



# Configuration Management

Norman Walsh, Lead Engineer  
01 May 2012

# Introduction

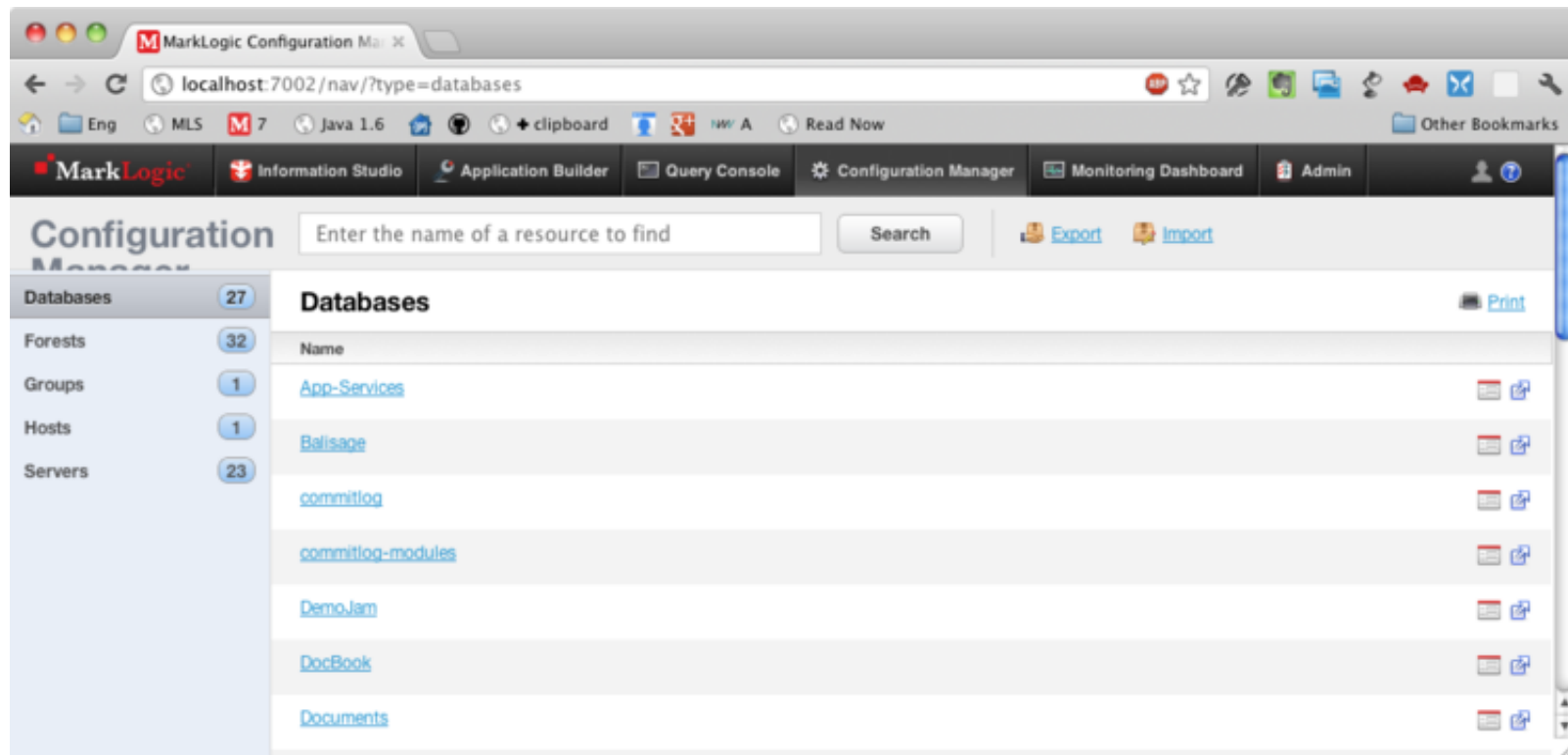


# Motivation

- What is the current configuration of this cluster?
- Is this cluster the same as that one?
  - How are they different?
  - How can I make them the same?

# Configuration Management

- Read-only configuration
  - What is the configuration of this cluster?

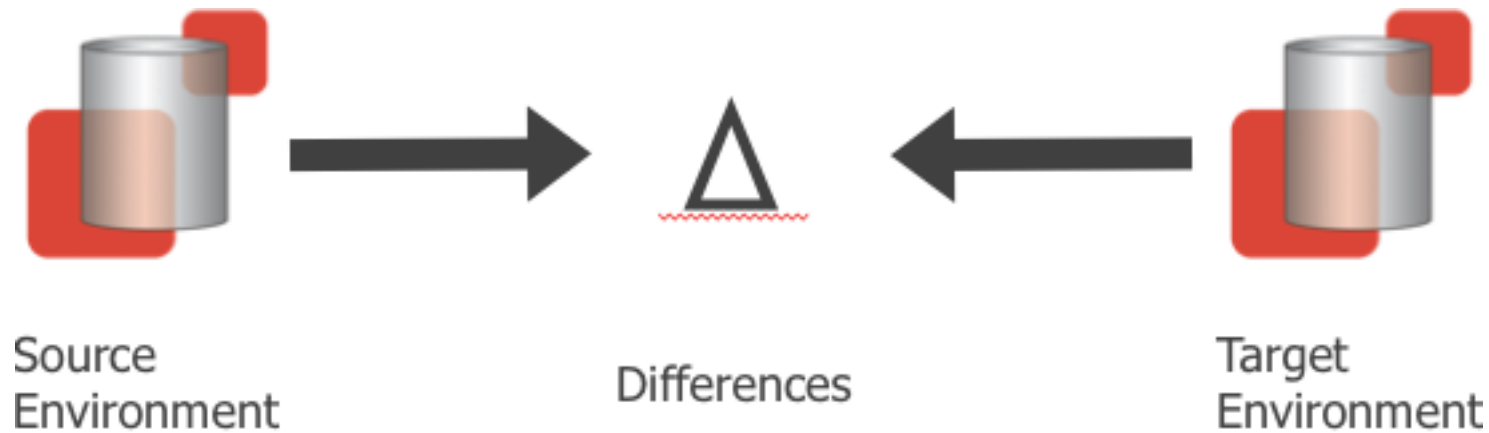


# Configuration Management

- More streamlined than the Admin UI
- Does not require “admin” privileges, users can be granted permission to view the configuration without being able to change it.
- For admins, the Admin UI is just a click away.

# Packaging

- Do the source and target environments differ? How can I make them the same?
- Source environment: certified master, developer system, staging cluster
- Target environment: additional clusters, staging cluster, production cluster



# What is it?

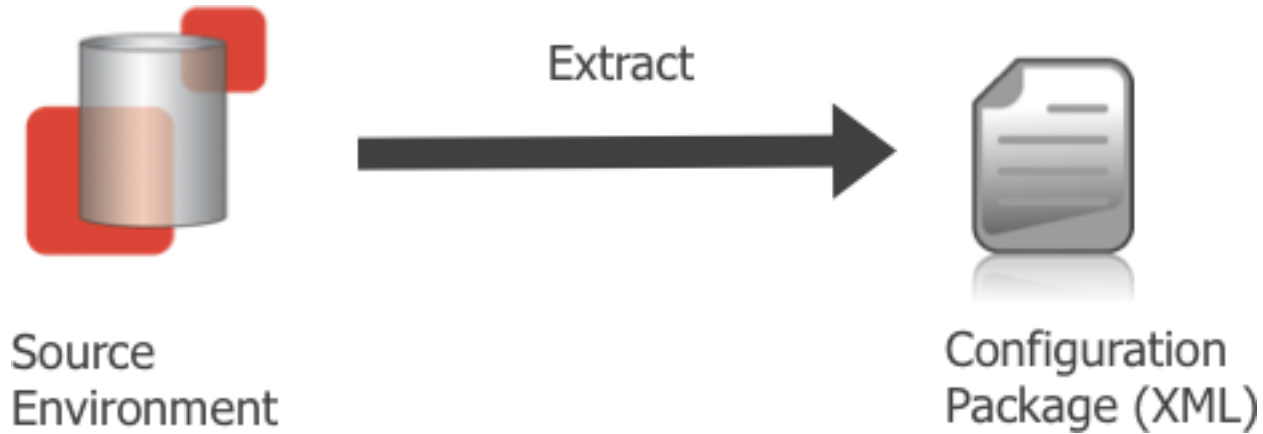
- A targeted solution to synchronize database and application server settings.
- A shared vocabulary to communicate configuration artifacts
- A set of tools to extract, compare, and apply changes:
  - A Browser UI
  - A REST API
  - An XQuery API

# Benefits

- Better integration with existing tools and practices
- Faster, more predictable deployments
- Shorter downtime windows
- Fewer “one off” errors
- More efficient communication between devs, ops, and support

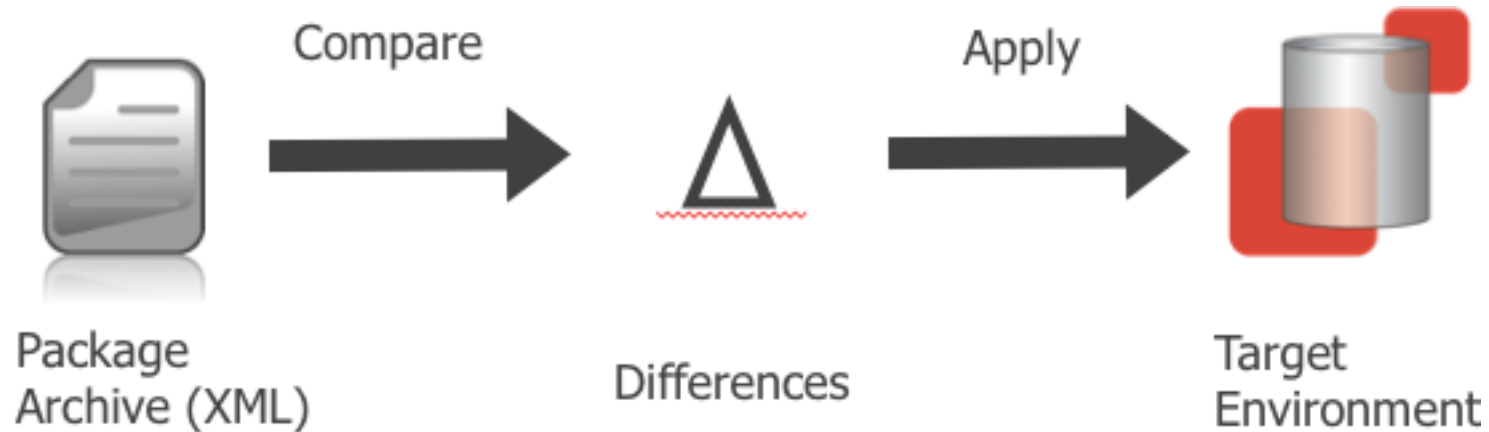
# Configuration push (1/2)

- Extract relevant configuration from source



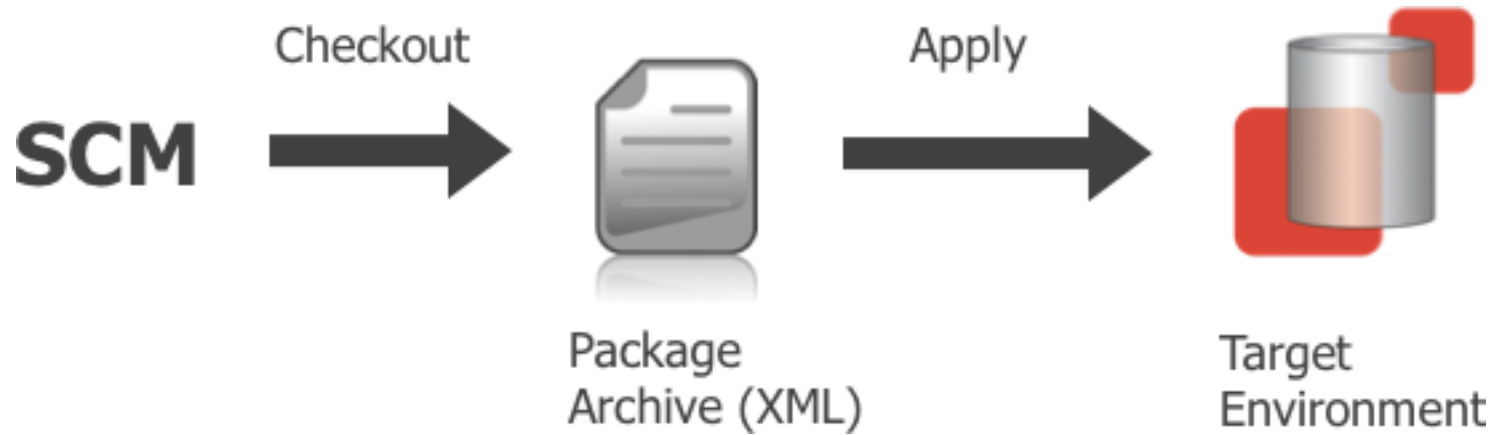
# Configuration push (2/2)

- Extract relevant configuration from source
- Compare against target
- Update target configuration



# Initial setup

- Developer setting up a new dev box



# Auditing production changes

- “Has something changed on the production server?”



# A few words about scope

- Handles configuration of databases and application servers only.
- Relies on other tools to coordinate code and data
- Doesn't manage relationships (users, schemas, etc.)

# Browser UI

# Demo

- Initial install
- Get updated package
- Compare updated package on target machine
- Apply changes

# REST API

# Download a package

```
GET http://localhost:8002/manage/v1/list/package?  
    database=dbname&appserver=group/servername  
package.xml
```

## Package XML:

```
<package xmlns="http://marklogic.com/package">  
  <package-database>  
    <metadata>  
      <package-version>1.0</package-version>  
    ...</metadata>  
    <name>Documents</name>  
    <package-database-properties>  
      <enabled>true</enabled>  
      <language>en</language>  
      <stemmed-searches>basic</stemmed-searches>...
```

# Compare a package

POST package.xml

<http://localhost:8002/manage/v1/package/compare>

"Delta" XML:

```
<pkg:package-diff xml:lang="en"
  xmlns:pkg="http://marklogic.com/package">
  <package-database-diff
    xmlns="http://marklogic.com/package/databases">...
    <package-database-properties>...
      <uri-lexicon>
        <pkg:delta>
          <pkg:del>>false</pkg:del>
          <pkg:add>>true</pkg:add>
        </pkg:delta>
      </uri-lexicon>...
```

# Install a package

POST package.xml

`http://localhost:8002/manage/v1/package/install`

"Result" XML:

```
<result xmlns="http://marklogic.com/package">  
  <ticket-id>/tickets/ticket/160...272</ticket-id>  
  <result-status>COMPLETED</result-status>  
  <result-event>PACKAGE_WRITTEN</result-event>  
</result>
```

# XQuery API

# The package API

```
import module namespace package
    = "http://marklogic.com/package/package"
    at "/MarkLogic/package/package.xqy" ;
```

- Public API contains all of the core functionality
  - `package:create`
  - `package:compare`
  - `package:install`
  - ...
- Like the `admin: API`: create a package, then update it
- Comparison returns an XML document detailing the differences
- Applying a package updates the server configuration.

# Reverting a package

- If it all goes horribly wrong...

```
let $ticket := "/tickets/ticket/2548827074908771567"  
return  
    package:revert($ticket)
```

# Q&A

Norman Walsh  
MarkLogic Corporation  
[Norman.Walsh@MarkLogic.com](mailto:Norman.Walsh@MarkLogic.com)



# Appendixes

# Creating a package

MarkLogic Information Studio Application Builder Query Console Configuration Manager Monitoring Dashboard Admin

## Packaging

[Export](#) [Import](#)

### Export a configuration package

Choose the application servers or databases whose configuration you would like to export [Download Package](#)

App servers	Databases
<input type="checkbox"/> Admin	<input type="checkbox"/> App-Services
<input type="checkbox"/> App-Services	<input type="checkbox"/> Documents
<input type="checkbox"/> Manage	<input type="checkbox"/> Fab
<input checked="" type="checkbox"/> Summit	<input type="checkbox"/> Last-Login
	<input type="checkbox"/> Modules
	<input type="checkbox"/> Schemas
	<input type="checkbox"/> Security
	<input checked="" type="checkbox"/> Summit
	<input checked="" type="checkbox"/> SummitLogging
	<input type="checkbox"/> Triggers

Package filename:

# Uploading a package

The screenshot shows the MarkLogic web interface. At the top is a navigation bar with the following items: MarkLogic, Information Studio, Application Builder, Query Console, Configuration Manager, Monitoring Dashboard, Admin, and a user profile icon. Below the navigation bar is a 'Packaging' section with two buttons: 'Export' and 'Import'. Underneath, there are three steps: 1. Upload package (selected), 2. Package comparison, and 3. Apply package. The main content area contains the text 'Choose a configuration package to upload.' followed by a text input field containing the path '/projects/presentations/2011/10-summit/summit-' and a 'Browse...' button. Below the input field is an 'Upload' button.

# Comparing a package

The screenshot shows the MarkLogic Packaging interface. The top navigation bar includes: MarkLogic, Information Studio, Application Builder, Query Console, Configuration Manager, Monitoring Dashboard, and Admin. The main header is 'Packaging' with 'Export' and 'Import' buttons. Below the header are three steps: 1. Upload package, 2. Package comparison (active), and 3. Apply package.

The comparison section shows:

- Package: summit-updated.xml
- Existing configuration
- Resulting configuration: 302 (green), 4 (yellow), 58 (red), 0 (grey) with an 'Apply' button.

The comparison table is for 'Database: Summit' and compares properties between the package and the existing configuration:

Property	Package	Existing configuration	Resulting configuration
Memory limits			Memory limits
Journal properties			Journal properties
Fast searches			Fast searches
Merge properties			Merge properties
Wildcard properties			Wildcard properties
Position properties			Position properties
Reindexing properties			Reindexing properties
Search properties			Search properties
Lexicon properties			Lexicon properties

Below the table, 'Collection Lexicon' is listed under the package column and 'None' under the existing configuration column.

# Applying a package

The screenshot shows the MarkLogic web interface. At the top is a navigation bar with the following items: MarkLogic, Information Studio, Application Builder, Query Console, Configuration Manager, Monitoring Dashboard, Admin, and a user profile icon. Below the navigation bar is a 'Packaging' section with two buttons: 'Export' and 'Import'. Underneath are three steps in a sequence: 1. Upload package, 2. Package comparison, and 3. Apply package. A 'Success' message is displayed, stating: 'The package was successfully applied. Ticket ID [/tickets/ticket/2548827074908771567](#)'.

# Creating a package

- Like the `admin: API`: create a package, then update it

```
let $pkg := package:create()  
let $pkg := pkg:add-database($pkg, "Summit")  
let $pkg := pkg:add-database($pkg, "SummitLogging")  
let $pkg := pkg:add-appserver($pkg, "Default", "Summit")  
return  
    $pkg
```

# Comparing a package

- Read the configuration off the current (target) machine
- Obtain the package XML from somewhere
- Compare the packages

```
let $target := package:create()  
let $target := pkg:add-database($pkg, "Summit")  
let $target := pkg:add-database($pkg, "SummitLogging")  
let $target := pkg:add-appserver($pkg,  
                                "Default", "Summit")  
  
let $source := xdmp:document-load("../package.xml")  
let $errors := package:errors($source)  
return  
  if (empty($errors))  
  then package:compare($source, $target)  
  else ...
```

# Applying a package

- If you're happy with the differences, you can install the package

```
let $source := xdmp:document-load( ".../package.xml " )
let $errors := package:errors($source)
return
  if (empty($errors))
  then package:install($source)
  else ...
```

# Database properties

The following properties are part of a package:

- Fragment Roots
  - namespace/localname pairs
- Fragment Parents
  - namespace/localname pairs
- Merge Policy
  - Priority
  - Max size/min size/min ratio
  - Merge timestamp
  - Blackout periods
    - Type
    - Disable/limit
    - Priority
    - Days
    - Duration
- Scheduled Backups
  - Directory
  - Type

# Database properties

(Continued)

- Period
- etc.
- Element Range Indexes
  - Type, namespace/localname, collation, positions
- Attribute Range Indexes
  - Type, parent namespace/localname, namespace/localname, collation, positions
- Element Word Lexicons
  - Namespace/localname, collation
- Attribute Word Lexicons
  - Parent namespace/localname, namespace/localname, collation
- Word Query
  - Index settings
  - Include root
  - Includes
    - Namespace/localname, weight, attr namespace/localname, value
  - Excludes
    - Namespace/localname
- Fields

# Database properties

(Continued)

- Name
- Index settings
- Include root?
- Field range indexes
- Phrase-Throughs
  - Namespace/localname
- Phrase-Arounds
  - Namespace/localname
- Element-Word-Query-Throughs
  - Namespace/localname
- Geospatial Indexes
  - Element indexes
    - Namespace/localname, point format, positions
  - Element child indexes
    - Parent namespace/localname, namespace localname, point format, positions
  - Element pair indexes
    - Parent namespace/localname
    - Latitude namespace/localname
    - Longitude namespace/localname
    - Positions

# Database properties

(Continued)

- Element attribute pair indexes
  - Parent namespace/localname
  - Latitude namespace/localname
  - Longitude namespace/localname
  - Positions

The following properties **are not** part of a package:

- Forests
- Flexible Replication
- Database Replication
- Triggers
- Content Processing

# Common server properties

The following properties are part of a package:

- Server name
- Root
- Port
- Database
- Last login
- Address
- Backlog
- Threads
- Request timeout
- Keep alive timeout
- Max time limit
- Default time limit
- Pre-commit trigger depth
- Pre-commit trigger length
- Collation

# Common server properties

(Continued)

- Authentication
- Concurrent request limit
- Log errors
- Debug allow
- Profile allow
- Default XQuery version
- Output SGML character entities
- Output options

The following properties **are not** part of a package:

- Privilege
- Request blackouts
- SSL certificate template
- SSL hostname
- SSL ciphers
- SSL require client certificate
- Documents database
- Modules database

# Common server properties

## (Continued)

---

- Last-login database

# HTTP server properties

The following properties are part of a package:

- Modules
- Display last login
- Session timeout
- Static expires
- Error handler
- URI rewriter

The following properties **are not** part of a package:

- Default user

# WebDAV server properties

The following properties are part of a package:

- Compute content length
- Session timeout
- Static expires

The following properties **are not** part of a package:

- Default user

# XDBC server properties

The following properties are part of a package:

- Modules
- Display last login