WE SCALED IT

Scale testing Eclipse Hono on OpenShift

Dejan Bosanac, Jens Reimann
Senior Software Engineer
EclipseCon Europe 2018

WHO WE ARE

Working for Red Hat

DEJAN BOSANAC

Twitter: @dejanb

GitHub: @dejanb

JENS REIMANN

Twitter: @ctron

GitHub: @ctron

Senior Software Engineers

Middleware / Messaging

Focus on IoT

ECLIPSE HONO

Project Introduction

Connect. Command. Control.

“Eclipse Hono™ provides remote service interfaces for connecting large numbers of IoT devices to a back end and interacting with them in a uniform way regardless of the device communication protocol. “

…

“Hono specifically supports scalable and secure ingestion of large volumes of sensor data by means of its Telemetry and Event APIs.“

THE CLAIM

redhat.

ECLIPSE HONO

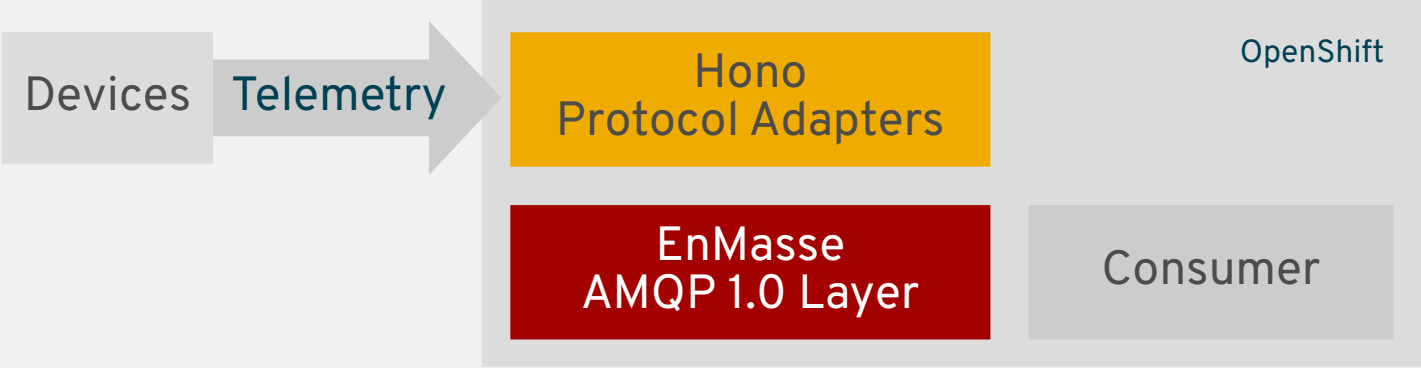
Connect. Command. Control.

Hono provides scalability for IoT

- Hono is scalable
- Hono allows to connect a large numbers of devices
- A core functionality is the ingestion of telemetry data

BASIC ARCHITECTURE

This is what was interesting to us.



WE'VE DONE THIS BEFORE

This wasn't the first test. We already tested Hono previously, but not with a focus on scalability.

Nov 2017

First attempt to run a larger test.

Jan 2018

Validate that our fixes actually worked.

Yes, they did.

July 2018

Test if Hono scales.

… does it?

THE LAB

redhat.

PICK A LAB

Luckily we have more than one, we went with the one we already knew.

Lab capabilities

- 148 nodes
- 1888 cores (3392 threads)
- 1.1PB of storage capacity

Our reservation – for 3+1 weeks

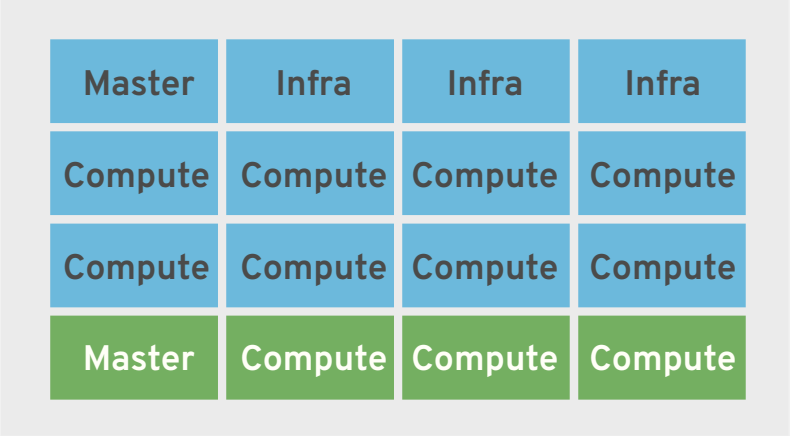
- 16 DELL R620 - 2x E5-2620, 64/128 GB RAM
- 192 total cores, 384 total threads
- 1536 GB RAM

HOW WE SET IT UP

A tale of two clusters.

Two separate OpenShift clusters.

- One for Hono and its infrastructure
  - 3 infrastructure nodes
  - Cluster native storage (Gluster)
- One for simulating devices and consumers
  - Was it necessary? No, but easier to work with.
- Dedicated masters

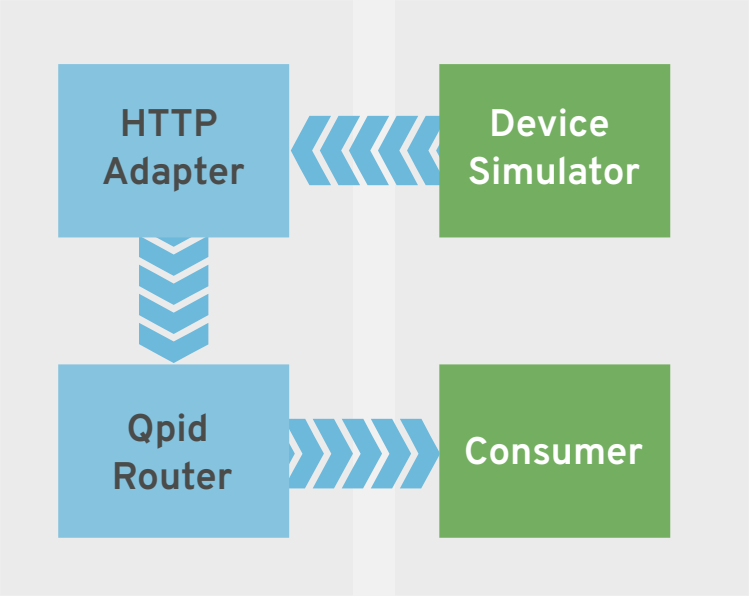


WHAT WE WANTED TO TEST

The use case we focused on.

Ingestion of telemetry data via HTTP.

- We also did tests around events, MQTT. But the focus was on telemetry over HTTP.
- We wanted to establish a baseline for telemetry over HTTP, and then start playing around, making modifications.
- We also wanted to validate a view fixes and ideas we had in the past.
- Take a top down approach. Use the defaults, then optimize.



SOFTWARE COMPONENTS

A few facts about the software we used.

- OpenShift (OCP) 3.9.1
- RHEL 7.5
- EnMasse 0.21

- Eclipse Hono 0.7
  - Snapshot version, close to M2
  - Forked on GitHub – “redhat-iot/hono”.
- Hono Simuator – “redhat-iot/hono-simulator”
  - Device simulator
  - Consumer simulator
- Hono Scale Test – “redhat-iot/hono-scale-test”
  - Scale test deployment and tools

MAKING PLANS

A few simple steps

- Get the test setup up and running
- Create a baseline for testing, max out the number of messages/s
- Start automated testing, ramping up the workload, measuring the results, seeing if it scales.
- Test a few scenarios: OpenJ9, Threading settings, events, …

UP AND RUNNING

SCALING EVERYTHING

SCALING EVERYTHING

SCALING EVERYTHING

MAXING IT OUT

PLEASE NOTE

Beware absolute numbers.

Results may be different

in your environment.

Do your own tests!

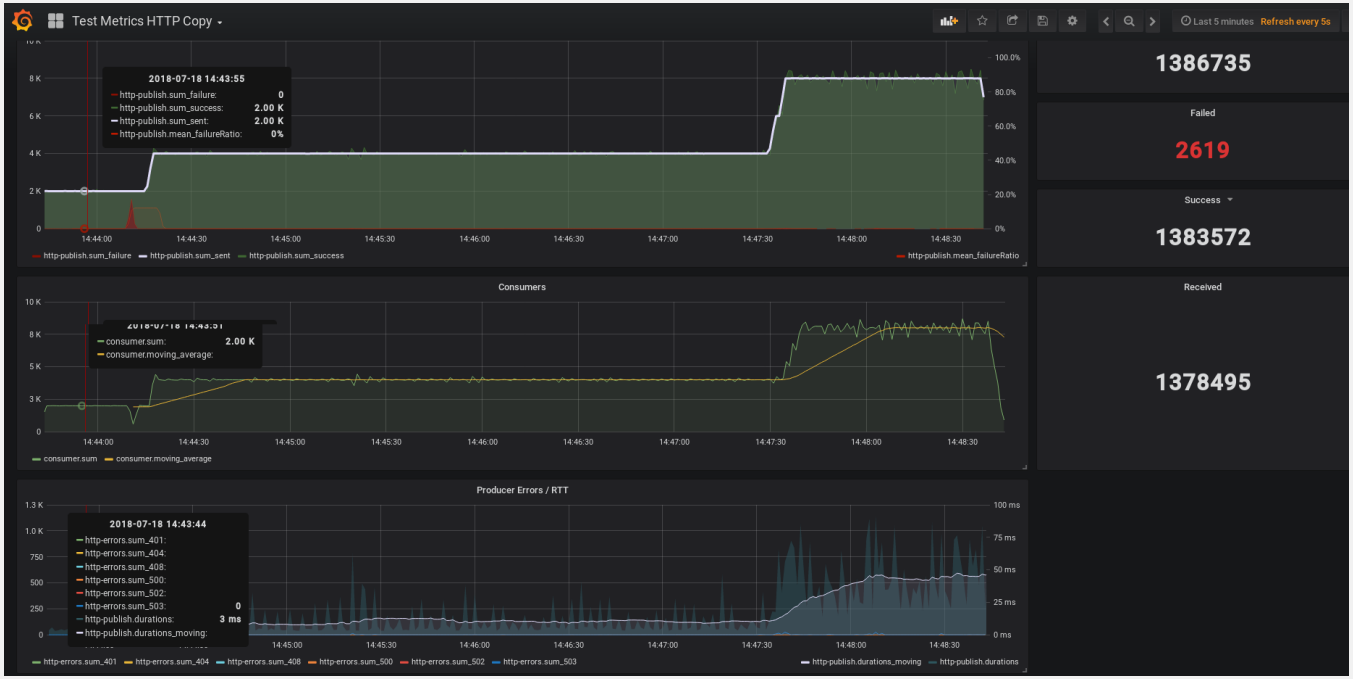
It is all Open Source!

MEASURING PERFORMANCE

Generating and consuming IoT workload.

- The simulation cluster runs X pods, simulating Y devices, sending one message each second.
  - How many attempts to send?
- The simulation cluster runs Z pods, consuming messages as fast as possible.
  - How many messages received?
- Hono handles back-pressure by rejecting messaging at the protocol adapter. Has an HTTP code for that.
  - Which errors?
- We can calculate the error rate, messages that couldn't be sent due to an error/reject on the cloud side.
- We can scale up producers, consumers and infrastructure components (EnMasse, Hono).

CATCH A GLIMPSE



CATCH A GLIMPSE

QPID Routers	1	1	1	1	1	2	2
Devices / Pod	1,000	1,000	1,000	1,000	1,000	1,000	1,000
Pods	1	2	4	8	16	32	40
Messages / second	1,000	2,000	4,000	8,000	16,000	32,000	40,000
Request RTT (ms)	<5	<5	11	45	20	50	150
Consumers	20	20	20	20	20	20	8
Consumer Credits	1,000	1,000	1,000	1,000	1,000	1,000	1,000
Hono Adapters	1	1	1	1	2	5	10
Receiver Link Credit	1,000	1,000	1,000	1,000	1,000	1,000	1,000
Hono Device Registry	10	10	10	10	10	10	10
Receiver Link Credit	1,000	1,000	1,000	1,000	1,000	1,000	1,000
Failure Ratio (%)	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 1
Average Throughput (msg	1,000	2,000	4,000	8,000	16,000	32,000	40,000
Messages / Instance	1000	2000	4000	8000	8000	6400	4000

THE BASELINE

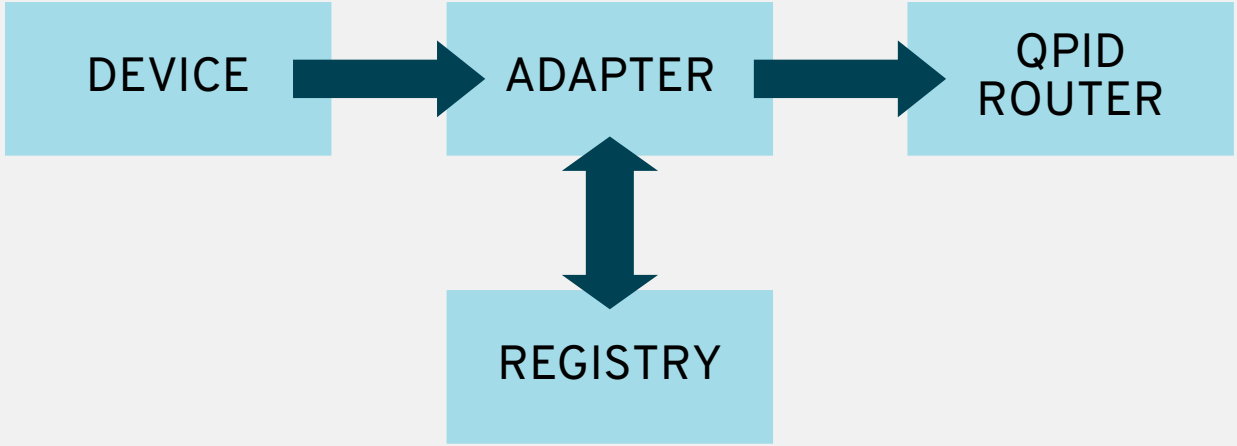
Maximum of 80.000 msgs/s

- 2 Dedicated QPID router nodes
- Device registry service as a sidecar
- 8 nodes running Hono services
- Using a HTTP client library which scales.

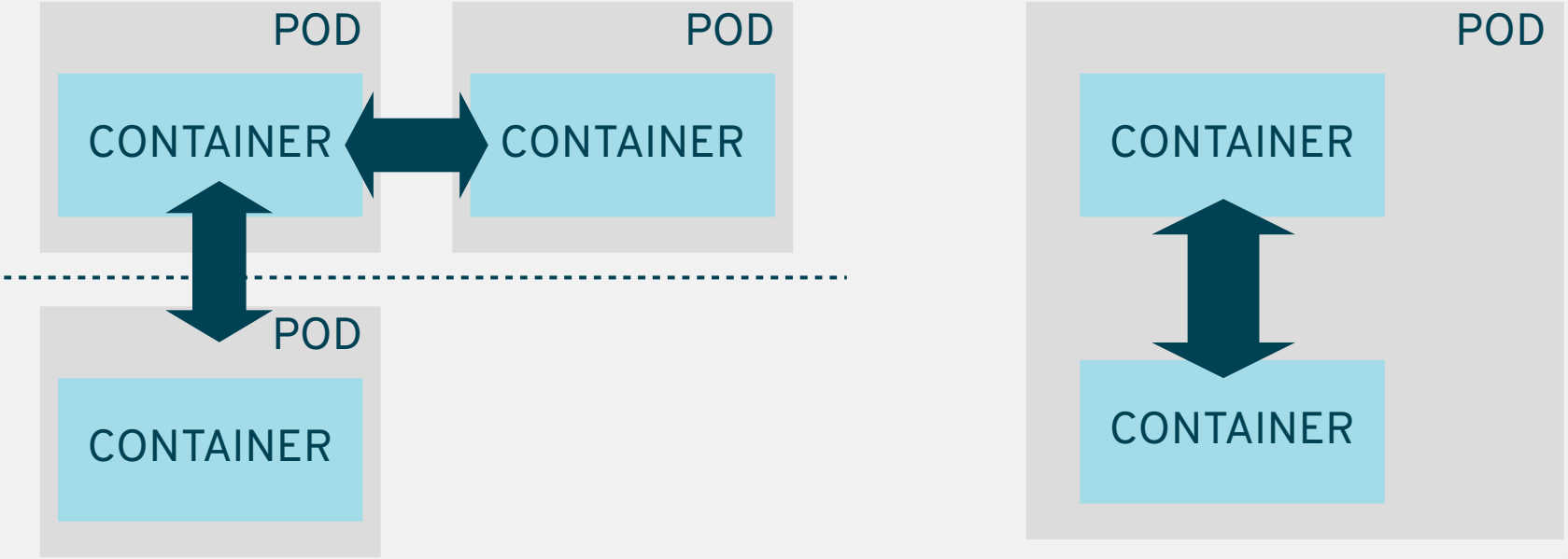
- Throughput is limited by CPU and Network resources
- RAM is not an limiting factor
- Neither is disk (for telemetry!)

THINGS WE LEARNED

DEVICE REGISTRY SIDECAR



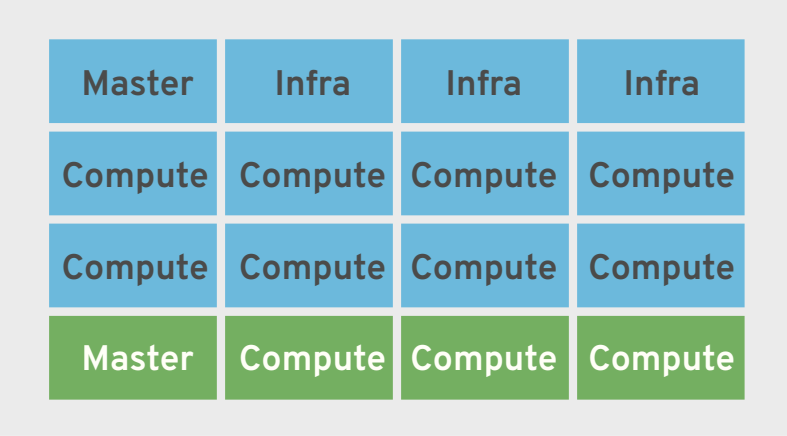
DEVICE REGISTRY SIDECAR



QPID ROUTER NODES

Putting QPID router on infra nodes helped streamline the message flow.

- Having dedicated QPID router nodes, did reduce the inter node traffic.
- Using node ports (vs routes) improved the performance a lot.



A PERFORMANT HTTP CLIENT

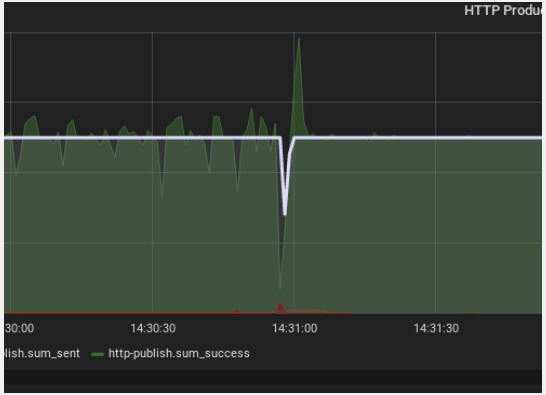
Finding a performant HTTP client library is tricky.

- Some of the initial limitations came from the device simulator, which could not keep up sending requests.
- **OkHttp** – Simple API, average performance
- **Java default HTTP client –** Complicated, bad performance
- **Apache HttpClient** – Complicated, bad performance
- **AHC** – Average API, good performance
- **Vert.x HTTP Client** – Good API, good performance
- We went with vert.x it also provides a simple API and we already knew it.

NATIVE SOCKETS AND TLS

Vertx/Netty allow the use of “epoll()” and “libssl.so”

- Switching to native TLS reduces the CPU load, especially over the long run.
- Switching to netty epoll() improves the throughput and takes load off the JVM.



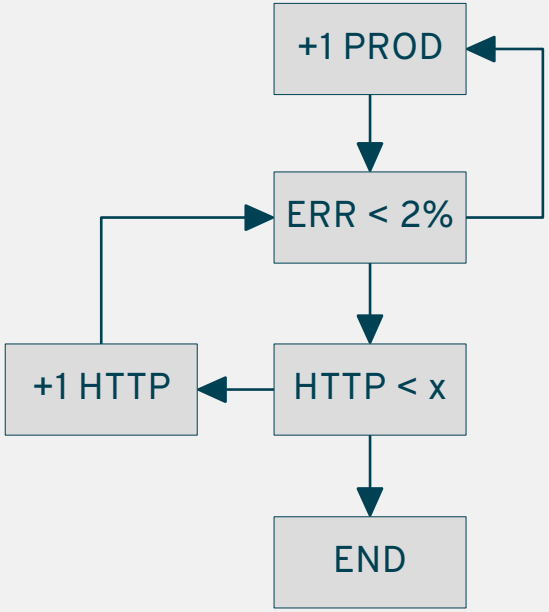
AUTOMATED SCALING TESTS

DEFAULT SCENARIO

Scale. Wait. Stabilize. Repeat.

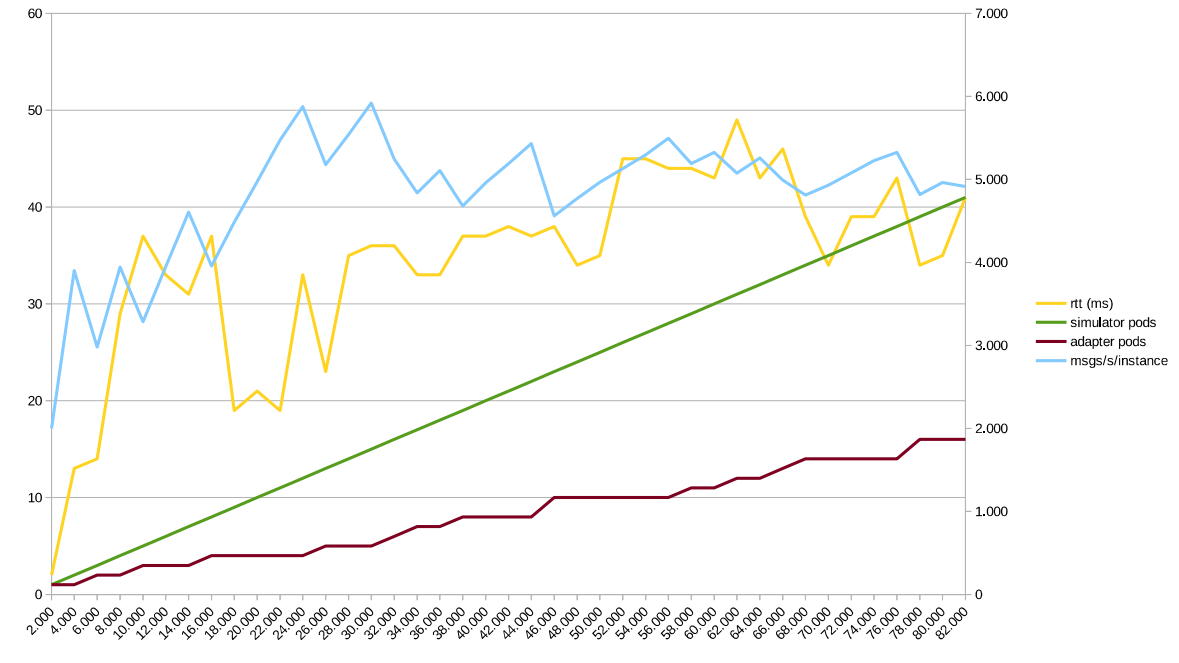
We wrote an automated test, to run the same scenario with different parameters.

1) Scale up producer
2) Wait for a stable message flow
3) If the flow is stable (error rate), continue with 1)
4) Otherwise scale up Hono adapters until we hit a limit
5) End the test when we reached the adapter limit



DEFAULT SCENARIO

Results.



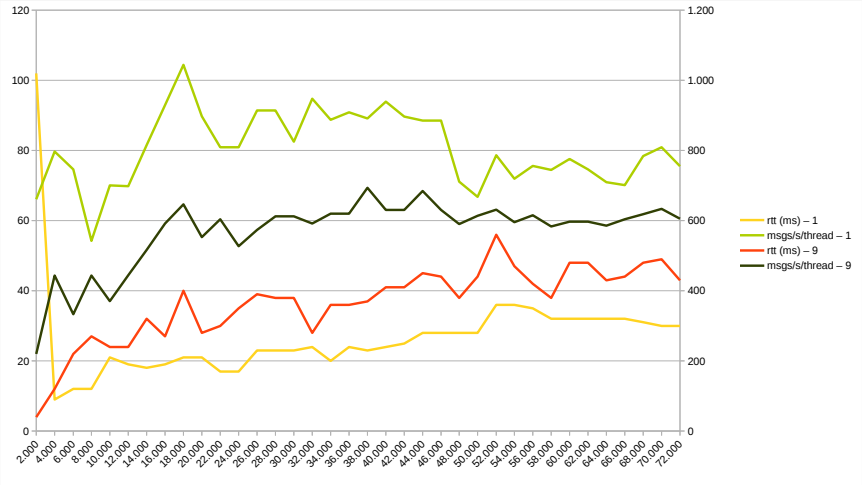
THINGS WE TRIED

I/O THREADS

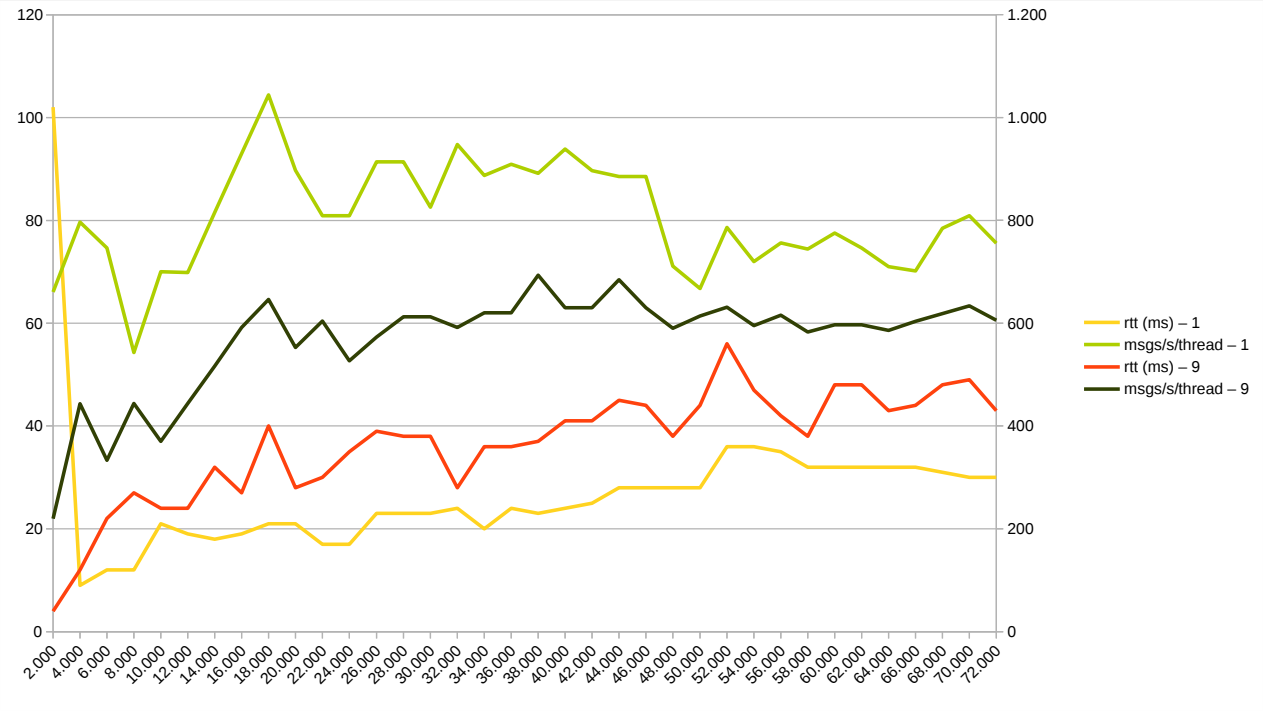
Comparing 1 vs 9 vert.x worker threads.

The same scenario was run, with a different thread configuration.

- Vert.x assigns “verticles” (services) to a worker thread.
- Is 1 JVM with 9 worker threads better than 9 JVMs with 1 worker thread?



1 vs 9 IO THREADS



OPENJ9 vs OPENJDK

Testing with Eclipse OpenJ9 vs OpenJDK

The claim is, that OpenJ9 consumes less memory, having the same performance.

- Create a fabric S2I builder for OpenJ9
  - ctron/s2i-java-openj9
- Run the default scenario with it

- No real difference could be seen.
- RAM never was a limiting factor.
- But the performance was the same.

SUMMARY

YES, IT SCALES!

YES, IT SCALES.

So, what does that mean?

In the amount of resources we had available, we could find that:

1) In the default scenario, with 9 worker threads, adding an additional pod increased the cluster capacity by ~5.000 msgs/s.
2) We could repeat that until we hit out cluster limit.

IS THAT GOOD ENOUGH?

Can we do better?

There is always room for improvement.

- The request/response client had some issues with back-pressure handling (fixed in 0.7)
- There was a memory leak, building up stress on the JVM (fixed in 0.7)
- There was a performance issue in the AMQP 1.0 encoding (fixed in 0.8-M1)
- And we continue to improve it …

THINGS TO CONSIDER

Deploying a cloud IoT solution can be tricky.

Keep in mind:

- The device registry, which is not part of Hono, has a huge impact on the overall performance.
- Consumers also play a role in the overall performance. We simply threw away the payload. This may not be viable business case for you ;-)

TRY THIS AT HOME

Everything is Open Source, you can do your own scale test.

- Eclipse Hono
  - eclipse/hono
- Scale Test Deployment
  - redhat-iot/hono-scale-test
- Hono Simulator
  - redhat-iot/hono-simulator

QUESTIONS?

THANK YOU


plus.google.com/+RedHat


facebook.com/redhatinc


linkedin.com/company/red-hat


twitter.com/RedHatNews


youtube.com/user/RedHatVideos