Posted in: Development (/spaces/127/sandpiper/forums/5366/development)

PostgreSQL Support for Large Objects

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
3/10/2020

> "PostgreSQL also supports a storage system called TOAST (https://www.postgresql.org/docs/12/storage-toast.html), which automatically stores values larger than a single database page into a secondary storage area per table. This makes the large object facility partially obsolete. One remaining advantage of the large object facility is that it allows values up to 4 TB in size, whereas **TOASTed fields can be at most 1 GB**."

So the question is if 1GB will be sufficient for our compressed files, especially the proposed support for **asset-archive** slices.

It seems to me that anything larger than 1GB should probably be broken up in any case (because that data will need to be synced as a unit (in a grain).

Thoughts?

1gb-max (/spaces/127/sandpiper/searchresults?keyword=%221gb-max%22&searchtags=1)

Like

Reply (/forums/post?tid=5805&ReplyPostID=5806&SpaceID=127)

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

3/10/2020

I wonder how realistic it is for large receivers to want "asset-archive" slices? The upside is the simplicity of data delivery. The downside might be disk space since these large grains would never be removed once processed.

If Epicor, for example, had 500 companies sending 1GB archives, that's a lot of disk space just hanging around. **Dave Hastings (https://autocare.communifire.com/people/dhastings)** , do you have any thoughts on this?

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

Like

Reply (/forums/post?tid=5805&ReplyPostID=5807&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
3/10/2020

I think the only "right way" to handle files of this size is by adding native support for online storage systems such as AWS S3 (https://aws.amazon.com/s3/) or Backblaze B2 (https://www.backblaze.com/b2/cloud-storage.html).

We previously stayed away from URL links, but this is different. This is a controlled service with security and the ability to gather metadata such as upload timestamps (versioning).

You might say, "well then, why not **store all 'payloads' as urls** to one of these storage services?" To which I would answer, "hmmm, that's an interesting thought!"

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

Like

Reply (/forums/post?tid=5805&ReplyPostID=5808&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
3/10/2020

Or... maybe a few extra terrabytes is "no big deal" for the added simplicity. Disk is cheap!

disk-is-cheap (/spaces/127/sandpiper/searchresults?keyword=%22disk-is-cheap%22&searchtags=1)

Like

Reply (/forums/post?tid=5805&ReplyPostID=5809&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
3/11/2020

I've changed my mind on this. I don't believe we should allow an "archive" type (zip file) for digital assets. It flies in the face of our **small object delivery** concept, especially since the archive is made up of (relatively) small objects to begin with!

Instead, I think we should add an option to our **sandpiper cli** tool to read an asset archive and create individual grains for distribution. This is a Level 2 concept, but nothing is keeping us from allowing this exception with the initial release.

The receiver side would then use the sandpiper cli to save these digital asset grains as files to a directory. The only thing we don't have is a way to see what has changed (because there is no concept of change in sandpiper, only existence). **Dave Hastings (https://autocare.communifire.com/people/dhastings)** , do you see any problem with this approach? Could you use the PIES information to determine changes?

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

Like

Reply (/forums/post?tid=5805&ReplyPostID=5811&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
3/18/2020

This discussion brings up another interesting question. Should we add a change date, hash or other key to the grain to help the PIM determine changes?

I was trying to figure out how a PIM would know which L2 grains to keep and which ones to remove when doing an update.

You wouldn't want to replace all of them just because a few updates were made. That would defeat the purpose of fine-grained syncing. Instead, the PIM should only remove the ones that are no longer valid and add new ones to make a complete set.

We could just leave this up to the PIM to determine, but in that case they will need to keep some kind of mapping to the grains they produce. The alternative is to store something with the grain for this purpose.

My thought is that we should leave sandpiper "pure" and let the PIM handle versioning.

Like

Reply (/forums/post?tid=5805&ReplyPostID=5828&SpaceID=127) Answer