DuraCloud Release 0.8

The following documentation pertains to DuraCloud Release 0.8. Documentation for the latest DuraCloud release can be found here

Introduction

DuraCloud is a hosted service and open technology developed by DuraSpace that makes it easy for organizations and end users to use cloud services. DuraCloud leverages existing cloud infrastructure to enable durability and access to digital content.

DuraCloud is particularly focused on providing preservation support services and access services for academic libraries, academic research centers, and other cultural heritage organizations.

The service builds on the pure storage from expert storage providers by overlaying the access functionality and preservation support tools that are essential to ensuring long-term access and durability. DuraCloud offers cloud storage across multiple commercial and non commercial providers, and offers compute services that are key to unlocking the value of digital content stored in the cloud. DuraCloud provides services that enable digital preservation, data access, transformation, and data sharing. Customers are offered "elastic capacity" coupled with a "pay as you go" approach. DuraCloud is appropriate for individuals, single institutions, or for multiple organizations that want to use cross-institutional infrastructure.

DuraCloud has been in a pilot phase since the beginning of Fall 2009 and will be released as a service of the DuraSpace not-for-profit organization in early 2011.

To see press releases, presentations, logos, and more, see the DuraCloud resources page

User Documentation

- Release Notes
- DuraCloud Features
- DuraCloud Services
- DuraCloud REST API
- DuraCloud Java Clients
- DuraCloud Sync Tool
- DuraCloud Retrieval Tool
- DuraCloud Chunker Tool
- DuraCloud Administration

Design Documentation

- DuraCloud Architecture
- DuraCloud Security
- DuraCloud Service Designs

Developer Documentation

- Building DuraCloud from Source
- Service Development Guide
- Logging Configuration

Source Code

- The DuraCloud 0.8 release (latest release)
  - Source can be download from Subversion: https://svn.duraspace.org/duracloud/tags/duracloud-0.8.0
  - Source can be viewed here: https://svn.duraspace.org/view/duracloud/tags/duracloud-0.8.0
  - JavaDocs for can be viewed here: http://duracloud.org/0.8.0/docs/javadoc
- The DuraCloud trunk (ongoing development)
  - Can be download from Subversion: https://svn.duraspace.org/duracloud/trunk/
  - Can be viewed here: https://svn.duraspace.org/view/duracloud/trunk/
- The DuraCloud software license can be seen here: http://duracloud.org/license

- We welcome code contributions, though we do request that all contributors submit a Contributor License Agreement

Accessing DuraCloud

1. If you are part of the DuraCloud pilot program, you will be notified when your DuraCloud instance has been provisioned and is ready to use.
    - Your instance will be available at a URL similar to the following:
        - https://[your-institution].duracloud.org/duradmin
        - https://[your-institution].duracloud.org/durastore
2. If you are not part of the DuraCloud pilot program, you will can either
    a. Wait until DuraCloud is made publicly available, which is planned for early 2011
    b. Download a copy of the source code and deploy your own DuraCloud instance

Support

- Bugs and feature requests
    - Please head over to our JIRA issue tracker and let us know about issues that you encounter.
- Discussion
    - Head over to the DuraCloud forum to discuss DuraCloud ideas and technology

Resources

- Presentations

DuraCloud Chunker Tool

Introduction

The Chunker Tool is a utility which was created in order to provide a simple way to copy files from a local file system to DuraCloud in a "one-off" manner. Actually, although the common case is to use this tool to copy one or more files to DuraCloud, it may also be run to copy files to another location on the local file system.

Download

Download the Chunker Tool here.

Operational notes

- If you want to jump directly into using the tool, download it from the link above and run the following command

    ```
    java -jar chunk-{version}-driver.jar
    ```

    The resulting usage statement (detailed below) should be enough to help you get started.

- The Chunker Tool allows you to copy multiple local files and directories into a single space within DuraCloud. The names of the objects which are added to DuraCloud will contain all of the directory elements in the path starting from the first element below the base directory down to the individual file names.

Using the Chunker Tool

- To run the Chunker Tool, open a terminal or command prompt and navigate to the directory where the Chunker Tool is located and run the above command.
- The following options are available when running the Chunker Tool

Short Option	Long Option	Arguments	Description

-a	--add	<f t s>	add content from directory:<f> to space or directory:<t> of maximum chunk size:<s>, where the chunk size must have a unit suffix of K,M, or G — If the -c option is provided, the destination space <t> will be interpreted as the name of a space in the DuraCloud account found at the host:port provided in the -c option, otherwise the destination space will be interpreted as a directory on the local file system.
-c	--cloud-store	<host:port>	use cloud store found at <host>:<port> as content destination
-d	--dir-filter	<l>	limit processed directories to those listed in file-list:<l> — If the -d option is not used, all directories under the base source directory provided in the -a option will be included. The file specified by this option is expected to contain a list of directory names each on there own line. The list is converted to an OrFileFilter from Apache Commons IO
-f	--file-filter	<l>	limit processed files to those listed in file-list:<l> – The file specified by this option is expected to contain a list of file names each on there own line. The list is converted to an OrFileFilter from Apache Commons IO
-g	--generate	<outFile numBytes>	generate test data to <outFile> of <size> bytes — This option does not copy any files, it only generates test data files of the size specified in the give argument.
-i	--ignore-large-files	no args	if this option is set, files over the chunk size specified in the 'add' option will be ignored.
-p	--password	<password>	password of duracloud instance
-u	--username	<username>	username of duracloud instance
-x	--exclude-chunk-md5s	no args	if this option is set, chunk MD5s will NOT be preserved in the manifest — It is expected that this option is rarely used, but in certain situations where the MD5s of the segments of a file that needed to be chunked because the parent file was larger than the limit set in the -a option, not generating these MD5s improves performance.

Creating your own Chunks

If you are interested in creating chunked files in DuraCloud using your own tools, you may do so by adhering to the XML schema used by DuraCloud to create chunks.

Download the Chunker XSD here

Software

Building DuraCloud Software from Source

- OSGi container

Introduction

DuraCloud application software is composed of many parts. A breakdown of the primary pieces is as follows:

- DuraStore - this web application provides the access to and management of storage resources, which includes handling the storage portion of the DuraCloud REST API
- StorageProviders - this set is made up of the StorageProvider interfaces and the implementations which connect to distinct cloud stores (currently Amazon S3, Rackspace CloudFiles, and EMC Atmos)
- DuraService - this web application handles the deployment and management of services within DuraCloud, which includes handling the services portion of the DuraCloud REST API
- Services - the set of all deployable services, as well as the support projects that allow the DuraCloud services infrastructure to function
- ComputeProviders - this set is made up of the ComputeProvider interfaces and the implementation which connect to distinct cloud compute services (currently Amazon EC2, using the typica library)
- Security - handles security for the DuraCloud applications
- Common - a set of projects which provide utilities for other portions of the codebase to reuse

The DuraCloud software, by its very nature, is designed to be integrated with underlying cloud storage and compute providers.
As may be expected, these integrations are exercised through many "integration tests". In order for DuraCloud to connect to these underlying providers, appropriate credentials must first be established and available to the build framework.
However, in order to get up and running quickly with the software, the baseline can be initially built with just its unit tests.
Once the developer has independently acquired credentials for one or more of the supported underlying storage providers (below) builds may then be run with integration tests.

- Amazon S3
- Rackspace CloudFiles
- Windows Azure

This guide lays out the three tiers of building/testing the baseline:

1. build with unit tests
2. build with storage and services integration tests
3. build with OSGi services container integration tests

Although this document is written from a Linux environment perspective, analogous builds/installations have been tested in Windows (but may have limitations, as noted below). Any comments or feedback are welcomed.

Prerequisites

Software that must be installed on your system prior to building/using DuraCloud

1. Maven 2.2.1 or above
2. Tomcat 6.x or above
3. Java 6 (note: the djatoka service has compatibility issues with open-jdk)
4. Subversion

Building DuraCloud

Any portions of the configuration below for which you need to include a replacement value will be written in all capital letters and included in brackets: [LIKE-THIS]

Build with unit tests

1. Check out latest stable release from Subversion repository

```
svn co https://svn.duraspace.org/duracloud/tags/duracloud-0.8.0
```

2. Set environment variables

```
export JAVA_OPTS="-XX:MaxPermSize=256m"
export MAVEN_OPTS="-Xmx1024m -XX:MaxPermSize=1024m"
```

3. Configure Tomcat
    a. Add to $CATALINA_HOME/conf/tomcat-users.xml

```
<tomcat-users>
  <role rolename="manager"/>
  <role rolename="admin"/>
  <user username="[ANY-USERNAME]" password="[ANY-PASSWORD]" roles="admin,manager"/>
</tomcat-users>
```

4. Start tomcat

```
$CATALINA_HOME/bin/startup.sh
```

5. Configure Maven2
    a. Add tomcat user to $M2_HOME/conf/settings.xml

```
<servers>
  <server>
    <id>tomcat-server</id>
    <username>[ANY-USERNAME]</username>
    <password>[ANY-PASSWORD]</password>
  </server>
</servers>
```

6. Build with only unit tests
    a. From top of source tree

```
mvn clean install -PskipIntTests
```

Note: Each of the maven2 project modules are configured to halt the build if there is a test failure. The only exception to this is the *integration-test* module which is configured to run through all of its tests regardless of failures. So the above build command can optionally exclude the '-PskipIntTests' flag if desired.

Building in Windows: Currently the service projects which deploy as WAR files (beginning with WebAppUtilService) do not pass unit tests on a Windows environment due to limitations regarding the deployment of Tomcat instances. You may choose to either perform a build skipping tests "mvn clean install -DskipTests" or to allow the tests to run but not stop the build "mvn clean install -Dmaven.test.failure.ignore=true"

Build with integration tests

This step assumes the successful completion of the previous build instructions.
As mentioned above, the integration tests require credentials in order to connect to the underlying storage providers. The tests themselves are configured to acquire these credentials from a locally created, encrypted database.

1. Create unit-test-db
    - From inside the //unit-test-db module, run:

```
mvn assembly:assembly
java -jar target/unit-test-db-[VERSION]-db-util.jar
```

2. Running the above db util jar will provide a commandline interface for adding credentials
3. Add credentials for s3, emc, rackspace, root-user
    a. Set the root-user to username: 'root', password: 'rpw' to just use the application default.
    b. See *root user* discussion below for ways of changing the default.
4. Add the connection details for the unit-test-db to $M2_HOME/conf/settings.xml

```
<profile>
  <id>always</id>
  <properties>
    <duracloud.home>[LOCATION-WHERE-APPLICATION-HAS-WRITE-ACCESS]</duracloud.home>

<unit.database.home.default>[LOCATION-WHERE-UNIT-TEST-DB-WAS-CREATED]</unit.database.home.default>
<unit.database.password.default>[UNIT-TEST-DB-BOOT-PASSWORD]</unit.database.password.default>
  </properties>
</profile>

<activeProfiles>
  <activeProfile>always</activeProfile>
</activeProfiles>
```

- where duracloud.home is the directory under which the logs and osgi-container will be placed
- where unit.database.home.default is the location of the unit-test-database created above
- where unit.database.password.default is the boot password used during creation of the unit-test-database above

5. The integration tests also expect a registry (read: 'space') of services to be available within the primary storage provider of service storage host
    a. The naming of the space follows the convention "duracloud-<version>-service-repo"
    b. The services to be loaded into the registry can be found in the target directories of each of the service projects
    c. See discussion below about application configuration, including the service storage host
6. Build with unit and integration tests
    a. From top of source tree

```
mvn clean install
```

7. As mentioned before, if there are any failures in the //integration-test module, the build will still complete, but the failures will be listed

Build with OSGi services container integration tests

This step assumes the successful completion of the previous build instructions

Linux/Mac

1. Start OSGi service container

```
cd //services/servicesadmin
mvn clean -f pom-run.xml pax:provision
cd runner
chmod +x run.sh
export BUNDLE_HOME=[DURACLOUD_HOME]/osgi-container
./run.sh
```

    a. Where [DURACLOUD_HOME] is a directory where the application has write access (can be same as <duracloud.home> set in Maven settings.xml above)
    b. The run.sh script will start an OSGi container and commandline interface to it
    c. The container starts with required bundles including the 'services-admin' installed
    d. See discussion below on OSGi container for more details
2. Once the 'services-admin' is running, tests that deploy services into the OSGi environment may be run
3. From inside the //integration-test module

```
mvn install -PrunServicesAdminTests
```

Windows

1. Set up OSGi service container

```
cd services/servicesadmin
mvn clean -f pom-run.xml pax:provision
cd runner
```

2. (Optional) Set the OSGi bundle storage location

```
set BUNDLE_HOME=[BUNDLE_HOME]
```

   a. Where [BUNDLE_HOME] is the full path to an empty directory where the osgi container content will be stored
   b. Open the run.bat file in the runner directory in a text editor and replace all instances of "$BUNDLE_HOME" with "%BUNDLE_HOME%"
   c. Note: A directory called "$BUNDLE_HOME" under the runner directory will be used as the default bundle home if one is not specified.

3. (Optional) Set up logging
   a. Download this logback.xml file file into your bundle home directory.
   b. Open the logback.xml file in a text editor and edit the LOG_FILENAME property to point to a full file path (including file name) for a log file.
   c. Note: One benefit to performing this step will be faster start time for your OSGi container.
4. Start OSGI service container

```
run.bat
```

   a. The run.bat script will start an OSGi container and commandline interface to it
   b. The container starts with required bundles including the 'services-admin' installed
   c. See discussion below on OSGi container for more details

5. Once the 'services-admin' is running, check to ensure that it was created properly
   a. In the console where run.bat was executed, an "osgi" prompt should be available. If it is not available, hitting enter should bring it up.
   b. Type "ss" and hit enter. This should list all of the available bundles. This list should include 50 items, all of which are either in the ACTIVE or RESOLVED state.

Optional items

Code coverage

1. If you plan on using Clover, the following element needs to be added to your maven 'settings.xml'

```
<profiles>
  <profile>
    <id>profile-clover</id>
    <activation>
      <property>
        <name>profile</name>
        <value>clover</value>
      </property>
    </activation>
    <properties>
      <cloverLicense>[LOCATION-OF-clover.license-FILE]</cloverLicense>
    </properties>
  </profile>
</profiles>
```

2. To run clover

```
mvn clover2:instrument clover2:aggregate clover2:clover -Pprofile-clover
```

3. A report will be generated in the following directory:
   //target/site/clover/

Logging

1. DuraCloud uses the SLF4j logging framework backed by the LogBack implementation
2. By adding either a logback.xml or logback-test.xml file on the classpath, logging configuration can be customized

DuraCloud internal tools

ServicesAdmin CLI

1. This tool provides a commandline interface for interacting with the 'services-admin' installed in a running OSGi container (see notes above for starting the container)
2. To build and run the CLI, from within the //servicesadminclient module

```
mvn assembly:assembly
java -cp target/servicesadminclient-[VERSION]-cli.jar
```

Application initialization utility

1. This utility takes a config file (example at //app-config/src/main/resources/init.props) and initializes an instance of duracloud
2. Until the applications durastore and duraservice are initialized, they are non-functional
3. To build and run the app-config utility, from within the //app-config module

```
mvn assembly:assembly
java -jar target/app-config-1.0.0-driver.jar
```

StoreClient package

1. To create a distributable zip of the storeclient and its dependencies, from within //storeclient run

```
mvn install -Ppackage-client
```

2. The zip will be found at /storeclient/target/store-client.zip

Misc configuration/discussion

Services on Windows

The following services do not function in a Windows deployment environment

- WebAppUtilService
- HelloWebappWrapper
- J2KService
- ImageMagickService

If you would like to run the ImageConversionService, you must install ImageMagick and have its /bin directory in your PATH, which is essentially what the ImageMagickService does in a linux environment.

root user

application config

OSGi container

Service Development Guide

Page under construction
AWoods 29-July-2010

Introduction

This document is intended to provide guidelines for developing your own custom services that can be deployed into the DuraCloud services framework.
The framework has been designed to allow any

- Java code that can be
- deployed as an OSGi bundle and
- implements the ComputeService interface
  to be dynamically installable into a running DuraCloud application instance.

Overview

In order to understand the flow of services within the services framework, there are several entities with which to become familiar:

1. **DuraService**
   - web application that exposes a REST-API for end-user management of services
     a. deploy service
     b. undeploy service
     c. configure service
     d. get service listing
     e. get service properties
   - responsible for retrieving a service to be deployed from the **Service Registry** and streaming it to the **ServicesAdmin**
   - responsible for mediating subsequent end-user management of deployed services to **ServicesAdmin**
   - responsible for passing metadata to the OSGi container when a service is being deployed.
     - metadata is defined in the service-repo.xml which is built based on the ServiceInfo class for each service

2. **Service Registry**
   - persistent holding area of deployable services
   - contains **Service Packages** and a single services-configuration descriptor XML object
   - implemented as one or more DuraCloud "space(s)"
     - the properties file to initialize DuraCloud contains duraservice.service-storage properties which allow configuration of the host, port, context and space

3. **Service Package**
   - for simple services, this can be a single OSGi bundle, jar file
   - for more involved services, this would be a zip file containing
     a. OSGi bundle jar that implements the ComputeService interface
     b. OSGi bundle jars that are both dependencies of the ComputeService bundle (above) and not among the base bundles installed in the **OSGi Container** by default
     c. any support files (see 'Example Services' below)

4. **ServicesAdmin**
   - management component resident in the OSGi container
   - exposes an internal REST-API through which communication from **DuraService** takes place
   - responsible for actually communicating with services via their common ComputeService interfaces
   - responsible for keeping track of currently deployed services and dependency-link counters

5. **OSGi Container**
   - production implementation currently in use is Equinox
   - bundles installed by default are listed as dependencies in the pom-run.xml file in the ServicesAdmin project

Example Services

All of the below services are currently functional and available in the DuraCloud source baseline.
They can all be found within the services directory

Pure Java Services

1. Hello Service
2. Replication Service
3. WebAppUtil Service
4. ImageConversion Service
5. MediaStreaming Service

Web Application Services

1. HelloWebAppWrapper Service
2. J2K Service

System Services

1. Script Service
2. ImageMagick Service

General Compute Services

Implementation Guidelines

OSGi Compatibility

- Although there are several available OSGi containers, and the DuraCloud test harnesses employ Equinox, Felix, and Knoplerfish, the production container into which services need to be able to deploy is **Equinox**.

ComputeService Interface

Dependency Packaging

Service Descriptor

Logging Configuration

Introduction

The logging framework used in the DuraCloud application is SLF4J with the LogBack implementation statically bound at runtime.
See the LogBack website for a detailed description of the configuration options.

The application also contains bridges for both Log4J and Commons-Logging which translates any underlying, dependency libraries which are configured to write to these frameworks into the SLF4J API. The effect is that all logging is channeled through the SLF4J configuration.

General Usage

- By default, if no configuration file is found by LogBack, the logging level is set to "DEBUG" and the appender is set to "STDOUT"
- When starting any DuraCloud application, a LogBack configuration file may be specified by using the following system variable

```
java -Dlogback.configurationFile={path-to-logging-configuration-file} -jar any-application
```

- Additionally, LogBack will use the file named "logback.xml" found at the top of the classpath for configuration
- An example logback.xml file

```
1<?xml version="1.0" encoding="UTF-8"?>
2
3<configuration >
4  <!--<configuration debug="true" scan="true">-->
5  <jmxConfigurator/>
6  <property name="LOG_FILENAME" value="/home/duraspace/logs/duracloud-osgi.log" />
7
8  <appender name="DURACLOUD" class="ch.qos.logback.core.rolling.RollingFileAppender">
9    <File>${LOG_FILENAME}</File>
10    <encoder>
11      <pattern>%-14p %d{yyyy/MM/dd HH:mm:ss} [%t] (%F:%L\\) [%M(\\)] - %m%n</pattern>
12    </encoder>
13    <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
14      <maxIndex>5</maxIndex>
15      <FileNamePattern>${LOG_FILENAME}.%i</FileNamePattern>
16    </rollingPolicy>
17    <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
18      <MaxFileSize>20MB</MaxFileSize>
19    </triggeringPolicy>
20  </appender>
21  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
22    <encoder>
23      <pattern>%-14p %d{yyyy/MM/dd HH:mm:ss} [%t] (%F:%L\\) [%M(\\)] - %m%n</pattern>
24    </encoder>
25  </appender>
26  <logger name="org.duracloud" level="DEBUG" additivity="false">
27    <appender-ref ref="DURACLOUD"/>
28  </logger>
29   <root level="WARN">
30    <appender-ref ref="STDOUT"/>
31  </root>
32</configuration>
```

- Notes on the above logback.xml file
  - on line 4, the attribute "debug" applies to displaying configuration information when LogBack starts up if set to "true"
  - on line 4, the attribute "scan" configures LogBack to re-read the given logback.xml every 60 seconds (by default) for updates
  - on line 26, the attribute "additivity" configures the given logger to inherit the configuration of the parent logger, in this case, the root logger
  - on line 26, if the "additivity" attribute were set to "true", all "DURACLOUD" log output would also log to "STDOUT"

DuraCloud REST API

DuraCloud REST API methods:

Notes

Each of the methods below has specific security requirements. See DuraCloud Security for more information

Due to an issue which does not properly handle requests redirected from http to https, it is recommended that all REST API requests use https directly.

Examples calling the API defined below with the Unix utility "curl" can be found here

DuraStore

Purpose: DuraStore is the application through which DuraCloud manages storage. The DuraStore REST API provides access to storage by mediating the underlying storage provider APIs to allow access to multiple cloud storage options through a single API.

Store REST Methods

Initialize Stores

- Purpose: Allows the initialization of storage provider accounts
- Request: POST https://host:port/durastore/stores
- Request Body: XML similar to:

```
<storageProviderAccounts>
  <storageAcct ownerId='0' isPrimary='true'>
    <id>1</id>
    <storageProviderType>AMAZON_S3</storageProviderType>
    <storageProviderCredential>
      <username>username</username>
      <password>password</password>
    </storageProviderCredential>
  </storageAcct>
</storageProviderAccounts>
```

- Response Code: 200 (on success)
- Response Body: "Initialization Successful" (on success)

Initialize Security Users

- Purpose: Allows the initialization of authorized users
- Request: POST https://host:port/durastore/security
- Request Body: XML similar to:

```
<?xml version="1.0" encoding="UTF-8"?>
<dur:security-users schemaVersion="0.2" xmlns:dur="duracloud.org">
  <security-user>
    <username>username-0</username>
    <password>password-0</password>
    <enabled>true</enabled>
    <accountNonExpired>true</accountNonExpired>
    <credentialsNonExpired>true</credentialsNonExpired>
    <accountNonLocked>true</accountNonLocked>
    <grantedAuthorities>ROLE_USER</grantedAuthorities>
  </security-user>
  <security-user>
    <username>username-1</username>
    <password>password-1</password>
    <enabled>false</enabled>
    <accountNonExpired>false</accountNonExpired>
    <credentialsNonExpired>false</credentialsNonExpired>
    <accountNonLocked>false</accountNonLocked>
    <grantedAuthorities>ROLE_USER ROLE_ADMIN</grantedAuthorities>
  </security-user>
</dur:security-users>
```

- Response Code: 200 (on success)
- Response Body: "Initialization Successful" (on success)

Get Stores

- Purpose: Provides a listing of available storage providers accounts (without credentials)
- Request: GET https://host:port/durastore/stores
- Parameters: None
- Response Code: 200 (on success)
- Response Body: XML similar to:

```
<storageProviderAccounts>
  <storageAcct isPrimary='true'>
    <id>1</id>
    <storageProviderType>AMAZON_S3</storageProviderType>
  </storageAcct>
  <storageAcct isPrimary="false">
    <id>2</id>
    <storageProviderType>RACKSPACE</storageProviderType>
  </storageAcct>
</storageProviderAccounts>
```

Space REST Methods

Get Spaces

- Purpose: Provides a listing of all of the spaces that a customer has created
- Request: GET https://host:port/durastore/spaces ? (storeID)
- Response Code: 200 (on success)
- Response Body: XML similar to:

```
<spaces>
  <space id="space1" />
  <space id="space2" />
</spaces>
```

Get Space

- Purpose: Provides a listing of the contents of a space along with space metadata
- Request: GET https://host:port/durastore/spaceID ? (storeID) (prefix) (maxResults) (marker)
    - storeID (optional) - ID of the content storage provider to query (default is primary store)
    - prefix (optional) - Only retrieve content ids with this prefix (default is all content ids)
    - maxResults (optional) - The maximum number of content IDs to return in the list (default is 1000)
      note: the maximum allowable value for maxResults is 1000. Any larger value will be reduced to 1000.
    - marker (optional) - The content ID marking the last item in the previous set (default is the first set of ids)
- Response Code: 200 (on success)
- Response Body: XML similar to:

```
<space id="space1">
  <item>Image 1</item>
  <item>Image 2</item>
</space>
```

- Response Headers: All available space metadata, example:

```
  x-dura-meta-space-count: 65
  x-dura-meta-space-access: OPEN
  x-dura-meta-space-created: Mon, 01 Jan 2000 08:00:00 EST
  x-dura-meta-custom-metadata: Custom Metadata Value
```

Get Space Metadata

- Purpose: Provides all space metadata
- Request: HEAD https://host:port/durastore/spaceID ? (storeID)
- Response Code: 200 (on success)
- Response Headers: Same as for Get space (above)

Create Space

- Purpose: Creates a new space
- Request: PUT https://host:port/durastore/spaceID ? (storeID)
- Request Headers: Metadata about the space, example:

```
  x-dura-meta-space-access: OPEN
  x-dura-meta-custom-metadata: Custom Metadata Value
```

- Response Code: 201 (on success)
- Response Headers: Location of the new space (i.e. the URL used to create the space), example:

```
  Location: https://myhost:8080/durastore/space1
```

Set Space Metadata

- Purpose: Updates the metadata associated with a space
- Request: POST https://host:port/durastore/spaceID ? (storeID)
- Request Headers: Same as Create space (above)
- Response Code: 200 (on success)
- Response Body: "Space $spaceID updated successfully" (on success)

Delete Space

- Purpose: Deletes a space
- Request: DELETE https://host:port/durastore/spaceID ? (storeID)
- Response Code: 200 (on success)
- Response Body: "Space $spaceID deleted successfully" (on success)

Content REST Methods

Get Content

- Purpose: Retrieves a piece of content along with its metadata
- Request: GET https://host:port/durastore/spaceID/contentID ? (storeID) (attachment)
    - if attachment param value is true, a Content-Disposition header is included with the response
- Response Code: 200 (on success)
- Response Body: The content stream
- Response Headers: All available content metadata, example:

```
Content-Type: text/plain
Content-Length: 5732
Content-MD5: 345670923478509747383920 2
ETag: 345670923478509747383920 2
x-dura-meta-content-name: Testing Content
x-dura-meta-content-owner: JSmith
```

Get Content Metadata

- Purpose: Retrieves the metadata of a piece of content without the content itself
- Request: HEAD https://host:port/durastore/spaceID/contentID ? (storeID)
- Response Code: 200 (on success)
- Response Headers: Same as Get content (above)

Store Content

- Purpose: Adds a piece of content to the store
- Request: PUT https://host:port/durastore/spaceID/contentID ? (storeID)
- Request Body: Content to be added
- Request Headers: Metadata about the content, example:

```
Content-Type: text/plain
Content-MD5: 4cd56e137a93a1accb43c5d32f4afffb
x-dura-meta-content-name: Testing Content
x-dura-meta-content-owner: JSmith
```

    - Note that when the optional Content-MD5 header is included, the final checksum of the stored file is compared against the MD5 value included in the header to ensure that the file was stored correctly. If the header is not included, an MD5 checksum is computed as the file is transferred to storage, and that value is used in the final comparison.
- Response Code: 201 (on success)
- Response Headers: Location of the new content (i.e. the URL used to create the content), example:

```
Location: https://myhost:8080/durastore/space1/content1
```

Set Content Metadata

- Purpose: Updates the metadata associated with a piece of content
- Request: POST https://host:port/durastore/spaceID/contentID ? (storeID)
- Request Headers: Same as Store content (above)
- Response Code: 200 (on success)
- Response Body: "Content $contentID updated successfully"

Delete Content

- Purpose: Removes a piece of content from the store
- Request: DELETE https://host:port/durastore/spaceID/contentID ? (storeID)
- Response Code: 200 (on success)
- Response Body: "Content $contentID deleted successfully"

Task REST Methods

Tasks are used to perform storage provider actions which cannot be performed in a generic manner across multiple providers.

Get Tasks

- Purpose: Provides a listing of all of the supported tasks for a given provider. Note that if no storeID parameter is included, the task listing is provided for the primary storage provider.
- Request: GET https://host:port/durastore/task ? (storeID)
- Response Code: 200 (on success)
- Response Body: XML similar to:

```
<list>
  <string>task1</string>
  <string>task2</string>
</list>
```

Perform Task

- Purpose: Performs a particular task. Note that most tasks can be performed by only one storage provider type.
- Request: POST https://host:port/durastore/task/taskName ? (storeID)
- Request Body: Parameters for task. Each task will expect parameters in a specific format, see task listing for more details.
- Response Code: 200 (on success)
- Response Body: Response value for task, format varies by task.

Tasks

taskName	Storage Provider	Name	Description	Request Body	Response Body
enable-streaming	Amazon S3	Enable Streaming task	Enables RTMP streaming for all files within a DuraCloud space through the use of Amazon's Cloudfront streaming capability. This task may take up to 15 minutes to complete.	Name of the space for which streaming is to be enabled	Text indicating the results of the task, including the streaming host
disable-streaming	Amazon S3	Disable Streaming task	Disables streaming by removing the ability for Cloudfront to access files within a space. This does not remove the streaming distribution, only disables its use, so enabling streaming on the same space again can be performed much more quickly. Some content in the space may continue to be available for streaming up to 24 hours after streaming has been disabled.	Name of the space for which streaming is to be disabled	Text indicating the results of the task
delete-streaming	Amazon S3	Delete Streaming task	Removes a streaming distribution created by the enable-streaming task. This task should be performed after performing the disable-streaming task. This task may take up to 15 minutes to complete, after which no content in the space will be available for streaming.	Name of the space for which streaming is to be deleted	Text indicating the results of the task
run-hadoop-job	Amazon S3	Run Hadoop Job task	Runs a hadoop job using Amazon's Elastic Map Reduce feature. A JAR which implements the hadoop interfaces is expected to have already been loaded into S3. This JAR is used to execute the hadoop job.	A map serialized into XML which includes, at a minimum, values for jarContentId, sourceSpaceId, destSpaceId, and workSpaceId.	A map serialized into XML which includes the jobFlowId
describe-hadoop-job	Amazon S3	Describe Hadoop Job task	Retrieves information about a hadoop job running in Amazon's Elastic Map Reduce	The Job Flow ID	A map serialized into XML which includes information about the running job
stop-hadoop-job	Amazon S3	Stop Hadoop Job task	Stops a hadoop job running in Amazon's Elastic Map Reduce	The Job Flow ID	A map serialized into XML which includes a results key with a value of either success or failure
noop	Amazon S3	Test task	Provides a simple way to test the calling of tasks	Body content is ignored	Text indicating successful task completion

DuraService

Purpose: DuraService is the application through which DuraCloud manages services. The DuraService REST API provides the means by which services available in the DuraCloud service repository are deployed, configured, and undeployed.

Resources: XML schema which define the service configuration can be found here

Service REST Methods

Initialize Services

- Purpose: Initializes the DuraService application
- Request: POST https://host:port/duraservice/services
- Request Body: XML similar to:

```
<servicesConfig>
  <primaryServiceInstance>
     <host>[PRIMARY-SERVICE-INSTANCE-HOST]</host>
     <servicesAdminPort>[PRIMARY-SERVICES-ADMIN-PORT]</servicesAdminPort>
     <servicesAdminContext>[PRIMARY-SERVICES-ADMIN-CONTEXT]</servicesAdminContext>
  </primaryServiceInstance>
  <userStorage>
    <host>[USER-STORAGE-HOST-NAME]</host>
    <port>[USER-STORAGE-PORT]</port>
    <context>[USER-STORAGE-CONTEXT]</context>
    <msgBrokerUrl>[USER-STORAGE-MSG-BROKER-URL]</msgBrokerUrl>
  </userStorage>
  <serviceStorage>
    <host>[SERVICES-STORAGE-HOST-NAME]</host>
    <port>[SERVICES-STORAGE-PORT]</port>
    <context>[SERVICES-STORAGE-CONTEXT]</context>
    <spaceId>[SERVICES-STORAGE-SPACE-ID]</spaceId>
  </serviceStorage>
  <serviceCompute>
    <type>AMAZON_EC2</type>
    <imageId>[MACHINE-IMAGE-ID]</imageId>
    <computeProviderCredential>
      <username>[USERNAME]</username>
      <password>[PASSWORD]</password>
    </computeProviderCredential>
  </serviceCompute>
</servicesConfig>
```

- Response Code: 200 (on success)
- Response Body: "Initialization Successful" (on success)

Initialize Security Users

- Purpose: Allows the initialization of authorized users
- Request: POST https://host:port/duraservice/security
- Request Body: XML similar to:

```
<?xml version="1.0" encoding="UTF-8"?>
<dur:security-users schemaVersion="0.2" xmlns:dur="duracloud.org">
  <security-user>
    <username>username-0</username>
    <password>password-0</password>
    <enabled>true</enabled>
    <accountNonExpired>true</accountNonExpired>
    <credentialsNonExpired>true</credentialsNonExpired>
    <accountNonLocked>true</accountNonLocked>
    <grantedAuthorities>ROLE_USER</grantedAuthorities>
  </security-user>
  <security-user>
    <username>username-1</username>
    <password>password-1</password>
    <enabled>false</enabled>
    <accountNonExpired>false</accountNonExpired>
    <credentialsNonExpired>false</credentialsNonExpired>
    <accountNonLocked>false</accountNonLocked>
    <grantedAuthorities>ROLE_USER ROLE_ADMIN</grantedAuthorities>
  </security-user>
</dur:security-users>
```

- Response Code: 200 (on success)
- Response Body: "Initialization Successful" (on success)

Get Services

- Purpose: Retrieves a listing of services, along with their configuration options
- Request: GET https://host:port/duraservice/services ? (show)
    - Parameter options for show (optional)
        1. available (default) - Includes only services which have not been deployed but are available for deployment
        2. deployed - Includes only services which have been deployed and started
- Response Code: 200 (on success)
- Response Body: XML list of services (see service config xsd)

Get Service

- Purpose: Retrieves information about a particular service including description, configuration options, and all deployments
- Request: GET https://host:port/duraservice/serviceID
- Response Code: 200 (on success)
- Response Body: XML service (see service config xsd)

Get Deployed Service

- Purpose: Retrieves information about a deployed service including description, configuration options, and a single deployment indicating the configuration options in use
- Request: GET https://host:port/duraservice/serviceID/deploymentID
- Response Code: 200 (on success)
- Response Body: XML service (see service config xsd)

Get Deployed Service Properties

- Purpose: Retrieves the runtime properties of a deployed service
- Request: GET https://host:port/duraservice/serviceID/deploymentID/properties
- Response Code: 200 (on success)
- Response Body: XML service (simple xml Map serialization)

Deploy Service

- Purpose: Deploys and starts an available service
- Request: PUT https://host:port/duraservice/serviceID ? (serviceHost)
    - Parameter value for serviceHost (optional) should indicate the services host on which the service should be deployed. Default is the primary customer host.
- Request Body: XML user configuration indicating the config selections for the service (see user config portion of service config xsd)
- Response Code: 201 (on success)
- Response Header: Location header indicates the URL at which information about the deployed service can be retrieved (the URL for a get deployed service call) which includes the deploymentID

Update Service Configuration

- Purpose: Updates the configuration of a deployed service
- Request: POST https://host:port/duraservice/serviceID/deploymentID
- Request Body: Updated XML user configuration indicating the config selections for the service (see user config portion of service config xsd)
- Response Code: 200 (on success)

UnDeploy Service

- Purpose: Stops and Undeploys a deployed service
- Request: DELETE https://host:port/duraservice/serviceID/deploymentID
- Response Code: 200 (on success)

DurAdmin

Purpose: DurAdmin is the user-facing application through which DuraCloud exposes DuraStore and DuraService functionality. The DurAdmin REST API provides the means by which DurAdmin is initialized.

Service REST Methods

Initialize Application

- Purpose: Allows the initialization of duradmin
- Request: POST https://host:port/duradmin/init
- Request Body: XML similar to:

```
<duradminConfig>
  <durastoreHost>[host]</durastoreHost>
  <durastorePort>8080</durastorePort>
  <durastoreContext>durastore</durastoreContext>
  <duraserviceHost>[host]</duraserviceHost>
  <duraservicePort>8080</duraservicePort>
  <duraserviceContext>duraservice</duraserviceContext>
</duradminConfig>
```

- Response Code: 200 (on success)
- Response Body: "Initialization Successful" (on success)

Initialize Security Users

- Purpose: Allows the initialization of authorized users
- Request: POST https://host:port/duradmin/security
- Request Body: XML similar to:

```
<?xml version="1.0" encoding="UTF-8"?>
<dur:security-users schemaVersion="0.2" xmlns:dur="duracloud.org">
  <security-user>
    <username>username-0</username>
    <password>password-0</password>
    <enabled>true</enabled>
    <accountNonExpired>true</accountNonExpired>
    <credentialsNonExpired>true</credentialsNonExpired>
    <accountNonLocked>true</accountNonLocked>
    <grantedAuthorities>ROLE_USER</grantedAuthorities>
  </security-user>
  <security-user>
    <username>username-1</username>
    <password>password-1</password>
    <enabled>false</enabled>
    <accountNonExpired>false</accountNonExpired>
    <credentialsNonExpired>false</credentialsNonExpired>
    <accountNonLocked>false</accountNonLocked>
    <grantedAuthorities>ROLE_USER ROLE_ADMIN</grantedAuthorities>
  </security-user>
</dur:security-users>
```

- Response Code: 200 (on success)
- Response Body: "Initialization Successful" (on success)

REST API Examples Using curl

Convenience Variables

The curl commands below can be used directly if you define the following variables in your shell

```
host=<duracloud-hostname>
space-0=<any-name>
space-1=<any-name>
user=<username>
pword=<password>
file=<any-file-name>
```

DuraStore Notes

The curl commands in the DuraStore section expect a test file for uploads.

```
echo hello > ${file}
```

Note that if the target of a content or space retrieval (GET) has access permissions set to "OPEN", then the "-u" option in the curl commands is not required.

DuraStore

Get Stores

```
curl -u ${user}:${pword} https://${host}/durastore/stores
```

Get Spaces

```
curl -u ${user}:${pword} https://${host}/durastore/spaces
curl -u ${user}:${pword} https://${host}/durastore/spaces?storeID=1
```

Create Space

```
curl -u ${user}:${pword} -X PUT https://${host}/durastore/${space-0}
curl -u ${user}:${pword} -H "x-dura-meta-city: arlington" -H "x-dura-meta-state: va" -X PUT
https://${host}/durastore/${space-1}?storeID=1
```

Store Content

```
curl -u ${user}:${pword} -T ${file} https://${host}/durastore/${space-0}/test.txt
curl -u ${user}:${pword} -T ${file} https://${host}/durastore/${space-0}/item.txt
```

Get Space

```
curl -u ${user}:${pword} https://${host}/durastore/${space-0}
curl -u ${user}:${pword} https://${host}/durastore/${space-1}?storeID=1
curl -u ${user}:${pword} https://${host}/durastore/${space-0}?prefix=test
```

Set Space Metadata

```
curl -u ${user}:${pword} -H "x-dura-meta-country: usa" -X POST https://${host}/durastore/${space-0}
```

Get Space Metadata

```
curl -u ${user}:${pword} -I https://${host}/durastore/${space-0}
curl -u ${user}:${pword} -I https://${host}/durastore/${space-1}?storeID=1
```

Get Content

```
curl -u ${user}:${pword} https://${host}/durastore/${space-0}/test.txt
curl -u ${user}:${pword} https://${host}/durastore/${space-0}/test.txt?storeID=0\&attachment=true
```

Set Content Metadata

```
curl -u ${user}:${pword} -X POST -H "x-dura-meta-color: green"
https://${host}/durastore/${space-0}/test.txt
```

Get Content Metadata

```
curl -u ${user}:${pword} -I https://${host}/durastore/${space-0}/test.txt
```

Delete Content

```
curl -u ${user}:${pword} -X DELETE https://${host}/durastore/${space-0}/test.txt
```

Delete Space

```
curl -u ${user}:${pword} -X DELETE https://${host}/durastore/${space-0}
curl -u ${user}:${pword} -X DELETE https://${host}/durastore/${space-1}?storeID=1
```

DuraService Notes

The commands in the DuraService section below demonstrate the deployment, inspection, reconfiguration, and undeployment of a service.
As of release 0.8.0, the Bit Integrity Checker has service id of '0' (noted in variable below).
The actual deployment id is dynamically generated based on the number of previous deployments within a given DuraCloud application.
After deploying the service, use the "Get Deployed Service" call to determine the specific *deployment id* of your deployed service.

```
bitintegrity=0
deployment=<determined-by-inspection>
```

The following two service configuration files are provided for deployment and reconfiguration of the Bit Integrity Checker.
They are based on the spaces that were created in the DuraStore section, but can be modified to indicate other configuration options or to execute over alternate spaces.

1. configuration file 0
2. configuration file 1

DuraService

Get Services

```
curl -u ${user}:${pword} https://${host}/duraservice/services
curl -u ${user}:${pword} https://${host}/duraservice/services?show=deployed
```

Deploy Service

```
curl -u ${user}:${pword} -X PUT -T deploy.xml https://${host}/duraservice/${bitintegrity}
```

Get Service

```
curl -u ${user}:${pword} https://${host}/duraservice/${bitintegrity}
```

Get Deployed Service

```
curl -u ${user}:${pword} https://${host}/duraservice/${bitintegrity}/${deployment}
```

Get Deployed Service Properties

```
curl -u ${user}:${pword} https://${host}/duraservice/${bitintegrity}/${deployment}/properties
```

Update Service Configuration

```
curl -u ${user}:${pword} -X PUT -T deploy-new.xml https://${host}/duraservice/${bitintegrity}
```

UnDeploy Service

```
curl -u ${user}:${pword} -X DELETE https://${host}/duraservice/${bitintegrity}/${deployment}
```

DuraCloud Service Designs

Service Design Documenation

Bit Integrity Checker

Overview

The Bit Integrity Checker is intended to provide a simple and easy to use way of assuring that the content stored in your DuraCloud account has maintained bit integrity. The basic idea is that a DuraCloud account administrator provides a listing of expected content IDs and their associated MD5s. This listing is then used by the service as a basis for comparison against MD5s found in DuraCloud.

There is a trade-off between cost (both in time and money) and assurance in the trustworthiness of the MD5 provided by DuraCloud. The service is designed to offer three options to address this balance, see "Levels of trust" below. The fastest and cheapest option is to use the MD5 stored in the metadata of the content item. The underlying storage providers assert that this value which is created on ingest is also checked when the content item is read, and retrieves a mirrored copy of the content if there is a mismatch. As a note, when content is pushed into DuraCloud via the DuraStore REST-API an MD5 can be provided by the user to be automatically checked with the one generated by the underlying storage provider, and if no MD5 is provided the DuraCloud application calculates it for this comparison. If the administrator does not trust the assertion of the underlying storage provider, the Bit Integrity Checker also provides the option of reading the content and recalculating the MD5. Finally, if the administrator does not trust this recalculation, then the Bit Integrity Checker also provides the option of passing in a "salt" character string which will be appended to the content during the recalculation of the MD5.

The input listing is expected to be found as a user-specified content item within DuraCloud, and the resultant output file will be stored to a user-specified location within DuraCloud. The formats of the input and output files are the same, so a previous run's output may be used as a subsequent run's input.

see July 2010 NDIIPP presentation

Requirements

1. User has the option to provide as input the listing of content items with expected MD5s
2. User has the option to provide as input the listing of content items without expected MD5s
3. Service will determine the system MD5 value for each content item according to selected "Level of trust" algorithm
4. Service will store the determined listing in a user-provided DuraCloud location
5. User has the option to specify any two listings of content items / MD5 pairs for comparison
6. Service will report on comparison of provided MD5s against system MD5s values of content items
7. Service will store report output in user-provided DuraCloud location
8. Service will run on a compute instance local to the input storeId - *future*

Design

Levels of trust

Provides the choice of balance amongst cost, time, and assurance

1. Trust in underlying storage providers
2. Trust in DuraCloud and opensource software
3. Trust in requester of service

The three levels of trust above are addressed by three implementations

1. System MD5s are determined by using the stored metadata values
2. System MD5s are determined by DuraCloud re-reading the content bytes and re-calculating the MD5s
3. System MD5s are determined by DuraCloud re-reading the content bytes appended with a 'salt' and re-calculating the MD5s

Operational modes

In order to address both scenarios of allowing the user to have certainty that MD5s are being generated/checked when requested and allowing the user to trust the service and have it execute with a single command, the following modes are available.

1. Single-step interaction
    a. User invokes service with a listing of contentId/**MD5 pairs** (and other options) to check
    b. Bit Integrity Checker generates a listing of contentId/MD5 pairs based on the input options **and performs comparison with input, expected MD5s**
    c. Service generates result report
2. Two-step interaction
    a. User invokes service with a listing of contentIds (and other options) to check
    b. Bit Integrity Checker generates a listing of contentId/MD5 pairs based on the input options
    c. User invokes service with a listing of expected contentId/MD5 pairs to be compared to the generated listing
    d. Service generates result report

Functionality spec

Single-step interaction

1. Service inputs
    a. spaceId & contentId where input listing is stored

        - comma-delimited listing of spaceId, contentId and expected MD5
        - each content item separated by newline character
        - first line in file will be ignored
    b. spaceId & contentId where results file should be written
    c. options (see below)
2. Service outputs
    a. comma-delimited listing of spaceIds, contentIds, expected MD5s, system MD5s, status state
    b. service status state

Two-step interaction

1. Step 1: Service inputs
    a. spaceId & contentId where input listing is stored
        - comma-delimited listing of spaceId, contentId and **without** expected MD5
        - each content item separated by newline character
        - first line in file will be ignored
    b. spaceId & contentId where results file should be written
    c. options (see below)
2. Step 1: Service outputs
    a. comma-delimited listing of spaceIds, contentIds, system MD5s
    b. service status state
3. Step 2: Service inputs
    a. spaceId & contentId where input listing is stored
        - comma-delimited listing of spaceId, contentId and expected MD5
        - each content item separated by newline character
        - first line in file will be ignored
    b. spaceId & contentId where service-generated listing is stored
        - comma-delimited listing of spaceId, contentId and system MD5
        - as generated and stored by the the service in step-1
4. Service outputs
    a. comma-delimited listing of spaceIds, contentIds, expected MD5s, system MD5s, status state
    b. service status state

Service options

1. trust level (stored value, recalculate, salt)
2. salt
    - arbitrary character string which will be appended to content in generating MD5
3. fail-fast boolean
    - service will exit when first error/mismatch found if 'true'
4. complete space(s) boolean
    - indicates if the input listing should be checked against the complete set of items in the space(s)
5. storeId of underlying storage provider
    - default to primary underlying storage provider

Service exceptions

1. Checked
    - Contains the following enum
    a. missing MD5 (expected or found)
    b. MD5 mis-match
    c. unequal content listings
2. Runtime
    a. internal error
    b. salt option set but salt not provided
    c. service level not supported
    d. input content item not exists
    e. output result content item already exists

Options / Mode matrix

The table below shows the possible usage scenarios across the top, and their associated input options.

1. One-step: hash from input list
    - user initiates Bit Integrity Checker as a single operation
    - an input listing of content-ids/hashes is provided
    - service generates hashes one-to-one for each item in the input listing
2. One-step: hash from complete space
    - user initiates Bit Integrity Checker as a single operation
    - an input listing of content-ids/hashes is provided
    - service generates hashes for **all** content-items in spaces found in the input listing
3. Two-step: hash from input list
    - user initiates Bit Integrity Checker as a two part operation, this being the first

    - an input listing of only content-ids is provided
    - service generates hashes one-to-one for each item in the input listing
4. Two-step: hash from complete space
    - user initiates Bit Integrity Checker as a two part operation, this being the first
    - user indicates space(s) of target content to hash
5. Two-step: compare two lists
    - user initiates Bit Integrity Checker as a two part operation, this being the second
    - two input listings of content-ids/hashes are provided

User input option	One-step: hash from input list	One-step: hash from complete space	Two-step: hash from input list	Two-step: hash from complete space	Two-step: compare two lists
hash approach	✓	✓	✓	✓	
salt	✓	✓	✓	✓	
fail-fast	✓	✓			✓
storage provider id	✓	✓	✓	✓	✓
space of provided listing	✓	✓	✓		✓
object-id of provided listing	✓	✓	✓		✓
space of provided listing-B					✓
object-id of provided listing-B					✓
space(s) of target content		✓		✓	
space for output	✓	✓	✓	✓	✓
object-id of result listing	✓	✓	✓	✓	
object-id of report	✓	✓			✓

Distributed Compute Services

Introduction

Due to the potential for DuraCloud users to store large number of files in the DuraCloud system, it becomes necessary to be able to run processing jobs over large data sets. This can be done in a variety of ways, but two primary distinctions come in the separation of services which take advantage of the capabilities of underlying cloud provider offerings, and services which require a DuraCloud-provided strategy for managing a distributed processing environment.

Cloud Provider offerings

Amazon

Amazon's Elastic Map Reduce offering makes use of the Hadoop project available from Apache. Amazon's service manages the server cluster on which Hadoop is run. Users provide the code necessary to actually perform the necessary computations using the Map/Reduce algorithm.

DuraCloud makes use of this capability in the Bulk Image Transformer, the Duplicate on Demand service, and the Bulk Bit Integrity Checker.

A DuraCloud service which uses Amazon's Elastic Map Reduce capability like the Bulk Image Transformer is made up of three parts:

1. The Processor Jar. This is a project which extends the base-processor project (the image-conversion-processor project does this for the Bulk Image Transformer.) This code extends the Hadoop framework to perform processing as part of the Map/Reduce algorithm. This Jar file is included as a resource within the DuraCloud service, then moved to S3 as part of service deployment to make it available to Amazon Elastic Map Reduce.
2. The DuraCloud Service, such as the bulkimageconversionservice project, is what is stored in the DuraCloud service registry and deployed into the DuraCloud service OSGi container. This code handles:
    a. Taking in all of the necessary parameters from the user
    b. Moving the processor jar and any bootstrap scripts to S3 storage
    c. Providing feedback to the user to indicate how the job is progressing
3. The DuraCloud Task. There are three tasks which communicate directly with the Elastic Map Reduce system to actually run the Hadoop jobs. This code is contained in the S3StorageProvider project, under the tasks package. New services making use of Hadoop may or may not need to update this code depending on whether there are job-specific parameter values that need to be processed. The tasks are:
    a. Run Hadoop Job Task - Actually starts the Hadoop job in Elastic Map Reduce with the parameters provided by the user and the processing jar and bootstrap script locations
    b. Describe Hadoop Job Task - Gets information about a Hadoop job in process, in order to provide this information back to the user
    c. Stop Hadoop Job Task - Stops a Hadoop job that is in process. If this call is made to a Hadoop job that is already complete, it is disregarded.

Microsoft Azure

Microsoft's Dryad project, which is still in the research phase, is said to be a very generic graph generation and processing engine which has the potential to be very powerful. Dryad should be able to handle the Map/Reduce algorithm as well as many other types of processing algorithms. Of course, with this added flexibility comes added complexity.

As noted here, Dryad is not yet available on Azure. So far there do not appear to be any published tests using Dryad.

A good comparison of Hadoop and Dryad

According to this article, Microsoft may be considering the use of Hadoop within Azure.

DuraCloud Distributed Processing

More to come...

Duplicate on Upload

Purpose

The Duplicate on Upload service can be used to copy a space (including all content and metadata) from the primary storage provider (currently Amazon S3) to another storage provider.

Options

The following options are available:

- Space to duplicate - A space which should be copied in its entirety to a secondary storage provider.
- Destination Store - The storage provider where content will be copied
- Space Name - the name of the replicated space

Notes

- After duplication, the original and destination spaces should contain identical content item lists
- Duplicated content items will include the full set of metadata available on the original content items

Flow

1. Service collects user input
    - space to replicate
    - store to replicate to
    - work space (for logs)
    - output space (for results file)

2. Service starts up Amazon Elastic MapReduce job via RunHadoopJobTaskRunner
    - Service passes the following info into the job:
        - set of input buckets
        - storeId of store to replicate to
        - work bucket
        - output bucket
        - duracloud account host/port/context
        - duracloud account username/password
    - Hadoop pulls files from S3 and splits into mapper functions
    - Each mapper runs by
        - picking up the file provided by hadoop
        - using StoreClient to create connections to both the source and destination ContentStores
        - ensuring that the destination space exists
        - checking to see if the destination space already contains the content item in question
            - if so: compare checksums, only move the file if they don't match
        - adding the S3 file to the destination bucket (if necessary)
        - retrieving the content metadata from the source content item and adding it to the destination content item
    - Reducer runs to collect final output, which is stored in output space

Image Server Service Design

Overview

When deployed in your DuraCloud server, the Image Server (imgserver) deploys an instance of the Djatoka JPEG2000 image server. This Djatoka server is itself a web application that needs to be deployed into a servlet container. That is why the System WebApp Utility (webapputil) must already be deployed/running on your DuraCloud server before it is possible to deploy the imgserver. In the process of deploying the imgserver, the webapputil installs a new instance of Apache Tomcat running on the next available port. See the design description of the webapputil service for more details. Then the Djatoka image server is deployed into the newly installed Tomcat. Along with the Djatoka image server, an IIPImage Viewer configured to reference the newly deployed Djatoka image server is deployed into the newly created Tomcat.
This image viewer is designed to act as the local mechanism for images hosted in your DuraCloud account to be viewable in the administration application (DurAdmin). If the imgserver is deployed and an item with mimetype image/<any> is select in DurAdmin, then both a thumbnail will be generated (with the help of the Djatoka server) and clicking on the thumbnail or the 'view' button will bring up the local IIPImage viewer. Note, if the mimetype of the image is set to a non-image value, such as binary/octet-stream, then the imgserver will not be active for that content item.

Integration with external applications

Using DuraCloud image viewer and image server

A common use case for the Image Server service is to integrate it with your existing website to serve the JP2 files that are hosted in your DuraCloud account. In order to do this you need to provide a link in your existing website to the imgserver viewer with an HTTP parameter specifying the direct DuraCloud URL of the image item. An example link would be

```
<a
href="http://180.12.34.56/adore-djatoka-p18080/viewer.html?rft_id=http://my.duracloud.org:80/durastore/jp
Image</a><br>
```

So in this example, the DuraCloud server has domain name "my.duracloud.org" with associated I.P. address of "180.12.34.56". The content is accessible through this address and the Djatoka image server is running (on a separate Tomcat/port) at the same address.
Notice, the I.P. address of your instance is required when referencing the image viewer. Using "http://my.duracloud.org/adore-djatoka-p18080" instead of "http://180.12.34.56/adore-djatoka-p18080" will not work.
Also note the "storeID" parameter. Here is the URL encoded value "?storeID=0". This is an optional parameter that can be used to specify the exact underlying storage provider from whence the image should come. See the REST API documentation for more details.

Using your own image viewer with DuraCloud image server

If on the other hand you would prefer to use your own image viewer instead of the IIPImage viewer deployed with the imgserver, this is also an option. In this case, within your viewer code, you need to configure it to reference the DuraCloud-hosted Djatoka resolver. In the example above, the resolver URL would be

```
http://180.12.34.56/adore-djatoka-p18080/resolver
```

Other considerations

Performance

The basic flow of interaction with the imgserver is that a

1. request is made from a client-side viewer (e.g IIPImage Viewer) to the Djatoka resolver
2. the Djatoka resolver retrieves the requested image as identified in the request URL
    a. converts it to JPEG2000 format if it is not already a JP2 file
3. the Djatoka server serves the requested tiles of the image in JPEG format

In this flow, there is considerable latency in the Djatoka resolver retrieving the image file from the underlying cloud content store. This latency is only experienced the first time the image is requested. After the initial request, the tiles are cached by the Djatoka server. Therefore, improved user performance can be achieved by pre-caching the images requesting them when they are made publicly available.

Access

When the Djatoka resolver is retrieving web-based content, the Djatoka software is not currently designed to supply authentication credentials with the HTTP request. As a result, the Djatoka server will not be able to serve content that resides in a "closed" DuraCloud space. We have a patch to the Djatoka baseline to address this issue and are working with the Djatoka developers to have it incorporated. In the meantime, all content that you would like to be served via the imgserver must reside in an "open" DuraCloud space.

Metadata Export Service

Design In-Progress
Comments/Suggestions welcome.
14-july-2010

Requirements

1. Service will create a listing of contentIds and associated MD5

Design

Functionality spec

1. Service inputs
    a. spaceId of contents for which associated metadata will be exported
    b. metadata field(s)
        i. MD5
        ii. TBD...
    c. spaceId where export file will be stored
    d. contentId of export file
2. Service outputs
    a. comma-delimited content item stored as input:spaceId/input:contentId
        - see output file description below
3. Service exceptions
    a. source space not found
    b. output space not found
    c. output content item already exists

Output File Description

1. The exported file will be comma-delimited
2. The first line of the file will contain a header for each column
3. If a metadata element is not found for any content item, the corresponding column will contain a message string

```
Space ID,Content ID,MD5
space0,path/contentA,15970caabc295dd928c0d3db20c72642
space0,path/contentB,MD5-not-found
space0,path/contentC,488e0b0466ee9c441fd20ff51a152a6c
```

DuraCloud Security

Overview

The security approach is divided into two distinct spheres of responsibility

1. Channel security (encryption)
2. Application security (AuthN / AuthZ)

The configuration of any given user compute instance will consist of an Apache HttpServer layered on top of Tomcat.

1. Apache HttpServer
    - All requests will come through Apache on port 443 (https) of the instance
    - The requests will internally be unencrypted, where encryption exists, and redirected to tomcat as open text
2. Tomcat
    - A defined set of resource endpoints will require AuthN and AuthZ
    - Spring-security is being leveraged to wire AuthN and AuthZ across relevant resources

Channel Security Implementation

1. Apache HttpServer is configured to require all requests to the three DuraCloud web applications (/duradmin, /durastore, and /duraservice) go over https.
2. Below are the https enforcement rules configured in Apache

```
###
# ensure 'duradmin' uses https
###

RewriteCond %{REQUEST_URI} /duradmin
RewriteCond %{SERVER_PORT} !^443$
RewriteRule ^(.*)$ https://%{SERVER_NAME}$1 [R=301,L]

###
# require https for 'durastore' & 'duraservice' for external requests
###

RewriteCond %{REQUEST_URI} ^(/durastore|/duraservice)
RewriteCond %{SERVER_PORT} !^443$
RewriteCond %{SERVER_NAME} !^localhost$
RewriteCond %{SERVER_NAME} !^127.0.0.1$
RewriteCond %{REMOTE_HOST} !^127.0.0.1$
RewriteCond ${local-ip-map:%{REMOTE_HOST}} !^localhost$
RewriteRule ^(.*)$ https://%{SERVER_NAME}$1 [R=301,L]
```

Application Security Implementation

The basic AuthN flow is as follows

1. User requests secured resource
2. If credentials not in request
    - response 401
3. Spring AuthenticationProvider performs AuthN
    a. AuthProvider asks UserDetailsService for GrantedAuthorities for given Principal
    b. notes
        i. DuraCloud provides custom UserDetailsService implementation to return UserDetails of requesting Principal
        ii. AbstractSecurityInterceptor permanently caches user AuthN decisions by default
4. Authentication object and "configuration attributes" are passed to AccessDecisionManager for AuthZ

Security Servlet Filters

DuraCloud leverages Spring's mechanism for wiring AuthN/Z into an application across servlet url patterns.
The following access rules are placed across the durastore and duraservice REST-APIs:

Store REST Methods

Action	Role
Initialize Stores	ROLE_ROOT
Initialize Security Users	ROLE_ROOT
Get Stores	ROLE_USER
Get Spaces	ROLE_ANONYMOUS if space 'open', else ROLE_USER
Get Space	ROLE_ANONYMOUS if space 'open', else ROLE_USER
Get Space Metadata	ROLE_ANONYMOUS if space 'open', else ROLE_USER
Create Space	ROLE_USER
Set Space Metadata	ROLE_USER
Delete Space	ROLE_USER
Get Content	ROLE_ANONYMOUS if space 'open', else ROLE_USER
Get Content Metadata	ROLE_ANONYMOUS if space 'open', else ROLE_USER
Store Content	ROLE_USER
Set Content Metadata	ROLE_USER
Delete Content	ROLE_USER

Service REST Methods

Action	Role
Initialize Services	ROLE_ROOT
Initialize Security Users	ROLE_ROOT
Get Services	ROLE_USER
Get Service	ROLE_USER
Get Deployed Service	ROLE_USER
Get Deployed Service Properties	ROLE_USER
Deploy Service	ROLE_USER
Update Service Configuration	ROLE_USER
UnDeploy Service	ROLE_USER

Administrative UI REST Methods

Action	Role
Initialize DurAdmin Application	ROLE_ROOT
Initialize Security Users	ROLE_ROOT

Roles

The fixed set of users/roles listed below are provided in DuraCloud. Each role in the list below represents a super set of the privileges of those above it.

1. ROLE_ANONYMOUS

    - no username/password
2. ROLE_USER
    - user created by DuraCloud-account admin
3. ROLE_ADMIN
    - owner of DuraCloud-account
4. ROLE_SYSTEM
    - internal user for delegation requests
5. ROLE_ROOT
    - DuraSpace personnel

User Management

1. In order for the administrator of a DuraCloud account to manage new users, an initial user with ROLE_ADMIN privileges is provided at start-up:
    - username: admin
    - password: changeme

Note: *Ultimately, the management of users will take place through the DuraCloud.org website (where users initially create accounts).*
User management is exposed in Duradmin in the upper-righthand corner of the console, but the usernames/passwords edited here are not persisted.
This means that when the application is migrated to the next release, those details will need to be re-entered/re-created.

Release Notes

Release 0.8

Released: Jan 26, 2011

The primary features of release 0.8 are:

- Simplified services
    - The listing of services has been better organized, to make finding the service you would like to run simpler.
    - All services now require you to set fewer options, simplifying the deployment process.
    - Bulk services (Image Transformer - Bulk, Bit Integrity Checker - Bulk, and Duplicate on Demand) now provide a *standard* configuration mode which handles the setting of server type, and number of servers used to perform the job, so that you no longer have to make those choices.
    - The output location for services has been set to the *x-service-out* space, which removes the need to set this value for each service, and provides a standard location to look for service output reports.
    - The work location for services has been set to the *x-service-work* space, which removes the need to set this value for each service, and provides a standard location to look for service logs and other run time artifacts.
- More reliable services
    - Several bugs which have caused services to fail have been resolved.
- Sync Tool command line flags now match those offered by the Retrieval Tool.
- UI updates which provide better visual cues for which storage provider is in use.
- A host of bug fixes and small tweaks

For more details about specific changes in release 0.8, see the JIRA issue tracker.

Release 0.7

Released: Oct 28, 2010

The primary features of release 0.7 are:

- A new Retrieval Tool, a companion to the existing Sync Tool, which is a command-line tool for the retrieving content from DuraCloud spaces.
- A new Bulk Bit Integrity Checker service, which can be run over content stored in Amazon to create a listing of checksums calculated for each file. This new service pairs well with the Bit Integrity Checker service (previously known as the Fixity Service), allowing the heavy processing to be handled in parallel using Hadoop on an Amazon EC2 cluster, and the simpler checks and comparisons to be handled by the DuraCloud instance.
- A new Duplicate on Demand service, which can be used to copy files from the primary Amazon store into another storage provider. This service pairs well with the Duplicate on Upload service (previously known as the Replication Service) by performing the large up-front copy using Hadoop on an Amazon EC2 cluster, then allowing Duplicate on Upload to watch for and add new files as they are uploaded.
- Integration of a new storage provider: Microsoft Windows Azure.

For more details about specific changes in release 0.7, see the JIRA issue tracker.

Note that there have been issues discovered during testing of the Bulk Image Transformer (included in release 0.6 as the Bulk Image Conversion Service). If you choose to run this service, it is recommended that the size of images being used be kept under 100MB. The likelihood of success appears to increase with server size, and number of servers being set to 3 or more is recommended. If you do run this service, please note the data set and configuration and make us aware of the outcome.

Release 0.6

Released: Sept 03, 2010

The primary features of release 0.6 are:

- Addition of a new Fixity Service, which allows for bit integrity checking on content stored within DuraCloud. This service has many options to fit various usage needs. For more information, see the Fixity Service page.
- Addition of a new Bulk Image Conversion Service, which, like the Image Conversion Service, allows for converting images into other formats. This new service, however, makes use of Hadoop in the background to run the conversion using multiple servers, allowing for much higher overall throughput.
- An updated handling of space metadata so that spaces with a large number of content items will not cause slow response times. Now spaces with more than 1000 items will initially show a value of 1000+ as the number of items in the space. DurAdmin, the administrative interface, will then calculate the total number of items on the fly.
- The Sync Tool has a new option (-e) which will cause the tool to exit once it has completed syncing rather than continually monitoring for changes. This makes it easier for administrators to include the Sync Tool in scripts which run daily or weekly to ensure all local content is moved to DuraCloud.
- DurAdmin now provides a way to delete groups of content items and spaces in one step.
- A host of bug fixes and small tweaks

For more details about specific changes in release 0.6, see the JIRA issue tracker.

Release 0.5

Released: July 28, 2010

The primary feature of release 0.5 is the addition of a completely new administrative user interface. This UI, called DurAdmin like its predecessor, provides for easy access to the primary features of DuraCloud.

For more details about specific changes in release 0.5, see the JIRA issue tracker.

Release 0.4

Released 0.4.1: June 30, 2010

- This build release is the first publicly available
- It primarily provides clean-up of projects and tests

Released 0.4: June 21, 2010

The primary features added in release 0.4 of DuraCloud were:

- Media Streaming Service
    - Provides a way to enable streaming for video and audio files as well as providing an example media player.
- Logging moved to SLF4J over Logback
    - Provides greater consistency in log output and greater flexibility in log configuration

For more details about specific changes in release 0.4, see the JIRA issue tracker.

Release 0.3

Released: May 17, 2010

The primary features in the third pilot release of DuraCloud are:

- Security
    - All DuraCloud applications now require authentication prior to performing write activities
    - Read activities on 'closed' spaces also require authentication, but 'open' spaces allow anonymous read access
- Sync Tool
    - Provides a command line utility for keeping DuraCloud content synchronized with the local file system

Other improvements in the 0.3 release:

- Image Conversion Service
    - Adds an option to convert images to the (web standard) sRGB color space
    - Adds the capability to perform multiple conversions at once (providing the compute capacity is available) and provides more frequent activity feedback through the continual writing of the conversion output file
- DuraStore
    - Adds an option for users to provide MD5 checksum when adding content. This disables the in-transfer MD5 computation (providing improved performance) and compares the final MD5 computed by the storage provider with the user provided MD5.

For more details about specific changes in release 0.3, see the JIRA issue tracker.

Release 0.2

Released: Feb 19, 2010

The second pilot release of DuraCloud focused on providing access to services which can be run over content, as well as improvements to the storage foundation provided by the first release.

Services available as of release 0.2:

- J2K service - serves J2K images, provides a J2K image viewer
- Image Conversion service - converts image files from one format to another
- Replication service - replicates content stored in one provider to another upon content upload
- Web Application Utility service - infrastructure service required by J2K service (allows for deployment of web applications)
- ImageMagick service - infrastructure service required by Image Conversion service (provides access to ImageMagick utilities)

Service functions available as of release 0.2:

- Services may be deployed with configuration
- Available and deployed services may be listed
- Deployed service configuration may be viewed and updated
- Deployed service properties may be viewed
- Deployed services may be undeployed and redeployed

New storage functions available as of release 0.2:

- Space content may be listed in chunks with an optional prefix filter
- Space and content metadata may be edited via the UI
- Space and content metadata tags may be added/removed via the UI

For more details about specific changes in release 0.2, see the JIRA issue tracker. Note that while most items included in the release are listed in the tracker, we migrated to using JIRA while working on release 0.2, so issues completed prior to the migration are not included.

Release 0.1

Released: Nov 2, 2009

The first pilot release of DuraCloud laid the foundation for storage across underlying providers.

Through either the web-ui or via direct REST calls

- underlying providers may be listed
- spaces may be created/deleted
- content may be uploaded/downloaded/deleted
- metadata may be viewed
- metadata may be modified
    - *modification is fully supported through the REST API*
    - *modification is partially supported through the web-ui*

DuraCloud Architecture

This diagram dep
are likely to occu
case where Dura
backup system fo
within an institutio
repository in this
being used as a

could be replaced
software system w
the file system.

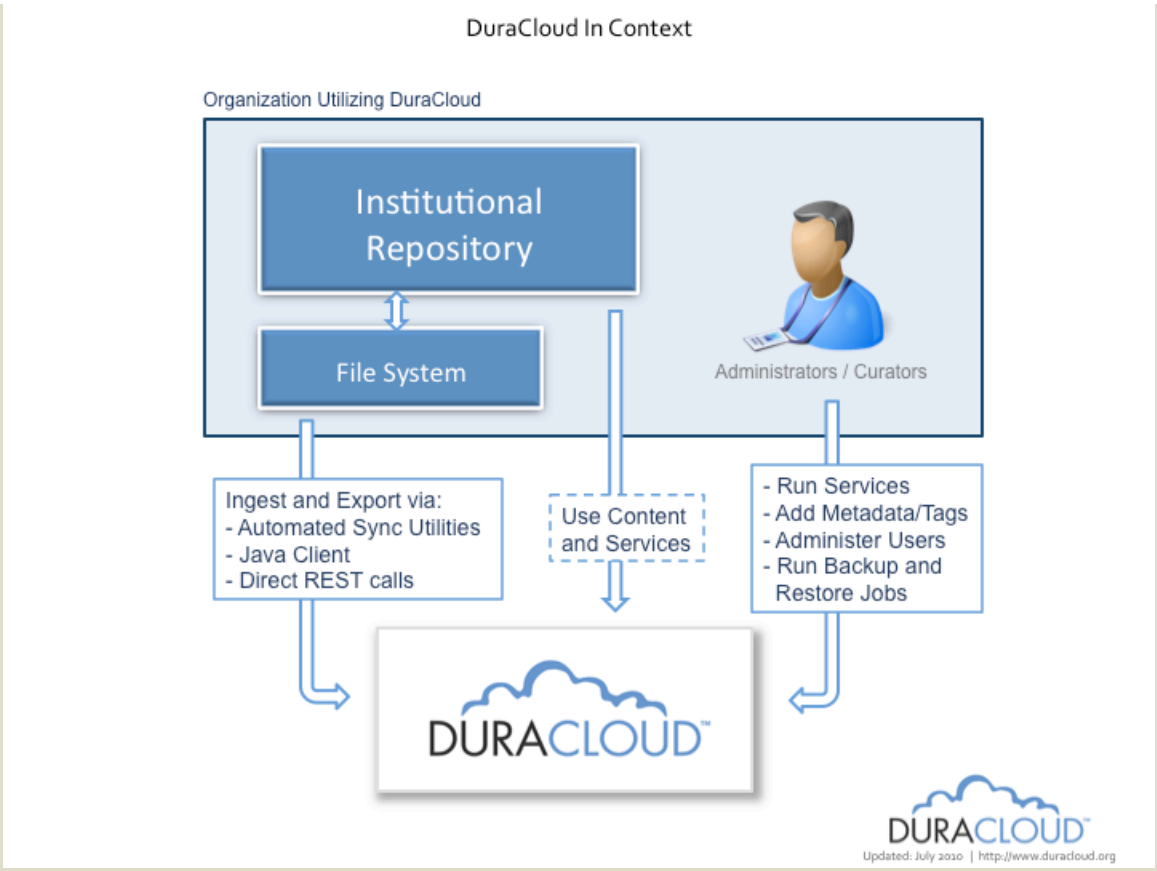
DuraCloud Intera

There are three ty
illustrated here:

1. Moving fi
system a
    - T
    h
    s

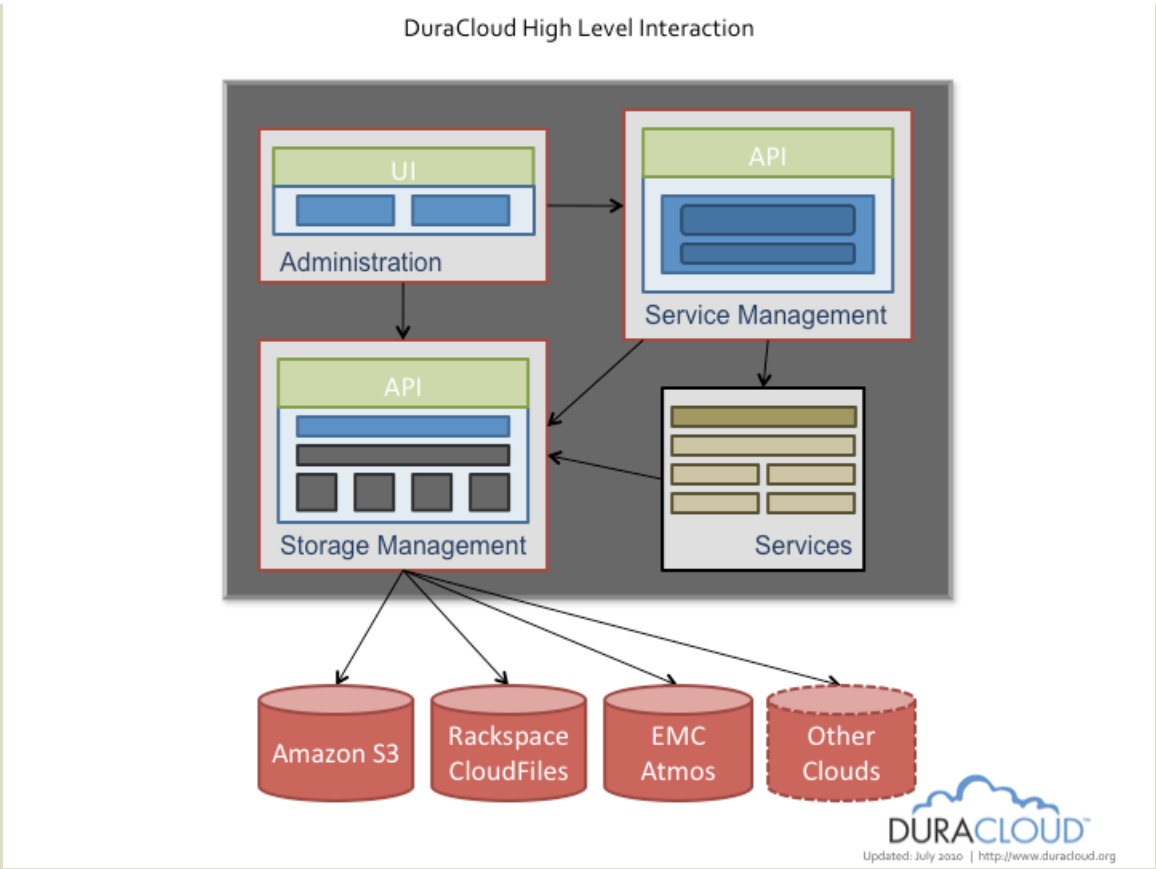
2. Applicati
repositor
content s
well as D
    - A
    h
    a
    a
    h
    T
    c
    a
    t
    c
    s
    h

3. Administ
    - T
    a
    a
    t
    h
    t
    i



This diagram dep
an example Prima
User Compute
Instance (*Primary*
UCI: gray box), a
four underlying
storage providers

**Primary User
Compute Instanc
(Primary UCI)**



Once a user crea a DuraCloud account, a new *Primary UCI* is started on their behalf and initializ based on the acc configuration. Most of the interaction with DuraCloud will be through the *Prima UCI*.

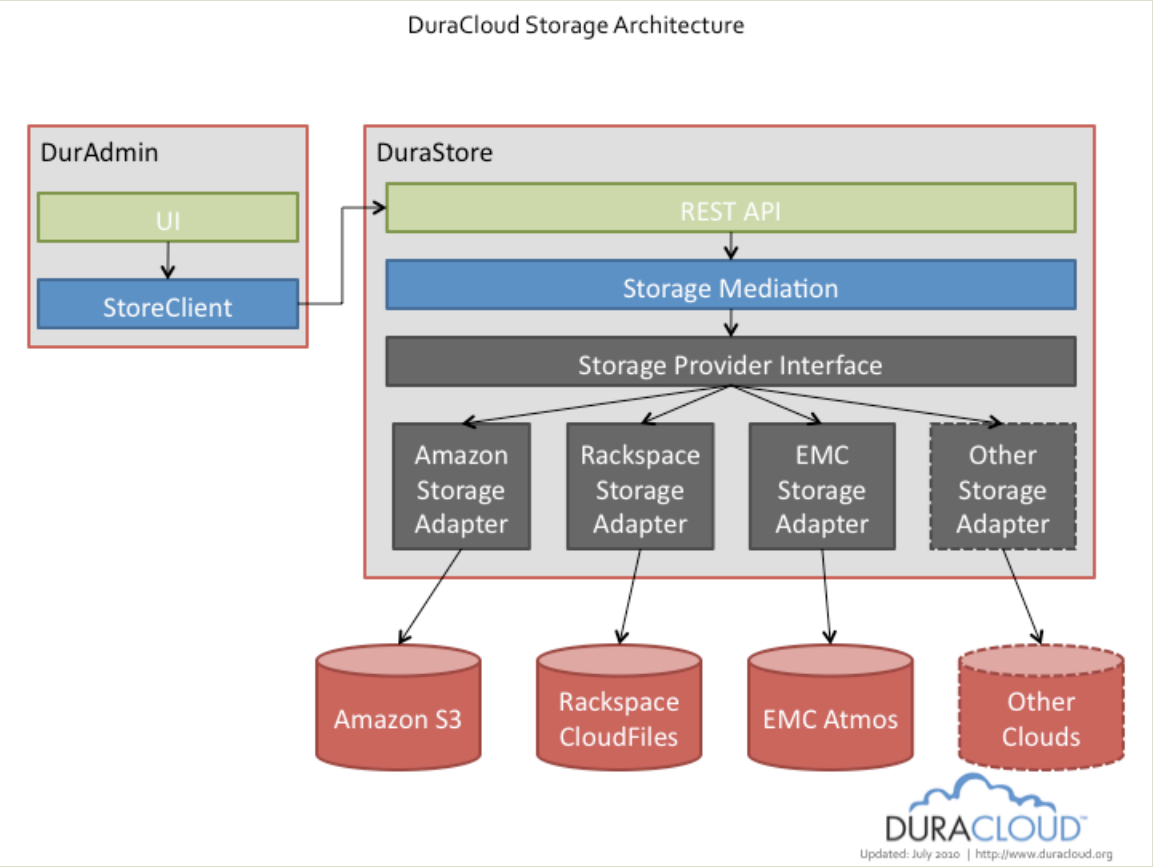
Fundamentally, th *Primary UCI* cons of a set of web applications that expose three gen capabilities

1. storage manager
2. service manager
3. a graphic web interface providing convenie access to the featu in 1 and

The *Primary UCI* assigned a public address which ca be mapped to a u owned domain na or a domain name provided by DuraCloud. Through this base URL and publishe storage and servi REST APIs, the u can by-pass the Administration UI *DurAdmin*) and interact directly w the Storage Management and Service Manager components.

The Services component and supporting storag providers are addressed below.

This diagram depicts a further introspection of the storage management component (*DuraStore*) and *DurAdmin*'s interaction therewith.



DuraStore

DuraStore is a web application deployed on the *Primary UCI* which is responsible for mediating the interaction with content stored in any one or more of the underlying storage providers. It exposes these content management functions via a public-facing REST API. Such functions include but are not limited to

- content upload / download / deletion
- metadata and tag creation / deletion
- space* creation / deletion
- space* visibility (public or private)

*A *Space* within the context of DuraCloud is a container within which content is stored.

Below the REST API is a *Storage Mediation* layer that translates requests into specific calls for the appropriate *Storage Provider* adapter implementation. One of the value propositions of DuraCloud is its ability to abstract away the details and idiosyncrasies of the varied underlying storage provider APIs. For each of the supported storage providers (currently: Amazon S3, Rackspace CloudFiles, and

EMC Atmos) we have created adapters that implement a common *Storage Provider Interface*.
In this way, new storage providers can be dropped into the architecture by implementing the same interface.

DurAdmin

DurAdmin is shown here to highlight the fact that its interaction with *DuraStore* happens exclusively through the *StoreClient* (provided with the application distribution), a Java wrapper over the storage REST API.

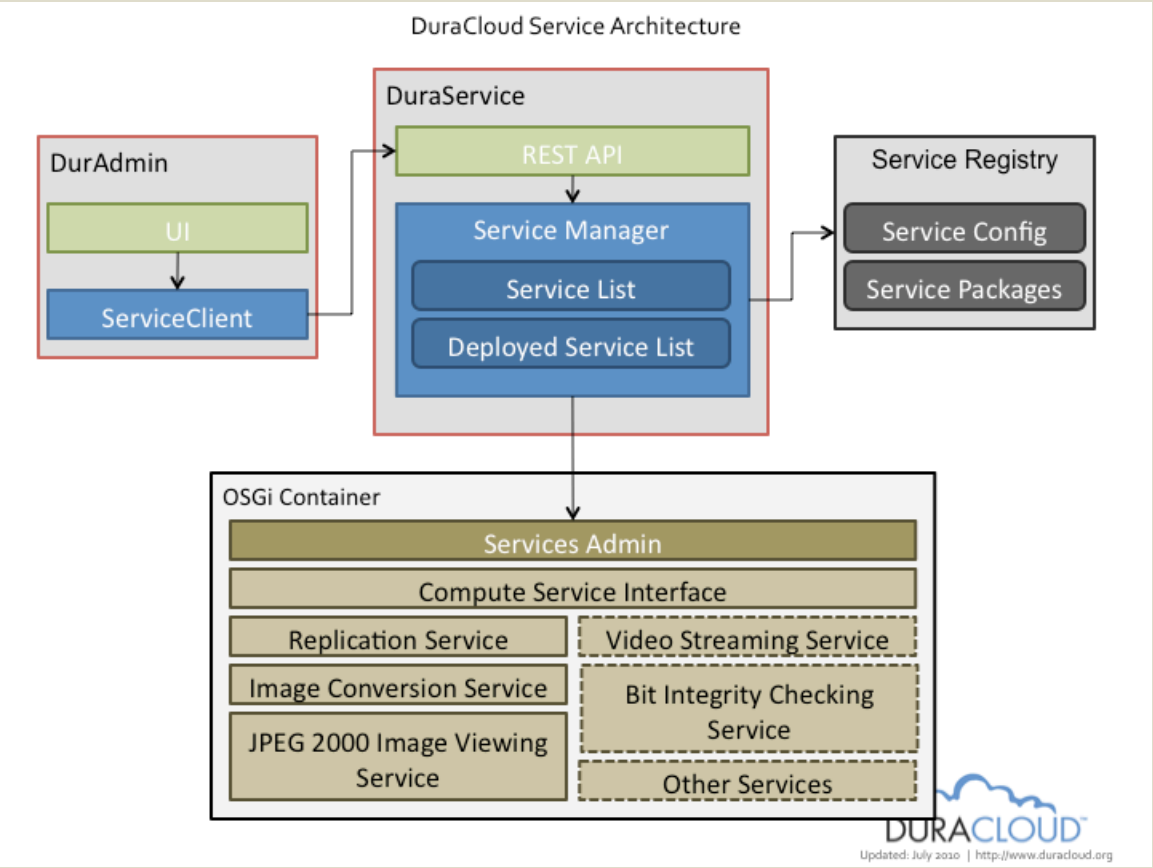
This diagram is th services counterp the previous stora management diag above.

DuraService

DuraService is a application deploy on the *Primary U* which is responsi mediating the interactions with deployed or availa user services.
It exposes these service managem functions via a public-facing RES API.
Such functions in but are not limited

- service deployme undeploy
- runtime service configura
- service property listings

The notion of a se in the DuraCloud context can be de as any compute a



that can be starte
stopped, configur
and monitored as
implementation o
*Compute Service
Interface*.
DuraService pass
service managem
requests to the *S
Manager* which h
access to

- available
services
their
configura
options f
the *Dura
Service
Reposito*
- deployed
services
with their
current
configura
and prop
from *Ser
Admin*

Service Registry

DuraCloud mainta
Service Registry
contains a set of
versioned *Service
Packages* availab
runtime deployme
These *Service
Packages* consist
OSGi bundles an
support files that
be community-pro
as well as develo
by DuraSpace.
The *Service Conf*
an XML documen
acts as a table of
contents for avail
services and their
associated
configuration opti

**OSGi Container:
Services Admin**

As an implementa
note, the need to
dynamically start,
and reconfigure
services at runtim
without performin
application restar
drove the decisio
employ an *OSGi
Container* as
DuraCloud's serv
hosting environm
Although the *OSG
Container* is depi
as a single compo
deployed in the
Primary UCI, if it
advantageous for
particular service

run on a *Manage*
a separate *OSGi Container* could ju
well be deployed
another compute
instance.

Services Admin a
each of the depic
Service elements
independent, OS
bundles.
All requests from
DuraService for
starting, configuri
etc., services pas
through *Services Admin*.
Such requests ar
mediated to the
appropriate servic
within the *OSGi Container* via the
common *Comput
Service Interface*.
Even though the
management of a
given service req
a representative
bundle that expor
implementation o
Compute Service Interface, a variet
actual service fla
supported.
Such service
implementations
include

- web applicatio (e.g. Adore-Dj
- comman utilities (e ImageMa
- pure Java Replicati Service)
- external compute instance

DurAdmin

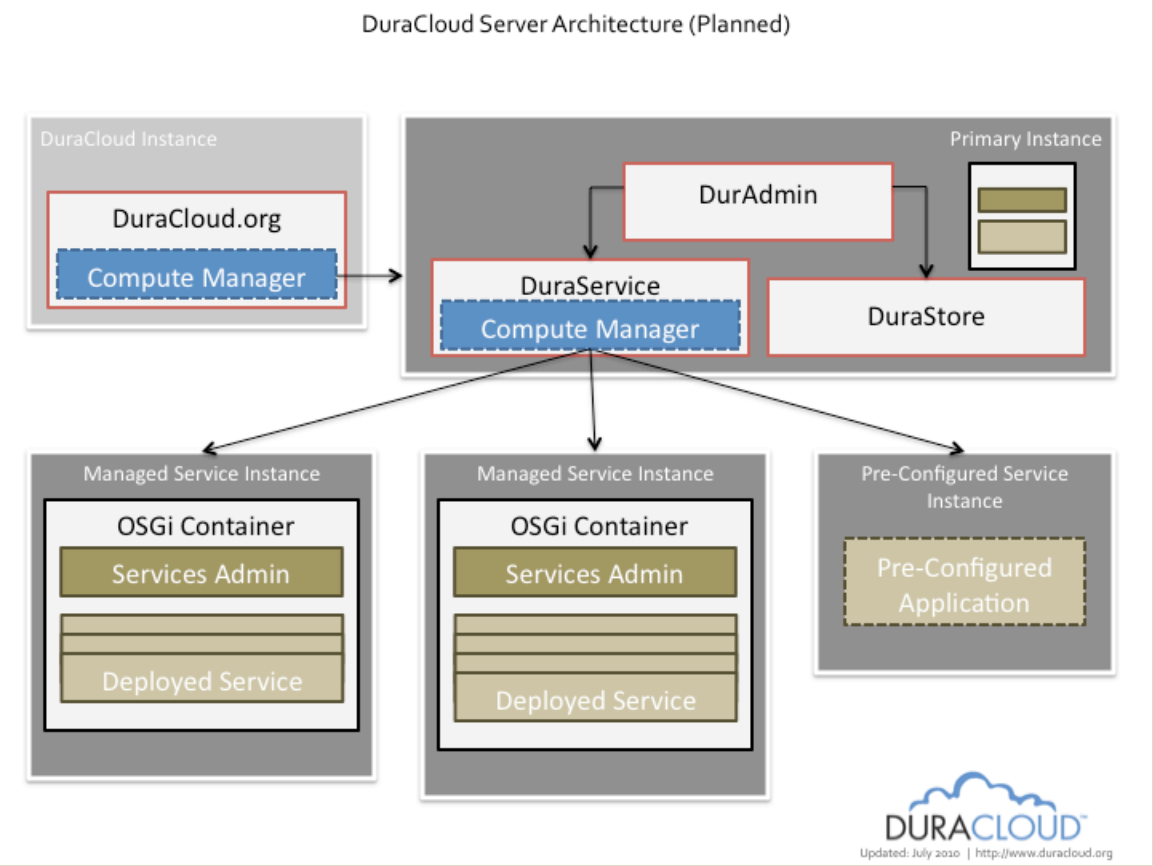
As in the previous
diagram, *DurAdm*
shown here to hig
the fact that its
interaction with
DuraService happ
exclusively throug
ServiceClient (wh
provided with the
application
distribution), a Ja
wrapper over the
services REST A

This diagram pull
up to a higher lev

two, depicting the
deployment topol
planned architect
been deployed.

DuraCloud is com
compute instance

1. **DuraClo**
   -
   -
   -
   -
2. **Primary Instance**
   -
   -
   -
   -
3. **Managed**
   -
   -
   -
4. **Pre-Con**
   -
   -
   -



- a
- I
  n
  g
  a

This diagram illus
the multiple levels
security provided
DuraCloud.

Instance Firewall

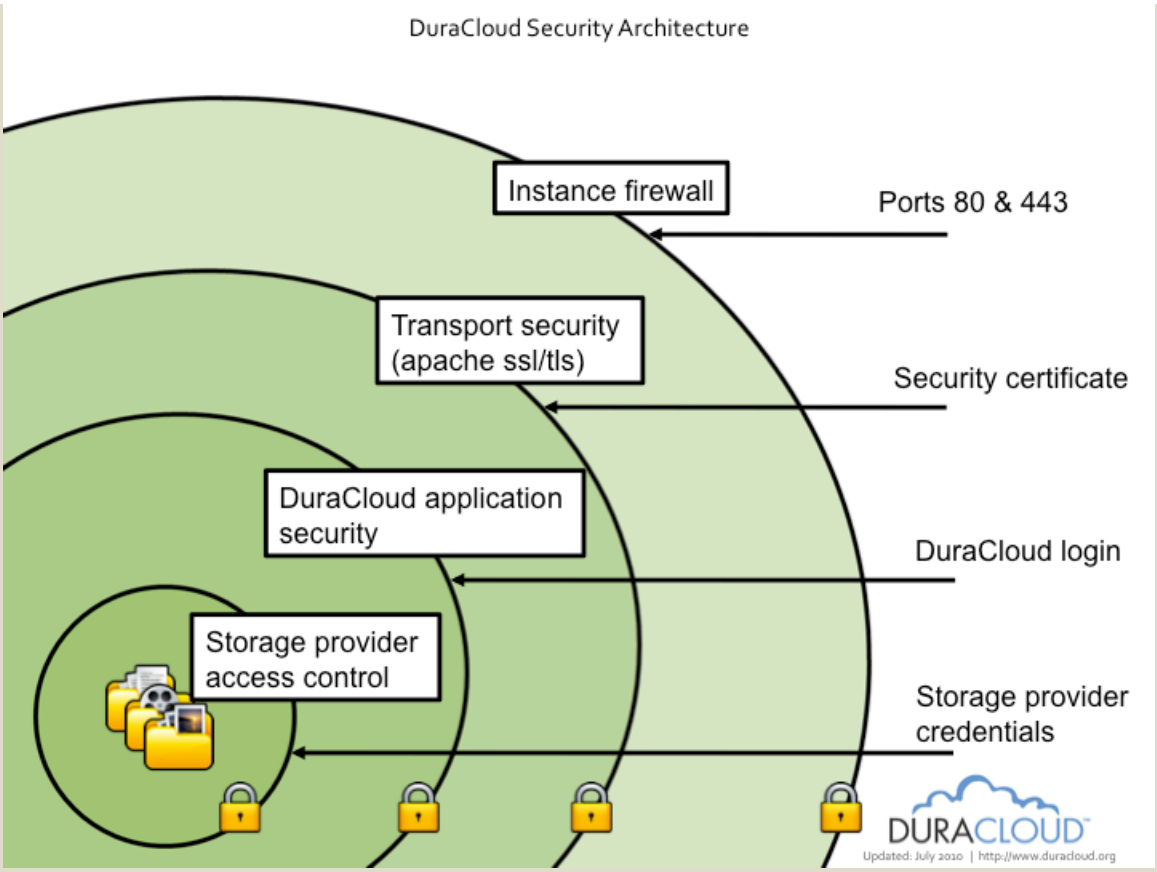
- The first
  defense
  DuraClou
  the firewa
  surround
  the *Prima*
  UCI. This
  firewall is
  construct
  allow
  connectio
  only thro
  the stand
  HTTP (80
  HTTPS (
  ports. Th
  requests
  coming in
  HTTP are
  redirecte
  HTTPS t
  ensure s
  transport

Transport Securi

- As noted
  above, al
  communi
  with
  DuraClou
  via HTTP
  meaning
  all inform
  that goes
  comes ou
  DuraClou
  encrypte
  transport
  ensures
  transmiss
  into or ou
  DuraClou
  be read o
  by the
  intended
  recipient.

DuraCloud Application Secu

- The
  DuraClou
  applicatio
  itself con
  access to
  content t

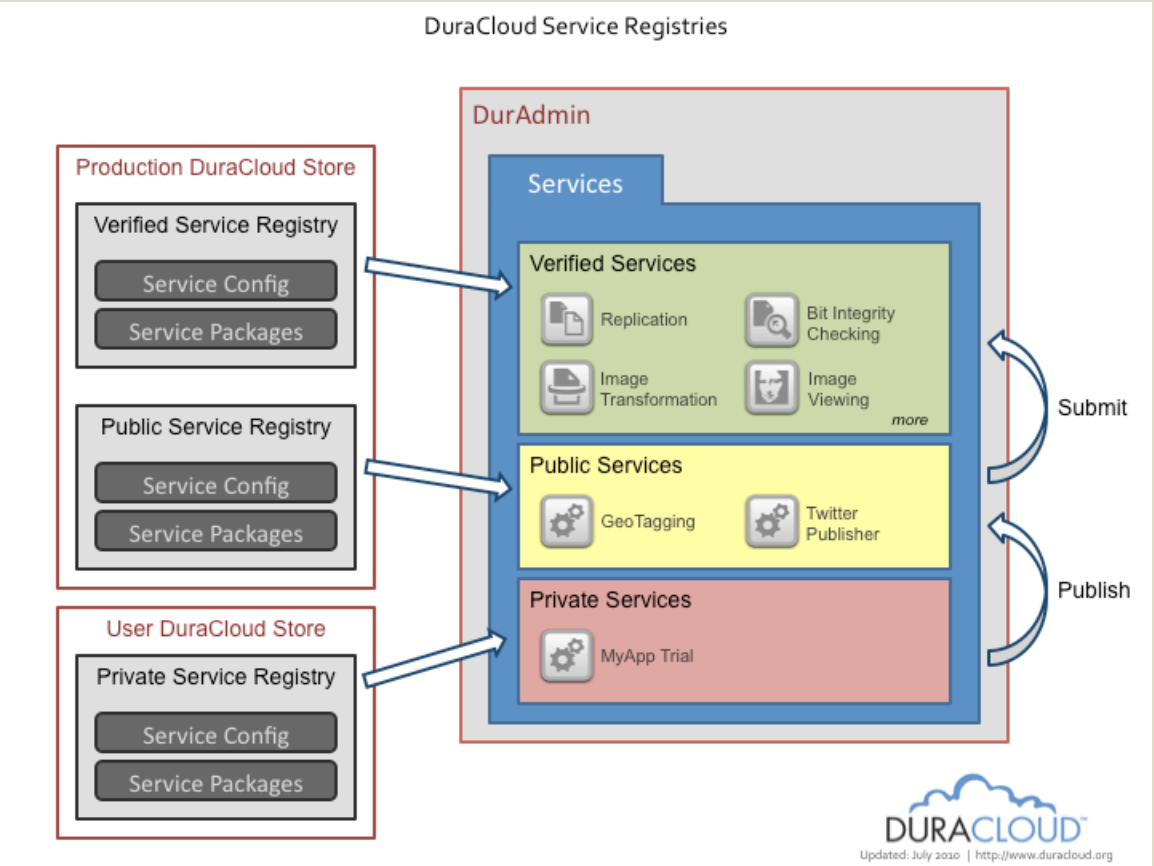


ensure th
only regis
users (wh
must be
by an
Administ
with the p
role, can
perform t
(such as
downloa
content).

Storage Provider Access Control

- DuraClou
uses the
access c
mechanis
the unde
storage
providers
lock dow
access,
ensuring
all action
involving
content n
occur thr
the Dura
applicatio
This allow
DuraClou
control a
to conten
describe
above wi
concerns
about wh
might be
occuring
underlyin
providers

This diagram illustrates the plan service registry architecture. A service registry is storage location which holds a set DuraCloud servic packages. The DuraCloud softwa can obtain information from t registry in order t display the list of services available from that registry, allow for the deployment of the services containe therein. This diag indicates the plan have three distinc service registries slightly different purposes:



Private Service Registry

- A private service registry w be where DuraClou account holders p their personal services. These services likely hav been cre by or for user to fu specific content processi needs. N other DuraClou users wil have acc to service located i private service registry. illustrate the diagr these services be store a user's personal storage a

Public Service Registry

- The Publ Service Registry be where services private service registries be publis to make available others to The inter is to allo these services be availa to use fo free or fo fee. The determin of the fee would be by the creator o service. While the services be availa to all, the use will b

on a "at y
own risk"
basis. Th
services
the Publi
Service
Registry
be stored
a centrali
DuraClou
storage
location.

Verified Service Registry

- Services
the Verifi
Service
Registry
those wh
have bee
submitte
and verifi
by the
DuraClou
committe
team. Th
services
tested to
ensure th
they func
properly
within the
DuraClou
system a
perform t
actions th
are inten
to perfor
They, like
services
the Publi
Service
Registry,
likely be
available
either for
or for a fe
as
determin
by the
service
creator.
These
services
also be
stored in
centralize
DuraClou
storage
location.

DuraCloud Sync Tool

Introduction

The Sync Tool is a utility which was created in order to provide a simple way to move files from a local file system to DuraCloud and subsequently keep the files in DuraCloud synchronized with those on the local system.

Download

Download the sync tool here.

How the Sync Tool Works

- When you run the Sync Tool for the first time, you must include DuraCloud connection information (host, port, username, password) as well as the space where you would like all of your files stored. You must also provide a list of directories which will be synced to DuraCloud and a directory for the Sync Tool to use for its own work.
- When the Sync Tool starts up, it will look through all of the files in each of the local content directories and add them to its internal queue for processing. Each of those files will then be written to your DuraCloud space. As this initial write is happening a listener is set up to watch for any file changes within each of the content directories. As a change occurs (a file is added, updated, or deleted), that change is added to the queue, and the appropriate action is taken to make the DuraCloud space consistent with the local file (i.e. the file is either written to the space or deleted from the space.)
- You can stop the Sync Tool at any time by typing 'x' or 'exit' on the command line where it is running. It will stop all listeners, complete any file transfers that are in progress, and close down.
- When you restart the Sync Tool, if you point it at the same work directory, it will pick up where it left off. While the Sync Tool is running, it is constantly writing backups of its internal queue, so it first reads the most current backup and begins processing the files there. It then scans the content directories to see if there are any files which have been added or updated since the last backup, and it also pulls a list of files from the DuraCloud space and scans that list to see if any local files have been deleted. Any changes detected are added to the internal queue, and the Sync Tool continues to run as usual.

Operational notes

- Restarting
  - You can perform a restart of the Sync Tool by using the -g command line option to point to the Sync Tool configuration file, which is written into the work directory (named synctool.config)
  - If you would like the Sync Tool to perform a clean start rather than a restart (i.e. you would like it to compare all files in the content directories to DuraCloud) you will need to either point it to a new work directory, or clear out the existing work directory.
  - The Sync Tool will perform a clean start (not a restart) if the list of content directories is not the same as the previous run. This is to ensure that all files in all content directories are processed properly.
- Collisions
  - The Sync Tool allows you to sync multiple local directories into a single space within DuraCloud. Because of this, there is the possibility of file naming collisions, where two local files resolve to the same DuraCloud ID. If this happens, one file will be overwritten by the other. There are a few ways to ensure that this does not occur:
    - Ensure that the top level files and directories within the set of content directories do not have overlapping names.
    - Sync only a single directory to a space. You can run multiple copies of the Sync Tool, each over a single local directory, syncing to its own DuraCloud space.
- Work Directory - these files and directories can be found in the work directory (specified using the -w command line parameter)
  - Config Files
    - When the Sync Tool starts up, it writes the list of parameters and values provided by the user on startup to a file called synctool.config in the work directory. This file can be used to restart the Sync Tool, using the -g parameter to point to the file's location. You can also restart the Sync Tool by indicating the same set of options as used originally. The -g parameter is for convenience only and is not required in any circumstance. Note that this file is overwritten each time the Sync Tool is run with a different set of parameters, so you may choose to copy the file elsewhere (or give it a new name) if you would like to keep a copy of a particular configuration set.
    - You may also see a file named synctool.config.bak in the work directory which is used to compare against the current config in order to determine if a restart is possible. In order for a restart to occur, the list of content directories (-c parameter) must be the same as the previous execution of the tool, and there must be at least one changed list backup (see below.)
  - Changed List Directory
    - While the Sync Tool is running it is constantly updating the list of files which have been changed (when starting the first time, this includes all files in the directories that need to be synced). In order to allow the Sync Tool to restart after it has been stopped, this list of files is continually backed up into the *changedList* directory. There is no reason to edit these files, but you may choose to delete the *changedList* directory along with the config files mentioned above to ensure that the Sync Tool does not attempt to perform a restart.
  - Logs Directory
    - Information about what the Sync Tool is doing while it is running can be found in the sync-tool.log file. It is a good idea to monitor this file for errors and warnings as this information is not printed to the console.
    - The duracloud.log file is useful for application debugging when the information in the sync-tool.log file is insufficient to understand a problem.

Prerequisites

- You must have Java version 6 or above installed on your local system. If Java is not installed, you will need to download and install it. To determine if the correct version of Java is installed, open a terminal or command prompt and enter

```
java -version
```

  The version displayed should be 1.6.0 or above. If running this command generates an error, Java is likely not installed.

- You must have downloaded the Sync Tool. It is available as a link near the top of this page.

Starting the Sync Tool

- To run the Sync Tool, open a terminal or command prompt and navigate to the directory where the Sync Tool is located
- To display the help for the Sync Tool, run

```
java -jar synctool-0.7.0-driver.jar
```

- When running the Sync Tool for the first time, you will need to use these options:

Short Option	Long Option	Argument Expected	Required	Description	Default Value (if optional)
-h	--host	Yes	Yes	The host address of the DuraCloud DuraStore application	
-r	--port	Yes	No	The port of the DuraCloud DuraStore application	443
-i	--store-id	Yes	No	The Store ID for the DuraCloud storage provider	The primary storage provider is used
-s	--space-id	Yes	Yes	The ID of the DuraCloud space where content will be stored	
-u	--username	Yes	Yes	The username necessary to perform writes to DuraStore	
-p	--password	Yes	Yes	The password necessary to perform writes to DuraStore	
-c	--content-dirs	Yes	Yes	A list of the directory paths to monitor and sync with DuraCloud. If multiple directories are included in this list, they should be separated by a space.	
-w	--work-dir	Yes	Yes	The state of the sync tool is persisted to this directory	
-f	--poll-frequency	Yes	No	The time (in ms) to wait between each poll of the sync-dirs	10000 (10 seconds)
-t	--threads	Yes	No	The number of threads in the pool used to manage file transfers	3
-m	--max-file-size	Yes	No	The maximum size of a stored file in GB (value must be between 1 and 5), larger files will be split into pieces	1
-d	--sync-deletes	No	No	Indicates that deletes performed on files within the content directories should also be performed on those files in DuraCloud; if this option is not included all deletes are ignored	Not set
-x	--exit-on-completion	No	No	Indicates that the sync tool should exit once it has completed a scan of the content directories and synced all files; if this option is included, the sync tool will not continue to monitor the content dirs	Not set

- When the Sync Tool runs, it creates a backup of your configuration in the work directory that you specify. When running the tool again, you can make use of this file to keep from having to re-enter all of the options specified on the initial run. In this case you need only a single option:

Short Option	Long Option	Argument Expected	Required	Description
-g	--config-file	Yes	Yes	Read configuration from this file (a file containing the most recently used configuration can be found in the work-dir, named synctool.config)

- An example for running the Sync Tool

```
java -jar synctool-0.8.0-driver.jar -w C:\tools\synctool\backup -c C:\files\important -f 2000 -h
test.duracloud.org -s important-dir-backup -t 5 -u myname -w mypassword
```

Runtime commands

- While the Sync Tool is running, these commands are available. Just type them on the command line where the tool is running.

Short Command	Long Command	Description
x	exit	Tells the Sync Tool to end its activity and close
c	config	Prints the configuration of the Sync Tool (the same information is printed at startup)
s	status	Prints the current status of the Sync Tool
l <Level>	N/A	Changes the log level to <Level> (may be any of DEBUG, INFO, WARN, ERROR)
h	help	Prints the runtime command help

DuraCloud Java Clients

StoreClient

Introduction

DuraCloud provides access to files stored in cloud storage systems through an application called DuraStore. DuraStore is installed and running on your DuraCloud instance and can be accessed via a REST interface. In order to aid Java developers in communicating with DuraStore, a Java client, called StoreClient was written.

Using StoreClient

To use the StoreClient, you will need all of the jars included in the libs directory of the download below to be available on your classpath. You will then be able to write code using the provided Javadocs to interact with the StoreClient.

Example Client

An example Java class has been provided to assist in set up and testing, as well as a starting point for writing your client code. The example client (found in ExampleClient.java) includes a simple main class to print out the list of spaces and space metadata available in your DuraStore account. To run the example:

- Extract the store-client zip file
- Update the HOST, PORT, USERNAME, and PASSWORD constant values in ExampleClient.java as needed to connect to your DuraCloud instance.
- Make sure Ant is available on your path and type "ant" to compile the example.
- Type "ant run" to run the example, which will print space names and metadata to the console.

ServiceClient

Introduction

DuraCloud provides management capabilities to deploy, undeploy, configure, and gain status of the services within DuraCloud through an application called DuraService. DuraService is installed and running on your DuraCloud instance and can be accessed via a REST interface. In order to aid Java developers in communicating with DuraService, a Java client, called ServiceClient was written.

Downloads

1. Store Client - Includes storeclient.jar and all dependent jars as well as Javadocs
2. Services Client - coming soon

DuraCloud Presentations

Presentations

- DuraCloud Pilot Program Webinar (video recording) – July 28, 2010*
- DuraCloud Pilot Program Panel at NDIIPP – July 21, 2010

- DuraCloud Fixity Session at NDIIPP – July 21, 2010
- DuraCloud Plenary Session at NDIIPP – July 21, 2010
- DuraCloud Plenary Session at OR2010 – July 7, 2010
- DuraCloud Session at CNI (video recording) – April 2010
- DuraCloud Session at Repositories in Cloud Seminar (video recording) – February 2010
- DuraCloud webinar presentation (video recording) – December 16, 2009*
- DuraCloud Session at Internet2 Meeting – October 2009
- DuraCloud Plenary Session at NDIIPP – September 2009
- DuraCloud Plenary Session at NDIIPP – June 2009
- DuraCloud Session at PASIG – June 2009
- DuraCloud Plenary Session at OR09 – May 2009
- DSpace/Fedora Collaboration Effort -- October 2008

*Please note that to play the WebEx audio recording, you will need to download and install the WebEx player available here.

DuraCloud Administration

This document details some of the considerations of concern to a DuraCloud administrator.

Naming restrictions

1. Space names
    a. The following restrictions apply to user-defined space names
        - only lowercase letters, numbers, periods, and dashes
        - no adjacent pair of "-" and/or "."
        - no number immediately following the last "."
        - between 3 and 63 characters
        - must start with a letter
        - may not end with a dash
    b. Note: Users can provide space names through the REST-API that do not follow these conventions, but the space actually created will have a different name under the covers.
2. Reserved space names
    a. Due to some specific operations exposed through the durastore REST-API, the following names are unavailable as user-defined space names
        - stores
        - spaces
        - security
        - task
3. Content object names
    a. The only restrictions are that a content object name
        - cannot include a question mark character
        - it is limited to 1024 bytes

DuraCloud Services

Introduction

The DuraCloud application provides a set of services which can be deployed and used for a variety of purposes, primarily to process the content which has been loaded into DuraCloud storage. The following list of services describes how each service is expected to be used and the options available for tailoring the service to your needs.

Note that the current way all DuraCloud services are configured, they will not auto-restart if they fail during processing. If you notice a failed job state, simply redeploy the service. Automatic service recovery is on the roadmap for DuraCloud development in the future and will be made available as soon as possible.

Duplicate on Upload

Description:

The Duplicate on Upload service provides a way to ensure that the content added to DuraCloud is stored with at least two storage providers. The Duplicate on Upload service performs on-ingest duplication of content. This means that once the Duplicate on Upload service is deployed, it watches for all content that is added to your DuraCloud account, determines if it should be copied to another DuraCloud store, and if so, performs the copy. All content that is copied will be placed in an identically named space in the secondary storage location.

Configuration Options:

1. Watch this store for uploads: The primary storage location which DuraCloud will monitor for file additions. When files are added to this store, they will be copied to the secondary store.
2. Copy to this store: The secondary store where content will be copied after it has been added to the primary store.

Duplicate on Demand

Description:

The Duplicate On Demand service provides a simple way to duplicate content from one space to another. This service is primarily focused on allowing the duplication of content from the primary storage provider to a secondary provider. To begin, a source space is chosen, along with a store and space to which content will be duplicated. The service then performs a copy of all content and metadata in the source space to the duplication space, creating the space if necessary. When the service has completed its work, a results file will be stored in the chosen space and a set of files (primarily logs) created as part of the process will be stored in the work space.

Configuration Options:

1. Source Space: DuraCloud space where source files can be found
2. Copy to this store: DuraCloud store to which content will be copied
3. Copy to this space: DuraCloud space where content will be copied
4. Standard vs. Advanced configuration
    a. Standard mode automatically sets up the service to be run
    b. Advanced mode allows the user to configure the number and type of servers that will be used to run the job
        i. Number of Server Instances: The number of servers to use to perform the duplication task.
        ii. Type of Server: The type (size) of server used as perform the task. The larger the server, the faster the processing will occur. Larger servers also cost more than smaller servers to run. For more information, see the Amazon EC2 documentation.

Image Server

Description:

The Image Server provides a viewer for image files through use of the Djatoka image server. While this service is geared towards serving JPEG 2000 images, it supports multiple image file types by converting them to JPEG 2000 format on the fly.

Note that the current implementation of this service requires that spaces be set to OPEN in order to use the viewer to view image files.

Configuration Options:

None

Media Streamer

Description:

The Media Streamer provides streaming capabilities for video and audio files. The service takes advantage of Amazon Cloudfront streaming, so files to be streamed must be within a space on an Amazon provider. Also, all media to be streamed by this service needs to be within a single space.

Amazon Cloudfront streaming uses the Flash Media Server to host streaming files over RTMP. File formats supported include MP3, MP4 and FLV among others. For a full listing of supported file types see the Flash Media Server documentation.

Configuration Options:

1. Source Media Space: The DuraCloud space where the source video and audio files to be streamed are stored. The Media Streamer service attempts to stream all files in this space.
2. Viewer Space: A DuraCloud space where example viewer files will be stored. After the service has started, this space will include a playlist including all items in the source media space as well as example html and javascript files which can be used to display a viewer.

Output Files, the following files can be found in the configured Viewer Space once the Media Streamer is running:

- player.swf - The flash-based video player JWPlayer
- playlist.xml - A playlist, created by DuraCloud, which includes all of the items in your Source Media Space
- playlistplayer.html - An HTML file, created by DuraCloud, which uses JWPlayer to display the items in the playlist
- singleplayer.html - An HTML file, created by DuraCloud, which uses JWPlayer to display a single media file (typically, the first item in your Source Media Space)
- stylish.swf - A supplementary flash file used to style the JWPlayer
- swfobject.js - A javascript file (available from here) used to embed the JWPlayer on a web page
- viewer.js - A javascript file, created by DuraCloud, used to simplify the loading of JWPlayer

All of the output files are intended as examples only. Their purpose is give developers a starting point for embedding video streamed by DuraCloud on their own web pages. Feel free to use, modify, ignore, or delete these files.

Bit Integrity Checker

Description:

The Bit Integrity Checker provides the ability to verify that the content held within DuraCloud has maintained its bit integrity. There are two modes of operation.
Modes:

1. Verify integrity of a Space
2. Verify integrity of an item list

When running in the *Verify integrity of a Space* mode, the checker performs the following steps

- collect the content hash values for each item from the underlying storage provider
- stream through each item recalculating their hashes
- compare the two listings

When running in the *Verify integrity of an item list* mode, the checker performs the following steps

- stream through each item in the provided listing, recalculating their hashes
- compare the newly generated listing with the provided listing

Configuration Options:

1. Stores: The underlying storage provider over which the service will run
2. Space containing content items: The DuraCloud space in which the content items to be verified reside
3. Verify integrity of an item list mode
    a. Input listing name: Name of the content item which contains the listing of items over which to run the service

Service Ouputs
All outputs of this service are placed in the system space, *x-service-out*.

1. bitintegrity/fingerprints-gen-<spaceId>-<date>.csv
    - interim listing generated with hash values from underlying storage provider
2. bitintegrity/fingerprints-<spaceId>-<date>.csv
    - interim listing with hashes recalculated from content streams
3. bitintegrity/fixity-report-<spaceId>-<date>.csv
    - final report with status of integrity check

Bit Integrity Checker - Tools

Description:

The Bit Integrity Checker Tools provide additional bit integrity checking utilities which can be used to perform specific integrity checking tasks.

Modes:

1. Generate integrity information for a Space
2. Generate integrity information for an item list
3. Compare two integrity reports

Configuration Options:

1. Mode 1 - *Generate integrity information for a Space*
    a. Get integrity information from...
        i. The storage provider: Determine the file MD5 by asking the storage provider for its stored MD5 value
        ii. The files themselves: Determine the file MD5 by retrieving them from the storage provider and computing the MD5
    b. Stores: The underlying storage provider in which the following space resides
        i. Space containing content items: The DuraCloud space in which the content items to be considered reside
2. Mode 2 - *Generate integrity information for an item list*

    a. Get integrity information from...
        i. The storage provider: Determine the file MD5 by asking the storage provider for its stored MD5 value
        ii. The files themselves: Determine the file MD5 by retrieving them from the storage provider and computing the MD5
        iii. Input listing name: Name of the content item which contains the listing of items over which to run the service
    b. Stores: The underlying storage provider in which the following space resides
        i. Space with input listing: The DuraCloud space in which the input listing file resides
3. Mode 3 - *Compare two integrity reports*
    a. Input listing name: Name of the first content item which contains a listing of items to be compared to the second listing
    b. Second input listing name: Name of the second content item which contains a listing of items to be compared to the first listing
    c. Stores: The underlying storage provider in which the following spaces reside
        i. Space with input listing: The DuraCloud space in which the first input listing file resides
        ii. Space with second input listing: The DuraCloud space in which the second input listing file resides

Service Ouputs

1. bitintegrity/fingerprints-<spaceId>-<date>.csv
    - listing of hashes when running in *from space* or *from list* modes
2. bitintegrity/fixity-report-<listingId-0>-vs-<listingId-1>-<date>.csv
    - comparison report of two hash listings

Bit Integrity Checker - Bulk

Description:

The Bulk Bit Integrity Checker provides a simple way to determine checksums (MD5s) for all content items in any particular space by leveraging an Amazon Hadoop cluster. This service is designed for large datasets (+10GB).

Configuration Options:

1. Space to verify: DuraCloud space where source files are stored
2. Service Mode
    a. Verify integrity of a Space: Retrieves all items in a space, computes the checksum of each, and compares that value with the MD5 value available from the storage provider
    b. Verify integrity from an item list: Retrieves all items listed in the item list, computes the checksum of each, and compares that value with the MD5 value provided in the item list
        i. Space with input listing: The DuraCloud space in which the input listing file resides
        ii. Input listing name: Name of the content item which contains the listing of items over which to run the service
3. Standard vs. Advanced configuration
    a. Standard mode automatically sets up the service to be run
    b. Advanced mode allows the user to configure the number and type of servers that will be used to run the job
        i. Number of Server Instances: The number of servers to use to perform the duplication task.
        ii. Type of Server: The type (size) of server used as perform the task. The larger the server, the faster the processing will occur. Larger servers also cost more than smaller servers to run. For more information, see the Amazon EC2 documentation.

Service Ouputs

1. bitIntegrity-bulk/bitIntegrity-report-<date>.csv
    - final report with status of integrity check
2. bitIntegrity-bulk/bitIntegrity-results.csv
    - interim listing with hashes recalculated from content streams

Image Transformer

Description:

The Image Transformer provides a simple way to transform relatively small numbers of image files from one format to another.

Note that the ImageMagick service must be deployed prior to using the Image Transformer

Configuration Options:

1. Source Space: DuraCloud space where source image files are stored
2. Destination Space: DuraCloud space where transformed image files will be placed, along with a file which details the results of the conversion process
3. Destination Format: The image format to which the source files will be transformed
4. Destination Color Space: The colorspace of the transformed files, either "Source Image Color Space", meaning that the colorspace of the original image will be used, or sRGB, meaning that the colorspace will be transformed to sRGB.
5. Source file name prefix: Only files beginning with the value provided here will be transformed. For example, if you enter ABC, only files whose names begin with the string ABC will be processed. This field is optional.
6. Source file name suffix: Only files ending with the value provided here will be transformed. For example, you enter .jpg, only files whose names ends with the string .jpg will be processed. This field is optional.

Image Transformer - Bulk

Description:

The Bulk Image Transformer provides a simple way to transform image files from one format to another in bulk. This service uses Amazon's Elastic Map Reduce capability to run the image transformation task within a Hadoop cluster.

Configuration Options:

1. Source Space: DuraCloud space where source image files are stored
2. Destination Space: DuraCloud space where transformed image files will be placed, along with a file which details the results of the transformation process
3. Destination Format: The image format to which the source files will be transformed
4. Destination Color Space: The colorspace of the transformed files, either "Source Image Color Space", meaning that the colorspace of the original image will be used, or sRGB, meaning that the colorspace will be transformed to sRGB.
5. Source file name prefix: Only files beginning with the value provided here will be transformed. For example, if you enter ABC, only files whose names begin with the string ABC will be processed. This field is optional.
6. Source file name suffix: Only files ending with the value provided here will be transformed. For example, you enter .jpg, only files whose names ends with the string .jpg will be processed. This field is optional.
7. Standard vs. Advanced configuration
    a. Standard mode automatically sets up the service to be run
    b. Advanced mode allows the user to configure the number and type of servers that will be used to run the job
        i. Number of Server Instances: The number of servers to use to perform the duplication task.
        ii. Type of Server: The type (size) of server used as perform the task. The larger the server, the faster the processing will occur. Larger servers also cost more than smaller servers to run. For more information, see the Amazon EC2 documentation.

System Transformer Utility

Description:

The System Transformer Utility deploys the ImageMagick application on a DuraCloud service instance, which allows other services to take advantage of its features. The Image Transformer requires that this service be deployed in order to operate correctly.

Configuration Options:

None

System WebApp Utility

Description:

The System WebApp Utility coordinates the installation, de-installation, startup and shutdown of Apache Tomcat servers on a DuraCloud service instance. These Tomcat servers are created to allow other DuraCloud services to deploy web applications. The Image Server requires that this service be deployed in order to operate correctly.

Configuration Options:

None

DuraCloud Features

- **Application Features**
    - Storage system REST API
        - Space (top level folder) listing
        - Space creation
        - Space deletion
        - Space metadata and tagging
        - Content (file) listing
        - Content storage
        - Content deletion
        - Content metadata and tagging
    - Service administration REST API
        - Available service listing
        - Deployed service listing
        - Service deployment
        - Service configuration
        - Service undeployment

    - Security requiring authentication on all DuraCloud applications
    - Web user interface (DurAdmin)
        - Access to all storage system capabilities, including space and content creation, updates, and deletes
        - Access to all service administration capabilities, including service deployment, configuration, and undeployment
        - Bulk deletion of spaces and content items
        - User account administration

- **Core Services**
    - **Media Streamer:**
        - Provides streaming capabilities for video and audio files.
    - **Duplicate on Upload:**
        - Provides a simple mechanism for synchronizing your content between two storage providers. A running Duplicate on Upload service will listen for updates on ingest which occur in one store and duplicate those activities in another store.
    - **Duplicate on Demand:**
        - Provides a simple way to duplicate content (and its associated metadata) from a space in the primary storage provider to a space in another provider.
    - **Image Server:**
        - Deploys an instance of the Adore Djatoka web application which provides serving and viewing of JPEG2000 images. Note that in order to view images using the Image Server, the images must be in an OPEN space.
    - **Image Transformer:**
        - Provides a simple way to convert image files from one format to another.
    - **Image Transformer - Bulk:**
        - Handles the conversion of large numbers of images files from one format into other.
    - **Bit Integrity Checker:**
        - Provides the ability to verify that the content held within DuraCloud has maintained its bit integrity.
    - **Bit Integrity Checker - Bulk:**
        - Provides a simple way to determine checksums (MD5s) for all content items in any particular space by leveraging an Amazon Hadoop cluster. This service is designed for large datasets (+10GB).
    - **Bit Integrity Checker - Tools:**
        - Provides additional bit integrity checking utilities which can be used to perform specific integrity checking tasks.

- **Utility Services**
    - **System WebApp Utility:**
        - Coordinates the installation/deinstallation and startup/shutdown of Apache Tomcat instances that are created to run web application services and are deployed outside of the hosting OSGi container.
        - *Required to run Image Server.*
    - **System Transformer Utility:**
        - Deploys the ImageMagick application which allows other services to take advantage of its features.
        - *Required to run Image Transformer.*

- **Tools**
    - **Sync Tool:**
        - Provides a command line utility for keeping DuraCloud content synchronized with the local file system.
    - **Retrieval Tool:**
        - Provides a command line utility for transferring content stored in DuraCloud to the local file system.
    - **Chunker Tool:**
        - Provides a command line utility for transferring single files to DuraCloud.

DuraCloud Retrieval Tool

Introduction

The Retrieval Tool is a utility which is used to transfer (or "retrieve") digital content from DuraCloud to your local file system.

Download

Download the retrieval tool here.

How the Retrieval Tool Works

- When the Retrieval Tool starts up, it connects to DuraCloud using the connection parameters you provide and gets a list of content items in the spaces you indicate. It will then proceed to download the files from those spaces, each into a local directory named for the space, which is placed within the content directory.
- For each content item, the Retrieval Tool checks to see if there is already a local file with the same name. If so, the checksums of the two files are compared to determine if the local file is the same as the file in DuraCloud. If they match, nothing is done, and the Retrieval Tool

moves on to the next file. If they do not match, the file from DuraCloud is retrieved.
- By default, when a local file exists and differs from the DuraCloud copy, the local file is renamed prior to the DuraCloud file being retrieved. If you would prefer that the local file simply be overwritten, you will need to include the overwrite command-line flag when starting the Retrieval Tool.
- As each content file is downloaded, a checksum comparison is made to ensure that the downloaded file matches the file in DuraCloud. If the checksums do not match, the file is downloaded again. This re-download will occur up to 5 times. If the checksums still do not match after the fifth attempt, a failure is indicated in the output file.
- As each file download completes, a new line is added to the retrieval tool output file in the work directory, indicating whether the download was successful or not. Files which did not change are not included in the output file.
- As the Retrieval Tool runs, it will print its status approximately every 10 minutes to indicate how many files have been checked and downloaded.
- Once all files are retrieved, the Retrieval Tool will print its final status to the command line and exit.
- As files are updated in DuraCloud, you can re-run the Retrieval Tool using the same content directory, and only the files which have been added or updated since the last run of the tool will be downloaded.

Operational notes

- Content Directory - the directory to which files will be downloaded. A new directory within the content directory will be created for each space.
- Work Directory - the work directory contains both logs, which give granular information about the process, and output files. A new output file is createdc for each run of the Retrieval Tool, and it stores a listing of the files which were downloaded.

Prerequisites

- You must have Java version 6 or above installed on your local system. If Java is not installed, you will need to download and install it. To determine if the correct version of Java is installed, open a terminal or command prompt and enter

  ```
  java -version
  ```

  The version displayed should be 1.6.0 or above. If running this command generates an error, Java is likely not installed.

- You must have downloaded the Retrieval Tool. It is available as a link near the top of this page.

Starting the Retrieval Tool

- To run the Retrieval Tool, open a terminal or command prompt and navigate to the directory where the Retrieval Tool jar file is located
- To display the help for the Retrieval Tool, run

  ```
  java -jar retrievaltool-[version]-driver.jar
  ```

- When running the Retrieval Tool, you will need to use these options:

Short Option	Long Option	Argument Expected	Required	Description	Default Value (if optional)
-h	--host	Yes	Yes	The host address of the DuraCloud DuraStore application	
-r	--port	Yes	No	The port of the DuraCloud DuraStore application	443
-u	--username	Yes	Yes	The username necessary to perform writes to DuraStore	
-p	--password	Yes	Yes	The password necessary to perform writes to DuraStore	
-i	--store-id	Yes	No	The Store ID for the DuraCloud storage provider	The default store is used
-s	--spaces	Yes	No	The space or spaces from which content will be retrieved. Either this option or -a must be included	
-a	--all-spaces	No	No	Indicates that all spaces should be retrieved; if this option is included the -s option is ignored	Not set
-c	--content-dir	Yes	Yes	Retrieved content is stored in this local directory	
-w	--work-dir	Yes	Yes	Logs and output files will be stored in the work directory	

-o	--overwrite	No	No	Indicates that existing local files which differ from files in DuraCloud under the same path and name sould be overwritten rather than copied	Not set
-t	--threads	Yes	No	The number of threads in the pool used to manage file transfers	3

- An example for running the Retrieval Tool

```
java -jar retrievaltool-0.8.0-driver.jar -c content -h test.duracloud.org -u myname -p
mypassword -w work -s space1 space1 -o
```