



# Unit 4: Security

© COPYRIGHT 2015 MARKLOGIC CORPORATION. ALL RIGHTS RESERVED.

# Learning Objectives

- Describe the Administrator role.
- Describe the security database.
- Create users and roles.
- Describe the relationship between the Node.js client API and REST APIs and the security implementation at the REST API level.
- Describe document permissions.
- Describe execute and URI privileges.
- Explore the Samplestack security model and LDAP integration.

# Administrator Role

- The admin role is a predefined role that is given all privileges and permissions to perform any action in the system.
- Think about the cURL statements we executed in the prior lab:
  - Why were we able to create resources and perform CRUD operations?

```
curl --anyauth --user admin:admin -X GET  
"http://localhost:7010/v1/documents?uri=/songs/song1.xml"
```

- We were able to perform all attempted operations because we authenticated as the admin user that was created when we initialized our MarkLogic instance.

# Security Database

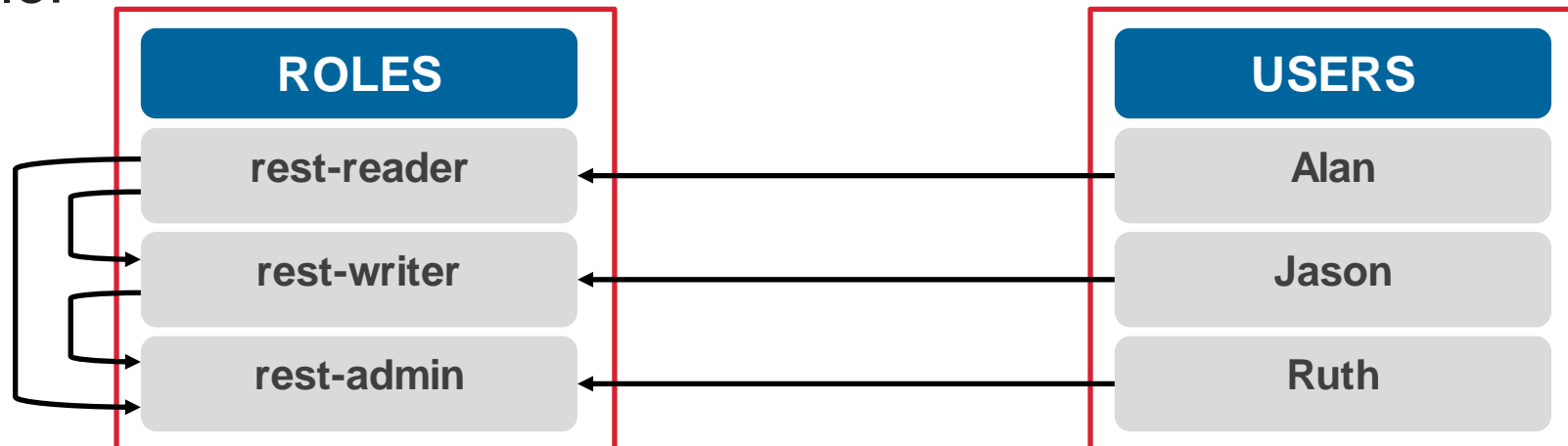
- The Security database is created on the first host that you initialize in a cluster.
  - That's why you create the admin account during initialization.
  - Users, roles, privileges and amps are stored here.
- Security database is shared across applications on that cluster.

<b>database name</b>	<input type="text" value="samplestack"/> The database name.
<b>security database</b>	<input type="text" value="Security"/> The security database.

<b>database name</b>	<input type="text" value="top-songs-content"/> The database name.
<b>security database</b>	<input type="text" value="Security"/> The security database.

# Users and Roles

- Roles are the foundation of the security model
  - Control what person(s) can do, see, change, etc.
- Users are assigned 1:many roles
- Roles can inherit other roles
- Example:



# REST API Security – Where it fits in the stack

- Remember:
  - The Node.js API communicates with MarkLogic through the REST API.
  - The user making the request must have appropriate privileges to perform the task.
    - For example: read a document, update a document, etc.



# REST API Security – Users & Roles

- User must have one of the predefined roles below or a custom equivalent:

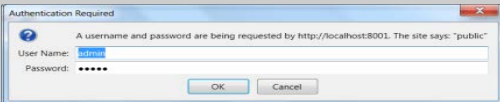
Role	Description
rest-reader	Enables read operations through the MarkLogic REST API, such as retrieving documents and metadata.
rest-writer	Enables write operations through the MarkLogic REST API, such as creating documents, metadata, or configuration information.
rest-admin	Enables administrative operations through the MarkLogic REST API, such as creating an instance and managing instance configuration

# REST API Security - App Server Authentication

- App servers support multiple types of authentication.
- A REST instance is an HTTP app server configured with Digest authentication.

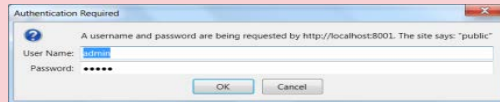
## BASIC

- Common
- Requires Login
- PW Masked
- Unencrypted



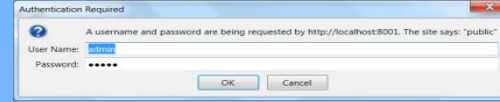
## DIGEST

- Requires Login
- Encrypted



## DIGEST-BASIC

- Digest First
- Basic Second
- In case user on old browser



## APP LEVEL

- None
- Authentication handled in App or will use default user

*We'll write our code to use  
Digest authentication against  
our REST instance*



# Creating Users

- Via the Admin interface (or a corresponding API):



The screenshot displays the MarkLogic Admin interface. On the left is a navigation tree under the 'Configure' tab, with 'Security' expanded to show 'Users' and 'NewUser'. The main panel has tabs for 'Summary', 'Create', and 'Help', with 'Create' selected. The 'New User' form contains a description 'user -- A database user.' and three input fields: 'user name' (required, unique), 'description' (optional), and 'password' (required, encrypted). 'ok' and 'cancel' buttons are in the top right.

Field	Description	Requirements
user name	User/login name (unique)	Required. You must supply a value for user-name.
description	An object's description.	
password	Encrypted Password.	Required.

# Creating Roles

- Via the Admin interface (or a corresponding API):



The screenshot displays the MarkLogic Admin interface. On the left is a navigation tree under the 'Configure' section, listing various system components like Groups, Databases, Hosts, Forests, Mimetypes, Clusters, and Security. Under 'Security', 'Users' and 'Roles' are visible, with 'NewRole' highlighted. The main panel on the right is titled 'New Role' and contains a 'Create' tab. It features a text area with the role description 'role -- A security role.' and two input fields: 'role name' and 'description'. The 'role name' field has a validation message: 'The Role name (unique) Required. You must supply a value for role-name.' The 'description' field has a placeholder text: 'An object's description.' At the top right of the dialog are 'ok' and 'cancel' buttons.

# Our Strategy for Training

- Implement Security early (rather than develop as admins and plug it in later).
- Utilize out of the box roles, but create project specific users

User Summary		
Summary	Create	Help
User	Description	Roles
admin	admin user	admin
healthcheck	Healthcheck application runner	healthcheck-user
infostudio-admin	Information Studio CPF pipeline and task runner	dls-user, dls-internal, infostudio-user, dls-admin, ...
nobody	nobody user	rest-reader, rest-extension-user, app-user
rest-admin-user	REST API admin user capable of managing configuration data	rest-reader, rest-extension-user, manage-user, rest-admin, ...
rest-reader-user	a user with rights to read documents via the REST API	rest-reader, rest-extension-user
rest-writer-user	a user with rights to read and insert/update documents via the REST API	rest-reader, rest-extension-user, rest-writer
samplestack-admin	user that can administer REST server	rest-reader, rest-extension-user, manage-user, rest-admin, ...
samplestack-contributor	user for write unrestricted access to samplestack data	rest-reader, rest-extension-user, samplestack-guest, samplestack-writer, ...
samplestack-guest	user for read-only, restricted access to samplestack data	rest-extension-user, samplestack-guest

# Document Permissions

- Document permissions authorize users with particular roles to access particular documents in queries.
- Permissions are assigned explicitly when loading documents.
- **Documents loaded with no document permissions set can only be accessed by users with the admin role.**
- **Documents loaded via REST API automatically have permissions granted to the appropriate REST roles.**
- 4 types of document permission capabilities:
  - Read
  - Update
  - Insert
  - Execute (for extension code in a modules database)

# Document Permissions

- Document permissions are metadata about the document and stored with the document in its respective database. Note the **Content Source** in the example.
- Roles and users are stored as documents in the **Security** database.



The screenshot shows the MarkLogic Query Console interface. At the top, there's a tab labeled "JS Query 1". Below it, the "Content Source" is set to "top-songs-content (top-songs-modules)" and the "Query Type" is "JavaScript". The query text is:

```
1 // show the permissions of a specific document
2 xdmp.documentGetPermissions("/songs/song2.json");
```

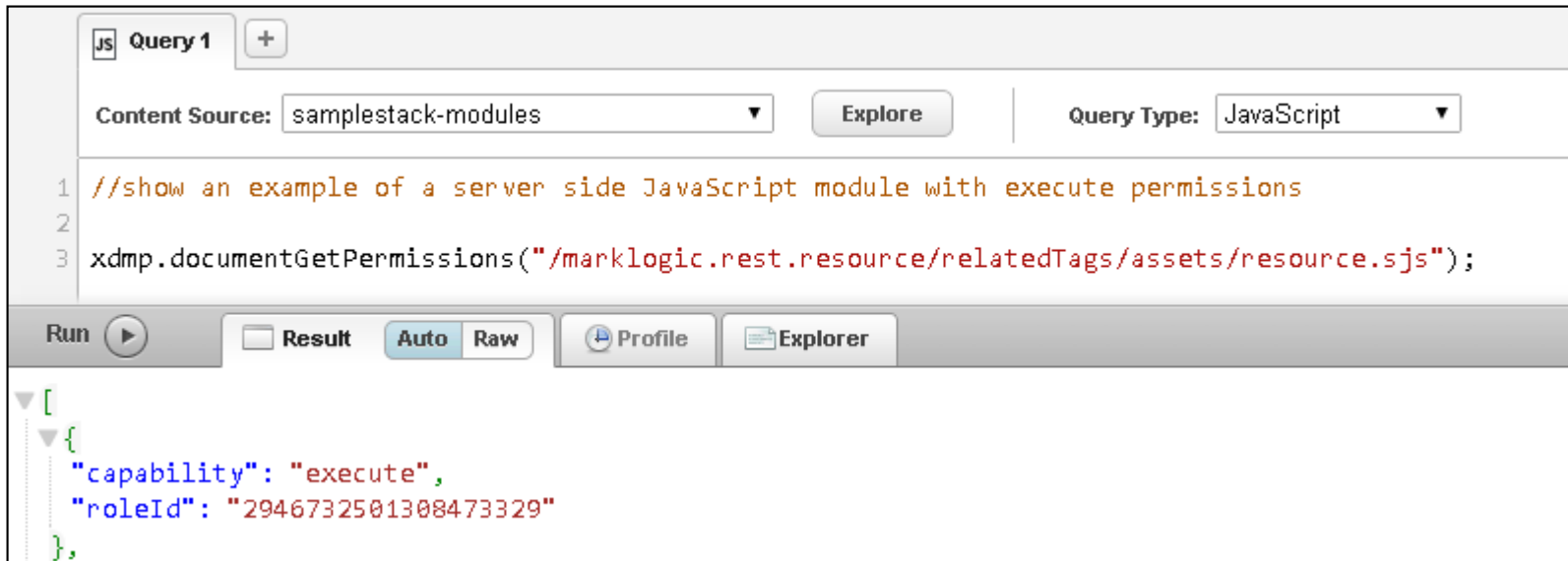
Below the query, there are buttons for "Run", "Result", "Auto", "Raw", "Profile", and "Explorer". The "Result" button is selected, and the output is displayed in a JSON format:

```
[
  {
    "capability": "read",
    "roleId": "7089338530631756591"
  },
  {
    "capability": "update",
    "roleId": "15520654661378671735"
  }
]
```

Handwritten blue arrows point from the JSON output to labels: "rest-reader role" points to the first object, and "rest-writer role" points to the second object.

# Execute Privileges

- Authorizes users to run specified pieces of code stored in a modules database.
- For the Node.js developer, execute privileges come into play if you write extensions or invoke server side JavaScript modules from Node.js.
- For example, Samplestack has several SJS extensions:



The screenshot displays the MarkLogic Query Editor interface. At the top, there's a tab labeled "JS Query 1" with a plus icon. Below the tab, the "Content Source" is set to "samplestack-modules" and the "Query Type" is set to "JavaScript". An "Explore" button is visible next to the content source. The main area contains a JavaScript code snippet:

```
1 //show an example of a server side JavaScript module with execute permissions
2
3 xdmp.documentGetPermissions("/marklogic.rest.resource/relatedTags/assets/resource.sjs");
```

Below the code editor, there's a "Run" button with a play icon. To the right of the "Run" button are tabs for "Result", "Auto", "Raw", "Profile", and "Explorer". The "Result" tab is currently selected, showing the output of the query:

```
[
  {
    "capability": "execute",
    "roleId": "2946732501308473329"
  },
]
```

# URI Privileges

- Authorizes users to create documents within a specific URI space.
- URI space is defined as a URI prefix. For example:
  - Users with role “X” can create URIs that begin with /songs/
  - Users with role “Y” can create URIs that begin with /movies/
- All roles must have some sort of URI privilege
  - The **any-uri** privilege will authorize a role to write with any URI

# Demo: MarkLogic Security in Samplestack

## Labs: Unit 4

Exercise 1: Create Users and Roles

Exercise 2: Test Document Permissions





# Unit Review Question 1:

Role and user data is stored in the:

1. Project database
2. Project modules database
3. Schemas database
4. Security database



# Unit Review Question 1:

Role and user data is stored in the:

1. Project database
2. Project modules database
3. Schemas database
4. **Security database**



## Unit Review Question 2:

Document permission data is stored in the:

1. Project database
2. Project modules database
3. Schemas database
4. Security database



## Unit Review Question 2:

Document permission data is stored in the:

1. **Project database**
2. Project modules database
3. Schemas database
4. Security database

## Unit Review Question 3:

A document is loaded into the database using the Node.js client API and no specific permissions were defined.

Who can read the document?

Update the document?

Delete the document?





## Unit Review Question 3:

A document is loaded into the database using the Node.js client API and no specific permissions were defined.

Who can read the document? Users with the rest-reader role

Update the document? Users with the rest-writer role

Delete the document? Users with the rest-writer-role



## Unit Review Question 4:

A copy of the Security database must be on each D Node host in the cluster.

1. True
2. False

## Unit Review Question 4:

A copy of the Security database must be on each D Node host in the cluster.

1. True
2. False

