Skeletal Support for APIv2 Implementation

Presentation to EdgeX Core WG
Michael Estrin
January 23, 2020

DELL Technologies

About Me

- Michael Estrin (m.estrin@dell.com).
- Software System Senior Principal Engineer at Dell Technologies.
- Part of the IoT Solutions Division's Platform Development Team.
- My team is responsible for
  - Dell's ongoing contributions to EdgeX.
  - Internal innovation projects built using EdgeX.
- My contributions to EdgeX
  - Common extensible bootstrapping package.
  - Simple dependency-injection container.
  - Helm Chart and Kustomize-based manifests to run EdgeX in Kubernetes.
  - Environment variable override support.
  - Refactored system management agent and Docker executor implementations.

Overview

Introduction.

Definitions.

Execution Examples.

Layered Architecture.

Project Structure.

Use-case Endpoints.

Batch Endpoints.

Use-Cases.

Router.

Controllers.

Middleware.

Common Behavior.

Wire Up.

Acceptance Tests.

Open Items.

Parting Comments.

Introduction

My goal is to provide a recorded comprehensive overview.

This is an extended speed presentation.

You won’t understand everything in this presentation after a single viewing.

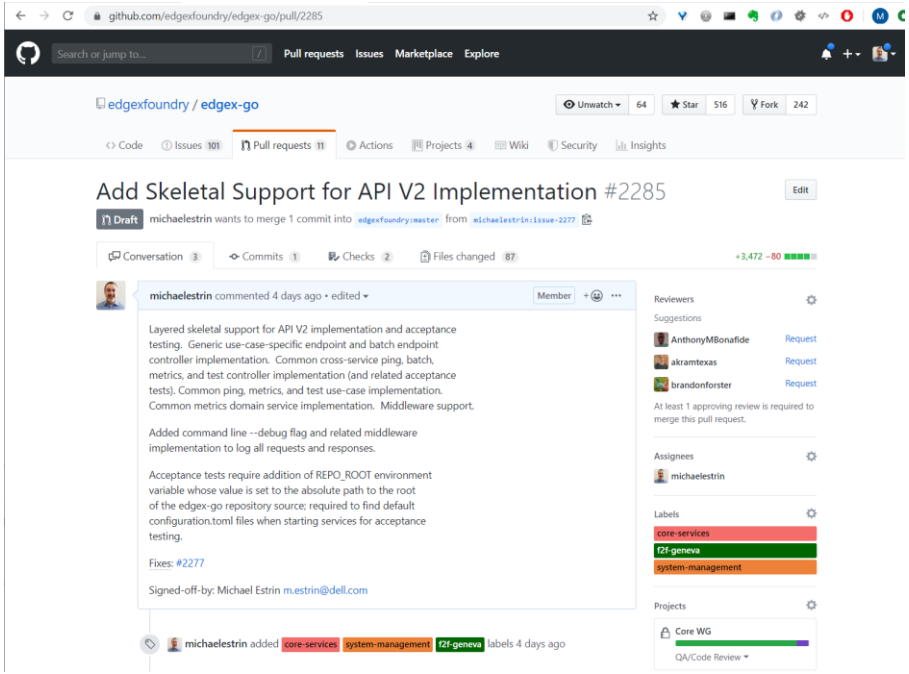
There is a lot to cover.

Unless I call for them, please hold questions.

I will address them after the presentation if there is time.

Regardless, time will be set aside in the next (January 30th) Core WG meeting for follow-up Q&A.

Draft Pull Request



https://github.com/edgexfoundry/edgex-go/pull/2285

I won’t be diving into the code in this presentation.

Work in Progress.

Incomplete.

(Enumerated later.)

Subject to change.

Demonstrate concepts.

Establish implementation patterns.

Separate concerns.

“Decouple all the things.”

Reduce risk by separating APIv2 implementation from APIv1 implementation.

Design supports the ability to add or substitute transports.

(e.g. asynchronous messaging, etc.)

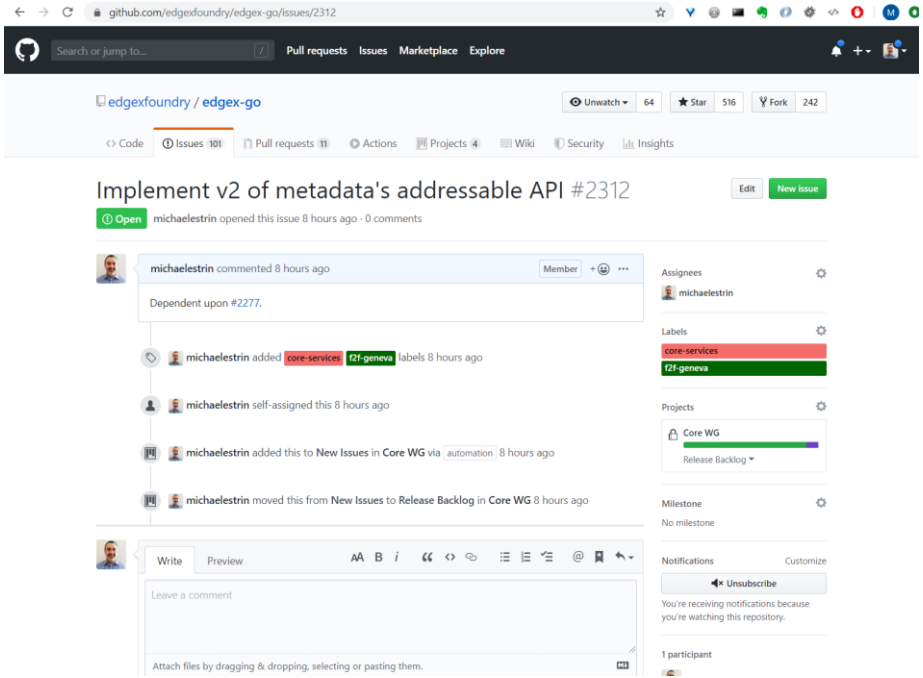
Design supports a single executable with endpoints from multiple services.

(Could be configuration-driven.)

PR does not include service-specific use-case implementation.

Implement Metadata's Addressable API; possible presentation at February 6th Core WG meeting.

Follow-on Issue.



https://github.com/edgexfoundry/edgex-go/issues/2312

Definitions

Use-case: discrete behavior initiated by external interaction.

(Could be user or another service.)

A use-case is associated with a version, a type, and an action.

Version: the major value of the semantic version of a specific use-case implementation.

(e.g. “2”)

Type: a mnemonic associated with a specific use-case implementation.

(e.g. “ping”)

Action: a generic name (HTTP method-equivalent) associated with a specific use-case implementation.

(e.g. “create”, “read”, “update”, etc.)

DTO: data transfer object. It has no behavior. It is used only for data transmission.

(Request DTO, Response DTO)

Use-case Requests vs. Transport Requests

Use-case request: a Request DTO instance that maps to single invocation of a use-case.

Use-case response: a Response DTO instance returned from a single invocation of a use-case.

Transport request: single HTTP request.

A single transport request can contain one or more use-case requests.

(Examples coming in next section.)

Use-case Endpoints vs. Batch Endpoint

Use-case endpoint: URL that takes one or more use-case-specific Request DTOs and returns corresponding Response DTOs.

Batch endpoint: “/api/batch” URL that takes one or more Request DTOs and returns corresponding Response DTOs. (Can be mixed types.)

Execution Examples

Ping (use-case endpoint, one uc-request)

```
$ curl -i -X GET  http://localhost:48082/api/v2/ping
HTTP/1.1 200 OK
Content-Type: application/json
Date: Tue, 21 Jan 2020 12:54:31 GMT
Content-Length: 19

{"response":"pong"}
```

Metrics (use-case endpoint, one uc-request)

```
$ curl -i -X POST -d "" http://localhost:48082/api/v2/metrics
HTTP/1.1 200 OK
Content-Type: application/json
Date: Tue, 21 Jan 2020 12:54:53 GMT
Content-Length: 134

{"memAlloc":696616,"memTotalAlloc":3770288,"memSys":11934200,"memMallocs":71874,"memFrees":67704,"memLiveObjects"
:4170,"cpuBusyAvg":0}
```

Test (use-case endpoint, one uc-request)

```
$ curl -i -X POST -d "{\"message\":\"foo1\",\"delay\":100}" http://localhost:48082/api/test
HTTP/1.1 200 OK
Content-Type: application/json
Date: Tue, 21 Jan 2020 12:55:22 GMT
Content-Length: 18

{"message":"foo1"}
```

(Illustrative only.)

Test (use-case endpoint, two uc-requests)

```
$ curl -i -X POST -d "[{\"message\":\"foo2\",\"delay\":0},{\"message\":\"foo3\",\"delay\":0}]" http://localhost:
48082/api/test
HTTP/1.1 207 Multi-Status
Content-Type: application/json
Date: Tue, 21 Jan 2020 12:56:21 GMT
Content-Length: 39

[{"message":"foo3"},{"message":"foo2"}]
```

(Illustrative only.)

Ping (batch endpoint, one uc-request)

```
$ curl -i -X POST -d "[{\"version\": \"2\", \"type\":\"ping\",\"action\":\"read\",\"content\":\"\"}]" http://loc
alhost:48082/api/batch
HTTP/1.1 207 Multi-Status
Content-Type: application/json
Date: Tue, 21 Jan 2020 12:58:01 GMT
Content-Length: 77

[{"Version":"2","type":"ping","Action":"read","Content":{"response":"pong"}}]
```

Metrics (batch endpoint, one uc-request)

```
$ curl -i -X POST -d "[{\"version\": \"2\", \"type\":\"metrics\",\"action\":\"command\",\"content\":\"\"}]" http
://localhost:48082/api/batch
HTTP/1.1 207 Multi-Status
Content-Type: application/json
Date: Tue, 21 Jan 2020 12:59:37 GMT
Content-Length: 215

[{"Version":"2","type":"metrics","Action":"command","Content":{"memAlloc":714576,"memTotalAlloc":4468176,"memSys"
:11934200,"memMallocs":86018,"memFrees":79871,"memLiveObjects":6147,"cpuBusyAvg":39.519524614641185}}]
```

Test (batch endpoint, one uc-request)

```
$ curl -i -X POST -d "[{\"version\": \"2\", \"type\":\"test\",\"action\":\"command\",\"content\":{\"message\":\"
foo1\",\"delay\":100}}]" http://localhost:48082/api/batch
HTTP/1.1 207 Multi-Status
Content-Type: application/json
Date: Tue, 21 Jan 2020 13:01:59 GMT
Content-Length: 79

[{"Version":"2","type":"test","Action":"command","Content":{"message":"foo1"}}]
```

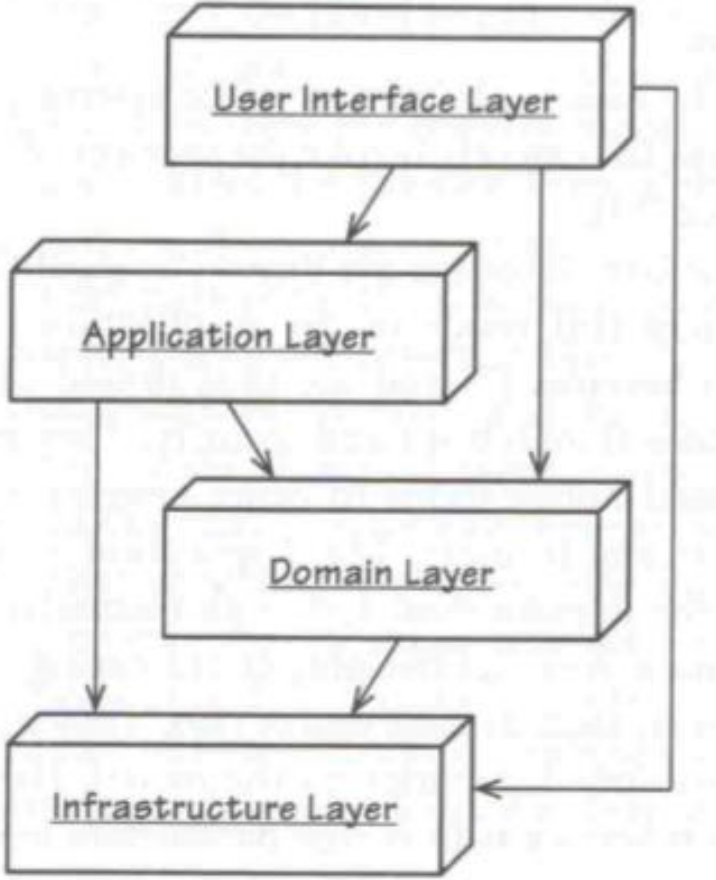
Mixed (batch endpoint, multiple uc-requests)

```
$ curl -i -X POST -d "[{\"version\": \"2\", \"type\":\"ping\",\"action\":\"read\",\"content\":\"\"},{\"version\"
: \"2\", \"type\":\"metrics\",\"action\":\"command\",\"content\":\"\"},{\"version\": \"2\", \"type\":\"test\",\"
action\":\"command\",\"content\":{\"message\":\"foo1\",\"delay\":100}}]" http://localhost:48082/api/batch
HTTP/1.1 207 Multi-Status
Content-Type: application/json
Date: Tue, 21 Jan 2020 13:02:46 GMT
Content-Length: 368

[{"Version":"2","type":"ping","Action":"read","Content":{"response":"pong"}},{"Version":"2","type":"metrics","Act
ion":"command","Content":{"memAlloc":646256,"memTotalAlloc":5029624,"memSys":11934200,"memMallocs":95492,"memFree
s":91691,"memLiveObjects":3801,"cpuBusyAvg":45.85220427638601}},{"Version":"2","type":"test","Action":"command","
Content":{"message":"foo1"}}]
```

Layered Architecture

Implementation is a relaxed layers architecture.



UI Layer.

Transport-specific.

Controller implementation.

Application Layer.

Use-case implementation.

DTO definitions.

Repository contracts.

Domain Layer.

Models.

Validation implementation.

Behavior (“business logic”).

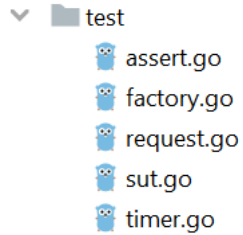
Infrastructure Layer.

Repository implementation.

Ancillary supporting structural implementation.

Project Structure

/internal/pkg/test

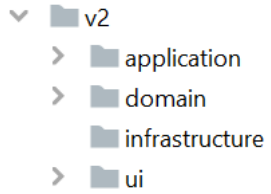


Supports Golang-based acceptance testing.

Implements the generic code to start a service within a separate goroutine in the test runner context.

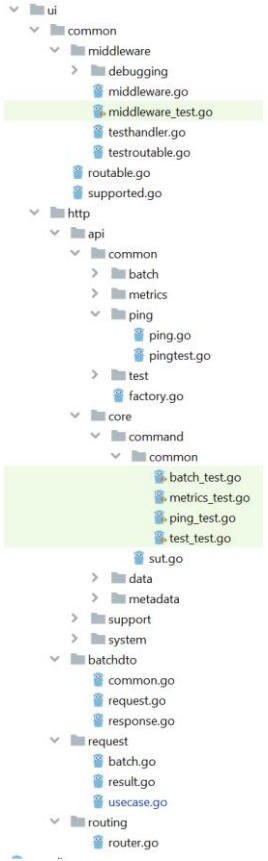
Common test-related assertions, factories, and timer implementation.

/internal/pkg/v2



Individual architectural layer implementation for APIv2 behavior.

/internal/pkg/v2/ui

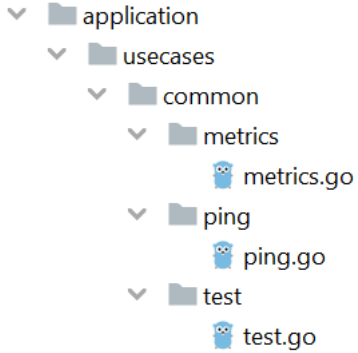


UI layer.

/common – transport-agnostic.

/http – HTTP-transport-specific.

/internal/pkg/v2/application

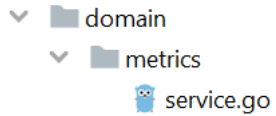


Application layer.

/common – cross-service use-cases.

Service-specific use cases will be in dedicated directories/packages (e.g. /core/data/).

/internal/pkg/v2/domain



Domain layer.

Domain service for metrics.

Content will grow with follow-on work.

/internal/pkg/v2/infrastructure

Infrastructure layer.

Content will be added in follow-on work.

Use-Case Endpoints

Use-case endpoint: URL that takes one or more use-case-specific Request DTOs and returns corresponding Response DTOs.

Endpoint is versioned.

(e.g. /api/v2/metrics)

Endpoint accepts either a single use-case request or an array of use-case requests.

Single use-case request returns HTTP 200 status code.

Array of use-case requests returns HTTP 207 status code.

Multiple use-case requests received in a single transport request are processed concurrently.

Batch Endpoint

Batch endpoint: “/api/batch” URL that takes one or more Request DTOs and returns corresponding Response DTOs. (Can be mixed types.)

Endpoint is not versioned.

(e.g. /api/batch)

Endpoint accepts an array of use-case requests and returns an array of corresponding responses.

Array of use-case requests returns HTTP 207 status code.

RequestEnvelope and ResponseEnvelope are the batch endpoint’s DTOs.

Envelopes contain version, type, and action (from definitions).

Envelopes also contain a content field – which holds a use-case-specific DTO.

Use-case requests sent in a single transport request can be of mixed version, type, and action.

Multiple use-case requests received in a single transport request are processed serially.

Batch endpoint is “free.”

Add a use-case endpoint; its behavior is available via the batch endpoint.

Use-Cases

Live in the application layer.

Single use-case can handle one or more version, type, and action (from definitions) variations.

Define their own private Request and Response DTOs.

Are implemented following GoF Command design pattern.

Implement the Routable contract.

Router

Lives in the UI layer.

Maps version, type, and action (from definitions) to a Routable contract implementation.

Controllers

Live in the UI layer.

Bridges the UI (i.e. endpoint invocation) to a use-case implementation.

Implement the Controller contract.

Provide two generic handler implementations.

Handler implementations provide generic marshalling and unmarshalling.

Handler implementations delegate to a use-case implementation.

One generic handler is called for all use-case endpoint invocations.

The other generic handler is called for all batch endpoint invocations.

Middleware

Lives in the UI layer.

Contains behavior executed for each use-case request.

Supports pre- and post-use-case processing.

Extensible.

Can be layered.

Optional.

Can be selectively enabled.

Can modify request and/or response content.

Can be added to all use-cases.

Can be selectively added to an individual use-case.

PR includes an example debugging middleware implementation that logs use-case requests.

Enabled by command line switch (–debug).

With switch off:

```
GET http://localhost:48082/api/v2/ping

HTTP/1.1 200 OK
Content-Type: application/json
Date: Tue, 21 Jan 2020 00:04:47 GMT
Content-Length: 19

{
  "response": "pong"
}

Response code: 200 (OK); Time: 18ms; Content length: 19 bytes
```

```
<4 go setup calls>
level=INFO ts=2020-01-21T00:04:45.0844505Z app=edgex-core-command source=database.go:152 msg="Database connected"
level=INFO ts=2020-01-21T00:04:45.0864518Z app=edgex-core-command source=telemetry.go:86 msg="Telemetry starting"
level=INFO ts=2020-01-21T00:04:45.0864518Z app=edgex-core-command source=httpserver.go:89 msg="Web server starting (localhost:48082)"
level=INFO ts=2020-01-21T00:04:45.0874492Z app=edgex-core-command source=message.go:50 msg="Service dependencies resolved..."
level=INFO ts=2020-01-21T00:04:45.0884512Z app=edgex-core-command source=message.go:51 msg="Starting edgex-core-command 1.1.0 "
level=INFO ts=2020-01-21T00:04:45.0894511Z app=edgex-core-command source=message.go:55 msg="This is the Core Command Microservice"
level=INFO ts=2020-01-21T00:04:45.0904515Z app=edgex-core-command source=message.go:58 msg="Service started in: 23.5485ms"
```

With switch on:

```
GET http://localhost:48082/api/v2/ping

HTTP/1.1 200 OK
Content-Type: application/json
Date: Tue, 21 Jan 2020 00:04:47 GMT
Content-Length: 19

{
  "response": "pong"
}

Response code: 200 (OK); Time: 18ms; Content length: 19 bytes
```

```
<4 go setup calls>
level=INFO ts=2020-01-21T00:07:36.1542553Z app=edgex-core-command source=database.go:152 msg="Database connected"
level=INFO ts=2020-01-21T00:07:36.1562647Z app=edgex-core-command source=telemetry.go:86 msg="Telemetry starting"
level=INFO ts=2020-01-21T00:07:36.1572891Z app=edgex-core-command source=httpserver.go:89 msg="Web server starting (localhost:48082)"
level=INFO ts=2020-01-21T00:07:36.1582939Z app=edgex-core-command source=message.go:50 msg="Service dependencies resolved..."
level=INFO ts=2020-01-21T00:07:36.1592578Z app=edgex-core-command source=message.go:51 msg="Starting edgex-core-command 1.1.0 "
level=INFO ts=2020-01-21T00:07:36.1612869Z app=edgex-core-command source=message.go:55 msg="This is the Core Command Microservice"
level=INFO ts=2020-01-21T00:07:36.1622643Z app=edgex-core-command source=message.go:58 msg="Service started in: 25.4581ms"
level=INFO ts=2020-01-21T00:07:38.1675026Z app=edgex-core-command source=debugging.go:60 msg="elapsed: 0ms, version: 2, kind: ping,
 action: read, request: null, response: {\"response\":\"pong\"}"
```

Other middleware possibilities – use-case usage metrics, use-case performance, etc.

Common Behavior

Single implementation of common behavior across services.

For example: configuration, metrics, ping, and version endpoints.

Test endpoint.

Echoes message after optional delay.

(/api/test, available in test context)

Wire Up

Routes for v2 are added to mux.Router in each service’s bootstrap handler (init.go).

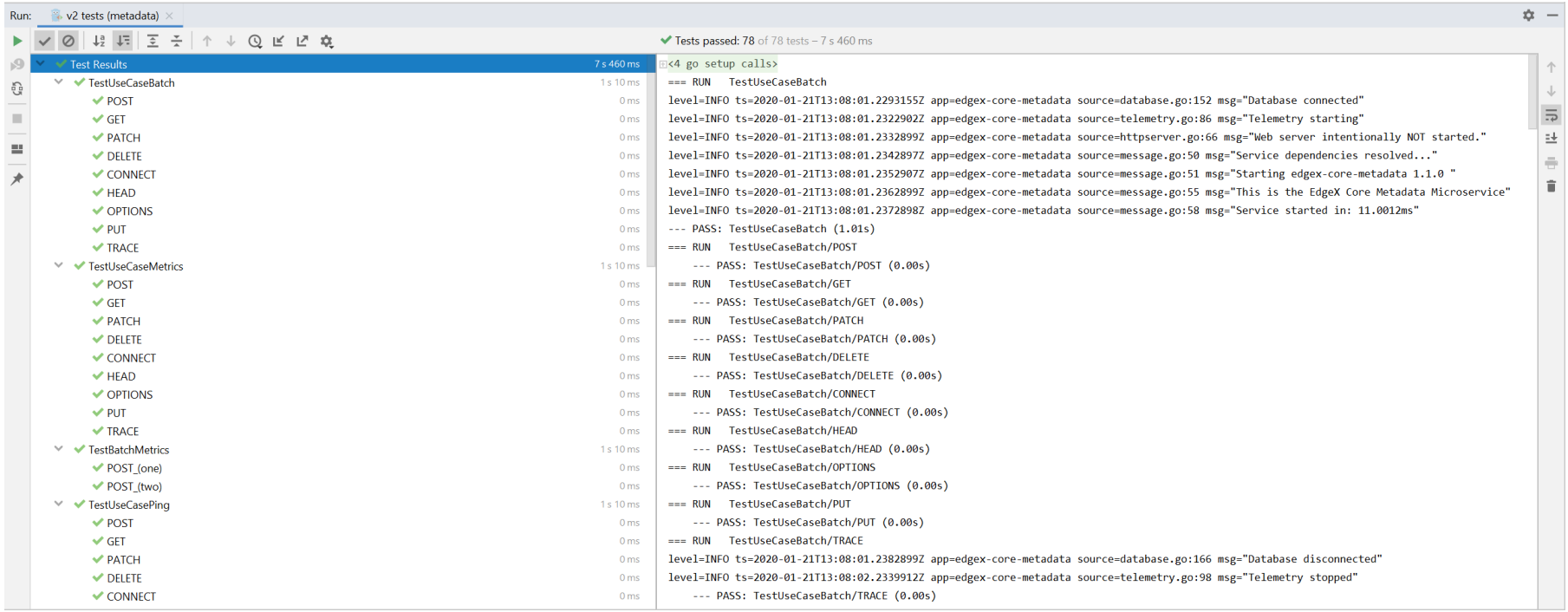
Acceptance Tests

Implementation is based on my December presentation to the QA/Test working group.

Tests spin up an instance of the service being tested inside the test runner context.

Leverage in-process, in-memory implementation of persistence contract.

Written to exercise all HTTP methods on endpoint.



Common behavior has a related common test.

(ping, metrics, etc.)

For common behavior, each service has its own test implementation that delegates to the related common test.

A test exists to verify concurrent execution of multiple use-case requests made on a use-case endpoint.

A test exists to verify serial execution of multiple use-case requests made on the batch endpoint.

Example tests are written to ensure backward- and forward- compatibility across minor API versions.

This is done by retaining each minor release’s DTOs and using them to execute requests against the latest implementation.

Example tests make heavy use of constants.

(v2/ui/http/api/common/ping/ping.go constant definitions and related usage in pingtest.go.)

Example tests have no hardcoded JSON.

(v2/ui/http/api/common/ping/pingtest.go usage of test.AssertJSONBodyEqualsForSingle().)

Open Items

Creation of a supporting ADR.

Still iterating on APIv2 specification.

https://github.com/edgexfoundry/edgex-go/pull/2309

Conformance to content and structure of APIv2 specification.

(Missing properties.)

Implement example service-specific functionality.

(Metadata’s addressable API.)

Implement request DTO validation.

(Handle edge case where we receive an empty request; unmarshal a valid object with invalid content.)

go-mod-core-contracts integration.

Implement strategy pattern-based abstraction for the RequestEnvelope strategy property.

(“sync”, “async-push”, “async-poll”)

Ongoing discussion in Certification WG on format of version in API URLs and batch DTOs.

Parting Comments

Review the code.

Run it.

Provide feedback.

https://github.com/edgexfoundry/edgex-go/pull/2285


DELL Technologies