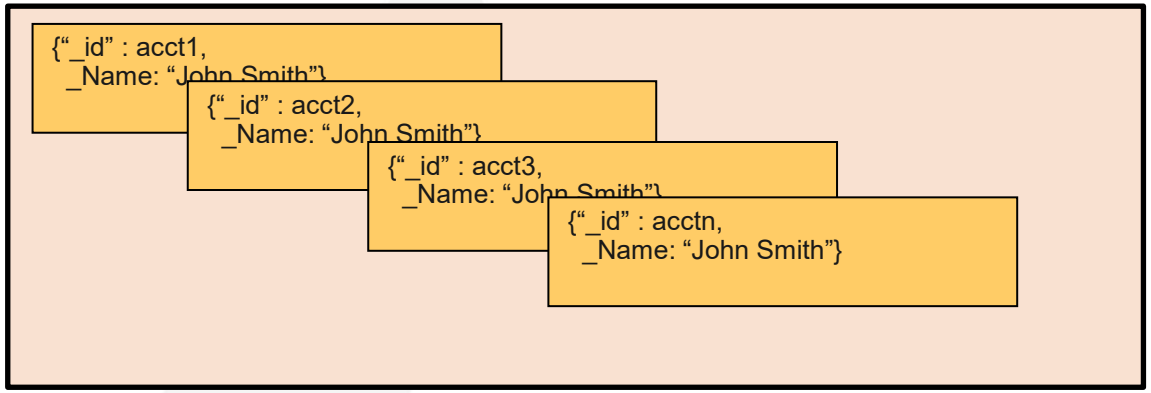
EzNoSQL
Key:Value JSON Store
for z/OS



Terri Menendez
STSM
IBM Corp
terriam@us.ibm.com
Aug 2022

Disclaimer

***Information on the new product is not a commitment, promise, or legal obligation to deliver any material, code or functionality. The development, release, and timing of any features or functionality described for our products remains at IBM's sole discretion.*

Agenda

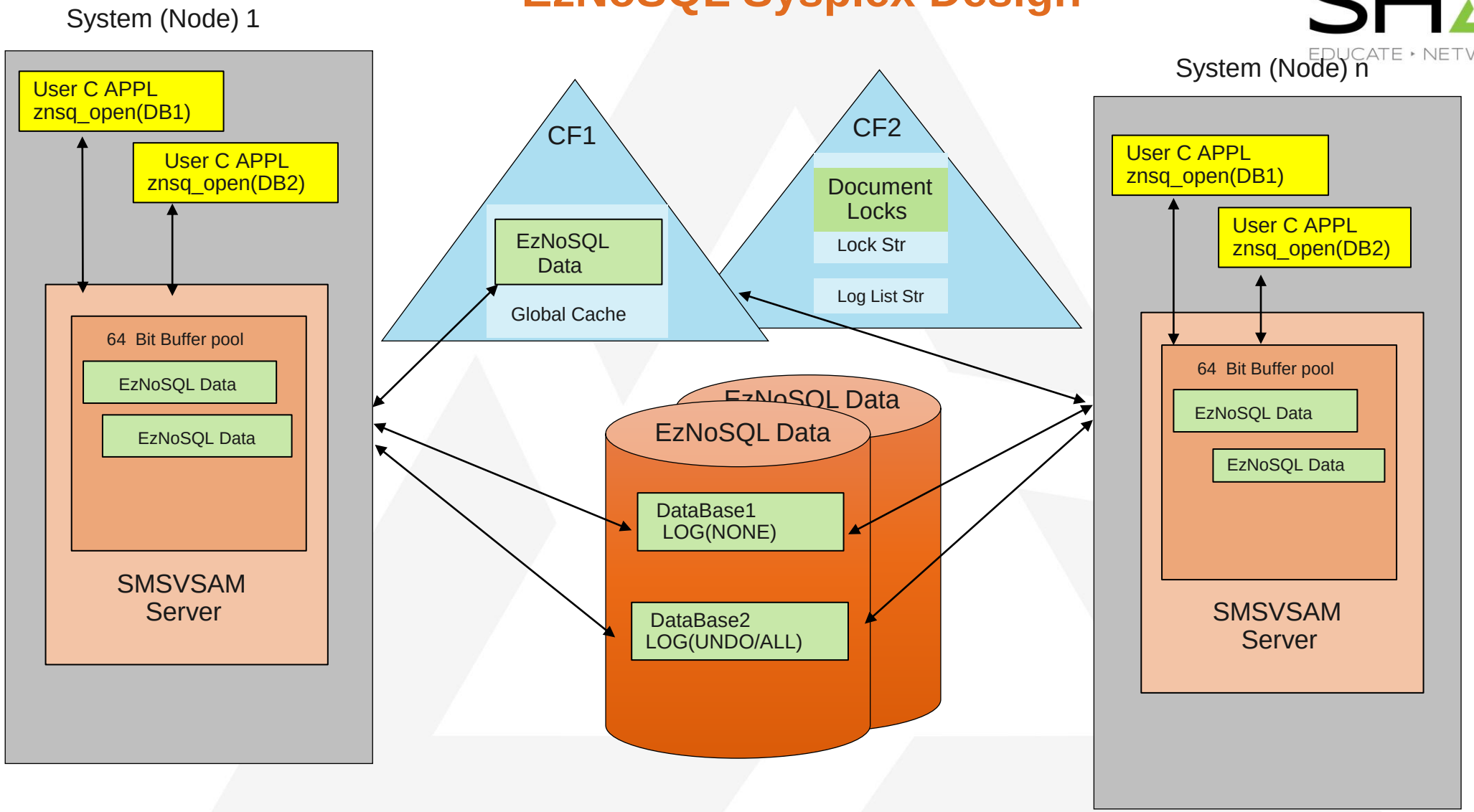
- Overview
- JSON Documents
  - Why JSON?
- Indexes
  - Index Examples
- Recoverable vs Non-Recoverable Databases
- APIs
- API Example
- Futures

EzNoSQL Overview

EzNoSQL Overview

- EzNoSQL on z/OS provides a comprehensive set of C based Application Progammer Interfaces (APIs), which enable applications to store JSON (UTF-8) documents while utilizing the full data sharing capabilities of IBM's Parallel Sysplex technology and System z operating system (z/OS).
- IBM's Parallel Sysplex Coupling Facility (CF) technology, enables separate processors to share a single instance of a data set (collection of documents), without the need for data sharding, replicating the updates, or programming for eventual consistency.
- Sysplex allows for easy horizontal scalability by adding additional processors (or z/OS instances) as required. Implementing EzNoSQL on z/OS will inherit many of the desired functions provided by z/OS such as system managed storage, data encryption and compression.
- The JSON documents can be up to 2 gig in size and can be updated either as non-recoverable, or with recoverable (transactional) consistency across the sysplex.
- The APIs also allow for the creation of secondary indexes, which provide for faster queries to specific key fields within the JSON data.
- Available on z/OS 2.4 and above with APAR OA62553.
- Refer to the following website for more information: https://www.ibm.com/support/z-content-solutions/eznosql

EzNoSQL Sysplex Design



JSON Documents

JSON (UTF-8) Documents:

Document:
{ element, element,… }

- One document (object) = one VSAM record
- Elements can vary in location, number, contents, and size.

Where an element is:
“key” : value

- Keys are character strings, Values can be strings, numbers, arrays, etc. JSON supports 5 types.

JSON Example:

```
{ “_id” : “00000001” ,
  “Name” : “John Smith” }
```

JSON UTF-8 representation:

```
7B225F696422 3A 22 30 30 30 30 30 30 30 31 22 2C 224E616D6522 3A 224A6F686E20536D697468227D
{ " _ i d " : " 0 0 0 0 0 0 0 1 " , " N a m e " : " J o h n S m i t h " }
```

For compete specification on JSON:
http://bsonspec.org/spec.html
www.json.org

Why JSON?

NoSQL (i.e. unstructured) key:value data bases have the following characteristics:

- The data (objects) may have widely varying formats:
  - {‘Name’ : ‘John Smith’, ‘Address’ : ‘24 First St’, ‘Email’ : ‘JohnSmith@gmail.com}
  - {‘Address’ : ‘1 North St’, ‘Name’ : ‘Joe Jones’ , ‘Gender’ : ‘M’ , ‘Married’ : ‘Y’}
- Data format not directly related to the file definition. Allows for rapid application changes.
- No need for a DBA to manage the data base.

VSAM data bases (e.g. “semi-structured”) have the following characteristics:

- The data (records) are partly structured and partly varying:
  - **#1256789 John Smith** (flags) 24 First St JohnSmith@gmail.com
  - **#3763521 Joe Jones** (flags) M Y
- Data format must have **primary** and **alternate** keys in the same location and length.
- Changes to the application related to the key location’s or length require the data base to be redefined.

EzNoSQL Indexes

EzNoSLQ Indexes

- Indexes:
  - All key names must be <256 characters
  - Indexes can be read forward or backward
  - Primary Indexes:
    - User supplied key values must be unique
    - If a key name is not supplied on the create, a primary key of “znsq_id” and unique key values will be auto-generated and pre-appended to the user document
    - No length restriction for the key value
    - Sequential inserts not allowed
    - Sequential reads will not maintain order
  - Secondary Indexes:
    - Optionally added by the application
    - Dynamically enabled/disabled however physically created only when the database is closed.
    - Key values maybe unique or nonunique, 4 gig duplicate keys allowed.
    - No length restriction, however, only the first 251 bytes are used as the value, otherwise treated as a non-unique key value
    - Sequential inserts allowed
    - Sequential reads will maintain order
  - Multkeys:
    - Can be used as a primary or secondary key
    - Multiple key names are concatenated via a reverse solidus character: \
    - Allows key values to be used within imbedded documents or arrays

EzNoSQL Index Examples

```
{
  "Customer_id":"4084",
  "Address": { "Street":"1 Main Street", "City":"New York", "State":"NY" }
  "Accounts": ["Checking","Savings"]
 }
```

- Unique primary key “Customer_id”:
  - Key Value: “4084”
  - “4084” is encrypted and randomized into an internal “derived key”
  - Document can be retrieved by with the argument “Customer_id” and “4084”
- Secondary key “Address”:
  - Key value: { "Street":"1 Main Street", "City":"New York", "State":"NY" }
  - Document can be retrieved with the argument “Address” and { "Street":"1 Main Street", "City":"New York", "State":"NY" }
- Secondary key “Accounts:
  - Key values: “Checking” and “Savings”
  - Document can be retrieved by either “Address” and “Checking” or “Savings”
- Secondary multikey “Address\Street”:
  - Key Value: “1 Main Street”
  - Document can be retrieved with the argument “Address\Street” and “1 Main Street”

EzNoSQL Recoverable vs Non-Recoverable Databases

Recoverable vs Non-Recoverable Databases

- Determined during create with the znsq_log_options parameter:
  - NONE – represents a non-recoverable data base
  - UNDO or ALL – represents a recoverable data base
- The recoverability of a database determines the duration of the locking and the transactional (atomic) capabilities:
  - Non-recoverable:
    - Exclusive document level locks (obtained for all writes) are held only for the duration of the write request
    - Shared locks (optionally obtained for reads) are held only for the duration of the read request
  - Recoverable:
    - Exclusive document level locks (obtained for all write requests) are held for the duration of the transaction.
    - Shared locks (optionally obtained for reads) have two options:
      - ✓ Consistent Reads (CR) are held for the duration of the read request
      - ✓ Consistent Read Extended (CRE) are held for the duration of the transaction
    - A transaction ends following a commit or backout.
    - Auto commits are issued by default after every write request. May be disabled with the znsq_set_autocomit

EzNoSLQ APIs

C Level APIs **

Four Elements

DB Management	Connection Management	Document Management	Document Retrieval
Create DB / Destroy DB Add Index / List Indicies	Open / Alt Open Close	Add Document Delete Document	Search Next Result Close Result Set
Primitives focused on Managing VSAM Datasets and indexes of entries	Primitives focused on Open and closing of VSAM datasets and managing indices	Primitives focused on Adding new entries to and removing from VSAM datasets	Primitives focused on Retrieving entries matching a criteria

C code located in USS: /usr/lib/ibm/libigwznsqd31.so
libigwznsqd31.x
libigwznsqd64.so
libigwznsqd64.x
/usr/include/zos/igwzbsq.h

C Level APIs **

Four Elements

DB Management	Connection Management	Document Management	Document Retrieval
Create DB / Destroy DB Add Index / List Indicies	Open / Alt Open Close	Add Document Delete Document	Search Next Result Close Result Set

znsq_create	znsq_open	znsq_update_result	znsq_position
znsq_create_index	znsq_close	znsq_delete_result	znsq_read
znsq_add_index		znsq_delete	znsq_next_result
znsq_drop_index		znsq_update	znsq_close_result
znsq_destroy		znsq_write	
znsq_report_stats		znsq_commit	
		znsq_backout	
		znsq_last_result	
		znsq_set_autocomit	

***Information on the new product is not a commitment, promise, or legal obligation to deliver any material, code or functionality. The development, release, and timing of any features or functionality described for our products remains at IBM's sole discretion.*

Example High Level C Program: **

znsq_create();	! Defines JSON dataset (name, keyname, ….)
znsq_create_index()	! Defines secondary index (name, altkey,…)
znsq_open();	! Returns a Connection_Token for this dataset/keyname and specified options
znsq_insert();	! Inserts and opens (1st Reference) base/path for specified key/altkey name
znsq_add_index();	! Defines and builds the index for the desired alternate key
znsq_read();	! Reads the document using either the base/ path for specified key/altkey name
znsq_update();	! Updates the document using either the base/ path for specified key/altkey name
znsq_commit();	! Commits last updates
znsq_report_stats();	! JSON document returned with database attributes
znsq_close();	! Closes the connection
znsq_destory();	! Deletes the JSON/BSON dataset

*** Executable example located in USS: /samples/IBM/igwznsqsamp1.c

Example znsq_report_stats API **

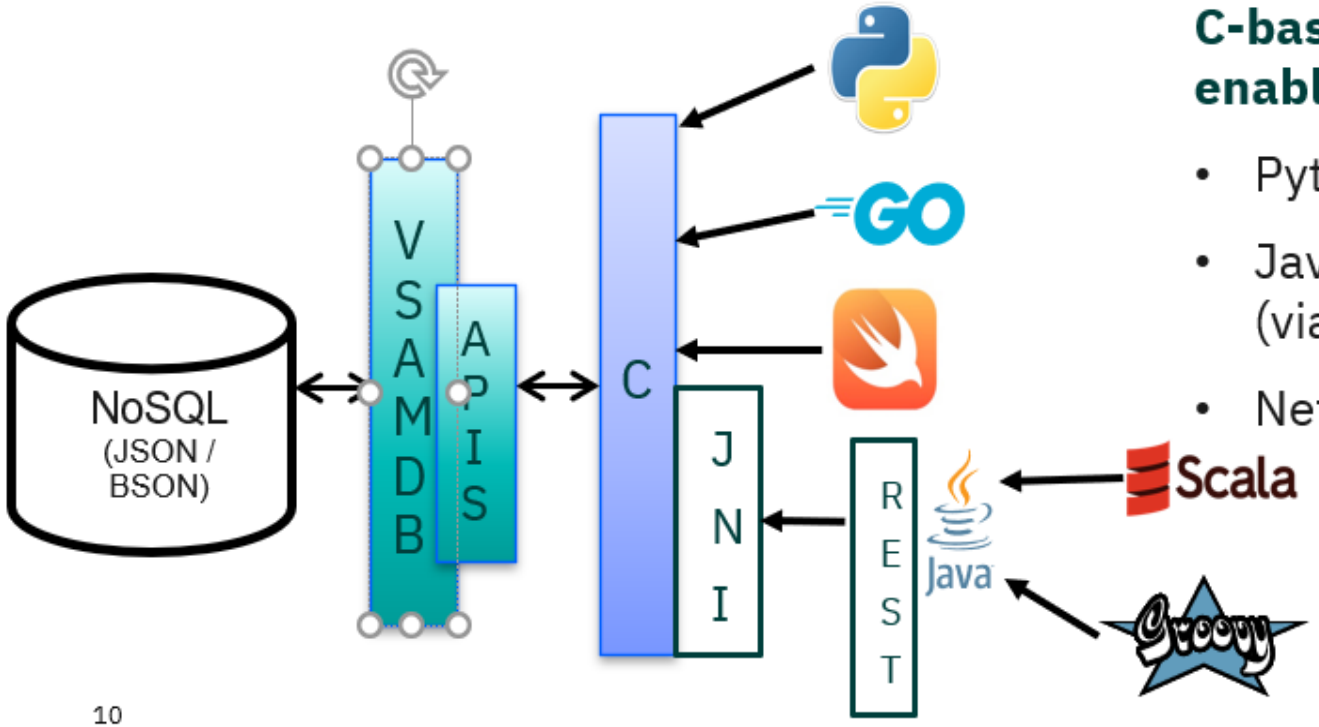

```
{"name":"HL1.JSON.KSDS1",
                "version":1,
                "documentFormat":"JSON",
                "keyname":"_id",
                "logOptions":"UNDO",
                "readIntegrity":"NRI",
                "readOnly":false,
                "writeForce":true,
                "autoCommit":false,
                "descendingKeys":false,
                "timeout":5,
                "avgDocumentSize":1000,
                "blockSize":26624,
                "avgElapseTime:150,
                "avgCPUTime:8,
                "statistics":
                  {"numberBlocksAllocated":1234,
                   "numberBlocksUsed":124,
                   "numberExtents":1,
                   "numberRecords":2,
                   "numberDeletes":5,
                   "numberInserts":10,
                   "numberUpdates":4,
                   "numberRetrieves":13},
                "numberIndices":1,
```

Example znsq_report_stats API (cont.)**


```
"indices":
                [{"name":"HL1.JSON.AIX.KSDS1",
                 "keyname":"Firstname",
                 "pathname":"HL1.JSON.PATH1",
                 "active":true,
                 "unique":true,
                 "descendingKeys":false,
                 "blockSize":26624,
                 "statistics":
                    {"numberBlocksAllocated":1234,
                     "numberBlocksUsed":124,
                     "numberExtents":1,
                     "numberRecords":2,
                     "numberDeletes":5,
                     "numberInserts":10,
                     "numberUpdates":4,
                     "numberRetrieves":13},
                 "numberCompoundKeys":1,
                 "compoundKeys":
                    [{"descendingKeys":T/F",name":"altkeyname"},
                     {}...]
                }]
              }
```

Futures

FUTURE Options**: Data stored in a platform independent format with full data sharing capabilities



C-based key-value interface to a NoSQL database enables higher level languages and interfaces

- Python, Go, Swift (interfaces to call directly)
- Java-based languages, like Groovy and Scala (via JNI)
- Network NoSQL interfaces

10

EzNoSQL Futures

Your feedback is important!

SHARE
EDUCATE • NETWORK • INFLUENCE

Submit a session evaluation for each session you attend:

www.share.org/evaluation