Posted in: API (/spaces/127/sandpiper/forums/5044/api)

Basic topology (client/server roles)

Unsubscribe from this discussion
Subscribe to RSS (../../../../../../spaces-cf/forums/rss-space-posts.ashx?spaceID=127&topicID=5067&forumID=5044&key=rrer%2B1xDo%2BlxpC7jBRbM5w%3D%3D)

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/17/2019

The fundamental question we need to resolve: Does the API server only represent the primary (origin) pool or can it represent the secondary (branch) also? My opinion is that the API methods we define should facilitate both - even if they are not all implemented.

Like

Reply (/forums/post?tid=5067&ReplyPostID=5068&SpaceID=127)

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/18/2019

One core concept in REST is that a request for output from an authoritative source (server) is conveyed through an http GET. This is so that the the result (the output) is unambiguously tied to a specific URL that contains all the query parameters. The point of this simple, unambiguous input/output correlation is that intermediate systems (proxy servers and indexer bots) can cache the answers that came out of the source recently. There is no "body" data conveyed in a GET - the entire request is contained in the URL. Most http servers limit the GET request size to 8K, while the limit of the POST request (that can't be cached) is many megebytes and intended to be large.

The reason it maters here is that in synchronization-type transactions (like git and rsync), the requester must tell the answerer what is already present in the secondary repository. For instance: *"hello, origin, this is Luke and I have id's x,y,x in my local pool. Give me the missing bits and tell me* *what to drop.*"

The x,y,z in my request is going to be way bigger than any server is going to tolerate in a GET. It will have to be POST. I realize this is not RESTful, but I think our situation calls for it.

Like

Reply (/forums/post?tid=5067&ReplyPostID=5069&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/18/2019

I see the sync request as a simple GET "/slice/{slice_id}". Which, if I'm authenticated, would return all object ids that have a status date after the last time we synced up.

I think it might help if I document some basic truths as I currently understand them. Let me know if you agree and/or if I left anything out:

1. All "product data" is delivered at some owner-defined level of granularity which becomes a syncable "product_data_object".
2. Each product_data_object is identified by a unique "oid" and has "owner", "status", "status_date", "payload_type" and "payload" attributes.
3. The payload of a product_data_object is determined by "payload_type" (e.g. "aces", "partspro", etc.) and must be a complete logical unit allowing net changes at that level using keys defined by the associated standard. For example, Delete with its associated Add. (Question: could it be an entire PIES file?)
4. The payload of a product_data_object cannot be changed, even by its owner (creator).
5. product_data_objects are created by the owner and stored in a "primary data pool" with a "status" of "Added" and a "status_date" of the current date-time.
6. The "status" of a product_data_object may be changed from "Add" to "Delete" by its owner (setting its "status_date" to the current date-time). (Question: how long must Deleted objects stay in the primary data pool?)
7. A "slice" is a set of product_data_objects. (This is still fuzzy in my mind because there are sync ramifications to changing the set and it feels too low of a level.).
8. A "slice" from a primary data pool can be synced with one or more secondary data pools.

9. A Sandpiper "server" provides an API for delivering all product_data_objects in a slice with a status_date after a supplied date (a "sync").
10. A Sandpiper "client" can ask for a sync of a "slice" (provided the server is on-line and credentials are authenticated). When the slice has been applied to the secondary data pool, the client tells the server that the sync was "completed".
11. A Sandpiper "client" can ask for a complete refresh of a slice (which will get all "Adds" in the slice).
12. Secondary data pools can sync with many primary data pools (because each product_data_object has an "owner").
13. A company can be a Sandpiper "client", "server" or both.
14. It is possible (but probably not recommended) that primary and secondary data pools share a single physical database.
15. A Sandpiper "server" can initiate a sync with a "client", even if the client is not currently listening. (I added that last part assuming we'd use a messaging queue to implement that requirement).

(Modified #10 to better show that a sync process is asynchronous, and Added #11 after initial posting)

sync-logic (/spaces/127/sandpiper/searchresults?keyword=%22sync-logic%22&searchtags=1)

Like

Reply (/forums/post?tid=5067&ReplyPostID=5070&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/19/2019

Doug, I'm going to speak as though what's in my head are the answers:

First of all, there would be a RESTish method like you describe, but it would only expose one specific object per request - not an entire slice.

Maybe something like: GET /oids/{id}

This method is one potential way that the branch would request the data for an object it needs - after the need has been determined. A more bulk-oriented method of syncing would be more desirable and part of the standard, but there may be scenarios where that cherry-picking of object data is appropriate.

A slice is simply an object selection criteria mutually-defined by the parties involved. "AB1" is mutually defined between me and WHI - to me it means a collection of internal proprietary PIM system application and part category id's. To WHI it means a specific ACABrandID, some specific ACApartTypeID's, and a bunch a black-box minutia that only they can explain. I know how to select for an AB1 dataset and they know what to expect and where to put AB1 when it arrives.

On your points above, we are in sync until #6. There is no status. There is no concept of "last synced date" If we approached it that way, we would have to maintain careful accounting of every secondary pool's last interaction on every object.

When an object is changed in the prim-db in any substantive way, it gets a new oid. The old oid could be saved for audit purposes but dosn't have to be. The current "answer" for what constitutes a slice (no matter what a secondary pool may or may not contain) is the list of oid's that the owner gets when it queries its local database of objects. That list of oid’s is the answer to “what’s the current state of AB1?”. It does not matter when I last synced with WHI or what they think the state is. They either have those oid’s or they don’t. There are no object-specific changes to be made - just adds and drops. If they already have and object, there’s no need to transmit it. When we sync-up, WHI must add the objects that they lack and drop the objects that no longer exist. This avoids two thorny issues.

1) Defining a “key” to update an existing object in the wild. (which has proven impossible over the last 20 years).

2) In a one-supplier-many-receivers scenario, you would have to keep track of last sync date for every relationship in a date-driven paradigm.

On Point #15: A full-featured server implements origin-out and branch-in methods so that it can represent either a branch or an origin - depending on the business scenario. Epicor is both a supplier and receiver of data depending on how you look at them. *sandpiper.epicor.com* would accept data from other authorities in some relationships (manufacturers like Brake Parts,inc) and be the authority on other relationships (third-party shop management systems that buy data from Epicor).

Like

Reply (/forums/post?tid=5067&ReplyPostID=5073&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/19/2019

It looks like we have two completely different approaches to the sync process. My approach is **time-based**, while yours (I believe) is **set-based**.

As you mentioned, time-based syncs work only if you know the "last_sync_time". I was suggesting the sync would be by slice, so you would keep a last_sync_time for each slice. It wouldn't need to be at that level, though. It could just as well be at a subscriber level, for example.

Set-based syncs work only if you can communicate the entire set to determine what is missing. I think it might be helpful to estimate how big these sets could be. If, for example, a single PIES file is broken into syncable objects, that could be hundreds of thousands of objects in a set.

Will it be efficient for a receiver to perform set-based syncs with hundreds of suppliers once a day? How about once an hour? I think we should put some solid numbers to this so we can determine if it is possible.

Even if a time-based sync was at the subscriber level, you could probably perform a sync every minute and still be performant. It would only return the objects that have changed in that minute. This would probably not be possible with a set-based sync, however.

A third approach (which does not use a traditional api with REST endpoints) might be to use message queues to communicate changes. They scale well and can be fault-tolerant. This method would be more transaction oriented... almost like a database replica being updated from a transaction log.

The receiver would read their input data queue and apply object adds/deletes. They could write to a command queue to ask for a complete refresh, etc. This even-driven pattern would allow near-real-time updates. There is no need to request a sync or exchange id sets. You simply initiate a subscription (which sends everything currently in the database) and then send change commands as they happen in the primary database.

I don't have much experience with this "transaction" approach, but it seems possible.

sync-method (/spaces/127/sandpiper/searchresults?keyword=%22sync-method%22&searchtags=1)

Like

Reply (/forums/post?tid=5067&ReplyPostID=5074&SpaceID=127)   Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)

8/19/2019

A full-coverage line (one brand) of brake rotors is about 100K ACES apps and about 4K PIES items. If the oid is 14 bytes and you have 7 bytes of overhead in a conveyance like:

<o>FLMK2sPWbn7HxR</o>

The conveyance of 100K oid's in this format is about 2 megabytes. That's a trivial amount of data in this day-and-age. Considering that 11 of those characters don't change (zero entropy), the set would zip by about half on average (to 1MB) if compression was used as part of the standard. If we used JSON encoding, it would be a little closer to ideal with no compression.

Hashing is another tool we could use. The client and server could speak in terms of the md5 or SHA hash of their respective sets of oid's to avoid having to convey them unless something has changed.

I feel pretty strongly that last-sync-time approaches are going to leave us open to systems that think everything is sync'd when it's not. This would lead to mistrust of the standard and it would all devolved back to the bad-old days for "full" submits.

Like

Reply (/forums/post?tid=5067&ReplyPostID=5077&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/19/2019

Your calculations assume just one oid per PIES "Item". I wonder if that will be an acceptable level of granularity. At the very least, I'd expect Item-Segment level (and when it comes to Interchange, maybe smaller than that). I've seen multi-gigabyte PIES files largely because each Item interchanged with hundreds of competitor part numbers.

But even if we double your estimate, it's still not very much raw data. It would certainly pale in comparison to the actual object payloads. I'm somewhat more concerned, however, about the processing required to find the diffs.

You'd probably handle the diff logic all in memory (database loading / indexing would be prohibitive). Whatever time that takes, though, is multiplied by the number of subscribes of that data set. But maybe that multiple is only on the order of 100 or so. I do like the md5 hash idea to keep from doing a comparison for nothing.

I'm not completely against the set-based method, but I'm not sure how a last-sync-time approach would ever get out of sync. The time is not updated unless the sync is completed successfully. A database corruption or hardware failure would just ask for a full load on a new database.

Even if we go with a set-based sync, I'd probably want a last-sync-time field on the subscriber slice anyway, if only for reporting purposes.

Like

Reply (/forums/post?tid=5067&ReplyPostID=5078&SpaceID=127)   Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/19/2019

One problem with a time-based sync is that it depends on an **accurate clock** on both ends. While this is fairly important for any server setup, it is a dependency that we should probably not overlook.

Maybe a "version number" instead of a date would handle this problem. The originator would increment the version_number each time a slice was updated, so any new/deleted objects would be assigned the new version_number.

A version approach still has the problem of deciding when to purge primary deleted objects, and it does not **absolutely guarantee** that both data pools are the same (because there is nothing that keeps an object from mistakenly being deleted from the secondary data pool by some errant process).

I'm coming around to your set-based approach (with a crc hash of some sort to avoid unnecessary checks).

sync-method (/spaces/127/sandpiper/searchresults?keyword=%22sync-method%22&searchtags=1)

Like

Reply (/forums/post?tid=5067&ReplyPostID=5079&SpaceID=127)   Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/20/2019

> "...the requester must tell the answerer what is already present in the secondary repository. For instance: *"hello, origin, this is Luke and I have id's x,y,x in my local pool. Give me the missing bits and tell me what to drop.*"
>
> The x,y,z in my request is going to be way bigger than any server is going to tolerate in a GET. It will have to be POST. I realize this is not RESTful, but I think our situation calls for it."

Doesn't this type of sync work just as well if the "client" (i.e. "requester") asks the "server" (i.e. "answerer") what they have in their repository? Does it matter which side decides the adds and drops? In that case, we could do a traditional GET request.

Like

Reply (/forums/post?tid=5067&ReplyPostID=5080&SpaceID=127)   Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/20/2019

You are correct about the simple GET. The branch would simply say (via an http GET): "*hello, origin. this is branch . Tell me the list of oid's in AB1.*" Origin's instant response to the GET would be the list of 100K oid's. The origin would close the socket and the branch would go away with it's list of truth. Over the next few seconds, the branch would decide what to drop and what it needs to acquire.

Then the branch would initiate a new and un-related connection to the origin (seconds or minutes after the first) and ask for a block of content (by list of oid's) that is found lacking after comparing lists of oid's from the previous connection.

This approach requires two separate connections in succession. It's not a huge issue, but it complicates the client side by need to keep track of the communication state and adds some overhead to the sync. The fundamental issue is that a web-style interaction (get/post/put/delete etc) is a simple request/response transaction. *There is no rebuttal after the response*. The socket is closed and the two parties move on. The whole exchange happens instantly. So if our protocol is going to leverage http, it must fit into request/response pairs. Follow-on actions must happen in new, follow-on request/response pairs. By contrast, something like an ftp or telnet session is a perpetual open channel until one side terminates. Commands and responses are pushed across the pipe in a persistent context. In such a persistent channel, the getting of the list and the asking for new objects could all happen in the same connection.

There is a caveat that I don't think makes sense in an API context. "Sessions" or "cookies". It's the mechanism by which we "log in" to websites. A webserver can "sessionize" a variable - which means telling the client to store a token to present on subsequent http requests in the next few minutes or hours. Even though it feels like you are logged into a persistent connection to a site like communifire, it's an illusion. Every interaction is a new request and response and the session token is the way that the server keeps track of the client's state.

Like

Reply (/forums/post?tid=5067&ReplyPostID=5081&SpaceID=127)   Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/20/2019

I agree, the api should be stateless. The only additional context the sync api needs, I believe, is authentication (but we can get into that later). I see the following pseudo code for a sync:

```
authenticate   // this is a topic for another discussion tread
get my.slices
for each slice in my.slices
  get slice.hash    // sha1 or md5 hash representing slice oids
  if slice.hash <> local.slice.hash    // saved from last sync or must we recalc?
    get slice.oid_list
    compare slice.oid_list with local.slice.oid_list
    for each new_oid in slice.oid_list
      get object.new_oid     // this object contains the payload
      store object in local.slice
    remove all obsolete local.slice.objects
  put slice.sync_completed   // for activity reporting purposes
```

The "get" and "put" commands are REST api calls to the server. The remaining logic is all local processing and database updates (on the client).

Like

Reply (/forums/post?tid=5067&ReplyPostID=5084&SpaceID=127)   Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/20/2019

YES! That is how I see this too. The one small thing: I would have a bulk get of objects in your inner-most loop - which would have to be a POST request because the list of needed oid's would be too big for a GET. You would have potentially 100K GET requests in succession on the initial sync. That would smell like a DoS attack :)

Like

Reply (/forums/post?tid=5067&ReplyPostID=5085&SpaceID=127)   Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/20/2019

(I included this response to document how I ended up agreeing with you <grin/>)

I understand the (latency) problem with many small requests, but seeing a `POST` for what is really a `GET` just doesn't feel right (although I know it's done).

We could allow an array of ids in a GET (/objects?oid=1,2,3), but that doesn't solve the size problem (allowing only ~80 oids in a 2K url).

One approach Google uses for their Translate api may make the `POST` approach more palatable (and self-documenting):

> "... *semantically* transform a `POST` request into a `GET` request."

https://stackoverflow.com/questions/19771031/rest-request-cannot-be-encoded-for-get-url-too-long (https://stackoverflow.com/questions/19771031/rest-request-cannot-be-encoded-for-get-url-too-long)

These discussions have some interesting thoughts as well:

https://stackoverflow.com/questions/1266596/what-is-the-best-way-to-design-a-http-request-when-somewhat-complex-parameters-a (https://stackoverflow.com/questions/1266596/what-is-the-best-way-to-design-a-http-request-when-somewhat-complex-parameters-a)

https://stackoverflow.com/questions/19637459/rest-api-using-post-instead-of-get (https://stackoverflow.com/questions/19637459/rest-api-using-post-instead-of-get)

So I'm good with the `POST` approach, but let's consider using the `X-HTTP-Method-Override: GET` header (unless it complicates the implementation).

Like

Reply (/forums/post?tid=5067&ReplyPostID=5089&SpaceID=127)   Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/20/2019

Now I just need to convince you that the server can represent an origin or a branch :)

I like the way the stackoverflow commenter referred to the "spirit of RESTfulness." REST is an admirable guiding principle. I think once you take cache-friendliness out of the equation (because we want **NOTHING** to do with caching this content), the case for strict adherence to REST gets weak.

Elements of sandpiper that are single-serving in nature could in-fact adhere to the REST ideals. For instance, we could define a method that GETs the ACESapp objects that that are within a slice and have a specific beasvehilceid - not just the oid's but the actual object content. You could even expose certain elements of the VCdb through sandpiper GET - like make names, model names and besevehilce records. My focus till now has been on syncing huge clunky "full" sets of data. I think there is a segment of the ecosystem that would trust and value Content-as-a-service through our standardized API. Imagine a light-weight e-commerce site that wants no parts of updating VCdb/PCDb/Qdb references each month and loading ACES and PIES files from several manufacturers. They could "plug into" the Autocare association's sandpiper endpoint and do real-time Make/Model/Year queries and "plug into" the manufacturer's endpoint to feed content into the lookup once the basevehicle id was in-hand. This nirvana scenario would be our real-time promised land - and it would not involve any oid's or hashes or add/drop lists.

I bet in 15 years, the discussion of origin vs branch will seem quaint. It will all be live access to single sources of truth. The elements of sandpiper that are used for effectively rsyncing supplier to receiver will become relics of the old days.

Like

Reply (/forums/post?tid=5067&ReplyPostID=5093&SpaceID=127)   Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/21/2019

> "Now I just need to convince you that the server can represent an origin or a branch :)"

Why is this a requirement? It seems to violate the "separation of concerns" principle. If the secondary-pool is really a "snapshot" copy and immutable, it makes sense to (logically) separate that from the primary-pool which is canonical and mutable.

A company can certainly have both roles and host both a Sandpiper "client" and "server". It might even be preferable for them to run these systems on separate machines (physical or virtual). Sandpiper will be built on technology with no cost penalty for multiple installations.

With this approach, we have a clear boundary between serving out data and receiving data. Of course, there is nothing to keep us from having a Sandpiper dashboard for configuring or managing both. If we are careful on how we design the database schema, we could even share a single physical database for each of these roles.

So maybe there's really no disagreement here. It might be more of an implementation discussion than a philosophical one.

Like

Reply (/forums/post?tid=5067&ReplyPostID=5100&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/21/2019

The simple answer is that you can't connect to a client. If you could, it would be called a server. It does not have a public presence waiting to answer connection requests.

If you draw a Vin diagram of Content authors in one circle and people who are good at maintaining API services in the other, there's a pretty slim overlap.

Draw that same diagram with *content receivers* in one circle and people who are good at maintaining API services in the other, you get way more overlap.

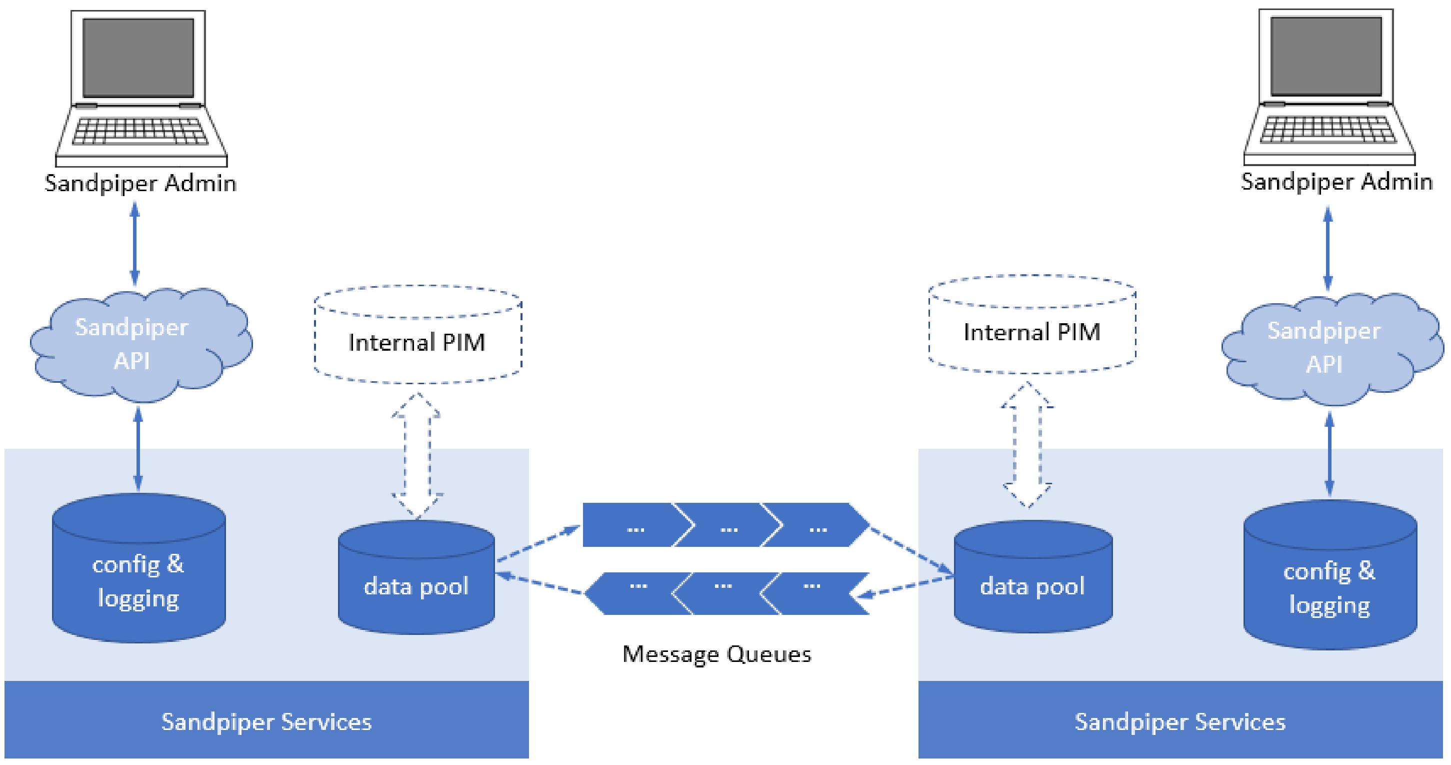
By dictating that the API server can only represent the author, we are not playing to the strengths of the people in our industry.

Like

Reply (/forums/post?tid=5067&ReplyPostID=5101&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/21/2019

I'd like to present a slightly different architecture that supports a two-way relationship in a natural way. It uses a traditional RESTful API for "command and control" (i.e. Sandpiper Admin) and **message queues** for data delivery. Here's a diagram of what I have in mind:



The "`config & logging`" database contains all Subscriber information including what slices they subscribe to, activity logs, etc.

The "`data pool`" contains both primary and secondary objects (because you seem to want to mix them).

"`Message Queues`" are created by the Sandpiper API and are specific to a trading partner relationship. These queues are used to communicate all sync requests and object responses (as previously discussed). The object responses are still one-way as indicated on the diagram, with a publisher (primary) and subscriber (secondary).

This design has several advantages, two of which are (1) ability to add near real-time updates in the future using an event driven approach, and (2) fault-tolerance (when one machine goes off-line).

There are several well-known open-source messaging libraries available which work with all popular programming languages. There are also hosted message brokers (although I don't see us needing to use them).

For a quick overview of message queues (including features, benefits and other resources) see this link:

https://aws.amazon.com/message-queue/ (https://aws.amazon.com/message-queue/)

Thoughts?

message-queues (/spaces/127/sandpiper/searchresults?keyword=%22message-queues%22&searchtags=1)

Like

Reply (/forums/post?tid=5067&ReplyPostID=5102&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/21/2019

Another benefit of the message-queue approach is the ability to support **unsolicited feedback** on data submissions (i.e. data assessment reports).

So when a receiver gets some data, they can do a validation and report back on a separate "data-feedback" channel. It will sit there until the author is ready to look at it. A queue is perfect for this. Hopefully, these assessments will be tied to the object id.

I feel that we should at least design this feedback loop, maybe as a Level 3 implementation.

feedback-loop (/spaces/127/sandpiper/searchresults?keyword=%22feedback-loop%22&searchtags=1)

Like

Reply (/forums/post?tid=5067&ReplyPostID=5103&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/21/2019

The fault-tolerance is an interesting feature of the message queue approach. Do you have any experience with them?

For the record, I don't necessarily want to mix the pools. What I want is to avoid creating two versions of the API (the origin version and the branch version),

Like

Reply (/forums/post?tid=5067&ReplyPostID=5104&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/21/2019

I've used named pipes, ipc, websockets and other messaging systems over the years, but never the latest batch of MQ technology. Looks much easier than it used to be. The only down-side I see is that it adds some infrastructure complexity (and dependencies). You need to have a message broker running somewhere.

My vision is that everything (in the reference implementation) would be bundled up in a Docker container, which could be deployed anywhere. That would make implementation much easier.

> "What I want is to avoid creating two versions of the API (the origin version and the branch version)"

I only wanted one version of the API too... the "Server" version. <grin/>. (As far as I can see, the only thing keeping us from that was the "requirement" to let the Server start a sync.)

Like

Reply (/forums/post?tid=5067&ReplyPostID=5105&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/21/2019

Ok, I'll let it go. The server will only be able to represent the author. But you will owe me $10 when NAPA, O'Reilly's, AutoZone, Advance, WHI, Epicor, OptiCat and Vertical Dev say: "your little sandpaper thing is clever and interesting, but it does not allow us to host the API how and where we want to."

Like

Reply (/forums/post?tid=5067&ReplyPostID=5106&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/22/2019

"Sand-Piper" :)

Like

Reply (/forums/post?tid=5067&ReplyPostID=5107&SpaceID=127)   Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/22/2019

Too funny! I hadn't thought about the "pipe" in Sandpiper...

I'm still not convinced message queues are the way to go. It does add infrastructure and support complexity. Conceptually, I like it a lot, but if we're trying to make deployment super-simple, it might be a barrier.

I think things may become more clear once we define the api endpoints (for both sides). I'm going to start working on that (just in a simple Excel file).

Can you post something (in a new discussion) about the "slice"? I'm still fuzzy on that piece.

Like

Reply (/forums/post?tid=5067&ReplyPostID=5108&SpaceID=127)   Answer

Adrian Parker (https://autocare.communifire.com/people/aparker)
8/14/2020

The server/publisher sends the time the push started. The receiver stores that value. When the receiver requests objects that changed it just uses the datetime it received from the publisher previously. The system time on the receiver is irrelevant.

Like

Reply (/forums/post?tid=5067&ReplyPostID=6116&SpaceID=127)   Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/15/2020

From above:

> A [time-based] approach still has the problem of deciding when to purge primary deleted objects, and it does not absolutely guarantee that both data pools are the same (because there is nothing that keeps an object from mistakenly being deleted from the secondary data pool by some errant process).

Like

Reply (/forums/post?tid=5067&ReplyPostID=6117&SpaceID=127)   Answer