EdgeX on NATS

Alex Ullrich
April 2022

onetrack CONSULTING

Introduction

The NATS messaging system (https://nats.io) is a top quality cloud messaging application with some unique characteristics that make it well suited to all kinds of EdgeX deployments.

To start NATS acts more like a messaging sink than a traditional broker - for a use case like ours in the standard “core” mode messages are routed to hot subscriptions and happily discarded if no subscription exists. This is part of what makes it so useful for request/reply APIs and other uses beyond simple pub/sub.

The relatively new NATS Jetstream (https://docs.nats.io/nats-concepts/jetstream) adds persistence that can be used for behavior more like we are used to with MQTT and others.

Why NATS?

NATS itself is an extremely high quality messaging platform and to me with the addition of Jetstream I think it is uniquely positioned to work throughout the entire stack. Its cloud capabilities are well proven at this point and I have been impressed with how it runs at the edge. It has a very small docker image size, excellent performance and is available on ARM64.

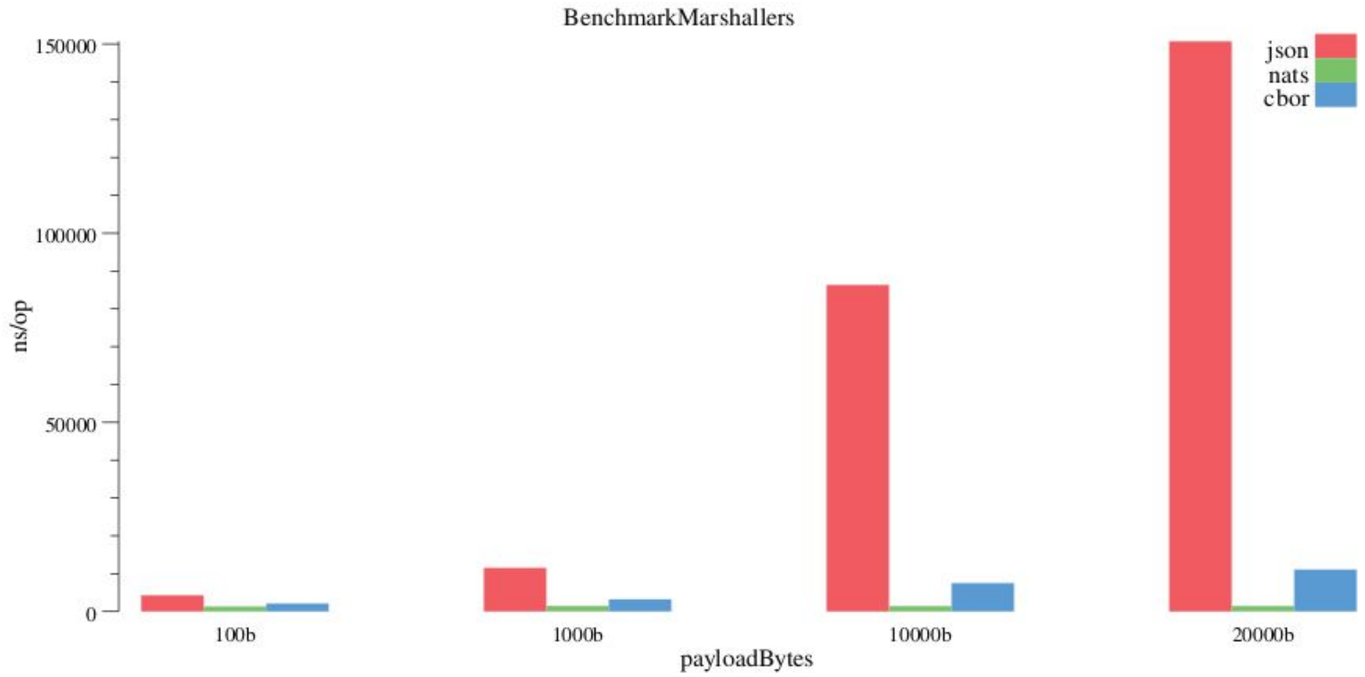
It also offers some interesting clustering capabilities that I have not looked into yet but would like to explore for HA options in multi-gateway deployments, and the ability to easily spin up embedded servers could make for some interesting unit test possibilities in the app service SDK.

NATS Protocol

NATS offers a simple text based wire protocol that performs well and is relatively easy to implement clients for. I do not have first-hand experience in this area but I don't see any reason it could not be used to publish by certain classes of device that use MQTT to communicate now.

The inclusion of metadata in the protocol enables "native" transmission of the EdgeX message envelope. This can yield tremendous throughput improvements by removing a now extraneous layer of marshaling / unmarshaling. JSON/CBOR content types can still be optionally supported for cross compatibility. Benchmarks included on next page.

Marshalling Benchmarks



JetStream

Beyond enabling durable messaging JetStream brings some interesting capabilities including:

- At-rest encryption for messages could simplify handling of sensitive data at the edge
- Streams can be consumed according to client needs
  - A normal app service can listen for only new messages on a topic
  - A less normal app service can listen for messages starting 15 minutes ago to build up internal state that it needs to operate.
  - A patch for a service that is known to have failed due to an upstream vendor change and lost messages from sequence id XXXX can subscribe from sequence id XXXX in stream.

More info can be found at https://docs.nats.io/nats-concepts/jetstream

Security / Authentication

This is another area I have not dug into a ton but NATS supports a wide array of authentication options that can simplify device onboarding including:

- Token
- Username/Password
- TLS Certificate
- NKEYS (https://docs.nats.io/running-a-nats-service/configuration/securing_nats/auth_intro/nkey_auth)
- Decentralized JWT (https://docs.nats.io/running-a-nats-service/configuration/securing_nats/auth_intro/jwt)

NATS also offers accounts for multi-tenant support but I am not sure how much that helps us in normal use.

MQTT Support

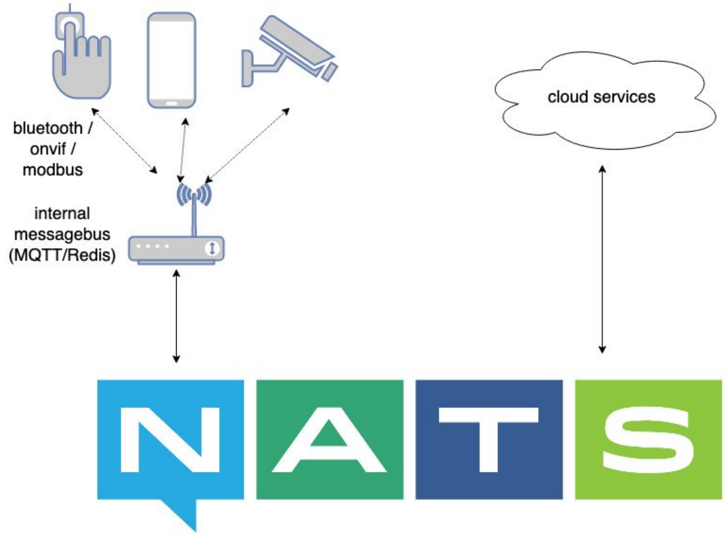
With the addition of Jetstream NATS can support MQTT type QoS levels, and in fact even offers support for the MQTT 3.1 wire protocol: https://docs.nats.io/running-a-nats-service/configuration/mqtt

This support could be leveraged to connect legacy devices to your application or to use prebuilt EdgeX services like device-mqtt-go. Note that because MQTT does not have metadata support in the protocol you would need to use JSON/CBOR marshaling here.

Network Topology Example - Current

It is outside the scope of the message bus but NATS offers a network topology unlike anything we currently have in the EdgeX ecosystem. In addition to the ability to form clusters in multi-machine environments it can form a mesh using the concepts of super-clusters and gateways (https://docs.nats.io/running-a-nats-service/configuration/gateways).

We'll explore this briefly using an example of a company with smart manufacturing operations using a NATS based backbone like NGS (https://synadia.com/ngs) for their enterprise messaging needs. Right now an EdgeX deployment into this company might use MQTT or redis for the message bus on their IoT gateways and have the last step of their pipeline publish to NGS via a Jetstream client.

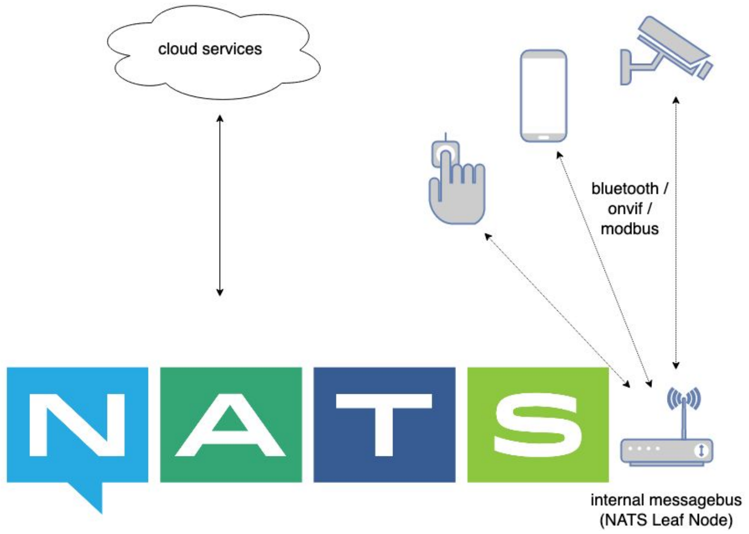


Network Topology Example - Pure NATS

With NATS running throughout the stack, brokers running in an edge deployment can enlist as leaf nodes to the company's greater NGS supercluster.

- If multiple hardware devices are available the brokers can form a local cluster that **then** enlists as a leaf node.
- Operating through NATS clients, edge services can act on local data for low latency (requests will only go to the supercluster if it is not found locally first).
- The final app service acting on a machine can then publish its output to the local broker - ***no additional northbound credentials needed***
- Data can then flow north through the NATS infrastructure itself to be consumed by cloud services.

This is vastly oversimplified for time but I think starts to show the flexibility that a system like NATS can offer for the increasingly interesting needs of EdgeX applications in the context of larger enterprise systems.


cloud services
bluetooth / onvif / modbus
NATS
internal messagebus (NATS Leaf Node)

Future Considerations: Key-Value Store

The persistence layer added with Jetstream has also been exposed as a (currently experimental) distributed key-value store. As this evolves I think it will be tremendously valuable to EdgeX. I have found this capable of driving the app service's store-forward loop quite well and in a way that offers true persistence to disk (not limited to memory allocated to redis container). This could prove useful for devices with extremely long intervals between network connectivity or large saved payloads.

I would like to try to back the core-metadata service with this store as well. We don't use the core-data service and this would effectively make redis an optional dependency for our deployments.

Why NOT NATS?

The major reported downside so far seems to be about a 1.5 MB increase in binary size. I need to find out more about how this is measured to see if I can play some golf with the client.

It also brings the baggage of more code to support but the NATS server and client are both mature at this point and seem very well supported to me, minimizing risk to the project.

- https://github.com/nats-io/nats-server/pulse/monthly
- https://github.com/nats-io/nats.go/pulse/monthly

Conclusions

I am sure that coming at this from a position of my own needs I have missed some of the negatives of adding NATS support to EdgeX but I think I would need to find a long list of them to make up for the positives outlined in this presentation. Removing the extra layer of envelope marshaling seems like it could really provide a step forward in message bus performance, maybe paving the way for a future where MQTT5 and NATS are the preferred ways to run it.

I think the key-value store is worth keeping a very close eye on as well. Redis' memory map limitations can be a struggle on some devices and I think the KV API could translate really well to backing the store and forward loop and metadata service. I doubt it will take on some of the features that make redis' read performance so appealing for core-data but with it being in active development there may be some opportunity to engage with the NATS team and help shape it to fit our needs.