Performance Testing Requirements

Formal EdgeX testing is currently focused on ensuring the functional aspects of EdgeX are working correctly. This level of testing involves a number of automated test suites for Unit, Integration and Blackbox API testing and validation. A set of tests that can be used to determine the non-functional characteristics of EdgeX, specifically performance, is also required. This document defines the scope of these tests and also identifies the key test cases required for each category of tests. Recommendations for possible testing approaches are also made.

Scope

EdgeX Performance Testing should minimally address the following general test case categories:

- Footprint – tests that measure both the executable file size and docker image size on disk for each EdgeX microservice.
- Memory – tests that measure the memory (RAM) consumed by each running EdgeX microservice for different load conditions. For example, measure the memory consumed when varying number of concurrent EdgeX Devices writing data into EdgeX, with varying data payload (Event /Reading collections) sizes and sampling rates.
- Service Startup - tests that measure how long it takes for each EdgeX microservice to startup.
- Latency – tests that measure the time it takes to take specific actions using EdgeX for different load conditions. For example, measure the time it takes for an EdgeX Device to read a data value and then export the same data value via the Export Service.
- CPU – tests that measure the CPU consumed (as a % of total CPU availability) when running EdgeX microservice under different load conditions.
- Throughput – tests that measure the data throughout (bytes/second) that can be achieved from data being read by an EdgeX Device to the data being exported via the Export Service for different load conditions.

In addition, there are an additional set of performance test that are specific to each microservices including:

- Core Data – tests that measure the read/write performance of Core Data and underlying database for different load conditions. For example, varying the number of concurrent clients writing to Core Data with different payload sizes and measuring the time it takes to issue each write call, or varying the number of concurrent clients reading different size collections of events from Core Data and measuring the time it takes to issue each read call.
- Support Logging – tests that measure log write and log query performance of the Logging Services for different load conditions. For example, measure the time it takes to write a log message for a varying number of Logging (write) clients and varying payload sizes, or measure the time it takes to query a log message for a varying number of Logging (read) clients, varying payload sizes, and varying number of log entries already present in the system.

(MKB: these should be do-able in isolation of EdgeX given it is a separate service?)

- Support Notifications – tests that measure the time it takes from when a Notification in sent to the service to the point the message has been pushed to all registered receivers for different load conditions. For example, measure the fan-out performance where one publisher sends a Notification to the service and a varying number of clients subscribe to receive the Notification or fan-in when where a varying number of concurrent publishers send Notifications to the service and a single client subscribes to receive all of the Notifications. (MKB: these should be do-able in isolation of EdgeX given it is a separate service?)
- Core Command – tests measure the time it takes to issue GET and PUT commands to a device/sensor via the Command Service for different load conditions. For example, measure the time it takes for a varying number of concurrent clients to each issue a GET command to read a property value from a device or for a varying number of concurrent clients to set a property on a device with a PUT command.
- Rules Engine - do we need dedicated Rules Engine tests ?
- Export Services – do we need dedicated Export Service performance tests ? For example, measure the performance when writing to a specific Cloud instance (e.g. Google IoT Core)?
- Device Services – for baseline and regression test purposes many of the general performance tests outlined above may be able to be performed using a Virtual Device Service. However, it is also necessary and desirable to be able to repeat at least a subset of these tests with real Device Services (e.g. Modbus, MQTT or BACnet Device Services), perhaps connected to real device or minimally connected to a simulator. The performance of each individual Device Service will be implementation is specific.

Requirements

1. Automated tests – the performance tests must be able to be integrated into the EdgeX build/test (CI pipeline) infrastructure and run on demand. (MKB: why only performance tests here?)
2. Standalone tests - the performance tests must be able to be run standalone on a developer's desktop without dependencies on the EdgeX build /test infrastructure. (MKB: why only performance tests here?)
3. Results logging and display – the results of the performance tests must be recorded in a format that enables a set of graphical performance test curves to be produced that can be easily displayed in a web browser. This includes being able to display historical performance trends to easily determine any regression in EdgeX performance.
4. The performance tests should be able to be run on different platforms - CPU (ARM 32 and 64 bit, x86 64 bit) and OS combinations (Linux and Windows).
5. Performance tests should be able to be run against both dockerized (default) and un-dockerized version of EdgeX.

Test Cases

Footprint

1. Measure the file size in bytes of each EdgeX Microservice executable.
2. Measure the file size in bytes of each EdgeX Microservice docker image.

Memory

1. Measure the memory (RAM) in bytes consumed by each EdgeX microservice in their idle state (running but no data flowing through the system).

2. Measure the memory (RAM) in bytes consumed by each EdgeX microservice with [1, 10, 20, 50, 100] devices, reading an event with [1, 5, 10, 50, 100] readings at a sample rate of [100 ms, 1 sec, 10 sec, 30 sec, 60sec]. (MKB: payload size? payload mix?)

Service Startup

1. Measure the time it takes to startup each EdgeX microservice, this includes the time it takes to create the docker container, configure and initialize the service.
2. Measure the time it takes to startup each EdgeX microservice with existing docker containers and initialized data.
3. Measure the time it takes to startup the complete set of EdgeX microservices required to enable data to be read by a device and exported.

Latency

1. Measure the time it takes to read an event with [1, 5, 10, 50, 100] readings from [1, 10, 20, 50, 100] devices (virtual) with a sample rate of [100 ms, 1 sec, 10 sec, 30 sec, 60sec] and export it (via the Export Service).

Throughput

1. Measure the throughput that can be achieved by read an event with [1, 5, 10, 50, 100] readings from [1, 10, 20, 50, 100] devices (virtual) with a sample rate of [100 ms, 1 sec, 10 sec, 30 sec, 60sec] and export it (via the Export Service). (MKB: payload size?)

Core Data

1. Measure the time it takes to write an event with [1, 100] readings from [1, 10, 20, 50, 100] devices (virtual or emulated) to Core Data.
2. Measure the time it takes to read an event by ID with [1, 100] readings from [1, 10, 20, 50, 100] clients from Core Data.
3. Measure the time it takes to read a set of events and readings with a complex query (all events for a given device, created within a time range and with reading attached (e.g. temperature).

Support Logging

1. Measure the time it takes to write a log message from a varying number of concurrent clients [1, 10, 20, 50, 100] to Support Logging.
2. Measure the time it takes to read a log message with a simple query (e.g. specify limit) from a varying number of concurrent clients [1, 10, 20, 50, 100] from Support Logging.
3. Measure the time it takes to read a log message with a complex query (e.g. specify logLevels, orginServices, labels) from a varying number of concurrent clients [1, 10, 20, 50, 100] from Support Logging.

Support Notifications

1. Measure the time it takes to post a notification for a varying number of concurrent publishers [1, 10, 20, 50, 100] to Support Notifications.
2. Measure the time it takes to post a notification (severity = “CRITICAL”) and for it to be published to a varying number of concurrent subscribers 1, 10, 20, 50, 100] from Support Notifications.
3. Measure the time it takes to read Notification with a simple query (e.g. specify slug) from a varying number of concurrent clients [1, 10, 20, 50, 100] from Support Notifications.
4. Measure the time it takes to read a Notification with a complex query (e.g. limit, start date) from a varying number of concurrent clients [1, 10, 20, 50, 100] from Support Notifications.

Core Command

1. Measure the time it takes to issue a GET command to read property value from a device (virtual) from a varying number of concurrent clients [1, 10, 20, 50, 100] via Core Command.
2. Measure the time it takes to issue a PUT command to set a property value on a device (virtual) from a varying number of concurrent clients [1, 10, 20, 50, 100] via Core Command.

Device Services

The test cases listed previously can be driven by a Virtual Device Service or device emulator. However, EdgeX also provides a number of Device Service for communicating with real physical devices/sensors. These include:

- Modbus Device Service
- BACnet Device Service
- MQTT Device Service
- BLE Device Service
- Serial Device Service
- SNMP Device Service

Test cases for Memory, CPU, Latency and Throughout should be repeated with the Virtual Device Service replaced with the each of the real Device Services listed above communicating with either a real device/sensor or possibly a device simulator.

Note: As of Delhi and Edinburgh releases, the Java device services have been replaced with Go and C versions.

Notes

1. To smooth out any outliers or “warm-up” effects each individual performance test result will typically be calculated by taking the average over a number of repeated test iterations (e.g. 1000, but may vary depending on sample rate for each test – more for low sample rates and less for high sample rates).
2. Will require a configurable Virtual Device Service to drive many of the tests listed above. The current Virtual Device Service is written in Java but to be able to support performance testing on other platforms (e.g. 32 bit ARM) then it is likely that Virtual Device Service implementations in either Go or C (or both) will be required.

3. It is expected that a test harness/framework that can satisfy the requirements and support the scope of tests outlined above will have to be created specifically for this task. However, there are many 3rd party performance testing technologies that may form some part of the overall framework. These includes basic command line utilities such as Top (Unix) or Docker (stats) for providing CPU and Memory consumption data. There are also more sophisticated commercial container monitoring tools such as DataDog with integrated dashboarding and reporting. To support API level load testing there are both open source tools such as Bender a Go-based framework for load testing services using protocols such as HTTTP and Thrift or commercial tools such as Load Impact that can be used with existing Postman test collections (Postman is currently used for EdgeX blackbox testing) to create realistic load scenarios with multiple clients issuing REST request simultaneously. Where appropriate use of these types of tools should be considered to complement any tests/scripts that have to be written specifically to support the implementation of the test cases listed in this document.
4. May require an emulated Cloud server (a dummy server that runs on the local test node and that be used to help log performance data) as an Export Service target that can be used to support the Latency and Throughput tests listed previously.