Reliable Messaging in Vantiq

Mission-critical applications often have reliability requirements. One of the most common requirements is that a consumer or subscriber of a messaging queue (or Vantiq topic) is guaranteed to see every event. Delivery guarantees for messaging and event systems typically fall into one of three categories:

- At-Most-Once: There is one and only one attempt to deliver the event to a subscriber.
- At-Least-Once: Each subscriber is expected to acknowledge receipt of the event, and if no acknowledgment is received the event will be redelivered until an acknowledgment is received.
- Exactly-Once: Each subscriber will successfully see the event once, and only once.

Vantiq supports the use of “at least once” delivery semantics for selected events. Exactly once delivery semantics are not supported.

Enabling Reliability

With Vantiq, there are three resources that produce events that can be delivered reliably: Service Event Types, Topics, and Sources. The default delivery guarantee is At-Most-Once. Vantiq will attempt to deliver an event to any rule or subscription that is triggered off of that resource, but if something goes wrong (for instance the rule fails, or a network outage occurs) then the event is effectively lost. Note that correlated rules that combine multiple triggering conditions cannot currently be made reliable.

Resources can be made more reliable by enabling the `isReliable` flag on creation, which will change the delivery guarantee from At-Most-Once to At-Least-Once. Resources that are marked as reliable can also be configured with a `redeliveryFrequency`, which is the number of seconds to wait for an acknowledgment from each subscriber before redelivering the event. Note that the first redelivery will take place sometime between one and two times the configured `redeliveryFrequency`. For instance, if the `redeliveryFrequency` for a reliable topic is `5` (i.e: 5 seconds), then an unacknowledged event will be redelivered between 5 and 10 seconds after the event initially arrived.

At-Least-Once delivery implies a chance of multiple redeliveries, so developers of reliable applications need to be prepared to deal with duplicate events. Wherever possible, the behavior of a rule triggered off of a reliable resource should be idempotent.

Note that a Source marked as reliable defines the delivery guarantee of inbound events, events received by a Source and delivered to a Vantiq resource such as a rule or subscription. Reliability of outbound events published through a Source to an external system (e.g., MQTT broker) is defined on the Source configuration document (e.g., MQTT QoS configuration property). For Source configuration, refer to the External Sources documents.

Ingesting Reliable Events

An interesting question in all reliable messaging systems is at what point does a message become reliable, and to what point is reliability guaranteed.

When a message arrives from a non-reliable source, or an event is published to a non-reliable topic, the event is held for just long enough to relay the event to every subscriber. This is insufficient to guarantee reliability because a hard crash of the machine the Vantiq server is running on will result in the loss of any data in memory. To ensure reliability it is necessary to write each event to persistent storage on disk so the event can be retrieved for redelivery on restart. An event cannot truly be considered reliable until it is recorded to persistent storage.

For events arriving on reliable service event types or topics, the event is stored on disk before a response is returned to the requester. The two most common ways to publish such events are through a POST to the REST API, or through a `PUBLISH` statement in VAIL. In the case of publishes via the REST API, the event will be stored persistently to disk prior to sending a response to the requester. If the response code is 200, the requester can assume the message will be reliably delivered. In VAIL, a `PUBLISH` statement that does not produce an exception constitutes a guarantee that the event will be reliably delivered.

Sources are more complicated, for two reasons. First, not all source types support reliability. Currently, only the following sources support reliability:

- AMQP
- GCPS
- Kafka
- MQTT

Second, reliability between the source and Vantiq depends on how the external source is configured and how the source in Vantiq is configured. For instance, if you configure an MQTT source in Vantiq for At-Least-Once delivery, but the MQTT server and topic are not similarly configured, then it is possible a message could be sent to the MQTT server and never arrive in Vantiq as an event. Getting these configurations to match up, and validating that the behavior is reliable, is challenging and dependent on the type of source. For assistance with reliable sources, please contact support@vantiq.com.

Acknowledging Events

When sources and topics are marked as reliable, all rules triggered off of events from that resource are expected to acknowledge the event by calling `Event.ack()`. For the sake of application development simplicity, it’s safe to call `Event.ack()` on events from an unreliable topic or source, in which case the ack is treated as a no-op.

If a rule fails to acknowledge the event, redelivery is guaranteed for the lifetime of the event. Reliable events have a time to live, which is configurable via the `redeliveryTTL` of the reliable source or topic, after which the event will be discarded. The `redeliveryTTL` is the number of seconds for which an event will be considered for redelivery after it is initially received. For instance, if the `redeliveryFrequency` is `30`, and the `redeliveryTTL` is `3600`, then the event will be redelivered every 30 seconds for up to an hour after the event was first created. If none of those redelivered events are acknowledged over the hour, then the event will be automatically ackd and redelivery will stop.

Any rule that is triggered off of a reliable topic is expected to acknowledge events after reaching a critical point in the rule execution. If a rule fails after an event has been acknowledged then no redelivery is guaranteed, so only acknowledge an event after the completion of any mission-critical work in the rule.

For Visual Event Handlers, the system handles processing of reliable events so that it can maintain reliability and preserve the handler's semantics.

Chaining Reliability

Applications in Vantiq are often comprised of a collection of rule and procedure executions, with these rules and procedures producing events that trigger more rules, which can produce more events, and so on. This composition is useful to separate out different pieces of logic, and since rules can execute in parallel, it is also a useful way to get the most performance out of the system. The downside is that this composition can create lots of places where events are created and can then be lost in the case of failures.

Reliability can be extended across multiple rules chained together by publishes to different topics by ensuring the chaining publishes happen before the original rule acknowledges the original event. For example, take the following two rules:

```
RULE FirstRule
WHEN EVENT OCCURS ON "/sources/MyReliableSource" AS event

// assume the event has a property called data, which is an array
// of objects, each of which we want to process independently
FOR (obj in event.data) {
    PUBLISH obj TO TOPIC "/my/reliable/topic"
}

Event.ack()
```

```
Rule SecondRule
WHEN PUBLISH OCCURS ON "/my/reliable/topic" AS event

UPSERT SensorData(event.newValue)

Event.ack()
```

Assuming the source named `MyReliableSource` and the topic `/my/reliable/topic` are both configured as reliable, this creates a chain of two reliable rule executions. From the point at which the initial source message arrives on `MyReliableSource`, we can guarantee that SecondRule will execute (at least) once for every object in the initial event's data array.

Consider the various points at which execution could fail:

- If the system fails after receiving the initial message on `MyReliableSource`, but before FirstRule acks the event, redelivery of the initial event is guaranteed to FirstRule.
- If the FirstRule reaches the `Event.ack()` call, that implies all of the above code ran successfully, meaning the `PUBLISH` calls in the for loop completed successfully, from which we can infer all of those events will be reliably delivered to the SecondRule.
- If the UPSERT in SecondRule fails, for instance because there were too many competing database requests and the organization maxed out their quota, the ack at the end will not be reached and the event on `/my/reliable/topic` will be redelivered.

Note that each of these places where reliability enables the redelivery of events is *also* a point at which duplicate events can be delivered if acknowledgments are missed. For instance, if the original event on `MyReliableSource` is delivered successfully, but the FirstRule fails midway through the execution of the for loop, then the original event will be redelivered and ALL iterations of the for loop will repeat. This can lead to the SecondRule seeing duplicates of events that it properly processed and acknowledged.

Reliability in Third Party Applications

External applications may subscribe to a reliable Vantiq resource through one of the offered SDKs and specify whether or not the subscription is persistent. Creating a persistent subscription guarantees at least once delivery of all events from reliable resources. This is accomplished by setting the persistent flag to true in the subscription parameters. In response to the initial subscribe request, the Vantiq server will send a message back to the client containing the `requestId` and unique identifier for the persistent subscription, `subscriptionName`. The combination of the `requestId` and unique `subscriptionName` are used to acknowledge a message or re-connect to a broken subscription.

The following is an example using the Vantiq NodeJS SDK to subscribe to a reliable topic and acknowledge messages as soon as they are received:

```
var SERVER   = "https://dev.vantiq.com";
v = new Vantiq({ server: SERVER, apiVersion: 1 });

 v.subscribe('topics', '/reliableTopic',  {persistent: true}, (r) => {
     if (r.body.subscriptionName) {
         //Server response with subscription information (not a topic event)
         subscriptionName = r.body.subscriptionName;
         requestId = r.body.requestId;
    } else {
        //r.body is an event on the subscribed topic. Acknowledge that we received the event
        v.acknowledge(subscriptionName, requestId, r.body);
    }
})
```

If the connection is broken, the subscription can be re-established using the same subscriptionName and requestId, and unacknowledged messages will be redelivered

```
v.subscribe('topics', '/reliableTopic', {subscriptionName: subscriptionName, requestId: "/topics/reliableTopic",
    persistent: true}, (r) => {
    v.acknowledge(subscriptionName, requestId, r.body);
})
```

The NodeJS SDK NodeJS SDK and Java SDK are available to download.

Performance Concerns

Reliability comes at a cost. That cost is primarily associated with the cost of storing every event during ingestion so it is available for redelivery. When a publish occurs on a non-reliable topic during normal system operation, the event can be delivered to locally subscribing rules without delay, in single digit milliseconds or even in sub-millisecond time depending on the hardware the Vantiq system is running on and what other work is going on in the system. Adding reliable storage means putting a database operation in the middle of this event delivery pathway and can lead to a 50% increase in the event delivery latency.

Given these costs, it's necessary to weigh the benefits of At-Least-Once delivery against the costs. For certain situations, the cost is an acceptable trade off:

- Almost always when safety is concerned, the benefits of reliability are worth the cost.
- Events that occur infrequently but are high-value, such as customers placing an order, are too expensive to lose.

On the other hand, there are plenty of situations where reliability isn't worth the price:

- Repetitive sensor reading that come in at regular intervals. It's often acceptable to just wait for the next regular reading.
- Slow changing data, where trends are detected over sequences of events and a single missing point won't matter.
- Events that can be recreated from state already stored persistently in the database can be made reliable by other means.

Keep in mind, you can achieve higher throughput with less hardware by not enabling reliable messaging.