Message Exchange Patterns
And Tools for Distributed Systems

SEPAD Track, Session 140
Thursday, May 23rd, 9:45AM
Room 13B
CLA Americas Summit 2019
Austin, September 25-27 @ NI Headquarters
Demo #1
What are the problems we want to solve?
Message Exchange Patterns
Message Brokers
Picking the Right Tools for the Job
MQTT
Demo #2
Takeaways
Q&A
DEMO #1
What are the problems we want to solve?
Problem Statement

How can we manage communication between components in our systems when:

- Components are distributed
- Components have interoperability issues
  - They are written in different languages
  - Are on different targets
  - Include 3rd party components
- The system has a dynamic topology
- The system needs to be scalable
- Message reliability and safety need to be considered (and the network is not reliable)
- A simple TCP connection is not enough

When it comes to building more complex systems we need to start from a higher level paradigm which defines how components in our system can interact.
Message Exchange Patterns
What do we mean by "Message Exchange Patterns"

- Common high-level patterns that define how components in a system connect and communicate with one another by passing messages.

- This discussion focuses on a few well-established patterns:
  - Request-Reply
  - Publish-Subscribe
  - Push-Pull (Pipeline)
  - Streaming
  - Re-direction
Request – Reply (1-1)

A simple and common pattern that allows a two-way message exchange between a pair of components. In this pattern the requestor (client) sends a query to the server and the server sends back a message in response.

Send a request to a parallel loop and wait for a response
Publish – Subscribe (1-N)

A one-to-many communication pattern (often referred to as a data distribution pattern). Senders and receivers can be decoupled. Publishers organize messages into “topics” and require no information about subscribers. Subscribers subscribe only to the messages of interest.

Diagram

Example

The publisher generates an event whenever a new value is available, multiple subscribers listen to the update.
Redirection & Routing

- Delegate work to other processes
- Balance Load on a network
- Provide a message abstraction layer

Can be extended to:
- Message Translation
- Pipelines
Pipeline

Push and pull is used for fan-out, fan-in one way communication.
Push typically distributes messages to all available clients evenly.
Pull will fairly queue messages from all connected clients.
This is a parallel task distribution and collection pattern.
Streaming

- High-Throughput One-Way communication
Message Brokers
Message Brokers

- Many messaging systems rely on a component to mediate the message exchanges. This intermediary component is called the broker.
- Brokers offer several advantages, including decoupling of components.
Picking the Right Tool for the Job
<table>
<thead>
<tr>
<th>Protocol/Tool</th>
<th>ZeroMQ</th>
<th>MQTT</th>
<th>AMQP</th>
<th>Kafka</th>
<th>JMS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Implementations and Brokers</strong></td>
<td><em>Not Required</em> Malamute</td>
<td>Mosquitto, HiveMQ</td>
<td>RabbitMQ, HornetQ</td>
<td>Apache Kafka</td>
<td>Java Exclusive</td>
</tr>
<tr>
<td><strong>Transport</strong></td>
<td>TCP, PGM, IPC, ITC</td>
<td>TCP (+Websocket)</td>
<td>TCP</td>
<td>TCP</td>
<td>TCP</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>A light-weight messaging library which extends the standard socket interface to easily support multiple messaging patterns.</td>
<td>A binary protocol that facilitates pub-sub messaging between servers and low-powered IoT devices.</td>
<td>A binary messaging spec implemented by a wide variety of brokers. Supports many routing and distribution strategies.</td>
<td>A publish-subscribe message streaming platform designed for high volume and durability.</td>
<td>Both a protocol and the components that implement it. A part of the Java EE that abstracts inter-service communication.</td>
</tr>
<tr>
<td><strong>When to use</strong></td>
<td>When complex workflows with higher speed and no central point of failure are required.</td>
<td>For IoT. Lots of clients available and plenty of MQTT brokers around</td>
<td>Require more than simple publish-subscribe messaging between components</td>
<td>A good solution for large scale message processing applications.</td>
<td>If already using the Java Enterprise Platform.</td>
</tr>
<tr>
<td><strong>Pro</strong></td>
<td>Fast, lightweight, no broker required, open source.</td>
<td>Light-weight. Designed to work in unstable and unpredictable network environments.</td>
<td>Reliable, Proven, Feature rich.</td>
<td>High-Throughput, distributed, scalable, high-performance, durable, open source.</td>
<td>Large enterprise solutions</td>
</tr>
<tr>
<td><strong>Con</strong></td>
<td>Complex, cannot extend</td>
<td>Security, Pub-Sub</td>
<td>Not particularly fast or lightweight</td>
<td>Requires Zookeeper.</td>
<td>Java Exclusive</td>
</tr>
</tbody>
</table>
MQTT
(Message Queuing Telemetry Transport)
What is MQTT

- Publish-Subscribe Transport Protocol (Application Layer)
- Lightweight
- Open Source standard
- Client, Broker
- Standard ISO/IEC 20922:2016 (MQTT 3.1.1)

Used for:
- IoT, M2M, Industry 4.0, etc.
Message Packets

- Connect, Connack, Disconnect
- Subscribe, Suback, Unsubscribe, Unsuback
- Ping, PingResp
- Publish, Puback, PubRec, PubRel, PubComp
- Auth (new to MQTT 5.0, 2019)

Source: LabVIEW Open Source (GitHub)

Why choose MQTT

- Lightweight (low-energy IoT devices, low bandwidth)
- Secure (SSL, TLS)
- Open Source or Scalable Cloud Infrastructure (AWS, Azure, Google, etc.)
- Interoperable (Linux, Mac, Windows, Cloud, Embedded)
MQTT Basic LabVIEW Client
Clients publish their messages with the following convention:

- Namespace/users/Client1/state
- Namespace/users/Client2/connection/state
- Namespace/users/Client2/service/state

Namespace/users/Client1/#
Subscriber receives all Client1 messages:
  - Namespace/users/Client1/state
  - Namespace/users/Client2/connection/state
  - Namespace/users/Client2/service/state

#/state
Subscriber can receive all "state" messages:
  - Namespace/users/Client1/state
  - Namespace/users/Client2/connection/state
  - Namespace/users/Client2/service/state

Namespace/users/Client2/+/state
Subscriber can receive only Client2 "{level}/state" messages:
  - Namespace/users/Client1/state
  - Namespace/users/Client2/connection/state
  - Namespace/users/Client2/service/state
Quality of Service

<table>
<thead>
<tr>
<th>QoS Level</th>
<th>Expected Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>QoS 0</td>
<td>None</td>
</tr>
<tr>
<td>QoS 1</td>
<td>PUBACK Packet</td>
</tr>
<tr>
<td>QoS 2</td>
<td>PUBREC Packet</td>
</tr>
</tbody>
</table>

Diagrams showing the handling of PUBLISH and PUBREL (Release) messages with different QoS levels.
Request-Reply in MQTT

MQTT is a Publish-Subscribe protocol.

- The resulting implementation of a Req-Reply pattern implies an asynchronous response.
- Requester includes a "return address" into the message.
- Requestee replies to that address.
- Possibility of routing messages (message abstraction layer)
Economics of MQTT Implementation

- Economics of MQTT web traffic (table or graph)
- Table of cost, etc.

<table>
<thead>
<tr>
<th>Deployment</th>
<th>Local Server</th>
<th>HiveMQ</th>
<th>Google Cloud</th>
<th>AWS IoT</th>
<th>Azure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base Cost</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cost per 1M messages</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Changes from MQTT 3.1.1 (2015) to 5.0 (2019)

- Keeps core functionality mostly intact
- Improves Error Reporting
- Formalizes Patterns such as Request-Response
- Authentication (Server-Client // Client-Server)
- Performance Improvements for small and large deployments
- Introduction of User Properties in packets

- [http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html)
- [https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html](https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html)
DEMO #2
Massive Scalability

- End with a diagram of how we would massively scale up our example to enable millions of nodes or thousands of nodes with larger bandwidth requirement.
Takeaways
Takeaways

- Plenty of solutions for Distributed Systems
- Scalable solutions to tried-and-true messaging patterns
- If it’s good for actor-to-actor communication (Inter-Process Communication), it is good for distributed systems too
- Solution is dependent on system topology
Get an exhaustive list of (more than 60) messaging patterns:


To get started with MQTT:

- https://www.hivemq.com/mqtt-essentials/

Some LabVIEW MQTT Links:

- https://github.com/cowen71/mqtt-LabVIEW  (Client)
- https://github.com/LabVIEW-Open-Source/LV-MQTT-Broker  (Broker – in dev)
Before you go, take the survey.
Stay Connected During and After NIWeek

ni.com/niweekcommunity
facebook.com/NationalInstruments
twitter.com/niglobal
youtube.com/nationalinstruments
Other Messaging Patterns With Honorable Mentions
Additional Slides to keep in reserve in case of questions:

- Maybe a slide to address any questions about application safety, data security and encryption example.
- …