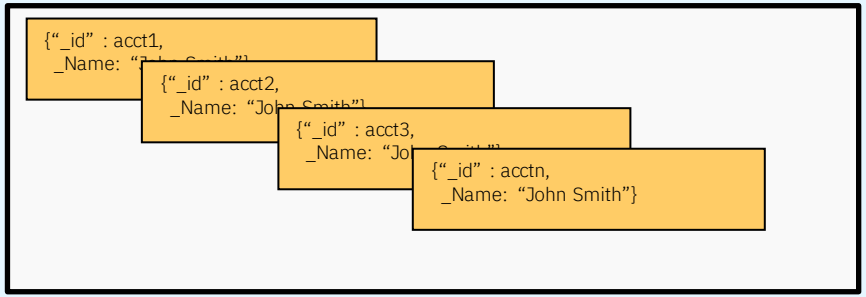
EzNoSQL Overview



Terri Menendez

STSM z/OS Development

terriam@us.ibm.com

Sept 2022

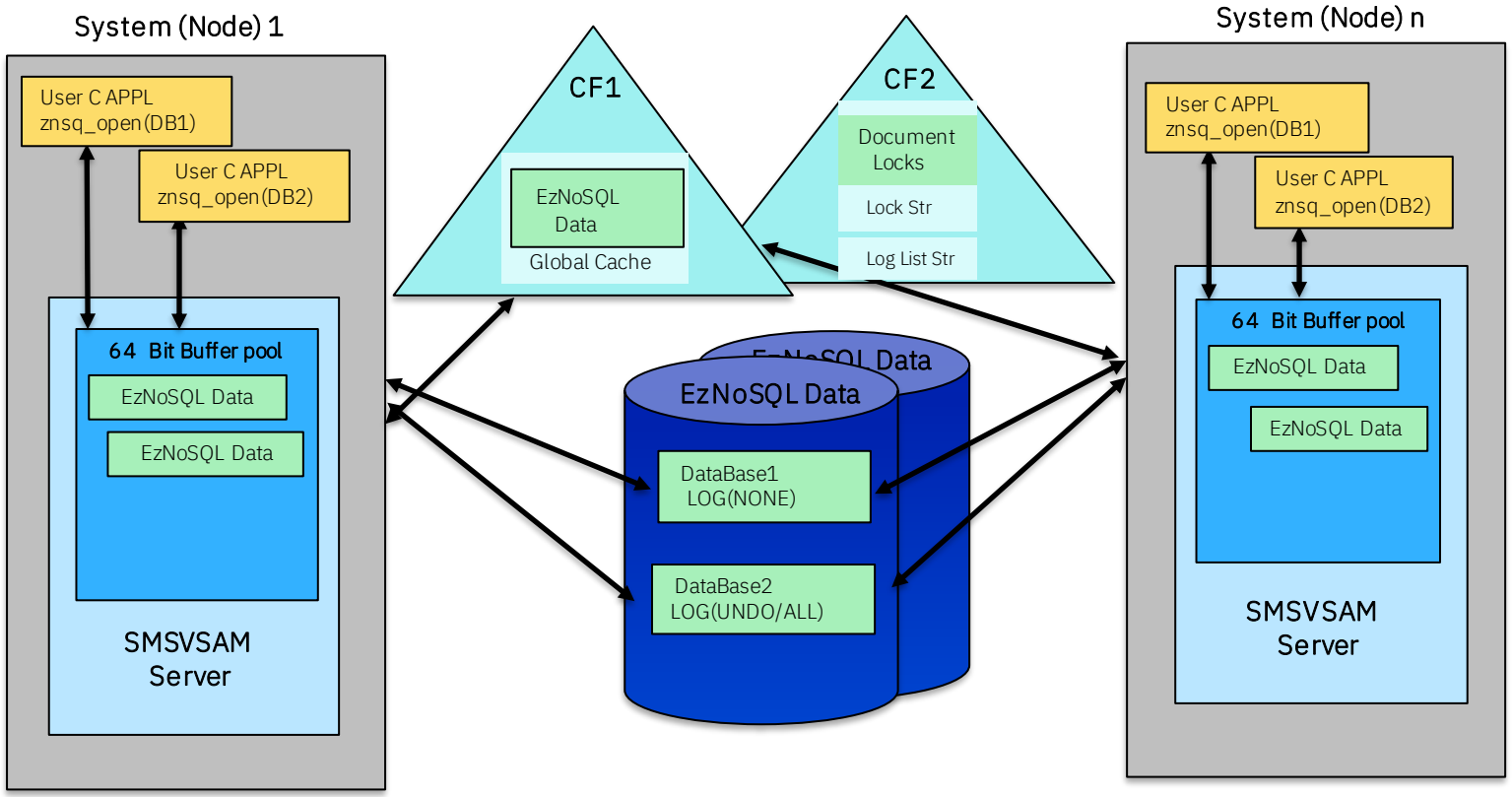
Agenda

- Overview
- JSON Documents
  - Why JSON?
- Indexes
  - Index Examples
- Recoverable vs Non-Recoverable Databases
- APIs
  - API Examples
- Performance
  - Internal Benchmark
- Content Solution Website
  - Getting Started
  - Technical documentation
  - Getting Started
  - Executables and Side Deck
  - Sample Program
- Futures
- Discussion / Questions

EzNoSQL Overview

- EzNoSQL on z/OS provides a **comprehensive set of C based Application Programmer Interfaces (APIs):**
  - enables applications to access JSON (UTF-8) documents
  - while utilizing the full data sharing capabilities of IBM's Parallel Sysplex Coupling Facility (CF) technology and System z operating system (z/OS).
- IBM's CF technology provides shared memory between processors which
  - enables separate processors to **share a single instance of a data base** (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**.
- Available on z/OS 2.4 and above with APAR **OA62553**.

EzNoSQL Sysplex Design



JSON Documents

JSON (UTF-8) Documents:

Document:
{ element, element,... }

Where an element is:
“key” : value

JSON Example:
{ “_id” : “00000001” ,
“Name” : “John Smith” }

- One document (object) = one VSAM record
- Elements can vary in location, number, contents, and size.
- Keys are character strings, Values can be strings, numbers, arrays, etc. JSON supports 5 types.

JSON UTF-8 representation:

```
7B225F696422 3A 22 30 30 30 30 30 30 30 31 22 2C 224E616D6522 3A 224A6F686E20536D697468227D
{  "_id" : " 0 0 0 0 0 0 0 1 " , " Name " : " 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 (e.g. 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’s location or length require the data base to be redefined.

EzNoSQL Indexes

EzNoSQL 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
- Multikeys:
  - 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 “Address” and either “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

* CICSVR currently does not support EzNoSQL forward recoveries.

EzNoSQL 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

***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.*

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_abort	
		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: **

```
return_code =  znsq_create();          ! Creates JSON dataset (name, keyname, UNDO....)
  if return_code != 0 {                ! If create failed  *
     return_code = znsq_last_result()  ! Obtain znsq _last_result diaganostic info
return_code  = znsq_create_index()     ! Defines secondary index (name, altkey,...)
return_code = eznsq_open();            ! Returns a Connection_Token for this dataset/keyname and specified options
return_code = znsq_insert();           ! Inserts and opens (1st Reference) base/path for specified key/altkey name
return_code = znsq_add_index();        ! Defines and builds the index for the desired alternate key
return_code = znsq_read();             ! Reads the document using either the base/ path for specified key/altkey name
return_code = znsq_update();           ! Updates the document using either the base/ path for specified key/altkey name
return_code = znsq_commit();           ! Commits last updates
return_code = znsq_report_stats();     ! JSON document returned with database attributes
return_code = znsq_close();            ! Closes the connection
return_code = znsq_destory();          ! Deletes the JSON/BSON dataset
```

** Ideally the znsq_last_result should be obtain after every failed API.l

Example: znsq_last_result API

```
znsq.last.result Report  2022.042 15:48:52
API Name: znsq_create          RC: 0000000C  RS: 82030004
Diagnostic Data:
IDCAMS  SYSTEM SERVICES                                    TIME: 15:48:51        03/03/22

DEFINE CLUSTER                      -
   (NAME(MY.JSON.DATA) -
   CYLINDERS(1000000 1)      -
   RECORDSIZE(500000 500000) -
   SHAREOPTIONS(2 3)        -
   STORCLAS(SXPXXS01)        -
   DATACLAS(KSX00002)        -
   LOG(NONE)                 -
   SPANNED                   -
   DATABASE(JSON)            -
   KEYNAMEU(_id)             -
   VOLUME (XP0201)           -
   FREESPACE (50 5))         -
   DATA(NAME(MY.JSON.DATA.D) -
   CONTROLINTERVALSIZE (32768)) -
   INDEX(NAME(MY.JSON.DATA.I) -
   CONTROLINTERVALSIZE (32768))
   IGD01007I DATACLAS ACSRTN VRS 02/19/92-1 ENTERED
```

Example: znsq_last_result API (cont.)

```
IGD01007I DATACLAS ACS EXECUTOR TEST ROUTINE IS ENTERED
IGD01007I DATACLAS DEFAULTING TO JCL OR TSO BATCH ALLOCATE
IGD01008I THE STORCLX ACSRTN VRS 07/06/99-1 ENTERED FOR SYSTEM X
IGD01008I STORCLAS ACS EXECUTOR TEST ROUTINE IS ENTERED
IGD01008I STORCLAS DEFAULTED TO JCL OR TSO BATCH ALLOCATE
IGD01009I MGMTCLAS ACSRTN VRS 07/28/88-1 ENTERED FOR SYSTEM X
IGD01009I   MGMTCLAS BEING DEFAULTED VIA JCL
IGD01010I STORGRP ACSRTN VRS 08/06/98-1 ENTERED FOR SYSPLEX
IGD01010I STORGRP BEING SET VIA STORCLAS UNLESS MULTIPLE GRPS SET
IGD01010I STORGRP BEING SET VIA STORCLAS UNLESS MULTIPLE GRPS SET
IGD17226I THERE IS AN INSUFFICIENT NUMBER OF VOLUMES IN THE ELIGIBLE
STORAGE GROUP(S) TO SATISFY THIS REQUEST FOR DATA SET
HL1.JSONWS01.BASE.KSDS1
IGD17290I THERE WERE 2 CANDIDATE STORAGE GROUPS OF WHICH THE FIRST 2
WERE ELIGIBLE FOR VOLUME SELECTION.
THE CANDIDATE STORAGE GROUPS WERE:SXP01 SXP02
IGD17279I 1 VOLUMES WERE REJECTED BECAUSE THEY WERE NOT ONLINE
IGD17279I 1 VOLUMES WERE REJECTED BECAUSE THE UCB WAS NOT AVAILABLE
IGD17279I 2 VOLUMES WERE REJECTED BECAUSE OF INSUFF TOTAL SPACE
IGD17219I UNABLE TO CONTINUE DEFINE OF DATA SET
HL1.JSONWS01.BASE.KSDS1
IDC3003I FUNCTION TERMINATED. CONDITION CODE IS 12

IDC0002I IDCAMS PROCESSING COMPLETE. MAXIMUM CONDITION CODE WAS 12
```

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"},
                {}...]
              }]
            }
```

Performance

Internal Benchmark

- System Configuration:
  - CPU 8561 Model 716 (z15) with 256G memory.
  - 1 LPAR
  - 8 Jobs, 1 TCB each

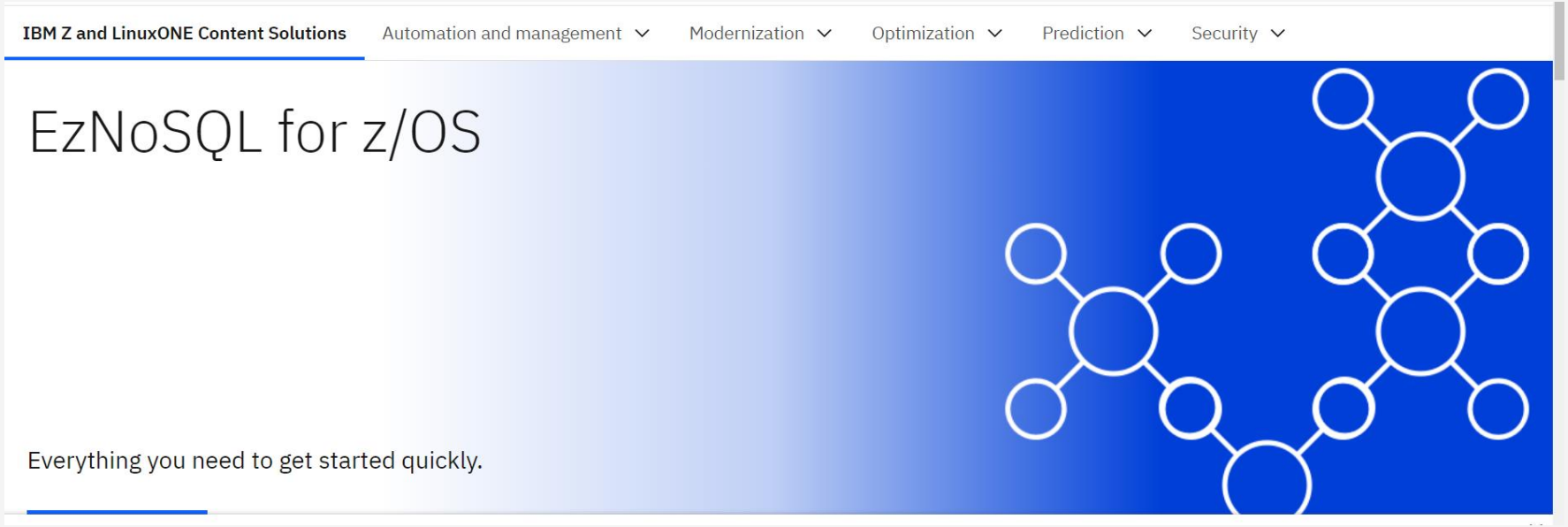
All number's in decimal				#		Index		Elapse	Trans	CPU	Number of dispatches per sec.				DASD
Runid	CPU/IIP	SSD	Option	Thrds	DB Size	Level	requests	Time (sec)	per sec	Busy	CPTCB	CPSRB	IIPTCB	IIPSRB	Rate
Jurgen's Benchmark															
T2179TA1	8/0	Y	Write	8	10,000,000	3	10,000,000	330	30,303	39.30	39,300	74,156	0	0	36,151
T2179TA4	8/0	Y	read	8	10,000,000	3	75,436,314	300	251,454	97.40	55,157	47,378	0	0	5,985
T2241AA1	8/0	N	read	8	10,000,000	3	81,650,817	300	272,169	97.24	75,982	66,947	0	0	204

Note: Performance APAR OA63680 is recommended for similar results in multitask configurations

Content Solution Web Site

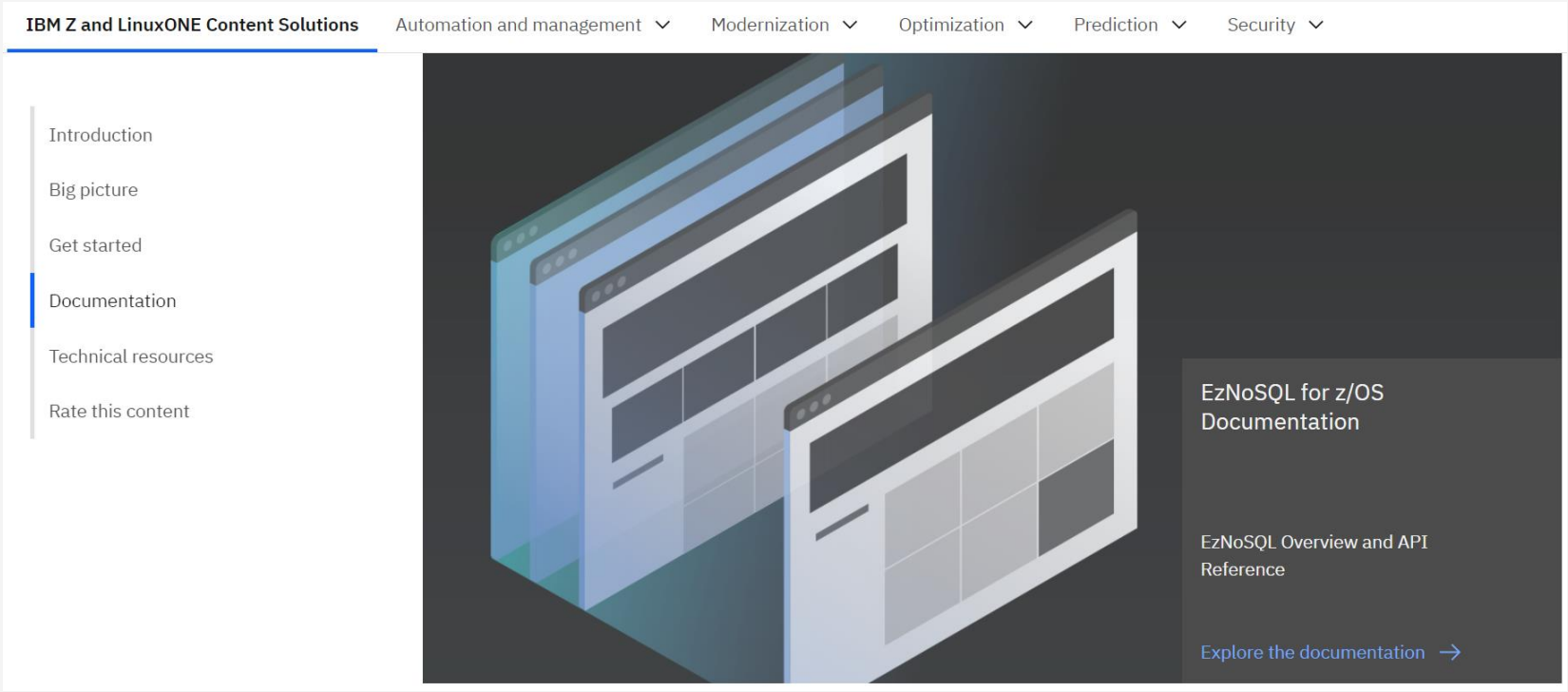
Getting Started…

- Website will be active when the PTFs for OA62553 close:
  - https://www.ibm.com/support/z-content-solutions/eznosql



Getting Started…

- Documentation:



Getting Started…

– Documentation in GitHub:

Getting Started...

Table of Contents

Getting Started...

Executables and Side Decks

The following table shows the names and locations of the EzNoSQL executables, side decks, and sample program:

Member	Location	Description
`libigwznsqd31.so`	`/usr/lib/`	31-bit API Library DLL
`libigwznsqd31.x`	`/usr/lib/`	31-bit x side deck
`libigwznsqd64.so`	`/usr/lib/`	64-bit API Library DLL
`libigwznsqd64.x`	`/usr/lib/`	64-bit APIs
`igwznsqdb.h`	`/usr/include/zos/`	EzNoSQL Header File
`igwznsqsamp1.c`	`/samples/`	Sample 31-bit application program

Getting Started...

– Compile and Link the Sample Program:

Compile and Link Procedure

To compile and link the sample program `/samples/ibm/igwznsqsamp1.c`:

```
xlc -c -qDLL -qcpluscmt -qLSEARCH="//'SYS1.SCUNHF'" igwznsqsamp1.c
xlc -o igwznsqsamp1 igwznsqsamp1.o -W l,DLL /usr/lib/libigwznsqd31.x
```

Example C Program:

- Executable example located in USS: /samples/igwznsqsamp1.c or /samples/IBM/igwvrivp
- (Minor customization required (HLQ and STORCLAS) documented in prolog)

```
//
// Program Name: igwznsqsamp1.c
//
// Function: Verify the EzNoSQL API's provided in z/OS 2.4 by:
//           Use the API's to create a JSON database and index,
//           write, position and read three JSON documents using an
//           alternate key, close and destroy the JSON database.
//
//           Note: Before running this program, be sure the PTF for
//                 the EzNoSQL enabling APAR OA62553 has been
//                 installed on your system.
//
// To compile and link this sample program:
// xlc -c -qDLL -qcpluscmt -qLSEARCH="//'SYS1.SCUNHF'" igwznsqsamp1.c
// xlc -o igwznsqsamp1 igwznsqsamp1.o -W l,DLL /usr/lib/libigwznsqd31.x
//
```

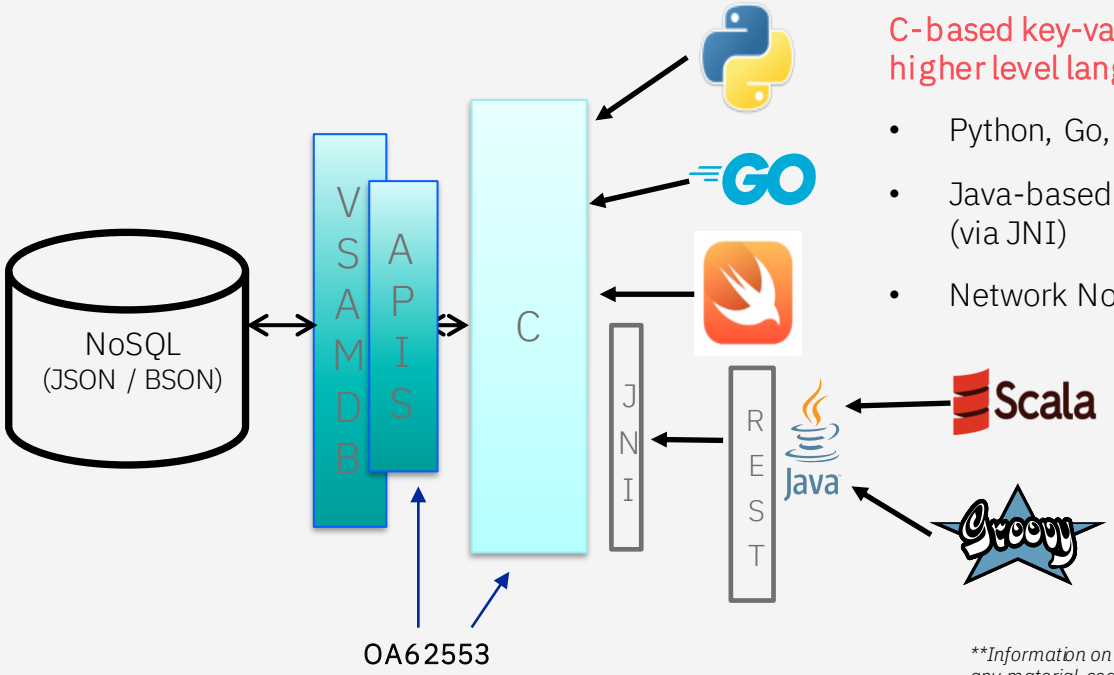
Example C Program:

```
# igwznsqsamp1
Database HL1.NOSQLDB created
Index HL1.NOSQLDB.AIX created
Database opened
Index for "Author" added
Document 1 written
Document 2 written
Document 3 written
Position to first document
Return code received from next_result was: X'34'
Verify first document
Return code received from next_result was: X'34'
Verify second document
Return code received from next_result was: X'0'
Verify third document
Database closed
Database destroyed
#
```

EzNoSQL Futures

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

***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.*