access to the same data but in a variety of different forms, for example Brokers client applications may consume data in raw format while Brokers users interact with Neuron UI. Experience APIs are the means by which data can be reconfigured so that it is most easily consumed by its intended audience, all from a common data source, rather than setting up separate point-to-point integrations for each channel.

The diagram below illustrates the API-led connectivity approach composed of three main layers:



**Commented [EA(1]:** @Shah, Reema (London) , So what is the final verdict. Is Quote and Appetite a part of the Risk itself or a separate entity.

If we merge, this would have a singular System API.

And only with merged data model, we use the NoSQL database benefits.

**Commented [S(2R1]:** @Einsteen, Aldrine (Reigate) , we are now going ahead with separate collection for Appetite, the Appetite will be linked to the Risk collection and Quote Collection with the appetite id

Note that Insurers Experience API is not described in this version of the document, the API is not exposed to Insurers in this first release of the product. The dotted lines, on the left hand side of the diagram, depict the solution is future enabled and can integrate other systems. For example Radar Live systems implementing other versions of the API or Third Party Insurers.

## Integration Catalog

The Integration Catalog is driven by the inbound trigger; a single integration flow will result in outbound interactions with one or more target systems. Note that multiple source systems can trigger the same integration flow.

| Catalog |
|---|

| Name | Source System(s) | Target System(s) | Comments |
|---|---|---|---|
| Risk Creation | Neuron UI, Broker System | Neuron DB, Radar-Live | Brokers create one risk per request. |
| Get Risks | Neuron UI, Broker System | Neuron DB | Returns a collection of risks linked to the Broker company making the API request. |
| Get Quotes for a specific  a Risk ID | Neuron UI, Broker System | Neuron DB | Returns a list of quotes associated with a Risk. |
| Get Neuron Config | Neuron UI, Broker System | Neuron DB | Retrieves reference data, to help the completion of a Risk (currency, class of business, countries, etc) stored in MongoDB.<br>*Note this API will be limited to a first set of reference data in Release 1.* |

## Risk Creation - Business Process Overview

Broker posts the risk submission to Neuron supplying all relevant information in the request.

*Patterns*

- Messaging Publish-Subscribe: Events carry the state, business processes are decoupled and implements reliability.
- Guarantee delivery: ensure the creation of Risk and quote messages.
- Synchronous Request-Reply: System APIs implement synchronous patterns.
- Process Orchestration.

*Flow*

- A Broker submits a Risk request to the Brokers Experience API, the request validates that all the details in the payload to create a risk and the Access Token generated by Neuron IDP are present before passing the Risk to the Risk Process API for further processing, see Authentication and Authorization section for the authentication details.
- Risk Process API executes the follow steps:
  - Call Brokers System API to retrieve the brokerId associated with the multi-tenant-id present in the request headers.
  - *Process API passes the risk request to Risk Neuron System API to create a risk, including the brokerId and an identification of the user (from the JWT). System API responds with the payload created in the MongoDB, including its unique Risk ID.*
  - Process API passes the risk request and the brokerId as a HTTP header to Risk Neuron System API to create a risk. System API responds with the payload created in the MongoDB, including the unique Risk ID.
  - For each insurer present in the Risk, a message is published into the *Create Quote* queue. Note that the insurer's field is removed before the message is published and the insurer id is passed to the queue as message property.
  - Process API sends the response to the Experience API, including the payload created in the Risk MongoDB collection as returned by Risk System API. Note that MongoDB creates a unique Id which will be used as Risk Id and returned to Neuron UI as part of the response.

*Assumptions*

Following items list the assumptions that have been made when working on designing the solution:

- Risk submission request payload doesn’t require neither transformation nor data enrichment.
- MongoDB document identifier will be used as the Risk unique ID.
- Experience API consumers can be Brokers Neuron UI and Brokers systems.
- In the first release, this solution doesn’t require validations of input data, for example the insurers IDs present in the request payload or the class of business values.

*Prerequisites*

Following prerequisites are required as part of the solution.

- Token includes the required metadata for API Policy to perform authorizations, for example the identifier of the Broker company (also called multi-tenantId), the user or system role and the metadata to identify the user triggering the action.
- Request payload is valid and includes all the necessary information required for the submission of a risk.

## Quote Creation - Business Process Overview

A risk message is consumed from *Create Queue* and a quote is created in MongoDB and dispatched to insurers.

*Patterns*

- Messaging Publish-Subscribe: Events carry the state, business processes are decoupled and implements reliability.
- Guarantee delivery: ensure the creation of the quote and creation quote messages.
- Circuit breaker: Prevents the flow from performing risk dispatching if they are likely to fail.
- Synchronous Request-Reply: System APIs implement synchronous patterns.

- Process Orchestration.

*Flow*

- Quote Process App consumes a Risk message from *Create Queue*.
- Process API calls endpoint POST /quotes exposed by Quotes System API to create a quote and return a new quoteId.
- Process API publishes the risk payload into the *Dispatch Quote Queue*, including as message properties the newly created quoteId and insurerId.
- Process API calls Vault SAPI to retrieve the connection details of the insurer, including the deliveryChannel, subscriptionKey, insurerServerHost, insurerURLPath and other technical details required for the delivery of the risk and quote.
- To finish the flow the risk is published to the *Submit Quote Exchange*, including the properties retrieved from Azure Vault and quoteId. The Exchange queue implements a content based router based on the value of deliveryChannel property.

**Commented [N(3]:** Can you please change this to Neuron Quote App?

**Commented [S(4R3]:** changed

*Assumptions*

This section covers any assumptions that have been made working on designing the solution.

- AnypointMQ Exchange is configured to route the incoming message to a destination queue based on the deliveryChannel property. For example if the value of deliveryChannel is 'radar-live-v1' for all Radar Live systems implementing the version 1, the queue Insurers Risk v1 is bound to that value.
- In Release 1, all Radar Live (RL) systems implement the same API contract.

- The current solution supports many RL  integrations. The Process and System API in this solution design can be reused If the RL system implements the same API contract and integration mechanism, for example a different authentication mechanism.
- AnypointMQ Exchange can be bound up to 10 different queue distributions. This means that the solution can support up to 10 versions of the Radar-Live API contract or Third Party insurers. Based on requirements, Third party insurers will poll for unprocessed risks.
- The system can retrieve from Azure Vault the connectivity details of the Insurer system based on a key known but both Neuron solution and Azure Vault, for example the Insurer ID.

*Prerequisites*

Following prerequisites are required as part of the solution.

- AnypointMQ Exchange configuration
- Vault System API can integrate with Azure Vault and retrieve the information required to integrate with Radar Live systems.

## Quote Distribution to Radar Live and response Stored - Business Process

A quote is consumed from *Radar Live Risk v1* queue and sent to the relevant Radar Live system specified in the message properties.

*Patterns*

- Messaging Publish-Subscribe: Events carry the state, business processes are decoupled and implements reliability.
- Synchronous Request-Reply: System APIs implement synchronous patterns.
- Process Orchestration.

*Flow*

- Radar Live Process API consumes a message from *Radar Live Risk v1* Queue and performs following actions:
    - Prepare the technical details required to deliver the risk to the insures, build the server URL including path & query parameters and HTTP headers (e.g. subscriptionKey, insurerServerHost, insurerURLPath)
    - Delivers the risk to the Radar Live System API in synchronous Request-Reply fashion. The response includes the status: acceptance, rejection or referred and the following percentage.
    - It creates a message payload including the response from Insurer and the quoteId. Error messages are also mapped to an errorMessage field so it can be audited later by operation users.
    - Publishes the message to the *Insurer Risk* queue.
- Quote Process API consumes a Quote Response message from *Insurer Response Queue* and performs following actions:
    - Calls endpoint PATCH /quotes/{quoteId} exposed by Quotes System API to update the quote with the response from the insurer.

*Assumptions*

- Radar Live systems expose the same API contract and authentication mechanism.

## Get A List of Risks - Business Process

Broker retrieves a list of submitted risks.

*Patterns*

- Synchronous Request-Reply: Implement synchronous patterns.

*Flow*

- A Broker sends a GET request to the Brokers Experience API, the request includes all the details in the access token to perform the authorization, see Authentication and Authorization section for details.
- Experience API delegates the request to the Broker System API to retrieve the list of risks linked to the broker (as a company).
- System API finds the collection of risks matching the filters specified in the API client request.
- The objects retrieved from MongoDB are returned as is to the API client.

Following flow diagram depicts the interactions between Neuron UI, Neuron IDP, MuleSoft and Neuron DB.

*Assumptions*

- Request includes in the query parameters the filters necessary to match the risks. For example "classOfBusiness=Casualty" to view all risks for Casualty.
- No message enrichment nor transformations are required for the object retrieved from the Neuron DB.

*Prerequisites*

- Token includes the required metadata for API Policy to perform authorizations, for example multi-tenant id of the Broker.

## Get Quotes linked to a Risk with a resource ID - Business Process

Broker retrieves a list of quotes associated with a risk.

*Patterns*

- Synchronous Request-Reply: Implement synchronous patterns.

*Flow*

- A Broker sends a GET request to the Brokers Experience API, the request includes all the details in the access token to perform the authorization, see Authentication and Authorization section for details.
- Brokers Experience API then sends a request to Risk Process API which performs the following actions:
  - Calls Brokers System API to retrieve the brokerId - based on the details included in the HTTP headers (extracted from the authentication token)
  - Calls a GET method exposed by Quotes System API to retrieve a list of quotes linked to risk and broker performing the request.

*Assumptions*

- Brokers Experience API receives in the request the path parameters matching the resource id of the risk associated with the list of quotes.
- No message enrichment nor transformations are required for the object retrieved from the Neuron DB.

*Prerequisites*

- Token includes the required metadata for API Policy to perform authorizations, for example the identifier of the Broker company (also called multi-tenantId), the role(s) or system.

## Appetite Check - Business Process

Broker does an appetite check before final risk submission by providing limited information of the risk.

### *Part 1: Send Appetite check request:*

*Patterns*

- Synchronous Request-Reply: System APIs and Broker Experience API implement synchronous patterns.
- Asynchronous: Appetite Process App Implement asynchronous patterns using publish subscribe.

*Flow*

- A Broker submits an appetite check request to the Brokers Experience API, the request validates that all the details in the payload and the Access Token generated by Neuron IDP are present before passing the Appetite check request to the Risk Process API for further processing, see Authentication and Authorization section for the authentication details.
- Risk Process API executes the follow steps:

- Call Brokers System API to retrieve the brokerId associated with the multi-tenant-id present in the request headers.
- Process API passes the appetite request and the brokerId as a HTTP header to Appetite Neuron System API to create an appetite. System API responds with the payload created in the MongoDB, including the unique Appetite ID.
- For each insurer present in the appetite check request, a message is published into the *Appetite request* queue. Note that the insurer's field is removed before the message is published and the insurer id is passed to the queue as message property.
- Process API sends the response to the Experience API, including the payload created in the Appetite MongoDB collection as returned by Appetite System API. *MongoDB creates a unique Id which will be used as Appetite Id and returned to Neuron UI as part of the response for the first call.*

- Appetite Process App execute the following steps:

  - Reads the message from the *Appetite request* queue.
  - Appetite process App calls Vault SAPI to retrieve the connection details of the insurer, including the deliveryChannel, subscriptionKey, insurerServerHost, insurerURLPath and other technical details required to connect to the insurer for the appetite check.
  - Publish the message to the appetite exchange including the properties retrieved from Azure Vault and appetiteId. The Exchange queue implements a content based router based on the value of deliveryChannel property which is bound to the *Radar Live Appetite* queue.

- Radar Live Process API consumes a message from *Radar Live Appetite* Queue and performs following actions:
    - Prepare the technical details required to get the appetite from the insurers, build the server URL including path & query parameters and HTTP headers (e.g. subscriptionKey, insurerServerHost, insurerURLPath)
    - Gets the appetite response from the Radar Live System API in synchronous Request-Reply fashion. The response includes the status: Open to submission, Not in Appetite.
    - It creates a message payload including the response from Insurer and the appetiteId. Error messages are also mapped to an errorMessage field so it can be audited later by operation users.
    - Publishes the message to the *Insurer appetite response* queue.
- Appetite Process App consumes the Insurer's Appetite Response message from *Insurer Response Queue* and performs following actions:
    - Calls endpoint PATCH /appetites/{appetiteId} exposed by Appetite System API to update the appetite with the response from the insurer.

### *Assumptions*

Following items list the assumptions that have been made when working on designing the solution:

- MongoDB document identifier will be used as the Appetite unique ID.

### *Prerequisites*

Following prerequisites are required as part of the solution.

- Token includes the required metadata for API Policy to perform authorizations, for example the identifier of the Broker company (also called multi-tenantId), the user or system role and the metadata to identify the user triggering the action.
- Request payload is valid and includes all the necessary information required for appetite check.

### *Part 2: Get Appetite linked with appetite ID.*

*Patterns*

- Synchronous Request-Reply: Implement synchronous patterns.

*Flow*

- A GET request is sent to the Brokers Experience API, the request includes all the details in the access token to perform the authorization, see Authentication and Authorization section for details.
- Brokers Experience API then sends a request to Risk Process API which performs the following actions:
  - Calls Brokers System API to retrieve the brokerId - based on the details included in the HTTP headers (extracted from the authentication token)
  - Calls a GET method exposed by Appetite System API to retrieve a list of quotes linked to risk and broker performing the request.
  - Respond the Insurer Appetite data to the Broker Experience API.
- Broker Experience  API transforms the response and returns the count of Insurer Appetite with the status "Open to submission".

*Assumptions*

- Brokers Experience API receives in the request on the intended endpoint to get the appetite based on the appetite.
- The count of appetite only includes the positive appetite responded by the insurers who returns the status “Open to submission “ in the given frame of time

*Prerequisites*

- Token includes the required metadata for API Policy to perform authorizations, for example the identifier of the Broker company (also called multi-tenantId), the role(s) or system.

## Onboard New Line of Business - Business Process

Broker gets to choose from more line of business to submit risk.

*Patterns*

- The message pattern for the new line of business would be the same as mentioned below.
- Messaging Publish-Subscribe: Events carry the state, business processes are decoupled and implements reliability.
- The flow to create the risk and get the quote remains the same for all the newly added line of business as mentioned below.
- Guarantee delivery: ensure the creation of Risk and quote messages.
- Synchronous Request-Reply: System APIs implement synchronous patterns.
- Process Orchestration.

- Request payload validations together with conditional validation is done via JSON schema validation.

*Flow*

- The flow to create the risk and get the quote remains the same for all the newly added line of business as mentioned below.
- A Broker choose from additional lines of business and submits a Risk request to the Brokers Experience API, the request validates that all the details in the payload to create a risk and the Access Token generated by Neuron IDP are present before passing the Risk to the Risk Process API for further processing, see Authentication and Authorization section for the authentication details.
- Risk Process API executes the follow steps:
  - Call Brokers System API to retrieve the brokerId associated with the multi-tenant-id present in the request headers.
  - *Process API passes the risk request to Risk Neuron System API to create a risk, including the brokerId and an identification of the user (from the JWT). System API responds with the payload created in the MongoDB, including its unique Risk ID.*
  - Process API passes the risk request and the brokerId as a HTTP header to Risk Neuron System API to create a risk. System API responds with the payload created in the MongoDB, including the unique Risk ID.
  - For each insurer present in the Risk, a message is published into the *Create Quote* queue. Note that the insurer's field is removed before the message is published and the insurer id is passed to the queue as message property.
  - Process API sends the response to the Experience API, including the payload created in the Risk MongoDB collection as returned by Risk System API. Note that MongoDB

creates a unique Id which will be used as Risk Id and returned to Neuron UI as part of the response.

*Assumptions*

Following items list the assumptions that have been made when working on designing the solution:

- Experience API consumers can be Brokers Neuron UI and Brokers systems.
- The request body should contain only those coverages which has been selected by the broker, if the specific coverage is not selected by the broker, the section including the coverage details should not be sent as part of the request payload.

*Prerequisites*

Following prerequisites are required as part of the solution.

- Token includes the required metadata for API Policy to perform authorizations, for example the identifier of the Broker company (also called multi-tenantId), the user or system role and the metadata to identify the user triggering the action.
- Request payload is valid and includes all the necessary information required for the submission of a risk for the given line of business. The coverage information should be present in the request body.

# API Application Catalog

A list of the proposed API applications is provided below. Note that this list is based on the information provided by the Neuron team during the discovery workshops. A single deployed API application can support multiple aspects of the logical integrations as described in the Integration Catalog section of this document.

To interpret this table it is important to first understand the MuleSoft API-led Connectivity approach as described in API-Led Solution Architecture and in more detail in this article API-led connectivity.

| API Application Catalog | | |
|---|---|---|
| API Name | Type (E/P/E) | Function (+ logical integration catalog references) |
| Brokers | Experience | Entry point for Neuron UI and Broker Systems to create Risks, Retrieve available Risk(s) and Quote(s). |
| Neuron Config | Experience | Entry point for the Neuron UI or Broker system to retrieve the platform reference data. |
| Insurers | Experience | Experience API exposed to Insurers. It allows the insurer to collect their quotes and provide final decisions.<br>*Not implemented in the phase* |
| Risk | Process | Logic for processing Risk Submission fulfillment and routing to the relevant Broker, Quotes and Risk System APIs.<br>Guarantees the delivery of the |
| Quote | Process | Orchestrates the dispatching of a quote and processes insurers responses. |
| Radar Live | Process | Implements logic that guarantees delivery of the quotes<br>Prepares the technical details to facilitate the quote delivery to the Insurers. |

| | | |
|---|---|---|
| Risks | System | Provides operations for creating and retrieving risks from MongoDB. |
| Neuron Configuration | System | Neuron configuration data, for example Broker and Insurers relationships are stored in MongoDB, this API deals also with reference data. |
| Radar Live | System | Provides functionality for pushing risks to Radar Live systems. This API handles risk submissions, insurer responses and error handling.<br>The same System API will be used for different underlying insurers implementing the same version and integration mechanisms (authentication, headers, etc.). |
| Vault | System | This system API integrates with Azure Vault to retrieve Insurers confidential data and other configuration required for delivering the risk. |
| Quotes | System | Provides operations to create, update and retrieve quotes. |
| Insurers | System | Exposes functions to retrieve and manage Insurers part of the Neuron platform. |
| Brokers | System | This system API facilitates the retrieval and management of brokers part of the platform. |

# Non-Functional Requirements

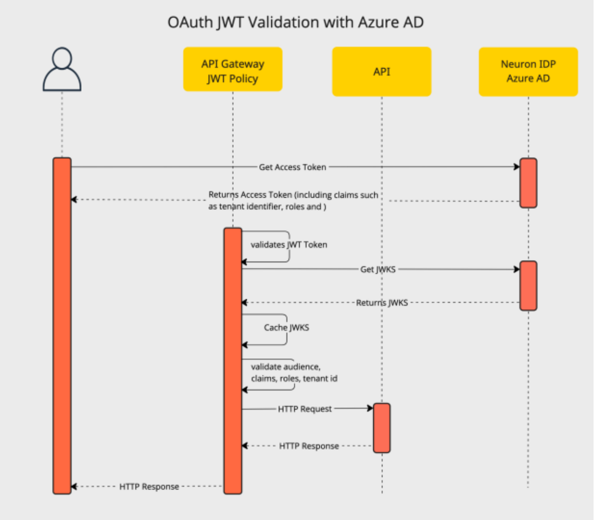
This section captures the non-functional requirements of all APIs related to the use-case(s)/solution.

## Security

This section describes the authentication flow for experience APIs.

| Authentication | |
|---|---|
| Application Name | Authentication Type/Policy |
| All Experience APIs | OAuth 2.0 access token enforcement, JWT Validation Policy |

| Authorization | |
|---|---|
| Application Name | Authorization Scopes/Roles |
| Brokers Experience API | brokerUser, brokerSystem |
| Neuron Management Experience API | neuronSupportUser<br>(not used in Release 1) |
| Insurers Experience API | insurerUser, insurerSystem<br>(not used in Release 1) |


OAuth JWT Validation with Azure AD
API Gateway JWT Policy
API
Neuron IDP Azure AD
Get Access Token
Returns Access Token (including claims such as tenant identifier, roles and )
validates JWT Token
Get JWKS
Returns JWKS
Cache JWKS
validate audience, claims, roles, tenant id
HTTP Request
HTTP Response
HTTP Response

*Flow*

- API consumer requests an access token from Neuron IDP using one of the Authorization Grant Types.
- Neuron IDP  authenticates the user (or application) and applies an access policy or a rule to determine what level of access should be granted. The access policy adds custom role(s).
- Neuron IDP responds to the consumer with the JWT token with an RSA signature. The JWT Token contains an Audience Claim with the Service API ID and the custom claim values populated.
- API consumer invokes the MuleSoft API and passes the JWT access token in the request.
- JWT policy makes a call to JWKS server that contains the public keys for the signature validation and cache it.
- MuleSoft API Gateway perform following:

  - Decrypts the JWT Token with the retrieve public key and validates the Audience Claim and Roles.
  - The TenantId is the same being used in the OAuth Token endpoint.
  - The roles claim value in the token matches with the value(s) required for invocation of the API. This can be done with the expression `#[vars.claimSet.roles contains('brokerUser') || vars.claimSet.roles contains('brokerSystem')]`
- If the JWT validation is successful, explode the claims as HTTP headers and allow the mule API to be called.

## SLA/Performance

*Response Times*

Below tables document the non-functional attributes that were communicated as part of the requirements gathering process.

| **Response Times** | | |
|---|---|---|
| S.No | Application Name | Response Time (95%) |
| 1 | Response time for all the transactions.<br>If the system can't handle such SLA's more granular requirements will be communicated, for example different SLA for writes, updates and reads. | Less than 1 second |

*Throughput*

| Throughput | | |
|---|---|---|
| S.No | Application Name | TPS |
| 1 | All external exposed APIs | The APIs are expected to handle at least 1 TPS for both read and writes at a peak time. Note that the number of concurrent users is up to 60  - 4 broker companies consisting of 15 Neuron registered users each. |

*Policies related to performance*

Following list of policies related to security and quality of service and performance have been identified so far:

| Policies | | |
|---|---|---|
| Application Name | Policy | Details |
| Experience APIs | All experience API are configured with:<br>- JWT-validation-policy,<br>- Rate-Limiting SLA Based<br>- JSON Threat Protection Policy for all PATCH/POST/PUT requests endpoints. | <TBD> Provide details such as Time Period, Number of Reqs, Delay Time, Delay Attempt, Queuing Limit etc. |
| Process APIs | Client ID enforcement | |
| System API: Radar-Live, MongoDB | Spike Control, Client ID enforcement | <TBD> add the number of allowed requests in the time window, window size and other parameters relevant to configure the policies |

| Health-check/Ping end-point | | |
|---|---|---|
| API Name | Type | End-point URL |
| Apply for all experience APIs | Health-Check | Experience APIs will implement a /health-check endpoint |

## Availability

The expected availability requirements is 99,90% of service availability. To ensure this attribute. all API should be deployed to at least two replicas.

## Scalability

It is assumed a vCore size of 0.1 will be configured for all the APIs. However load tests will confirm the memory capacity and processing power of the replicas and whether the implementation should scale the replicas vertically or horizontally.

## Reliability

Use cases that must ensure reliability and guaranteed delivery will implement Anypoint MQ queues.

## Reuse Considerations

At this stage of the project, it was identified that MongoDB APIs will be re-used for all the use cases part of the release 1 and care will be taken in future release to create standardized reusable components.

About MuleSoft

## MuleSoft, a Salesforce company

MuleSoft's mission is to help organizations change and innovate faster by making it easy to connect the world's applications, data, and devices. With its API-led approach to connectivity, MuleSoft's market-leading Anypoint Platform™ empowers over 1,600 organizations in approximately 60 countries to build application networks. By unlocking data across the enterprise with application networks, organizations can easily deliver new revenue channels, increase operational efficiency, and create differentiated customer experiences.

For more information, visit **mulesoft.com**