JetStream based Object Stores

Metadata	Value
Date	2021-11-03
Author	@scottf
Status	Implemented
Tags	jetstream, client, objectstore, spec

Revision	Date	Author	Info
1	2021-11-03	@scottf	Initial design
2	2023-06-14	@Jarema	Add metadata
3	2024-02-05	@Jarema	Add Compression

Context

This document describes a design of a JetStream backed object store.

Overview

We intend to hit a basic initial feature set as below, with some future facing goals as indicated:

Current feature list:

- Represent an object store.
- Store a large quantity of related bytes in chunks as a single object.
- Retrieve all the bytes from a single object
- Store metadata regarding each object
- Store multiple objects in a single store
- Ability to specify chunk size
- Ability to delete an object
- Ability to understand the state of the object store
- Data Compression of Object Stores for NATS Server 2.10

Possible future features

- Event Notifications (add/delete/lock)
- Locking
- Archiving (tiered storage)
- Searching/Indexing (tagging)
- Versioning / Revisions
- Overriding digest algorithm
- Capturing Content-Type (mime type)
- Per chunk Content-Encoding (i.e. gzip)
- Read an individual chunk.

Basic Design

- Object store or bucket is backed by a stream
- Multiple objects can be placed in each bucket
- Object Info is stored as json in the payload of the message on the Object Info message subject.
- The Object Info subject is always rolled up (per subject)
- Object chunks are stored as the payload of messages on the Chunk message subject

Naming Specification

Protocol Naming Conventions are fully defined in ADR-6

Object Store

The object store name or bucket name (`bucket`) will be used to formulate a stream name and is specified as: `restricted-term` (1 or more of `A-Z, a-z, 0-9, dash, underscore`)

Object Id

Object ids (`object-nuid`) are a nuid.

Object Name

An individual object name is not restricted. It is base64 encoded to form `name-encoded`.

Digest

Currently `SHA-256` is the only supported digest. Please use the uppercase form as in RFC-6234 when specifying the digest as in `SHA-256=IdgP4UYMGt47rgecOqFoLrd24AXukHf5-SVzqQ5Psg8=`.

Modified Time

Modified time is never stored.

- When putting an object or link into the store, the client should populate the ModTime with the current UTC time before returning it to the user.
- When getting an object or getting an object or link's info, the client should populate the ModTime with message time from the server.

Default Settings

Default settings can be overridden on a per object basis.

Setting	Value	Notes
Chunk Size	128k (128 * 1024)	Clients may tune this as appropriate.

ObjectStore / Stream Config

The object store config is the basis for the stream configuration and maps to fields in the stream config like in KV.

```
type ObjectStoreConfig struct {
    Bucket      string      // used in stream name template
    Description string      // stream description
    Metadata    map[string]string   // stream metadata
    TTL         time.Duration   // stream max_age
    MaxBytes    int64       // stream max_bytes
    Storage     StorageType     // stream storate_type
    Replicas    int         // stream replicas
    Placement   Placement       // stream placement
    Compression bool        // stream compression
}
```

- If Compression is requested in the configuration, set its value in the Stream config to `s2`. Object Store does not expose internals of Stream config, therefore the bool value is used.

Stream Configuration and Subject Templates

Component	Template
Stream Name	`OBJ_<bucket>`
Chunk Stream subject	`$O.<bucket>.C.>`
Meta Stream subject	`$O.<bucket>.M.>`
Chunk message subject	`$O.<bucket>.C.<object-nuid>`
Meta message subject	`$O.<bucket>.M.<name-encoded>`

Example Stream Config

```
{
  "name": "OBJ_MY-STORE",
  "description" : "description",
  "metadata": [{"owner": "infra"}],
  "subjects": [
    "$O.MY-STORE.C.>",
    "$O.MY-STORE.M.>"
  ],
  "max_age": 0,
  "max_bytes": -1,
  "storage": "file",
  "num_replicas": 1,
  "rollup_hdrs": true,
  "allow_direct": true,
  "discard": "new",
  "placement": {
    "cluster": "clstr",
    "tags": ["tag1", "tag2"]
  },
```

```
  compression: "s2"
}
```

Structures

ObjectLink is used to embed links to other buckets and objects.

```
type ObjectLink struct {
    // Bucket is the name of the other object store.
    Bucket string `json:"bucket"`

    // Name can be used to link to a single object.
    // If empty means this is a link to the whole store, like a directory.
    Name string `json:"name,omitempty"`
}
```

ObjectMetaOptions

```
type ObjectMetaOptions struct {
    Link      ObjectLink `json:"link,omitempty"`
    ChunkSize uint32      `json:"max_chunk_size,omitempty"`
}
```

ObjectMeta

Object Meta is high level information about an object.

```
type ObjectMeta struct {
    Name        string            `json:"name"`
    Description string            `json:"description,omitempty"`
    Headers     Header            `json:"headers,omitempty"`
    Metadata    map[string]string `json:"metadata,omitempty"`

    // Optional options.
    Opts ObjectMetaOptions `json:"options,omitempty"`
}
```

ObjectInfo

Object Info is meta plus instance information. The fields in ObjectMeta are serialized in line as if they were direct fields of ObjectInfo

```
type ObjectInfo struct {
    ObjectMeta
```

```
    Bucket  string    `json:"bucket"`
    NUID    string    `json:"nuid"`
    // the total object size in bytes
    Size    uint64    `json:"size"`
    ModTime time.Time `json:"mtime"`
    // the total number of chunks
    Chunks  uint32    `json:"chunks"`
    // as in http, <digest-algorithm>=<digest-value>
    Digest  string    `json:"digest,omitempty"`
    Deleted bool      `json:"deleted,omitempty"`
}
```

ObjectInfo Storage

The ObjectInfo is stored as json as the payload of the message under the Meta message subject. The `ModTime` (`mtime`) is never written as part of what is being stored.

When the ObjectInfo message is retrieved from the server, use the message metadata timestamp as the `ModTime`

Example ObjectInfo Json

```
{
    "name": "object-name",
    "description": "object-desc",
    "metadata": [{"owner": "infra"}],
    "headers": {
        "key1": ["foo"],
        "key2": ["bar", "baz"]
    },
    "options": {
        "link": {
            "bucket": "link-to-bucket",
            "name": "link-to-name"
        },
        "max_chunk_size": 8196
    },
    "bucket": "object-bucket",
    "nuid": "CkuyLEX4z2hbyjj1aWCfiH",
    "size": 344000,
    "chunks": 42,
    "digest": "SHA-256=abcdefghijklmnopqrstuvwxyz=",
    "deleted": true
}
```

ObjectStoreStatus

The status of an object

```
type ObjectStoreStatus interface {
    // Bucket is the name of the bucket
    Bucket() string

    // Description is the description supplied when creating the bucket
    Description() string

    // Bucket-level metadata
    Metadata() map[string]string

    // TTL indicates how long objects are kept in the bucket
    TTL() time.Duration

    // Storage indicates the underlying JetStream storage technology used to store
data
    Storage() StorageType

    // Replicas indicates how many storage replicas are kept for the data in the
bucket
    Replicas() int

    // Sealed indicates the stream is sealed and cannot be modified in any way
    Sealed() bool

    // Size is the combined size of all data in the bucket including metadata, in
bytes
    Size() uint64

    // BackingStore provides details about the underlying storage.
    // Currently the only supported value is `JetStream`
    BackingStore() string

    // IsCompressed indicates if the data is compressed on disk
    IsCompressed() bool
}
```

The choice of `IsCompressed()` as a method name is idiomatic for Go, language maintainers can make a similar idiomatic choice.

Functional Interfaces

ObjectStoreManager

Object Store Manager creates, loads and deletes Object Stores

API:

```
// ObjectStore will lookup and bind to an existing object store instance.
ObjectStore(bucket string) -> ObjectStore
```

```
// CreateObjectStore will create an object store.
CreateObjectStore(cfg ObjectStoreConfig) -> ObjectStore

// DeleteObjectStore will delete the underlying stream for the named object.
DeleteObjectStore(bucket string)
```

ObjectStore

Storing large objects efficiently. API are required unless noted as "Optional/Convenience".

Put

- Put will place the contents from the reader into a new object.
- It is an error to Put when ObjectMeta contains a Link. Use AddLink or AddBucketLink

```
Put(ObjectMeta, [input/stream/reader]) -> ObjectInfo
```

Optional/Convenience Examples

```
Put(name string, [input/stream/reader]) -> ObjectInfo
PutBytes(name string, data []byte) -> ObjectInfo
PutString(name string, data string) -> ObjectInfo
PutFile(name string, file [string/file reference]) -> ObjectInfo
PutFile(file [string/file reference]) -> ObjectInfo
```

Notes

On convenience methods accepting file information only, consider that the reference could have operating specific path information that is not transferable. One solution would be to only use the actual file name as the object name and discard any path information.

Get

Get will retrieve the named object from the object store.

- Deleted objects should be treated the same as objects that don't exist.

At least one [language specific] variant of:

```
Get(name string) -> [Language specific handling output/stream/writer]
Get(name string, [output/stream/writer]) -> ObjectInfo
Get(name string) -> ObjectResult, error
```

Optional/Convenience examples:

```
GetBytes(name string) -> []byte
GetString(name string) -> string
GetFile(name string, file string)
```

GetInfo

GetInfo will retrieve the current information for the object.

- Do not return info for deleted objects, except with optional convenience methods.

```
GetInfo(name string) -> ObjectInfo
```

Optional/Convenience examples:

```
GetInfo(name string, includingDeleted bool) -> ObjectInfo
GetInfo(name string, opts ...WatchOpt) -> ObjectInfo
```

UpdateMeta

UpdateMeta will update **some** metadata for the object.

- Only the name, description and headers can be updated.
- Objects, Links and Bucket Links are all allowed to be updated.
- It is an error to update metadata for a deleted object.
- It is an error to rename an object to the name of an existing object

```
UpdateMeta(name string, meta ObjectMeta)
```

Delete

Delete will delete the named object.

- It's acceptable to no-op for an object that is already deleted.
- It's an error to delete an object that does not exist.

At least one variant of:

```
Delete(name string) -> void
Delete(name string) -> ObjectInfo
```

Seal

Seal will seal the object store, no further modifications will be allowed.

At least one variant of:

```
Seal() -> void
Seal() -> ObjectStoreStatus
```

Watch

Watch for changes in the underlying store and receive meta information updates.

```
Watch(opts ...WatchOpt) -> ObjectWatcher
```

List

List will list all the objects in this store.

- When listing objects, filter those objects that have been deleted. If necessary or requested, provide convenience methods or optional arguments to include them if the API user desires them.

```
List() -> List or array of ObjectInfo
```

Status

Status retrieves run-time status about the backing store of the bucket.

```
Status() -> ObjectStoreStatus
```

ObjectStore Links

Links are currently under discussion whether they are necessary. Here is the required API as proposed. Please note that in this version of the api, it is possible that `obj ObjectInfo` or `bucket ObjectStore` could be stale, meaning their state has changed since they were read, i.e. the object was deleted after it's info was read.

AddLink

AddLink will add a link to another object into this object store.

- It is an error to link to a deleted object.
- It is an error to link to a link.
- It is okay to name the link if the name does not exist.
- It is okay to name the link to that of an existing but deleted object.
- It is okay to name the link to that of another link or bucket link (overwrite).
- It is an error to name the link to that of an existing but not deleted regular object.

```
```

```
AddLink(name string, obj ObjectInfo) -> ObjectInfo
```

AddBucketLink

AddBucketLink will add a link to another object store.

- It is okay to name the link if the name does not exist.
- It is okay to name the link to that of an existing but deleted object.
- It is okay to name the link to that of another link or bucket link (overwrite).
- It is an error to name the link to that of an existing but not deleted regular object.

```
AddBucketLink(name string, bucket ObjectStore) -> ObjectInfo
```