First look at MarkLogic

- EASY TO GET DATA IN
- EASY TO GET DATA OUT
- ENTERPRISE READY
- FLEXIBLE DEPLOYMENT
Brief history

• FOUNDED IN THE YEAR 2001.
• FOUNDERS: CHRISTOPHER LINDBLAD, PAUL PEDERSEN AND FRANK R. CAUFIELD
• INITIALLY BAPTIZED AS CERISENT.
• INITIALLY FOCUSED TO ADDRESS SHORTCOMINGS WITH EXISTING SEARCH AND DATA PRODUCTS BY USING XML DOCUMENT MARKUP.
• USED XQUERY AS THE QUERY STANDARD FOR ACCESSING COLLECTIONS OF DOCUMENTS.
RDBMS v MarkLogic

Typical RDBMS-Based Application Architecture

- Internet and Mobile clients
- HTTP processors (caches, proxies, balancers, SSL, etc.)

Application Server
- Application logic
- Data Access Objects
- Object/Relational Mapper (alternate)
- Database Driver (JDBC, ODBCP)
- Search Interface (Lucene)
- RDBMS (Oracle, MySQL, Postgres, etc.)
- Search Indexes

MarkLogic-based Application Architecture

- Internet and Mobile clients
- HTTP processors (caches, proxies, balancers, SSL, etc.)

Application Server
- Application logic
- Data Access Objects
- JSON, XML Object Mapping
- MarkLogic REST Client

Key

optional
MarkLogic Architecture

- **Interfaces**
  - Java
  - REST
  - XQuery
  - ODBC

- **Evaluation Layer**
  - Evaluator
    - XSLT | XPath | XQuery | SQL | SPARQL
  - Cache
    - Expanded Tree Cache | Module Cache
  - Broadcaster / Aggregator

- **Data Layer**
  - Transaction Controller
    - Multiversion Concurrency Control
  - Cache
    - Compressed Tree Cache | List Cache
  - Transaction Journal
  - Indexes
    - Data | Structure | Text | Scalar | Metadata | Security | Geospatial | Reverse | Triple
  - Compressed Storage
    - XML | JSON | Binary | Text

*Same process runs both evaluation and data layers*
Key features

- STRUCTURE AWARE
- SCHEMA AGNOSTIC
- DOCUMENT CENTRIC
- MULTI MODEL
- SEARCH ORIENTED
- TRANSACTIONAL (ACID)
- HIGH PERFORMANCE AND SCALABILITY
- HIGH AVAILABILITY
Document centric

- SUPPORTED DOCUMENT TYPES:
  - XML
  - JSON
  - TEXT DOCUMENTS
  - RDF TRIPLES
  - BINARY DOCUMENTS
Multi-model

- **TYPES OF DATA MODEL:-**
  - Document Store
  - Native XML
  - Resource Description Framework (RDF)
  - Search Engine
Search oriented

• SIMPLE QUERIES (URI/KEY-VALUE LOOK UP)

```
curl -X GET --anyauth --user username:password \\
```

• COMPLEX QUERIES (BASED ON WORDS/PHRASES/DOCUMENT STRUCTURE)

```xml
for $result in cts:search(
    /article[@year = 2010],
    cts:and-query(
        cts:element-word-query(xs:QName("description"),
            cts:word-query("pet grooming"))
        , cts:near-query(
            (cts:word-query("cat"), cts:word-query("puppy dog")), 10
        ),
        cts:not-query(cts:element-word-query(xs:QName("keyword"),
            cts:word-query("fish"))
    ))[1 to 10]
return
```
Data management

In Memory to On-Disk Stand

Merger Process
Transactional (ACID)
High availability

SHARED-DISK FAILOVER

1. NORMAL OPERATION
   - E-NODE
   - D-NODE PRIMARY
   - D-NODE FAILOVER 1
   - D-NODE FAILOVER 2
   - FOREST1

2. FAILURE
   - E-NODE
   - D-NODE PRIMARY
   - D-NODE FAILOVER 1
   - D-NODE FAILOVER 2
   - FOREST1
High availability (cont.)
Scaling with MarkLogic

GBs

$ $ $

Fast and easy to add or remove nodes

ELASTICITY

Inexpensive commodity hardware

TBs

$ $ $...

$ $ $...

$ $ $...
• FREE DEVELOPERS LICENSE.
• ESSENTIAL ENTERPRISE AT $18K/YEAR.
• ESSENTIAL ENTERPRISE ON AMAZON WEB SERVICES AT $0.99/HR.
DEEP IN FUNCTIONALITY
Basics

• QUERY
  - Standard text search
  - Element-level XML search
  - Native XQuery interface

• MANIPULATE
  - Navigate within content
  - Modify content programmatically
  - Combine content from multiple sources

• RENDER
  - Transform XML schema or DTDs
  - Output to various formats
Advanced

- SECURITY
- SEMANTIC INFERENCE OF FACTS
  - USING RULE SETS, AND SPARQL
- GEOSPATIAL
- DATABASE REPLICATION
- TIERED STORAGE
- BITEMPORAL
Security

- ROLE-BASED ACCESS CONTROL
  - SECURITY DATABASE, ADMINISTRATION
- AUTHENTICATION
  - INTERNAL OR EXTERNAL USING LDAP AND KERBEROS SECURITY
- CONFIGURATION MANAGEMENT
- ATOMIC FORESTS
Semantics

- Data is stored as triples
  - Subject, Predicate, Object
- Triple index used for efficient query
- Generate new facts and meta data
- Work as a graph model
- Combination query

Geospatial

- Points and regions of interest, intersecting paths.
- Geospatial queries, indexes and shapes
  - Points, (complex) polygons, circles, boxes
- Text (WKT) and well-known binary (WKB)
  - Point, linestring, triangle, multipoint, multilinestring, multipolygon, geometrycollection
- Integration with leading geospatial vendors
  - Robust visualization

"Show me a list of hospitals that fall within the boundaries of this certain set of coordinates"
Database replication

- FLEXIBLE REPLICATION
  - FILTERED AND MANIPULATED BEFORE REPLICATION
  - QUERY-BASED: UPDATES OF QUERY DYNAMICALLY UPDATE REPLICATED DATA.
- GEOGRAPHICALLY DISPERSED CLUSTERS AND MOBILE USERS
- MASTER-SLAVE ARCHITECTURE
- TRANSITIVE REPLICATION
- SAFE UPDATES
Tiered storage

- Data in different tiers based on cost and performance
- Reduce the cost of storage
- Save time managing storage
Update

• USING TEMPORAL DATABASE
  - No update! No delete!
  - Only insert and read-at-a-time
  - Every document has two timestamps
    - “created”, “expired”

• HIGH THROUGHPUT

• BITEMPORAL
  - Rewind the information
  - Capture evolving data and business through time

Valid Time – Real-world time, information “as it actually was”
System Time – Time it was recorded to the database
Query/answer processing

Browser Tier
- User submits a question
- Question as JSON

Middle Tier
- Business Logic
  - Create Q&A object
  - Build patch
- Submit document
- Database Client
  - MarkLogic REST API

Database Tier
- MarkLogic REST API
- Insert Q&A document

Browser Tier
- User submits an answer
- Answer as JSON

Middle Tier
- Business Logic
  - Augment answer
  - Build patch
- Submit patch
- Database Client
  - MarkLogic REST API

Database Tier
- MarkLogic REST API
- Update Q&A document
- Patch as JSON
Developer tools

JSON
Unified indexing and query for today's web and SOA data

Node.js Client API
Enterprise NoSQL database for Node.js

Java Client API
NoSQL agility in a pure Java interface

Server-Side JavaScript
JavaScript runtime inside MarkLogic using Google's V8

Node.js Client API

Xquery API
Query XML documents using XPath expressions

e.g. Construct a JSON object

```javascript
object-node {
  "p1": "v1", "p2": [1, 2, 3], "p3": fn:true(), "p4": null-node()
}
e.g. Iterate through the results (the raw documents)

```javascript
DocumentPage page
= client.newDocumentManager().search(query, 1);

for (DocumentRecord doc : page) {
    System.out.println(doc.getContent(new JacksonParserHandle()));
}
e.g. Delete from the database every document in a collection

```javascript
xdmp:collection-delete("collection-uri")
```
SampleStack

- **END-TO-END THREE-TIERED APPLICATION IN JAVA AND NODE.JS**
  - QUESTION AND ANSWER SITE
- **ENCAPSULATES BEST PRACTICES AND INTRODUCES KEY MARKLOGIC CONCEPTS**
- **USE SAMPLE CODE AS A MODEL FOR BUILDING APPLICATIONS**
  - UI, FULL TEXT SEARCH, SEARCH RESULT FILTERING, USERS AND ROLES, FACETS
  - DOCUMENT MODEL, DOCUMENT INSERTION AND UPDATE
  - TRANSACTIONS AND DATA INTEGRITY
- **MODERN TECHNOLOGY STACK SHOWS WHERE MARKLOGIC FITS IN YOUR ENVIRONMENT**
IMPLEMENTATION CONCEPTS
Word indexing

INVERTED INDEX

• WORD -> DOCUMENT RELATION
• EVERY ENTRY IS CALLED A TERM LIST

HOW DOES IT SEARCH TWO DIFFERENT WORDS ??

• USE THE SAME DATA STRUCTURE AND GET THE INTERSECTING DOCUMENTS
Phrase indexing

- Use the same word-indexing data structure
- Use word positioning information
- Enhance the inverted index with additional information such as multiple words
Which indexing is used in MarkLogic??...

- ANYONE OF THESE SETTINGS IS USED AT RUNTIME
- EACH APPROACH HAS ITS OWN ADVANTAGE AND DISADVANTAGE
Indexing structure

- PARENT-CHILD INDEX FOR MAINTAINING HIERARCHICAL STRUCTURE OF XML AND JSON DOCUMENTS
- IT’S SIMILAR TO FAST PHRASE SEARCH BUT USES CONSECUTIVE TAGS
- SEARCHING AN ADVANCE DATABASE BOOK TITLED “INSIDE MARKLOGIC SERVER” USES THE FOLLOWING PARENT-CHILD HIERARCHY

  <BOOK><METADATA>ADVANCE DATABASE</METADATA>
  <TITLE>INSIDE MARKLOGIC SERVER</TITLE> .......... </BOOK>
Indexing structure (cont.)

```
<a>
  <b>
    <c>foo</c>
    <d>baz</d>
  </b>
  <e>baz</e>
</a>
```

<table>
<thead>
<tr>
<th>Term</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>a/b</td>
<td>1</td>
</tr>
<tr>
<td>a/e</td>
<td>1</td>
</tr>
<tr>
<td>b/c</td>
<td>1</td>
</tr>
<tr>
<td>b/d</td>
<td>1</td>
</tr>
</tbody>
</table>
Range index

- **SUPPORT** **FAST RANGE** **QUERIES**, - DOCUMENTS WITHIN PARTICULAR SET OF DATES
- **DATA TYPE AWARE EQUALITY** **QUERIES** – COMPARE DATES BASED ON SEMANTIC VALUE RATHER THAN ITS LEXICALLY CORRECT INITIALIZED VALUE
- **GET** **ORDER BY** **RESULTS** – SEARCH RESULTS SORTED BY ITEM PRICE
- **CROSS DOCUMENT** **JOINS** – MERGING TWO DOCUMENTS, ONE CONTAINING THE NAME OF THE PEOPLE AND THE OTHER CONTAINING THE DATE OF BIRTH OF THE PEOPLE
Metadata indexing and relevance

- PARENT-CHILD INDEX FOR MAINTAINING HIERARCHICAL STRUCTURE OF XML AND JSON DOCUMENTS
- SHORT DOCUMENTS WITH EQUAL NUMBER OF HITS OR DOCUMENTS CONTAINING RARE HIT WORDS ARE PRIORITIZED
- TERM LISTS ARE USED TO INDEX DIRECTORIES, COLLECTIONS AND SECURITY RULES -> UNIVERSAL INDEX

RELEVANCE = \log(\text{TERM FREQUENCY}) \times (\text{INVERSE DOCUMENT FREQUENCY})
Geospatial index

- Query terms based on geospatial indexes present in the document
- Match by exact latitude longitude or against an ad hoc polygon of vertices, which can be used to draw city boundaries
- Supports polar region coordinates, and anti-meridian longitude boundary near the international date line and considers the ellipsoid shape of Earth
- Point queries are resolved by range indexes and polygon queries are resolved by using high speed comparators to determine point position
- Special trigonometry operations to resolve searches related to polar coordinates
Point in time query

- In database each query is registered with a time stamp when the query starts.
- At present time, we can query the database as it was at an arbitrary time in the past.
- Useful for locally testing a feature (database roll back).

```
xdmp:eval("doc('/json/sample_doc.js on')",
<options xmlns="xdmp:eval">
<timestamp>96825</timestamp>
</options>)
```
Advance text handling

- TEXT SENSITIVITY – SUCH AS CASE-SENSITIVE, E.G.- ‘POLISH’ AND ‘POLISH’
- FROM MARKLOGIC 8.0 STEMMED INDEXING IS BY DEFAULT ENABLED
- WILDCARDED SEARCH QUERIES, SUCH AS MARK*, MAR*LOG*
Optimistic lock

- DOES NOT HOLD LOCK ON THE DOCUMENT IN BETWEEN READ AND UPDATE OPERATION
- CONDITIONAL UPDATE USING VERSION ID
- IT'S CONTENT VERSIONING NOT DOCUMENT VERSIONING

```bash
HTTP/1.1 200 Document Retrieved
Content-type: application/xml
ETag: "254768939037681240"
Connection: close

```
PROGRAMMING
WITH REST API
REST API Insert (PUT / POST) request

```shell

```
REST API Insert/Update content and metadata

REST API Data retrieval (GET Request)

**DOCUMENT**


**METADATA**


**CONTENT AND METADATA**

REST API Searching

SEARCHING

curl --anyauth --user user:password -X
GET -H "Accept: application/json"
http://localhost:8000/LATEST/search?q=hamlet

..."matches":
[ { "path": "fn:doc("/shakespeare/plays/hamlet.json")/PLAY/TITLE",
"match-text": [ "The Tragedy of ",
{ "highlight": "Hamlet" },
", Prince of Denmark"
] } ]
...
REST API Streaming

STREAMING

NO NEED TO LOAD THE ENTIRE CONTENT INTO MEMORY

REST API Patch UPDATE

curl --anyauth --user user:password -x post -d
@./patch_example.xml -i -h "content-type: application/xml" -h "x-http-method-override: patch"
/patch_example.xml

PATCH TEMPLATE

<rapi:patch
xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:insert />
  <rapi:replace-insert />
  <rapi:replace/>
  <rapi:delete/>
</rapi:patch>
REST API Patch UPDATE (cont.)

```xml
<api:patch
xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:insert context="/header/p[1]">
    <rapi:attribute-list attr1="val1" />
  </rapi:insert>
</api:patch>
```

<table>
<thead>
<tr>
<th>Before Update</th>
<th>After Update</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;header&gt;</code></td>
<td><code>&lt;header&gt;</code></td>
</tr>
<tr>
<td><code>&lt;p&gt;one&lt;/p&gt;</code></td>
<td><code>&lt;p&gt;one&lt;/p&gt;</code></td>
</tr>
<tr>
<td><code>&lt;p&gt;two&lt;/p&gt;</code></td>
<td><code>&lt;p&gt;two&lt;/p&gt;</code></td>
</tr>
<tr>
<td><code>&lt;p&gt;three&lt;/p&gt;</code></td>
<td><code>&lt;p&gt;three&lt;/p&gt;</code></td>
</tr>
</tbody>
</table>

Before Update: `<header>` `<p>one</p>` `<p>two</p>` `<p>three</p>`

After Update: `<header>` `<p attr1="val1">` one `</p>` `<p>two</p>` `<p>three</p>`
REST API DELETE Request

BLANK DIRECTORY OR COLLECTION NAME DELETES THE ENTIRE DATABASE

SINGLE DOCUMENT


MULTIPLE DOCUMENTS

APPLICATIONS
When MarkLogic?

- SPARSE, DIVERSE DATA
- QUERIES DATA ACCORDING TO POWER LAW
- RENDER RESULT IN SPECIFIC FORMAT DIRECTLY
- TERABYTES OF DATA IN DIFFERENT GEOGRAPHICAL LOCATIONS.
- NEED FASTER RESULTS.
- ELASTIC SECURITY, REPLICATION
- .......

MarkLogic™
Use cases

HealthCare.gov

BBC

Mitchell1
Project - HealthCare.gov

- Faster time to production: 18 months, within next 6 months – 5500+ transactions per second
- Scalability: 160,000 concurrent users, 99.9% availability, query response time <0.1 second
- Schema-agnostic data model: seamless online shopping for users
- Enterprise grade database platform: high availability and security
Project – BBC (London Olympics)

- DYNAMIC UPDATE ON EACH OF 10,000 ATHLETE PAGES
- OLYMPIC VIDEO CONTENT REQUESTS: 106 MILLIONS
- 2.8 PETABYTES OF DATA ON BUSIEST DAY
- EASY LOADING OF DATA: VIDEOS, ARTICLES, TWEETS, IMAGES, STATISTICS
Project – Mitchell1

- Complex data management and integration
- Enhancements every 2 weeks compared to once or twice per year
- Increase in revenue with better customer experience
- Cost reduction with less manual data transfer
And many more...
Trend charts

DB-Engines Ranking of Native XML DBMS

DB-Engines Ranking of RDF Stores

© October 2016, DB-Engines.com
Trend charts (cont.)

DB-Engines Ranking of Document Stores

DB-Engines Ranking of Search Engines

© October 2016, DB-Engines.com
Why not MarkLogic?
References


• J HUNTER. INSIDE MARKLOGIC SERVER, 2011.

• DB-ENGINES RANKING. KNOWLEDGE BASE OF RELATIONAL AND NOSQL DATABASE MANAGEMENT SYSTEMS, 2015.


• MARKLOGIC SERVER, CONCEPTS GUIDE. HTTPS://DOCS.MARKLOGIC.COM/GUIDE/CONCEPTS.PDF

thank you

GROUP 11

AVIRUP CHAKRABORTY
RASHA ELHESHA
SAPTARSHI CHAKRABORTY
DEBARSHI MITRA