BP-18: LedgerType, Flags and StorageHints

Status

Current state: *["Under Discussion"]*

Discussion thread: [link]

JIRA: [link]

Released: 4.7

Motivation

In implementing and reviewing BP-14 Relax durability, there was a lot of confusions and discussions about `LedgerType`, `Flags` and `StorageHints`. They are a bit confused and mixed with BP-14 Relax durability.

This BP is to propose defining clearly and allowing extensions in future, to support features:

- bypass journal
- different storage types (NVMe, SSD, HDD)
- different caching behaviors

Proposed Changes

This BP will introduce `LedgerType`, `Flags` and `StorageHints`.

LedgerType

The ledger type indicates what is the type of this ledger. The ledger type will be supplied on creating a ledger and recorded as part of the metadata. The ledger type will mostly used for segregating metrics.

A few examples for this:

- for distributed metadata store, we want to separate the ledgers used for users and the ledgers used for bookkeeper itself to store metadata.
- in salesforce use case, some ledgers are used for storing logs and some ledgers are used for storing data
- in pulsar, some ledgers are used for storing topics and some are used for storing cursors.

A LedgerType is a string provided by user when creating a ledger. It will be recorded in the metadata.

```
BookKeeper#newCreateLedgerOp()
        .withLedgerType("catlog")
        ...
```

The type names prefixed with "." are reserved for system predefined types. The available system types:

- metadata: reserved for metadata ledgers used by bookkeeper itself to store metadata for distributed metadata store

StorageHints

The `StorageHints` provides the hints for bookies on where is the best location to store data. The storage hints are provided on creating a ledger and recorded as part of the metadata.

A typical use case for this is to support storing different ledgers on different storage mediums.

```
enum StorageHints {
}

BookKeeper#newCreateLedgerOp()
        .withStorageHints(StorageHints... hints)
```

Flags

Flags defines the write/read behaviors attached to a ledger handle. The flags are categorized into WriteFlags and ReadFlags. WriteFlags defines the flags used for adding entries to ledgers, while ReadFlags defines the flags used for reading entries from ledgers.

```
enum WriteFlag {
        NONE                        = 0;
        DEFERRED_FORCE        = 1; // the flag to relax durability
}

public static EnumSet<WriteFlag> getWriteFlags (int flagValue) {
...
}
public static int getWriteFlagsValue(EnumSet<WriteFlag> flags) {
...
}

enum ReadFlag {
        NONE                        = 0;
        DONT_CACHE                = 1; // don't cache the data on reading entries
}

public static EnumSet<ReadFlag> getReadFlags (int flagValue) {
...
}
public static int getReadFlagsValues(EnumSet<ReadFlag> flags) {
...
}

CreateBuilder#withFlags(EnumSet<WriteFlag> flagSet)
CreateBuilder#withFlags(WriteFlag… flags)
OpenBuilder#withFlags(EnumSet<ReadFlag> flagSet)
OpenBuilder#withFlags(ReadFlag… flags)
```

Compatibility, Deprecation, and Migration Plan

No issue about migration and compatibility. All these are only available in the new API.

Rejected Alternatives

N/A