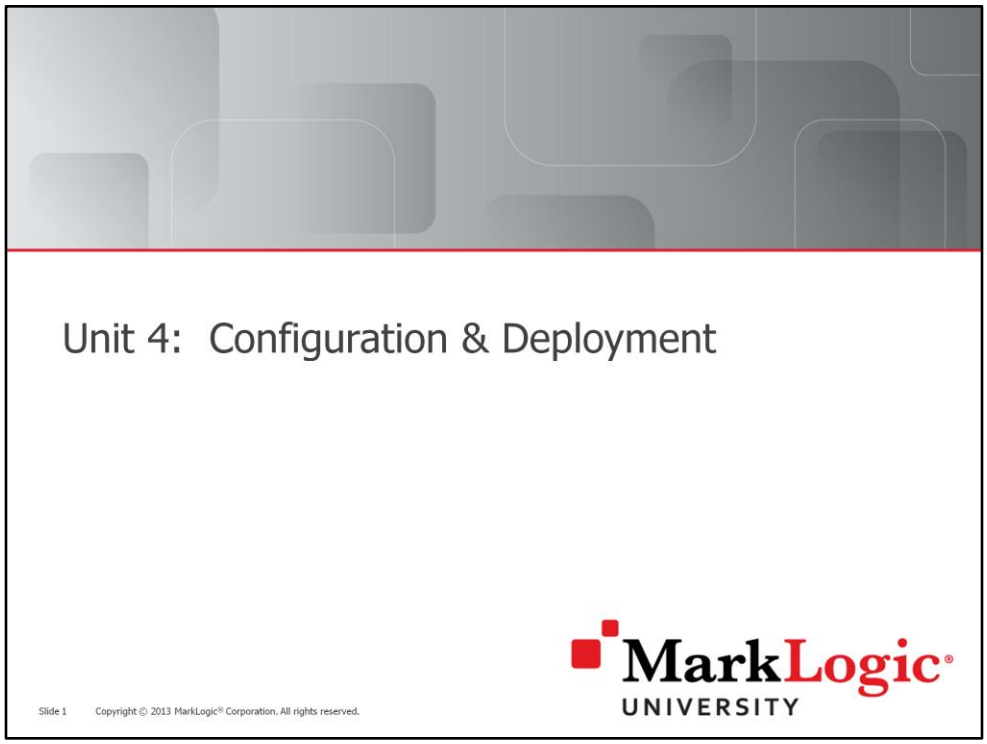
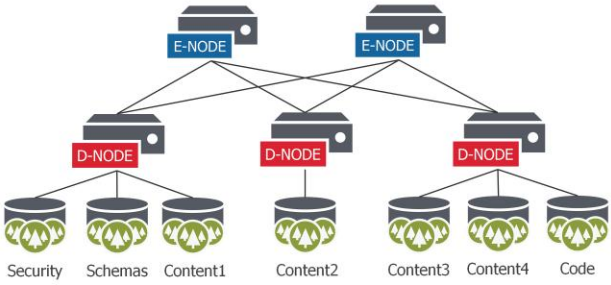


Learning Objectives

- Deploy and configure a cluster and an application using the management, packaging, and client REST APIs.
- Configure a database for tiered storage.
- Build partitions for tiered storage.
- Test tiered storage

Architecture Review

- How would you deploy an application across a cluster?

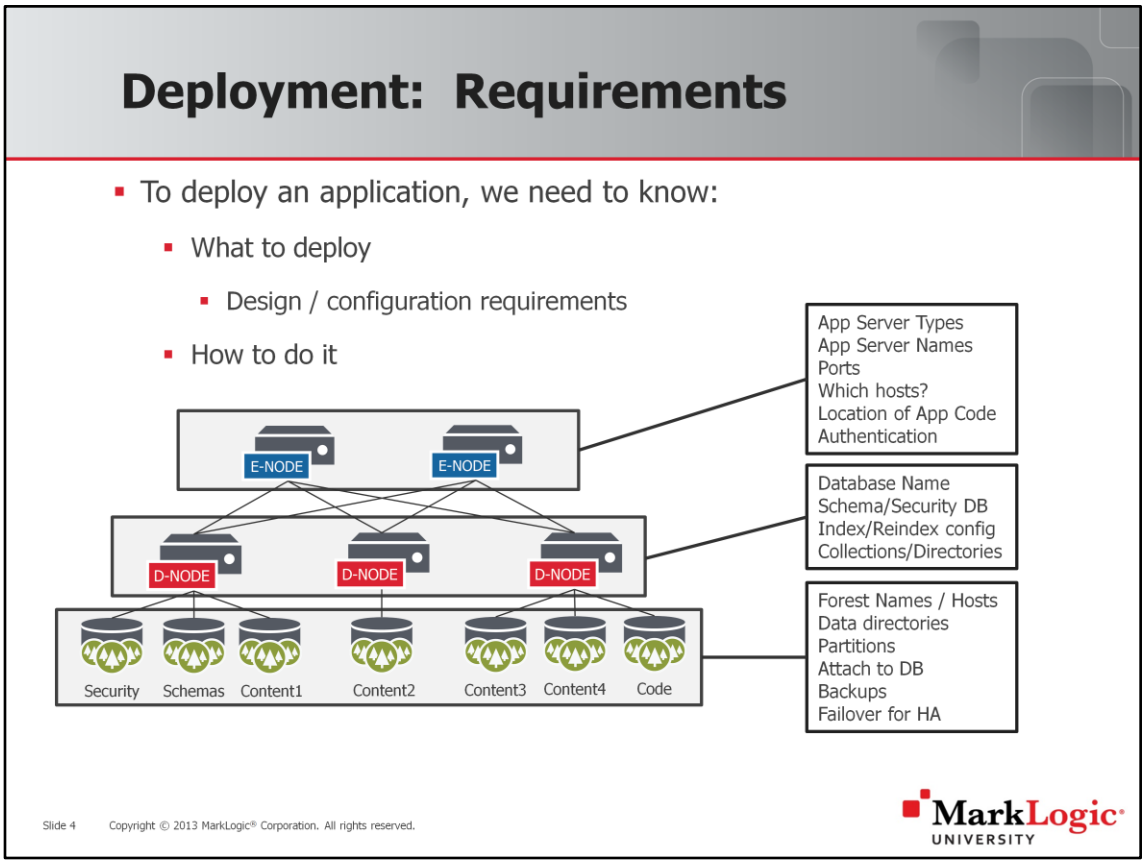


Slide 3

MarkLogic UNIVERSITY

Remember the architecture components we discussed earlier. Regardless of whether you are in a cluster of many hosts or a cluster of a single host, we need a method to deploy and manage application configuration. There are many ways to accomplish this task in MarkLogic, and we will dig into a few here such as:

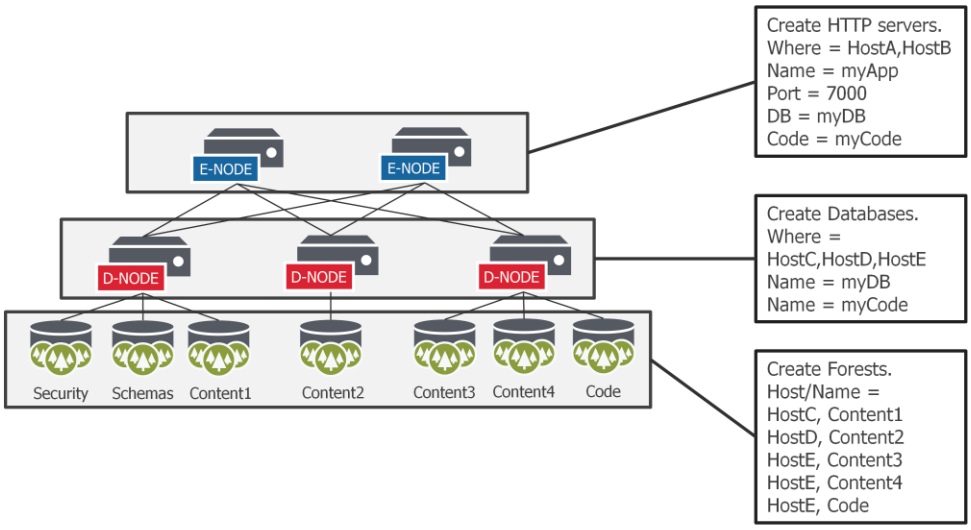
- The Admin interface
- REST
- Configuration Manager
- And later we will take a look at the Admin API



In order to build the appropriate environment for an application, we need to have specifications for how the app should be deployed. These are some examples of the information we need – but this is not an all inclusive list!

Deployment: Scenario

- Objective: Deploy the configuration outlined below



In order to build the appropriate environment for an application, we need to have specifications for how the app should be deployed. These are some examples of the information we need – but this is not an all inclusive list!

Deployment: Admin Interface

- Port 8001
- /MarkLogic Install Location/Admin/
- Perform Administrative Functions

Slide 6

MarkLogic UNIVERSITY

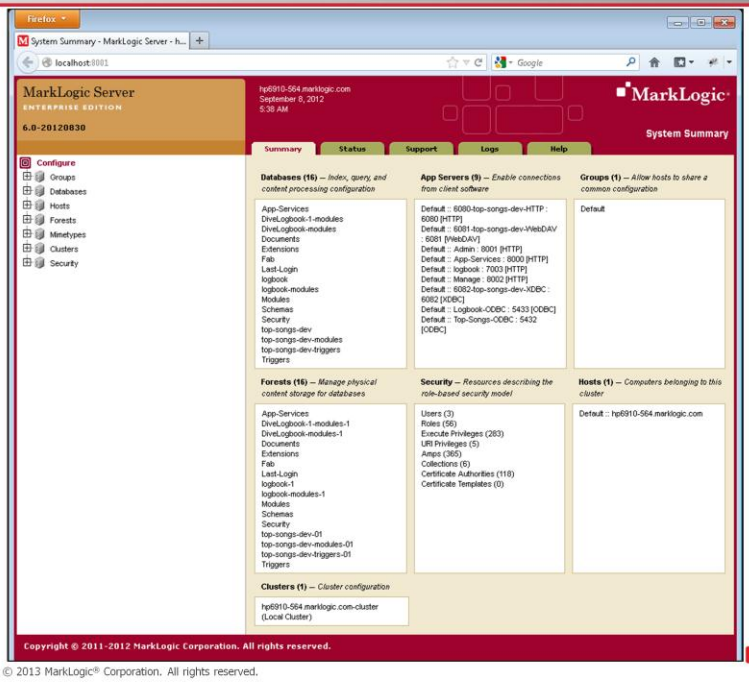
The MarkLogic Administration tool is a prebuilt XQuery application that is available on port 8001 after installation of the product. The codebase for this application can be found in the location where MarkLogic was installed, in the /admin/ folder.

The admin tool provides a UI for performing common tasks related to:

- Application Servers
- Databases and Forests
- Security settings
- Managing groups and clusters
- Monitoring the system

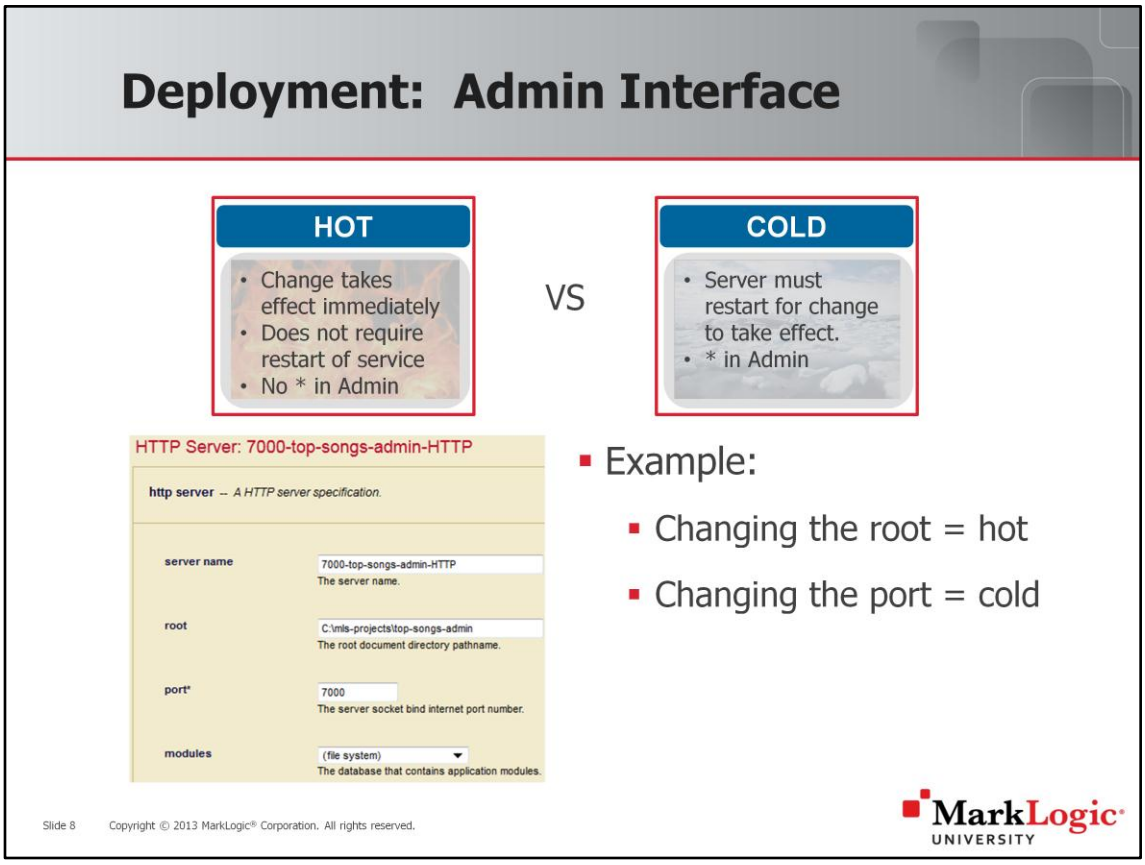
The core functions provided by the admin tool are also packaged as an API that can be used to script many of the tasks performed in the UI.

Deployment: Admin Interface



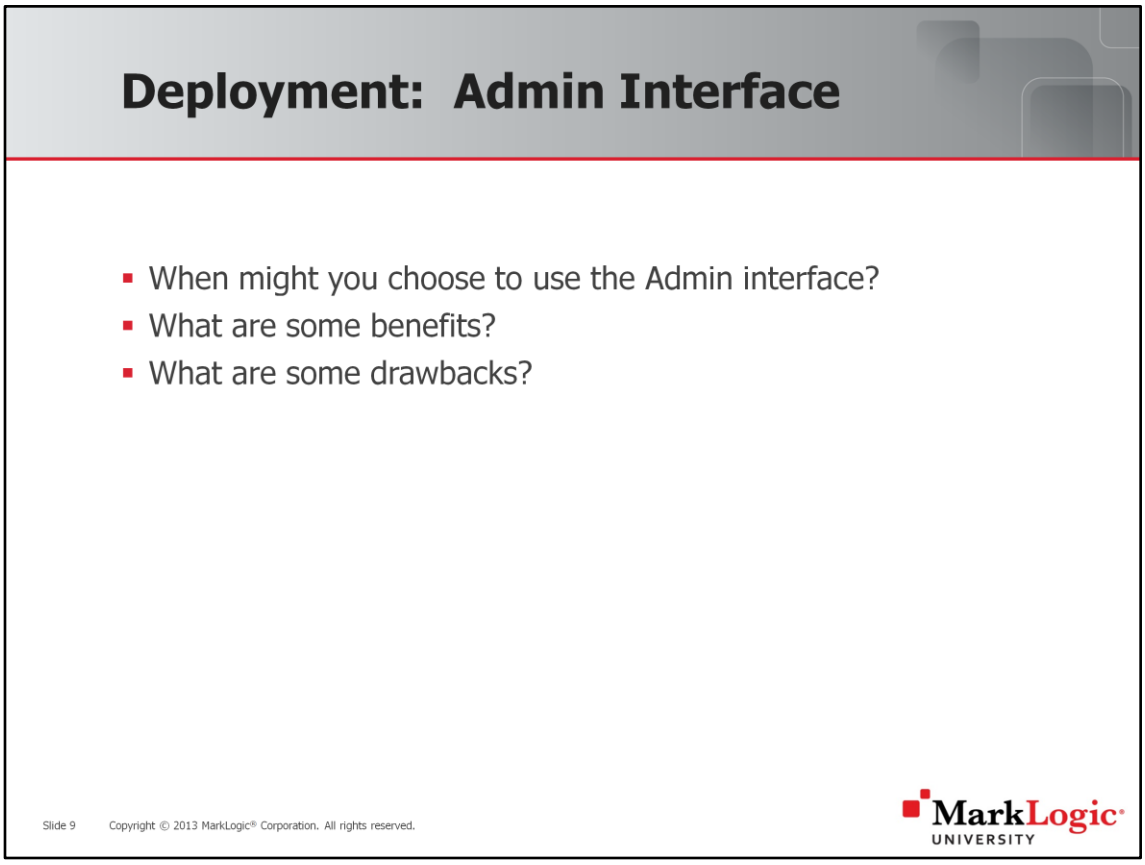
Note that the Admin Interface also has an underlying API that may also be used to script tasks using XQuery code. Since we have yet to discuss XQuery, we will not use the Admin API in this module (there will be an example later).

Instead we will focus on using REST and the Configuration Manager tool to accomplish our goals.

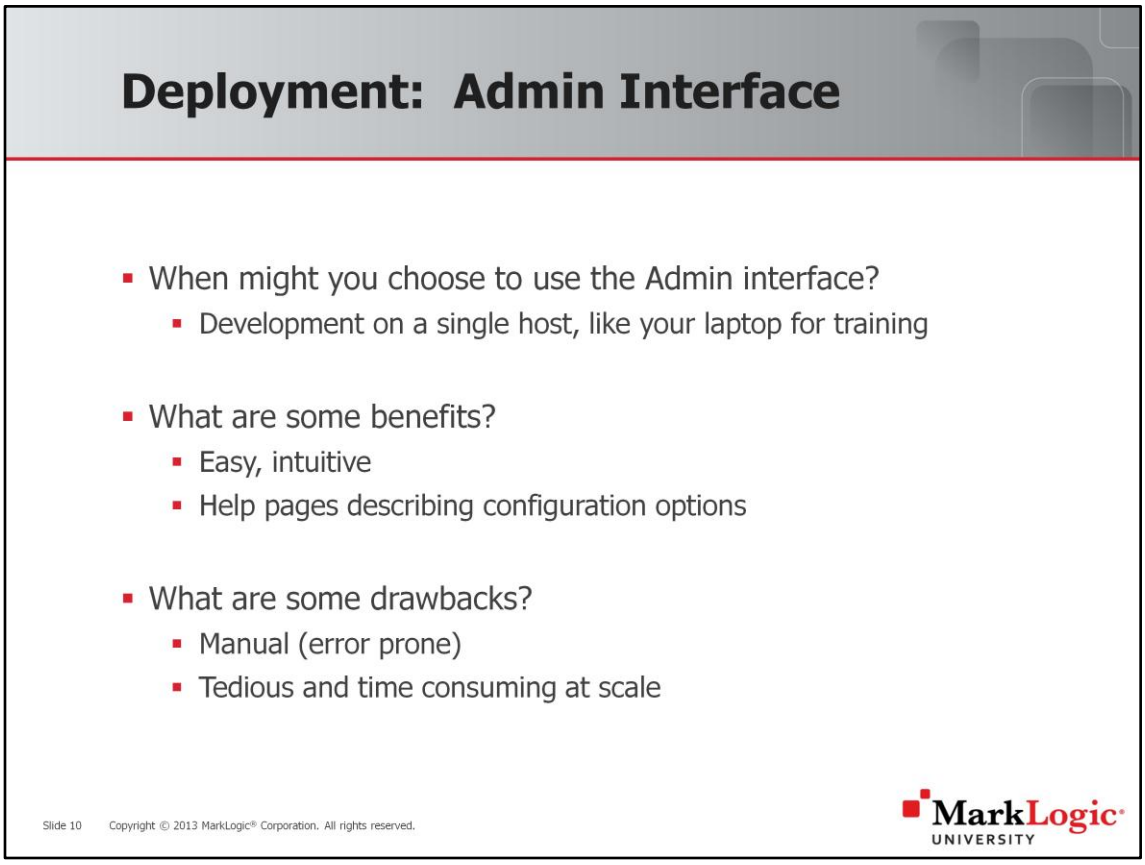


A configuration change is described as COLD if it requires a restart of the MarkLogic Server service. HOT configuration changes do not require a restart of the service.

In the Admin tool, settings with an asterisk (*) indicate a COLD configuration change.



The Admin interface is a valuable tool and should be used for appropriate scenarios. Think a little bit about when / where the Admin interface would be valuable versus where it may be less useful.



The Admin interface is a valuable tool and should be used for appropriate scenarios. Think a little bit about when / where the Admin interface would be valuable versus where it may be less useful.

REST Overview

REST: What, Why and When

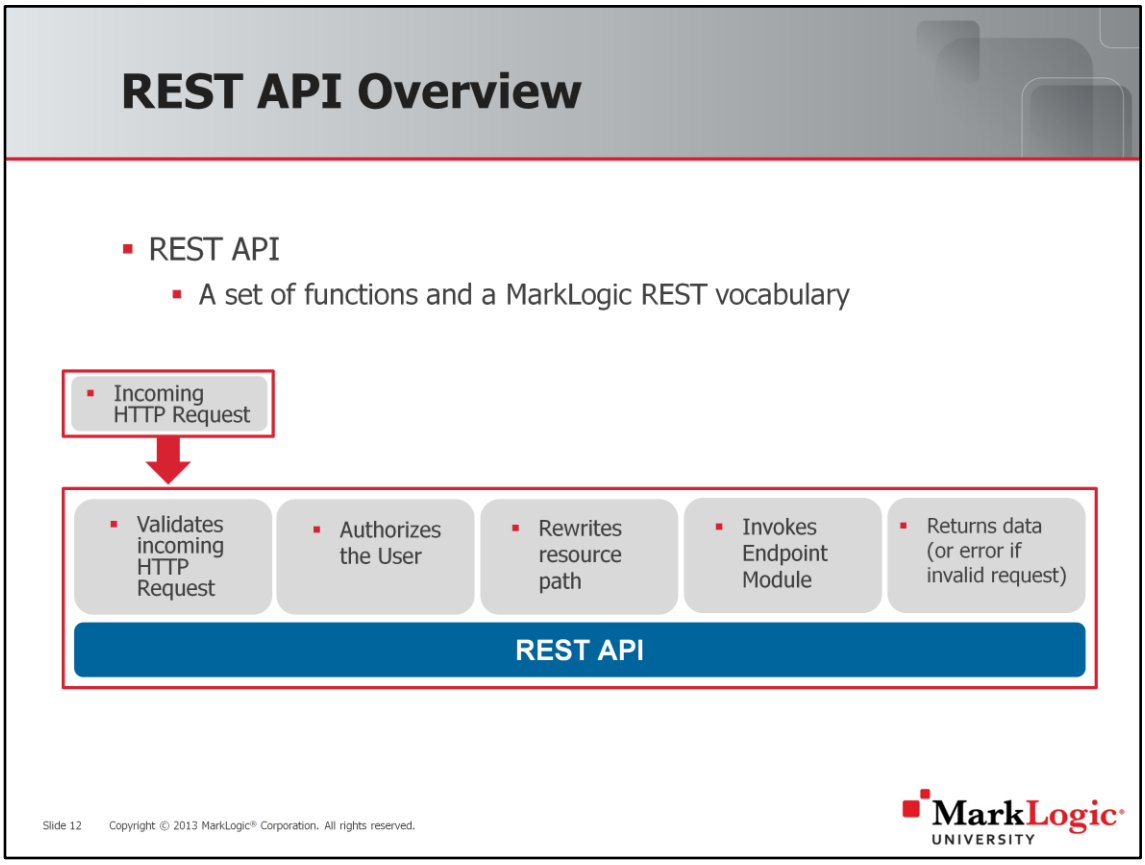
- REST
  - **RE**presentational **S**tate **T**ransfer
- REST API
  - A set of services for CRUD, search, and management of documents, metadata and resources in your MarkLogic environment.
- Why?
  - Take advantage of MarkLogic as your database using a variety of programming languages.

MarkLogic exposes the core functionality as a set of REST API services. What is REST? REST is a lot like the web in the sense that everything you access is a URL. You speak HTTP natively and make good use of HTTP verbs (like PUT or GET for example). Browsers like to say PUT something at this URL or GET me this URL.

By exposing online services and features as URLs, you can view, modify, maybe even delete (if you're on an authenticated client). With REST, every resource (like a document) is a first class entity with it's own URL target space that you can talk to.

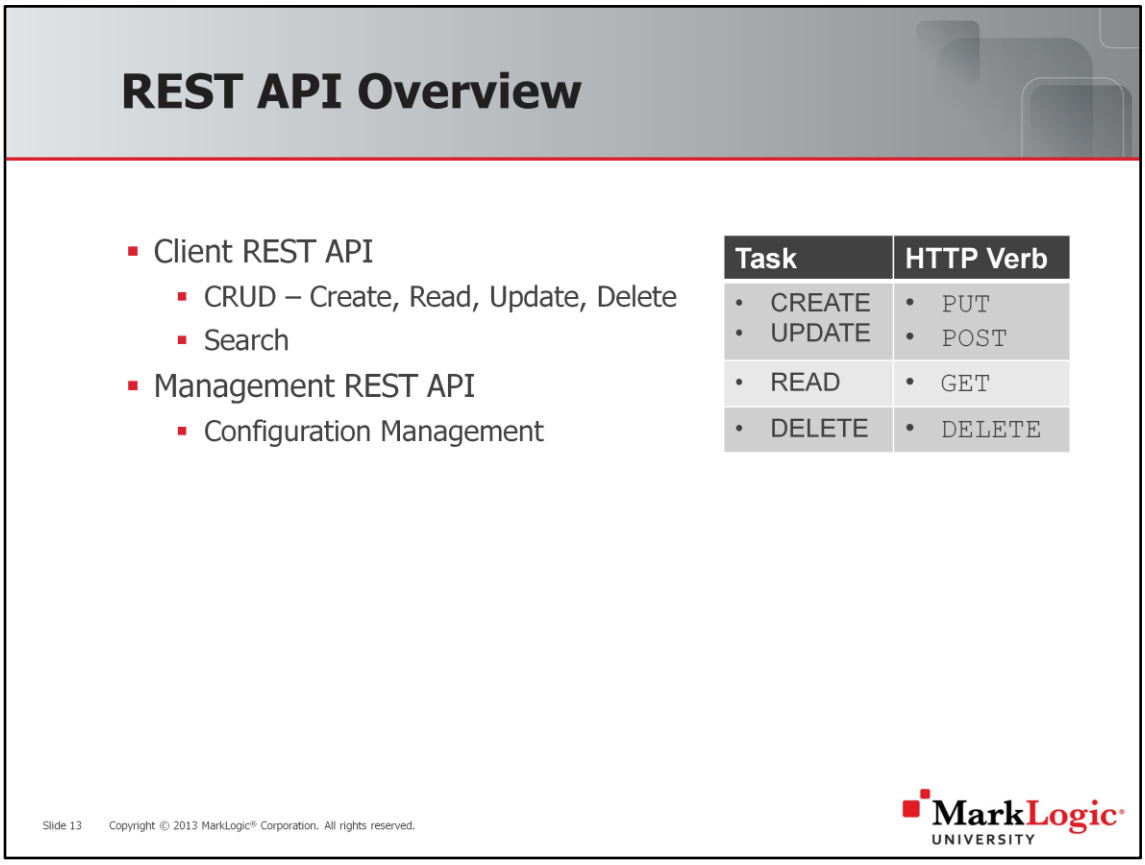
REST stands for Representational State Transfer. REST is an architectural style that, in the context of monitoring MarkLogic Server, describes the use of HTTP to make calls between a monitoring application and monitor host.

The REST API enables developers to easily integrate MarkLogic into existing enterprise IT environments through a common HTTP interface. This ensures that the REST API works well with load balancers, caching proxies and SSL.



The REST API contains a set of XQuery functions and a MarkLogic REST vocabulary of common operations that simplifies the task of describing web service endpoints. These descriptions include the mapping of URL parts to parameters and conditions that must be met in order for the incoming request to be mapped to an endpoint.

The REST API does the following:
- Validates the incoming HTTP request
- Authorizes the user
- Rewrites the resource path to one understood internally by the server before invoking the endpoint module
- If the request is valid, the endpoint module executes the requested operation and returns any data to the application.
- Otherwise, if the request is invalid, the endpoint module returns an error message

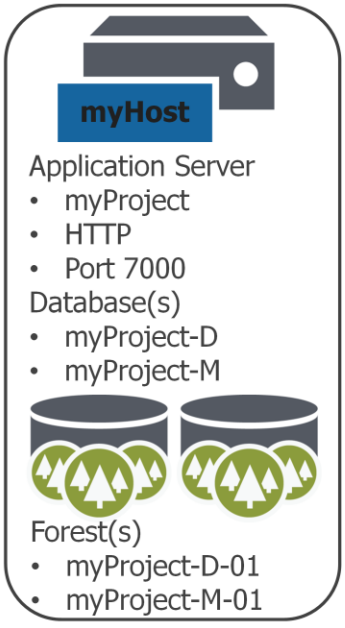


The REST API contains a set of XQuery functions and a MarkLogic REST vocabulary of common operations that simplifies the task of describing web service endpoints. These descriptions include the mapping of URL parts to parameters and conditions that must be met in order for the incoming request to be mapped to an endpoint.

The REST API does the following:

- Validates the incoming HTTP request
- Authorizes the user
- Rewrites the resource path to one understood internally by the server before invoking the endpoint module
- If the request is valid, the endpoint module executes the requested operation and returns any data to the application.
- Otherwise, if the request is invalid, the endpoint module returns an error message

Deployment: REST



- Assume a single host for first example
- Steps to perform:
  - Install MarkLogic
  - Start the service
  - License MarkLogic (REST)
    - Certain license types require manual acceptance of license terms
  - Create Admin account (REST)
    - Check server restart timestamp (REST)
  - Create databases (REST)
  - Create forests (REST)
  - Create application servers (REST)

Slide 14

In order to setup a project on an instance of MarkLogic we must perform several steps of configuration. The REST API enables us to do this from any host by simply building an HTTP request. In the next few slides we will look at how to use the REST API to accomplish this objective.

Note that order does matter. When creating the forests, you can attach them to the database at the same time. Therefore it makes sense to create the database first and then create the forests. Likewise, the application server needs to be configured to associate with a particular database, therefore it also makes sense to create the database before the application server.

REST API – License MarkLogic

- myLicense.xml:

```
<init xmlns="http://marklogic.com/manage">
    <license-key>key provided by MarkLogic</license-key>
    <licensee>licensee provided by MarkLogic</licensee>
</init>
```

- Invoking the REST API with cURL:

```
curl --anyauth --user user:password -X POST \
-d@"./myLicense.xml" -i -H "Content-type:application/xml" \
http://localhost:8001/admin/v1/init
```

- So what's going on inside this cURL statement?

REST API – License MarkLogic

- myLicense.xml:

```
<init xmlns="http://marklogic.com/manage">
    <license-key>key provided by MarkLogic</license-key>
    <licensee>licensee provided by MarkLogic</licensee>
</init>
```

- Invoking the REST API with cURL:

```
curl --anyauth --user user:password -X POST \
-d@"./myLicense.xml" -i -H "Content-type:application/xml" \
http://localhost:8001/admin/v1/init
```

- --anyauth
  - "Figure out the authentication method by yourself; use the most secure"
  - This is done by first sending a request and checking the response-headers, thus possibly inducing an extra network round-trip.
  - Eliminate this extra round trip by specifying **--digest** instead.

MarkLogic UNIVERSITY

REST API – License MarkLogic

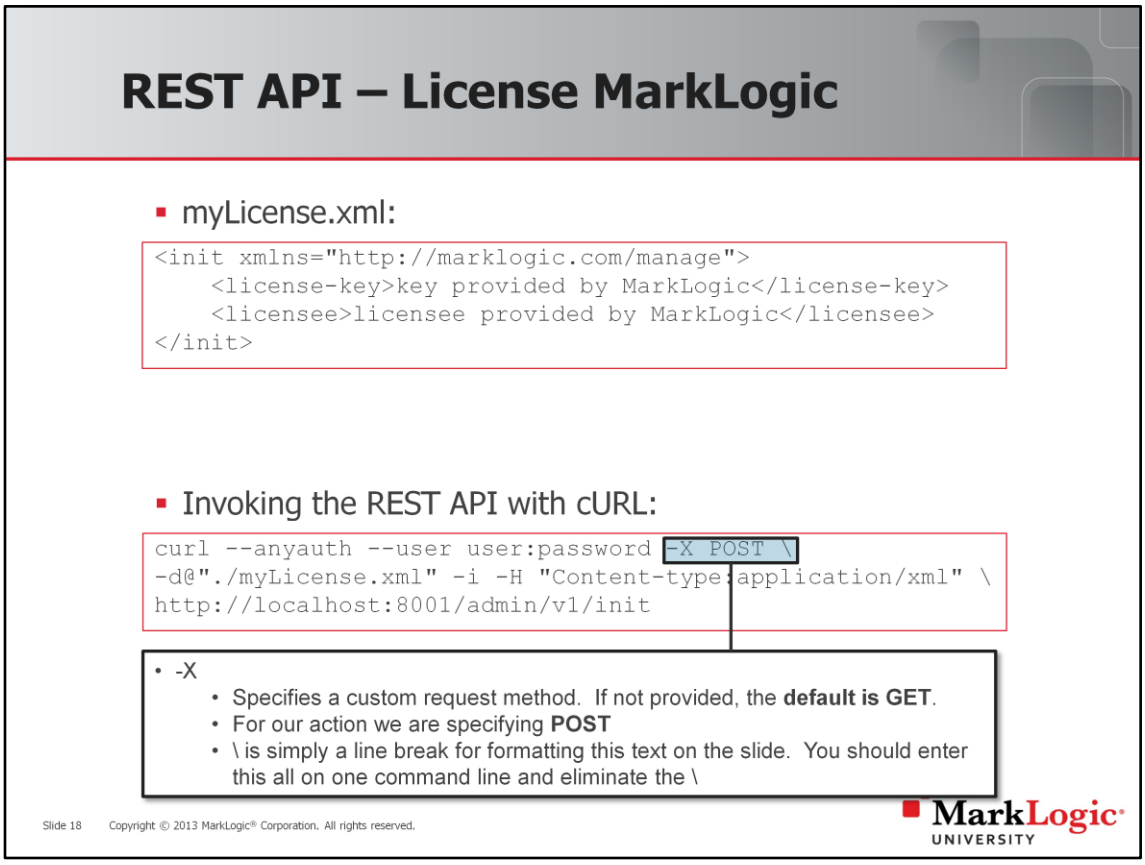
- myLicense.xml:

```
<init xmlns="http://marklogic.com/manage">
    <license-key>key provided by MarkLogic</license-key>
    <licensee>licensee provided by MarkLogic</licensee>
</init>
```

- Invoking the REST API with cURL:

```
curl --anyauth --user user:password -X POST \
-d@"./myLicense.xml" -i -H "Content-type:application/xml" \
http://localhost:8001/admin/v1/init
```

- --user
  - Specifies that authentication will be provided in the form of USERNAME:PASSWORD
  - Only applicable if updating the license (must be admin)
  - Initial license install does not need these because Security DB does not exist



- -d
  - (HTTP) Sends the specified data in a POST request to the HTTP server, in the same way that a browser does when a user has filled in an HTML form and presses the submit button. This will cause curl to pass the data to the server using the content-type application/x-www-form-urlencoded.
  - If you start the data with the letter @, the rest should be a file name to read the data from, or - if you want curl to read the data from stdin. The contents of the file must already be URL-encoded. Multiple files can also be specified. Posting data from a file named 'foobar' would thus be done with --data @foobar.
- -i
  - Include the HTTP-header in the output. The HTTP-header includes things like server-name, date of the document, HTTP-version and more.
- -H
  - Extra header to use when getting a web page.

REST API – License MarkLogic

- myLicense.xml:

```
<init xmlns="http://marklogic.com/manage">
    <license-key>key provided by MarkLogic</license-key>
    <licensee>licensee provided by MarkLogic</licensee>
</init>
```

- Invoking the REST API with cURL:

```
curl --anyauth --user user:password -X POST \
-d@"./myLicense.xml" -i -H "Content-type:application/xml" \
http://localhost:8001/admin/v1/init
```

- -d
  - Sends the specified data in a POST request to the HTTP server, in the same way that a browser does when a user has filled in an HTML form and presses the submit button.
  - @ is a reference to a file containing configuration info.

Slide 19

MarkLogic UNIVERSITY

- -i
  - Include the HTTP-header in the output. The HTTP-header includes things like server-name, date of the document, HTTP-version and more.
- -H
  - Extra header to use when getting a web page.

REST API – License MarkLogic

- myLicense.xml:

```
<init xmlns="http://marklogic.com/manage">
    <license-key>key provided by MarkLogic</license-key>
    <licensee>licensee provided by MarkLogic</licensee>
</init>
```

- Invoking the REST API with cURL:

```
curl --anyauth --user user:password -X POST \
-d@"./myLicense.xml" -i -H "Content-type:application/xml" \
http://localhost:8001/admin/v1/init
```

- -i
  - Include the HTTP-header in the output. The HTTP-header includes things like server-name, date of the document, HTTP-version and more.
  - Typically omitted during production scripting
- -H
  - Extra header we are going to send indicating content-type as XML

MarkLogic UNIVERSITY

REST API – License MarkLogic

- myLicense.xml:

```
<init xmlns="http://marklogic.com/manage">
    <license-key>key provided by MarkLogic</license-key>
    <licensee>licensee provided by MarkLogic</licensee>
</init>
```

- Invoking the REST API with cURL:

```
curl --anyauth --user user:password -X POST \
-d@"./myLicense.xml" -i -H "Content-type:application/xml" \
http://localhost:8001/admin/v1/init
```

- The host and endpoint that we are invoking to accomplish the task.

REST API – License MarkLogic (JSON)

- myLicense.json:

```
{
  "license-key" : "key provided by MarkLogic",
  "licensee" : "licensee provided by MarkLogic"
}
```

- Invoking the REST API with cURL:

```
curl --anyauth --user user:password -X POST \
-d@"./myLicense.json" -i -H "Content-type:application/json" \
http://localhost:8001/admin/v1/init
```

- If you prefer JSON, just change the header and configuration information format

REST API – Create Admin Account

- admin.xml:

```
<instance-admin xmlns="http://marklogic.com/manage">
   <admin-password>admin</admin-password>
   <admin-username>admin</admin-username>
   <realm>public</realm>
</instance-admin>
```

- admin.json:

```
{ "admin-username" : "admin",
  "admin-password" : "admin",
  "realm" : "public" }
```

- Invoking the REST API with cURL:

```
curl --anyauth --user user:password -X POST \
-d@"./admin.xml" -i -H "Content-type:application/xml" \
http://localhost:8001/admin/v1/instance-admin
```

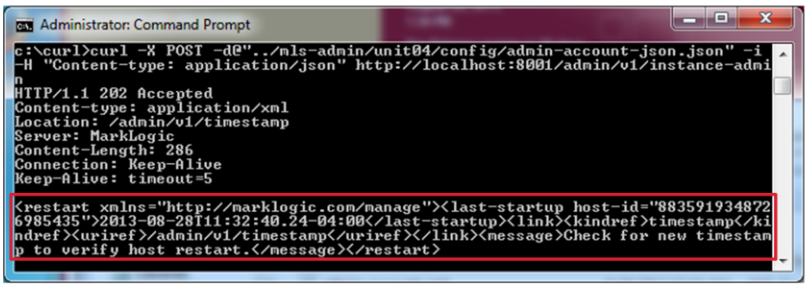
```
curl -i -X POST \
--data "admin-username=admin&admin-password=admin" \
http://localhost:8001/admin/v1/instance-admin
```

Slide 23

Note the final cURL example shows an alternate way to pass the configuration information using the cURL –data flag.

REST API – Check for Server Restart

- Note that requests to `/admin/` or `/manage/` REST API endpoints include a timestamp in the response
  - Timestamp = 2013-08-28T11:32:40.24-4:00
  - Indicates timestamp of the **last** server restart prior to request



REST API – Check for Server Restart

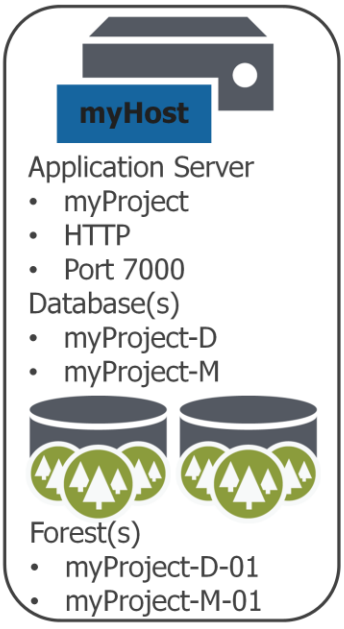
- Get current timestamp

```
curl --anyauth --user admin:admin -X GET -i \
http://localhost:8001/admin/v1/timestamp
```

- 200 OK = MarkLogic service is running on host
- String timestamp for comparison
  - Timestamp 1 = 2013-08-28T11:32:40.24-04:00
  - Timestamp 2 = 2013-08-28T11:37:07.329-04:00

MarkLogic UNIVERSITY

Creating a REST Instance - XML



- myconfig.xml:

```
<rest-api xmlns="http://marklogic.com/rest-api">
  <name>myProject</name>
  <group>Default</group>
  <database>myProject-D</database>
  <modules-database>myProject-M</modules-database>
  <port>7000</port>
</rest-api>
```

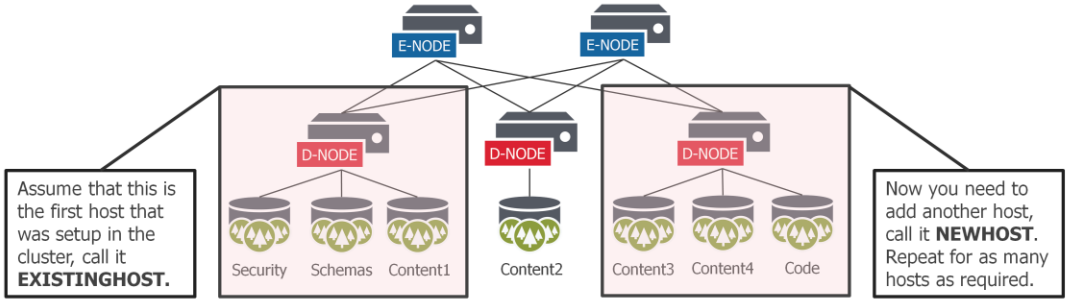
- Invoking the REST API with cURL:

```
curl --anyauth --user admin:admin -X POST \
-d@"./myconfig.xml" -i \
-H "Content-type:application/xml" \
http://localhost:8002/v1/rest-apis
```


You can use the REST API to deploy an HTTP Application Server (configured to use the REST API for your applications) as well as a database to contain your data and a second database to contain your code and REST API configurations.

Managing a Cluster

- Common Questions:
  - What about building out a cluster?
  - Managing the deployment across a cluster?
- Answer:
  - REST API and Configuration Manager



Adding a Host to a Cluster

```
# Assumption: MarkLogic is installed on NEWHOST and service is running
# Assumption: License is installed on NEWHOST (see earlier example)

# Get the new server's configuration and put it in a temp file
curl --silent http://NEWHOST:8001/admin/v1/server-config -o "/tmp/scfg.xml"

# Hand NEWHOST config to EXISTINGHOST and get the combined configuration
curl -i -X POST -H "Content-type: application/x-www-form-urlencoded"
--digest --user admin:admin --data-urlencode "server-config@/tmp/scfg.xml"
-d "group=Default" http://EXISTINGHOST:8001/admin/v1/cluster-config
-o "/tmp/cfg.zip"

# Check the timestamp of NEWHOST
TS=$(curl --silent http://NEWHOST:8001/admin/v1/timestamp)

# Now install the combined config on NEWHOST, which will restart
curl -X POST -H "Content-type: application/zip" --data-binary @/tmp/cfg.zip
http://HOSTB:8001/admin/v1/cluster-config

# Loop logic to compare timestamps to determine when restart is complete
# Clean up tmp files, respond, and exit
```

Adding a Host to a Cluster

```
# Assumption: MarkLogic is installed on NEWHOST and service is running
# Assumption: License is installed on NEWHOST (see earlier example)

# Get the new server's configuration and put it in a temp file
curl --silent http://NEWHOST:8001/admin/v1/server-config -o "/tmp/scfg.xml"

# Hand NEWHOST config to EXISTINGHOST and get the combined configuration
curl -i -X POST -H "Content-type: application/x-www-form-urlencoded"
--digest --user admin:admin --data-urlencode "server-config@/tmp/scfg.xml"
-d "group=Default" http://EXISTINGHOST:8001/admin/v1/cluster-config
-o "/tmp/cfg.zip"

# Check the timestamp of NEWHOST
TS=$(curl --silent http://NEWHOST:8001/admin/v1/timestamp)

# Now install the combined config on NEWHOST, which will restart
curl -X POST -H "Content-type: application/zip" --data-binary @/tmp/cfg.zip
http://HOSTB:8001/admin/v1/cluster-config

# Loop logic to compare timestamps to determine when restart is complete
# Clean up tmp files, respond, and exit
```

Adding a Host to a Cluster

```
# Assumption: MarkLogic is installed on NEWHOST and service is running
# Assumption: License is installed on NEWHOST (see earlier example)

# Get the new server's configuration and put it in a temp file
curl --silent http://NEWHOST:8001/admin/v1/server-config -o "/tmp/scfg.xml"

# Hand NEWHOST config to EXISTINGHOST and get the combined configuration
curl -i -X POST -H "Content-type: application/x-www-form-urlencoded"
--digest --user admin:admin --data-urlencode "server-config@/tmp/scfg.xml"
-d "group=Default" http://EXISTINGHOST:8001/admin/v1/cluster-config
-o "/tmp/cfg.zip"

# Check the timestamp of NEWHOST
TS=$(curl --silent http://NEWHOST:8001/admin/v1/timestamp)

# Now install the combined config on NEWHOST, which will restart
curl -X POST -H "Content-type: application/zip" --data-binary @/tmp/cfg.zip
http://HOSTB:8001/admin/v1/cluster-config

# Loop logic to compare timestamps to determine when restart is complete
# Clean up tmp files, respond, and exit
```

Adding a Host to a Cluster

```
# Assumption: MarkLogic is installed on NEWHOST and service is running
# Assumption: License is installed on NEWHOST (see earlier example)

# Get the new server's configuration and put it in a temp file
curl --silent http://NEWHOST:8001/admin/v1/server-config -o "/tmp/scfg.xml"

# Hand NEWHOST config to EXISTINGHOST and get the combined configuration
curl -i -X POST -H "Content-type: application/x-www-form-urlencoded"
--digest --user admin:admin --data-urlencode "server-config@/tmp/scfg.xml"
-d "group=Default" http://EXISTINGHOST:8001/admin/v1/cluster-config
-o "/tmp/cfg.zip"

# Check the timestamp of NEWHOST
TS=$(curl --silent http://NEWHOST:8001/admin/v1/timestamp)

# Now install the combined config on NEWHOST, which will restart
curl -X POST -H "Content-type: application/zip" --data-binary @/tmp/cfg.zip
http://HOSTB:8001/admin/v1/cluster-config

# Loop logic to compare timestamps to determine when restart is complete
# Clean up tmp files, respond, and exit
```

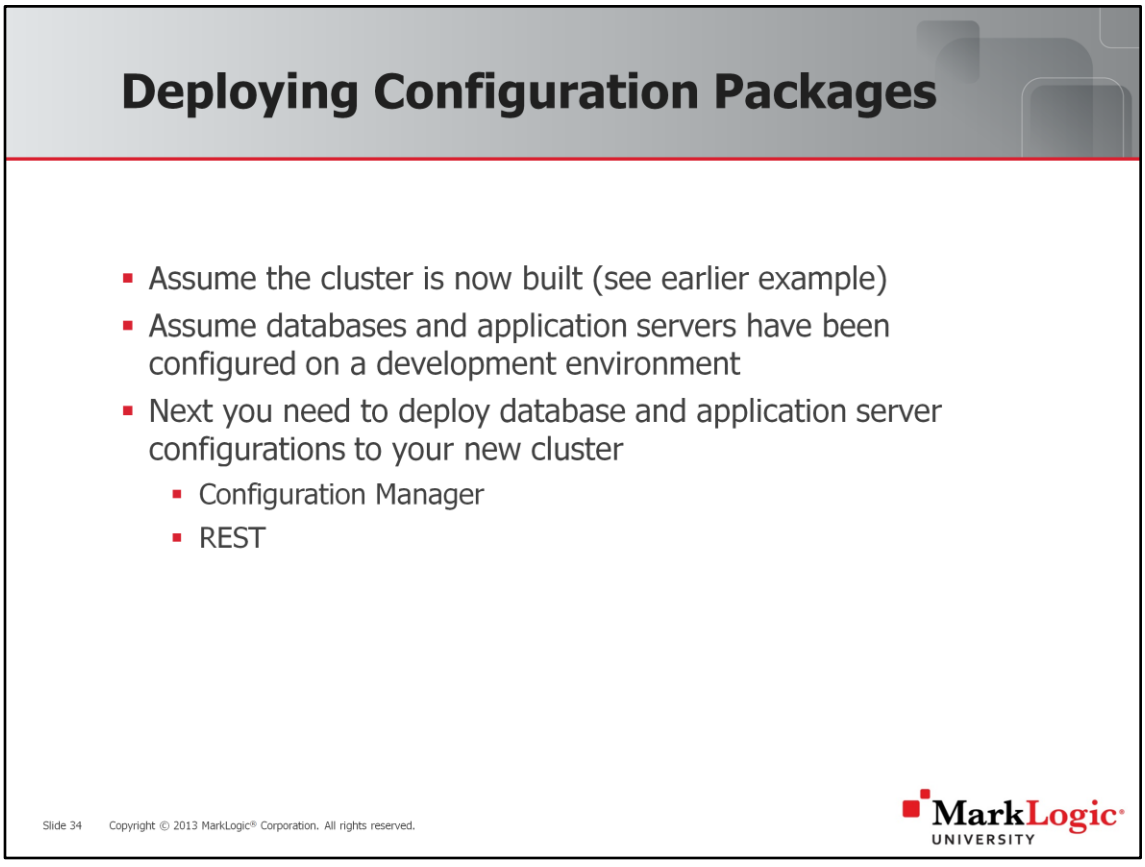
Adding a Host to a Cluster

```
# Assumption: MarkLogic is installed on NEWHOST and service is running
# Assumption: License is installed on NEWHOST (see earlier example)

# Get the new server's configuration and put it in a temp file
curl --silent http://NEWHOST:8001/admin/v1/server-config -o "/tmp/scfg.xml"

# Hand NEWHOST config to EXISTINGHOST and get the combined configuration
curl -i -X POST -H "Content-type: application/x-www-form-urlencoded"
--digest --user admin:admin --data-urlencode "server-config@/tmp/scfg.xml"
-d "group=Default" http://EXISTINGHOST:8001/admin/v1/cluster-config
-o "/tmp/cfg.zip"

# Check the timestamp of NEWHOST
TS=$(curl --silent http://NEWHOST:8001/admin/v1/timestamp)

# Now install the combined config on NEWHOST, which will restart
curl -X POST -H "Content-type: application/zip" --data-binary @/tmp/cfg.zip
http://NEWHOST:8001/admin/v1/cluster-config

# Loop logic to compare timestamps to determine when restart is complete
# Clean up tmp files, respond, and exit
```


MarkLogic UNIVERSITY

Adding a Host to a Cluster

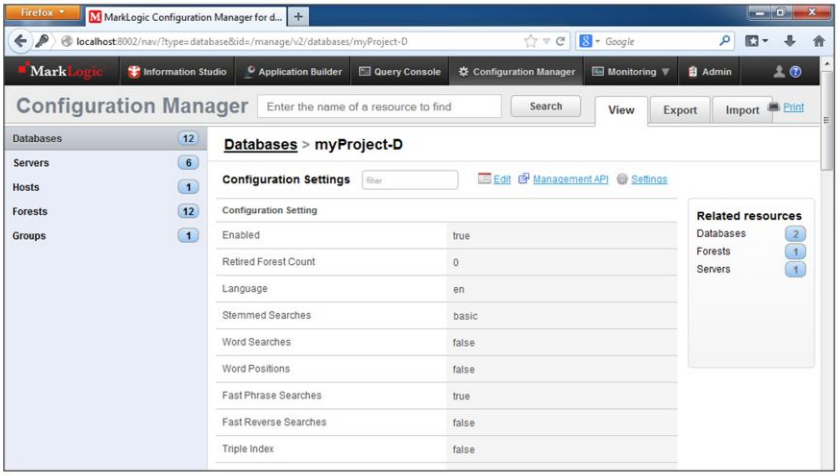
```
# Assumption: MarkLogic is installed on NEWHOST and service is running
# Assumption: License is installed on NEWHOST (see earlier example)

# Get the new server's configuration and put it in a temp file
curl --silent http://NEWHOST:8001/admin/v1/server-config -o "/tmp/scfg.xml"

# Hand NEWHOST config to EXISTINGHOST and get the combined configuration
curl -i -X POST -H "Content-type: application/x-www-form-urlencoded"
--digest --user admin:admin --data-urlencode "server-config@/tmp/scfg.xml"
-d "group=Default" http://EXISTINGHOST:8001/admin/v1/cluster-config
-o "/tmp/cfg.zip"

# Check the timestamp of NEWHOST
TS=$(curl --silent http://NEWHOST:8001/admin/v1/timestamp)

# Now install the combined config on NEWHOST, which will restart
curl -X POST -H "Content-type: application/zip" --data-binary @/tmp/cfg.zip
http://HOSTB:8001/admin/v1/cluster-config

# Loop logic to compare timestamps to determine when restart is complete
# Clean up tmp files, respond, and exit
```



So now lets assume that we have built our cluster. The next step is to deploy database and application server configurations across that cluster. Usually, these configurations already exist – most likely on a development environment. So there is no value in rebuilding these manually or even via scripts.

Rather, it makes more sense to package a configuration from your development environment and migrate it to your cluster. The Configuration Manager tool and its underlying REST API makes this easy to accomplish.

Configuration Manager



- Port 8002
- Read only view
- Export / Import
- Compare
- Underlying API

Configuration Manager

- Example of exported configuration data

```
  </package-database-properties>
  <links>
    <forests-list>
  <forest-name>top-songs-admin-01</forest-name>
    </forests-list>
    <security-database>Security</security-database>
    <schema-database>Schemas</schema-database>
    <triggers-database>top-songs-admin-triggers</triggers-database>
  </links>
</package-database>
<package-http-server xmlns="http://marklogic.com/package/servers">
  <metadata>
    <package-version>1.0</package-version>
    <user>admin</user>
    <group>Default</group>
    <host>hp8440-1501.marklogic.com</host>
    <timestamp>2012-02-03T14:02:59.917-08:00</timestamp>
    <platform>winnt</platform>
  </metadata>
  <server-type>http-server</server-type>
  <group-name>Default</group-name>
  <name>7000-top-songs-admin-HTTP</name>
  <package-http-properties>
    <enabled>true</enabled>
    <root>C:\mls-admin\Unit03\top-songs-admin</root>
    <port>7000</port>
```


Once you have exported configuration data, you can then import it on another host. While this could be accomplished using the GUI, more likely in a large cluster you would prefer to have scripts which automate this process. Next we will look at how to do the same thing via REST.

Configuration Manager & REST

- Goal:
  - Register a package and install it on a host
- Assumption:
  - Package has already been exported somewhere on the machine

```
# Register package
# Must be a user assigned the manage-admin role.
# MarkLogic admin role is not required to create a package

curl -i -X POST --digest --user admin:admin --header \
"Content-Type:application/zip" -T data/package1372355991796.zip \
http://localhost:8002/manage/v2/packages?pkgname=YOURPACKAGE
```

```
# Install package
# Requires admin role

curl -i -X POST --header "Content-type:application/x-www-form-urlencoded" \
--data "" --digest --user admin:admin \
http://localhost:8002/manage/v2/packages/YOURPACKAGE/install
```

Slide 37

MarkLogic UNIVERSITY

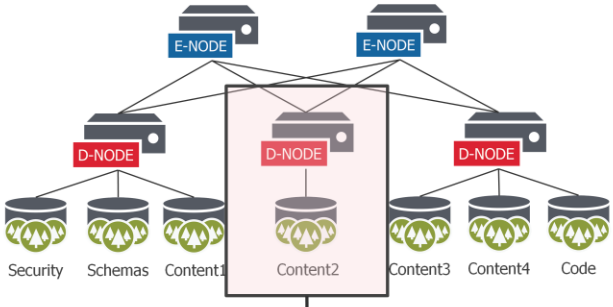
Top top example shows you how to register a package based on an already exported configuration (the zip file). It creates a package called “YOURPACKAGE”.

Now that the package has been registered, it can be installed if desired. Simply reference “YOURPACKAGE” when invoking the /install/ endpoint.

Note the security roles required. In order to register a package the user must belong to the manage-admin role. In order to install the package, the user must be a MarkLogic admin.

Managing a Cluster: Create a Forest

- Adding a new forest and attaching it to a database



Assume the highlighted host has additional capacity and your projects data needs are growing.

Lets add another forest to this host, assuming it is called **HostD**.

Managing a Cluster: Create a Forest

- forest-config.xml:

```
<forest-create xmlns="http://marklogic.com/manage">
    <forest-name>myNewForest</forest-name>
    <host>HostD</host>
    <database>myDatabaseName</database>
</forest-create>
```

- Invoking the REST API with cURL:

```
curl -i -X POST --digest --user admin:admin \
--header "Content-Type:application/xml" \
-d@"./forest-config.xml" \
http://localhost:8002/manage/v2/forests
```

- What is going to happen after the forest is added?

Slide 39
MarkLogic UNIVERSITY

We can scale out by adding a new forest on an existing host (or of course adding a new host + new forests). What is going to happen to our data when we add the new forest?

MarkLogic will rebalance the data across the database according to assignment policy defined on the database. The default assignment policy is called “bucket”.

The bucket policy uses the URI of a document to decide which forest the document should be assigned to. The URI is first "mapped" to a bucket then the bucket is "mapped" to a forest. The mapping from a bucket to a forest is kept in memory for fast access.

Unit 4: Applying the Learning Objectives

- Deploy and configure a cluster and an application using the management, packaging, and client REST APIs.
  - Exercise 1 | Exercise 2 | Exercise 3 | Exercise 4 | Exercise 5
- Configure a database for tiered storage.
  - Exercise 6 | Exercise 7
- Build partitions for tiered storage.
  - Exercise 8
- Test tiered storage
  - Exercise 9