In het bijgevoegde Zip-bestand zitten twee Java demo's die het gebruik en de werking van queues laten zien. Demo-1 laat zien wat het verschil is tussen queues en topic met betrekking tot guaranteed delivery en distributie van de berichten. Demo-2 laat zien hoe je XML berichten op een queue kunt zetten.

Demo-1 kun je draaien door de runner te starten, deze runner bevat drie voorbeelden: "Produce, wait, consume", "Consume, wait, porduce, topics", en "Consume, wait, produce, Queues". In elk voorbeeld wordt een aantal producers en consumers aangemaakt. Een producer/consumer kan verbinden met een queue of een topic. De producers versturen één bericht naar de queue of het topic. De consumers maken altijd gebruik van de synchrone manier van communiceren met de queue/topic, namelijk door de receive methode aan te roepen. Hierbij wordt een time-out van 2 seconden meegegeven. Verder kan aan de consumer worden meegegeven of er slechts één of meerdere berichten ontvangen moeten worden. Als er gekozen wordt om meerdere berichten te ontvangen wordt de receive aangeroepen totdat er geen bericht meer binnen de time-out wordt ontvangen.

In het eerste voorbeeld worden eerst de producers gestart, twee producers die een bericht naar een queue versturen en twee die een bericht naar een topic versturen. Vervolgens wordt er 5 seconden gewacht om vervolgens drie consumers te starten. Twee queue consumers en één topic consumer die meerdere berichten kan ontvangen.

Het tweede voorbeeld start eerst drie consumers op die naar topic luisteren. Twee consumers die meerdere berichten kunnen ontvangen en één die slechts één bericht kan ontvangen. Vervolgens wordt er 1 seconde gewacht en worden er vier producers (topic) gestart.

Het derde voorbeeld is gelijk aan het tweede, alleen worden er nu queues gebruikt.

Naast de consumer is er ook een listener class. Hiermee kan asynchroon worden geluisterd naar de queues/topics.

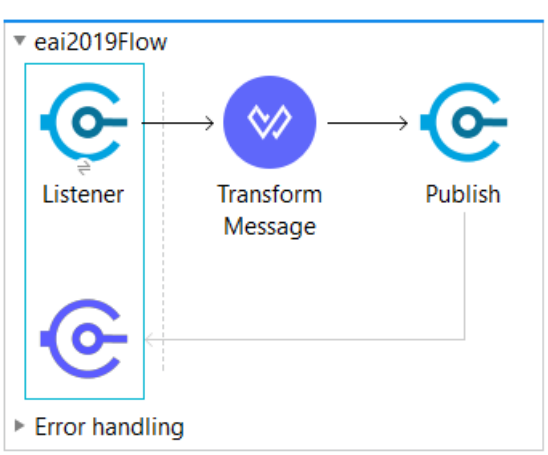
1. Voordat je het voorbeeld draait, probeer te voorspellen welke consumer welk bericht ontvangt.
2. Start de runner en bestudeer de logging in de console. Klopt wat je ziet met wat je voorspeld had?
3. Verander de programmatuur zodat in plaats van synchroon, asynchroon geluisterd wordt naar de queues/topics. Verklaar wat er nu te zien is in de logging.

Demo-2 stuurt een XML object van de producer naar de consumer, dit is een eenvoudige manier om instanties van java-klasses te versturen. Hierbij wordt gebruik gemaakt van de Xstream library. Je kunt deze demo draaien door eerst de consumer te starten en vervolgens de producer.

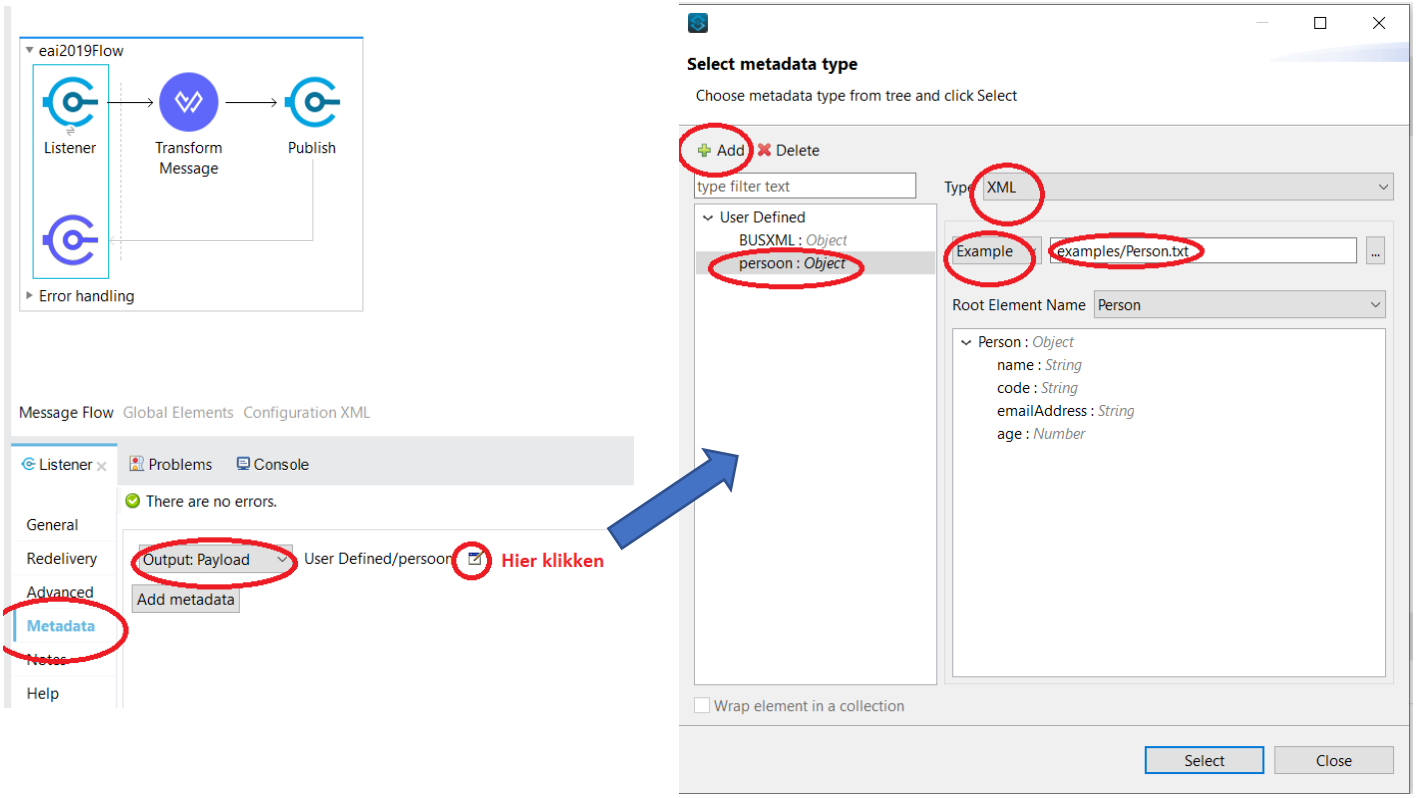
1. Voer het programma uit en kijk wat er op de console getoond wordt.
2. In de SendPerson methode in de producer en in de CreatePerson methode in de consumer staan regels met de xstream.alias instructie. Haal deze regels uit de comments en draai het programma weer. Wat is er veranderd?

We gaan nu Anypoint gebruiken om de leeftijd van de persoon die verstuurd wordt met één op te hogen.

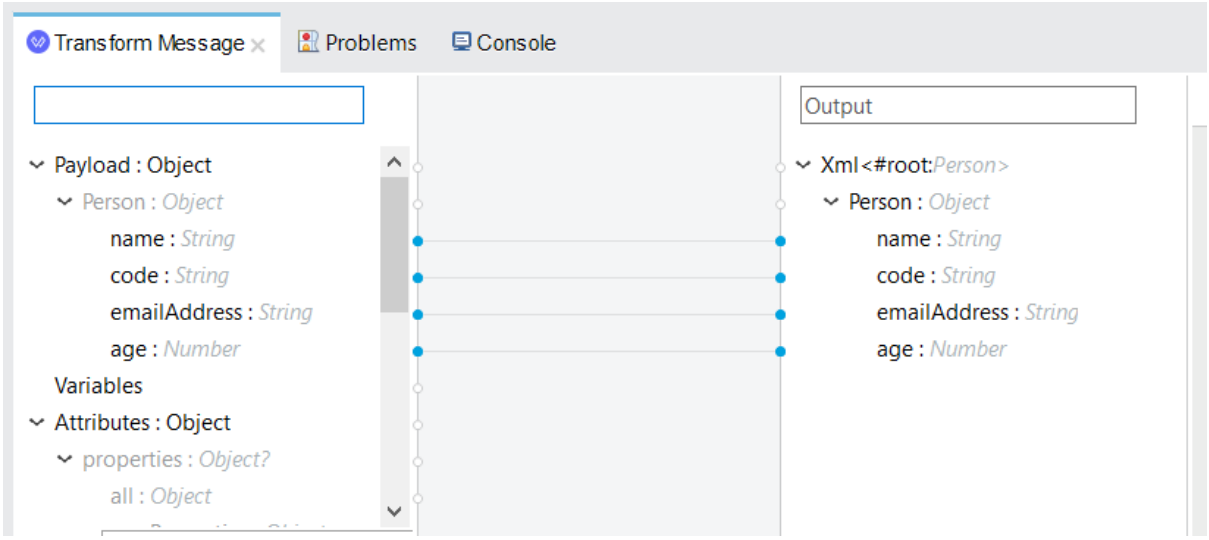
1. Verander de naam van de queue waar de consumer naar luisterd naar 'TESTQUEUE2'.
2. Maak in Anypoint de onderstaande flow aan, waarbij de listener naar 'TESTQUEUE1' luistert en de Publish naar 'TESTQUEUE2' schrijft.



3. Bij een connector zoals de Listener kan Anypoint niet automatisch bepalen wat de output is. Door op de connector metadata te definiëren is het mogelijk om dit handmatig te doen. Zie onderstaande illustraties.

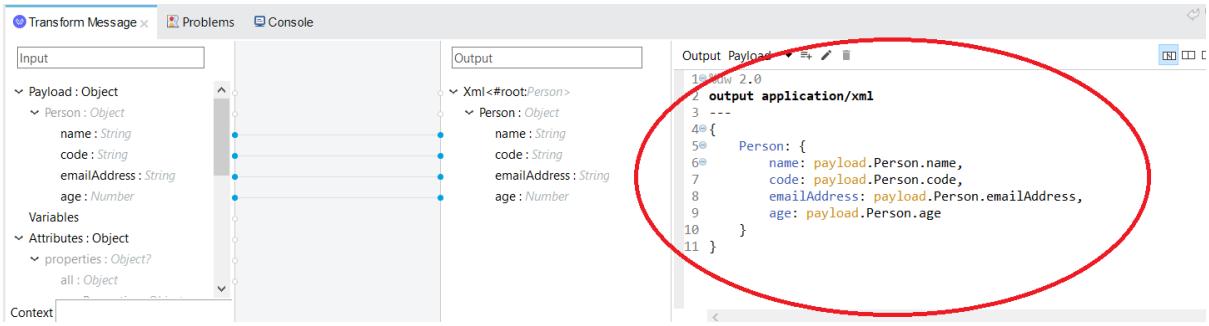


4. Het type data dat uit de connector komt moet dus gedefiniëerd worden. Je kunt het type persoon aanmaken als XML object. Door een voorbeeld XML bestand (kopiëren uit de console van het java programma) in te laden wordt automatisch de structuur bepaald. Door op de Publish ook de metadata van de input te definiëren (dit is natuurlijk hetzelfde type, namelijk een XML persoon object, zal de tranform Message automatisch het juiste input en output weergeven. Verbind de input en output zoals in het onderstaande plaatje, en start vervolgens alle onderdelen, i.e. ActiveMQ, Anypoint flow, consumer en producer.



Komt het bericht aan? Kijk in de admin console van ActiveMQ welke queues er zijn. Welk pattern zoals beschreven in de Enterprise Integration Patterns is hier geïmplementeerd.

5. Zoals je in het vorige onderdeel hebt gezien forceert Anypoint extra controles op de input op Queues. Net als bij HTTP kun je de content-type van het bericht namelijk vastleggen. Hiervoor worden dezelfde beschrijvingen gebruikt als bij HTTP. In dit geval dus 'application/xml'. Pas dit voor beide JMS connectors aan. Zorg er ook voor dat de leeftijd met één opgehoogd wordt. Dit kan in de transform message in het venster dat hieronder wordt aangegeven (zoals hier aangegeven is nog niet de juiste code!).



In deze oefening is het niet strikt noodzakelijk om een transform message te gebruiken. De tekst die hierboven is aangegeven kan in de Publish connector ook rechtstreeks in de Body gezet worden. In dat geval is het ook niet nodig om de metadata op de Publish te definiëren. De content type is echter nog wel nodig. De code die hier gebruikt wordt roept de DataWeaver 2.0 functionaliteit aan van Anypoint. De '%dw 2.0' zorgt hier voor. De regel die hierop volgt bepaald welk formaat de output heeft, in dit geval dus XML, waarna de beschrijving van het object volgt, in iets dat lijkt op JSON. De regel met de minnetjes (---) is ook onmisbaar.