

1. DuraCloud Release 0.6	2
1.1 DuraCloud Chunker Tool	3
1.2 Software	4
1.2.1 Building DuraCloud Software from Source	4
1.2.2 Service Development Guide	8
1.3 Logging Configuration	10
1.4 DuraCloud REST API	11
1.5 Services	21
1.5.1 Fixity Service	21
1.5.2 Metadata Export Service	24
1.6 DuraCloud Security	25
1.7 Release Notes	28
1.8 DuraCloud Architecture	30
1.9 DuraCloud Sync Tool	42
1.10 DuraCloud Java Clients	44
1.11 DuraCloud Presentations	45
1.12 DuraCloud Administration	45
1.13 DuraCloud Services	46

# DuraCloud Release 0.6

## 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](#)



The following documentation pertains to release 0.6. The most up-to-date documentation can be found [here](#)

## User Documentation

- [Release Notes](#)
- [DuraCloud Administration](#)
- [DuraCloud Services](#)
- [DuraCloud REST API](#)
- [DuraCloud Java Clients](#)
- [DuraCloud Sync Tool](#)
- [DuraCloud Chunker Tool](#)
- [Logging Configuration](#)

## Design Documentation

- [DuraCloud Architecture](#)
- [DuraCloud Security](#)
- [Services](#)

## Developer Documentation

- [Building DuraCloud from Source](#)
- [Service Development Guide](#)

## Source Code

- The DuraCloud 0.6 release (latest release)
  - Source can be download from Subversion: <https://svn.duraspace.org/duracloud/tags/duracloud-0.6.0>
  - Source can be viewed here: <https://svn.duraspace.org/view/duracloud/tags/duracloud-0.6.0>
  - JavaDocs for can be viewed here: <http://duracloud.org/0.6/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/duradmin)
    - [https://\[your-institution\].duracloud.org/durastore](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

## Presentations

- [List of 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 <a href="#">Apache Commons IO</a>
-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 <a href="#">Apache Commons IO</a>
-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.

## Software

### Building DuraCloud Software from Source

- [Introduction](#)
- [Prerequisites](#)
- [Building DuraCloud](#)
  - [Build with unit tests](#)
  - [Build with integration tests](#)
  - [Build with OSGi services container integration tests](#)
- [Optional items](#)
  - [Code coverage](#)
  - [Logging](#)
- [DuraCloud internal tools](#)
  - [ServicesAdmin CLI](#)
  - [Application initialization utility](#)
  - [StoreClient package](#)
- [Misc configuration/discussion](#)
  - [Services on Windows](#)
  - [root user](#)
  - [application config](#)
  - [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](#)
- [EMC Atmos](#)

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 djabatoka service has compatibility issues with open-jdk)
4. Subversion

## Building DuraCloud

### Build with unit tests

1. Check out latest stable release from Subversion repository

```
svn co https://svn.duraspace.org/duracloud/tags/duracloud-0.6.0
```

2. Set environment variables

```
export JAVA_OPTS="-XX:MaxPermSize=256m"
```

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

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
```

## 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>[specify-location-of-clover.license]</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**

### 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)"

### 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:

- DuraStore
  - Initialize Stores
  - Initialize Security Users
  - Get Stores
  - Get Spaces
  - Get Space
  - Get Space Metadata
  - Create Space
  - Set Space Metadata
  - Delete Space
  - Get Content
  - Get Content Metadata
  - Store Content
  - Set Content Metadata
  - Delete Content
  - Get Tasks
  - Perform Task
    - Tasks
- DuraService
  - Initialize Services
  - Initialize Security Users
  - Get Services
  - Get Service

- Get Deployed Service
- Get Deployed Service Properties
- Deploy Service
- Update Service Configuration
- UnDeploy Service
- DurAdmin
  - Initialize Application
  - Initialize Security Users

 Each of the methods below has specific security requirements. See [DuraCloud Security](#) for more information

## 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 <http://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 <http://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 <http://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 <http://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 [`http://host:port/durastore/spaceID ? \(storeID\) \(prefix\) \(maxResults\) \(marker\)`](http://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 [`http://host:port/durastore/spaceID ? \(storeID\)`](http://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 [`http://host:port/durastore/spaceID ? \(storeID\)`](http://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=http://myhost:8080/durastore/space1
```

## Set Space Metadata

- Purpose: Updates the metadata associated with a space
- Request: POST [`http://host:port/durastore/spaceID ? \(storeID\)`](http://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 [`http://host:port/durastore/spaceID ? \(storeID\)`](http://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 [`http://host:port/durastore/spaceID/contentID ? \(storeID\)`](http://host:port/durastore/spaceID/contentID ? (storeID) (attachment)) (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=3456709234785097473839202
ETag=3456709234785097473839202
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 [`http://host:port/durastore/spaceID/contentID ? \(storeID\)`](http://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 [`http://host:port/durastore/spaceID/contentID ? \(storeID\)`](http://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=4cd56e137a93alaccb43c5d32f4afffb
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=http://myhost:8080/durastore/space1/content1
```

### Set Content Metadata

- Purpose: Updates the metadata associated with a piece of content
- Request: POST [`http://host:port/durastore/spaceID/contentID ? \(storeID\)`](http://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 [`http://host:port/durastore/spaceID/contentID ? \(storeID\)`](http://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 <http://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 <http://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 <http://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 <http://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 <http://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 <http://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 <http://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 <http://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 <http://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 <http://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 <http://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 <http://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 <http://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)

## Services

### Under development

1. Fixity Service
2. Metadata Export Service

# Fixity Service

- Overview
- Requirements
- Design
  - Levels of trust
  - Operational modes
  - Functionality spec
    - Single-step interaction
    - Two-step interaction
    - Service options
    - Service exceptions
    - Options / Mode matrix

## Overview

The Fixity Service 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 Fixity Service also provides the option of reading the content and recalculating the MD5. Finally, if the administrator does not trust this recalculation, then the Fixity Service 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. Fixity Service 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. Fixity Service 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. spaceld & contentId where input listing is stored
    - comma-delimited listing of spaceld, contentId and expected MD5
    - each content item separated by newline character
    - first line in file will be ignored
  - b. spaceld & contentId where results file should be written
  - c. options (see below)
2. Service outputs
  - a. comma-delimited listing of spacelds, contentIds, expected MD5s, system MD5s, status state
  - b. service status state

### Two-step interaction

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

## 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 0.6

**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 depicts scenarios that are likely to occur in a real-world case where DuraCloud is used as a backup system for a repository within an institutional repository in this case. The repository could be replaced by any other software system or the file system.

### DuraCloud Interactions

There are three types of interactions illustrated here:

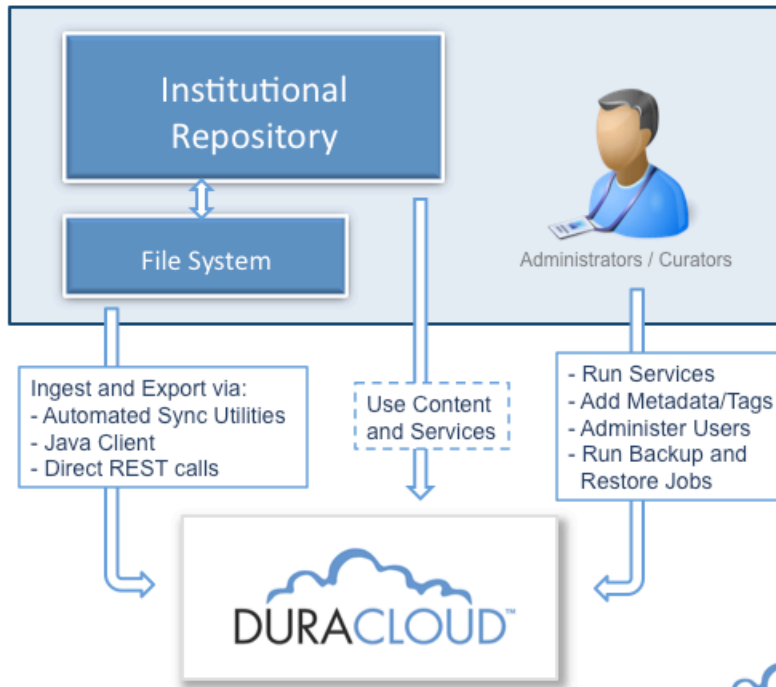
1. Moving files from the local system a

- -
- -
- -

2. Application repository content s

## DuraCloud In Context

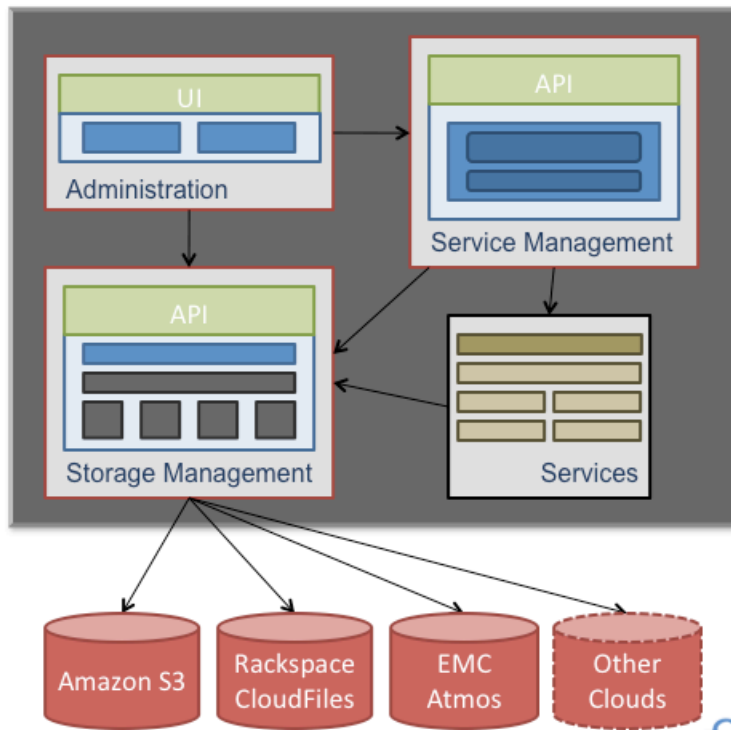
Organization Utilizing DuraCloud



well as D

3. Administ

### DuraCloud High Level Interaction



**DURACLOUD**  
 Updated: July 2010 | <http://www.duracloud.org>

This diagram depicts an example Primary User Compute Instance (Primary UCI), and its interaction with four underlying storage providers.

#### Primary User Compute Instance (Primary UCI)

Once a user creates a DuraCloud account, a new Primary UCI is started on their behalf and initialized based on the account configuration. Most of the interaction with DuraCloud will be through the Primary UCI.

Fundamentally, the Primary UCI consists of a set of web applications that expose three general capabilities:

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

The Primary UCI is assigned a public address which can be mapped to a user-owned domain name or a domain name provided by DuraCloud. Through this base URL and published storage and service REST APIs, the user can by-pass the Administration UI (DurAdmin) and interact directly with the Storage Management and



Service Management components.

The Services component and supporting storage providers are addressed below.

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

#### **DuraStore**

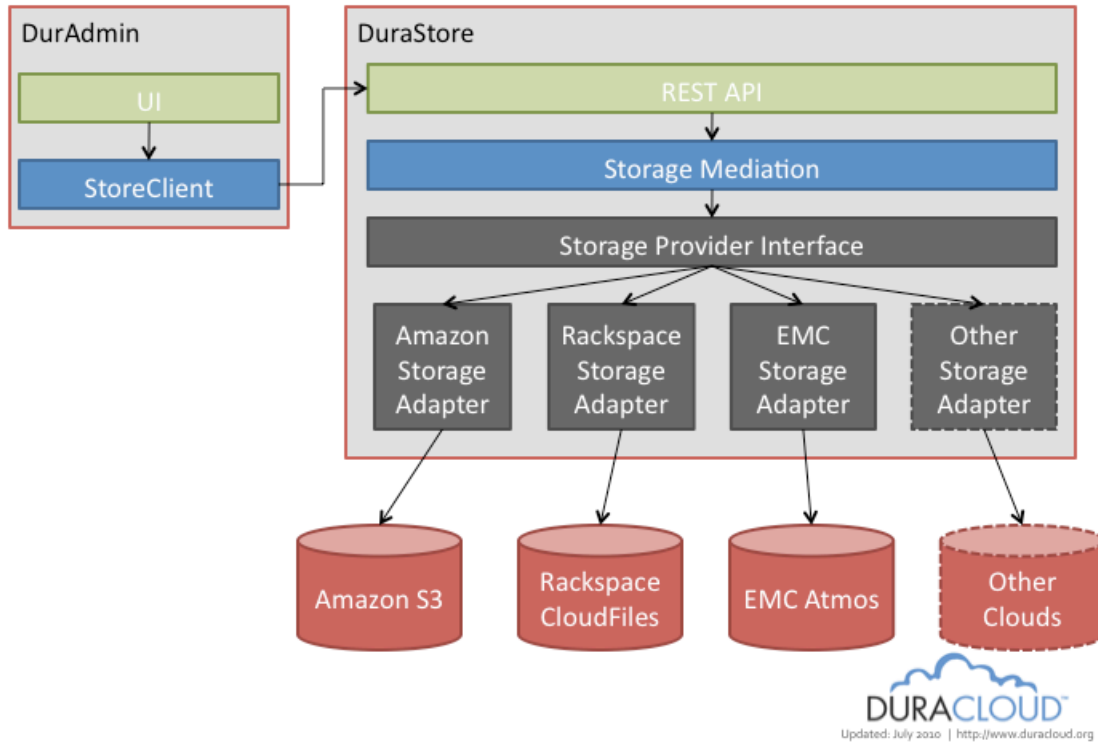
*DuraStore* is a web application deployed on the *Primary UCI* which is responsible for mediating the interaction with content stored in any one or more of the underlying storage providers. It exposes these content management functions via a public-facing REST API. Such functions include but are not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

## DuraCloud Storage Architecture



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 the services counterpart to the previous storage management diagram above.

### DuraService

*DuraService* is an application deployed on the *Primary UI* which is responsible for mediating the

interactions with deployed or available user services. It exposes these service management functions via a public-facing REST API.

Such functions include but are not limited to:

- service deployment/undeployment
- runtime service configuration
- service property listings

The notion of a service in the DuraCloud context can be defined as any compute artifact that can be started/stopped, configured and monitored as implemented by the *Compute Service Interface*.

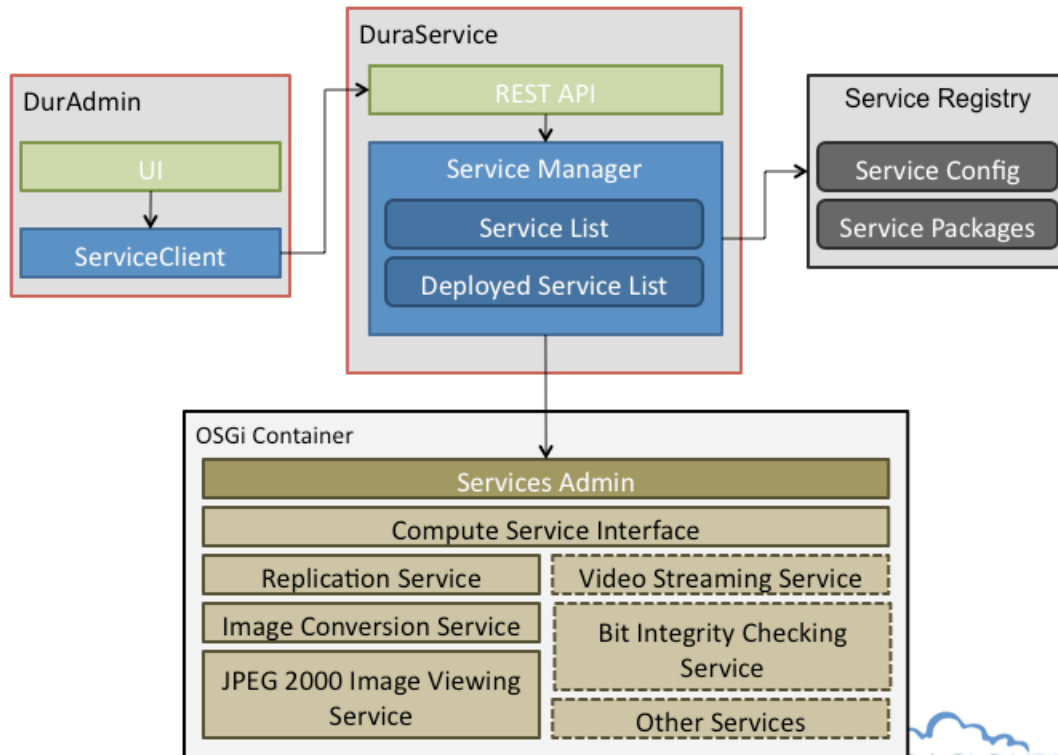
*DuraService* passes service management requests to the *Service Manager* which has access to:

- available services and their configuration options from the *Dura Service Repository*
- deployed services with their current configuration and properties from *Service Admin*

### **Service Registry**

DuraCloud maintains a *Service Registry* which contains a set of versioned *Service Packages* available for runtime deployment. These *Service Packages* consist of OSGi bundles and support files that may be community-provided as well as developed by DuraSpace. The *Service Content* is an XML document that acts as a table of contents for available services.

## DuraCloud Service Architecture



**DURACLOUD**  
Updated: July 2010 | <http://www.duracloud.org>

services and their associated configuration options.

### OSGi Container: Services Admin

As an implementation note, the need to dynamically start, and reconfigure services at runtime without performing application restarts drove the decision to employ an *OSGi Container* as DuraCloud's service hosting environment. Although the *OSGi Container* is depicted as a single component deployed in the *Primary UCI*, if it is advantageous for particular services to run on a *Managed* separate *OSGi Container* could just as well be deployed on another compute instance.

*Services Admin* and each of the depicted *Service* elements are independent, OSGi bundles.

All requests from *DuraService* for starting, configuration, etc., services pass through *Services Admin*.

Such requests are mediated to the appropriate service within the *OSGi Container* via the common *Compute Service Interface*. Even though the management of a given service requires a representative OSGi bundle that exports implementation of *Compute Service Interface*, a variety of actual service flavors are supported. Such service implementations include

- web applications (e.g. Adore-DJ)
- command-line utilities (e.g. ImageMagick)
- pure Java (e.g. Replication Service)

- external compute instances

### DurAdmin

As in the previous diagram, *DurAdmin* is shown here to highlight the fact that its interaction with *DuraService* happens exclusively through *ServiceClient* (which is provided with the application distribution), a Java wrapper over the services REST API.

This diagram pulls up to a higher level, depicting the deployment topology planned architecture that has been deployed.

DuraCloud is composed of compute instances.

### 1. DuraCloud

- I
- V
- S
- i
- I
- S
- C
- S
- J
- a
- C
- C

### 2. Primary Instance

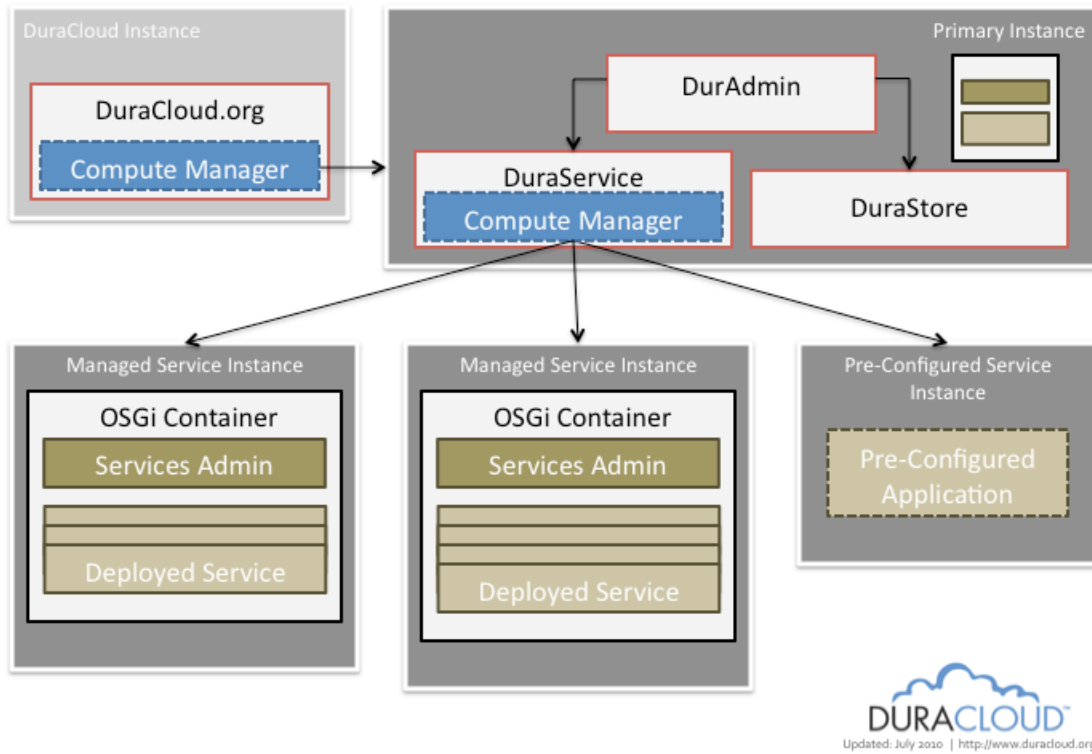
- I
- I
- a
- f

- I
- C
- S

### 3. Managed

- C

## DuraCloud Server Architecture (Planned)



#### 4. Pre-Config

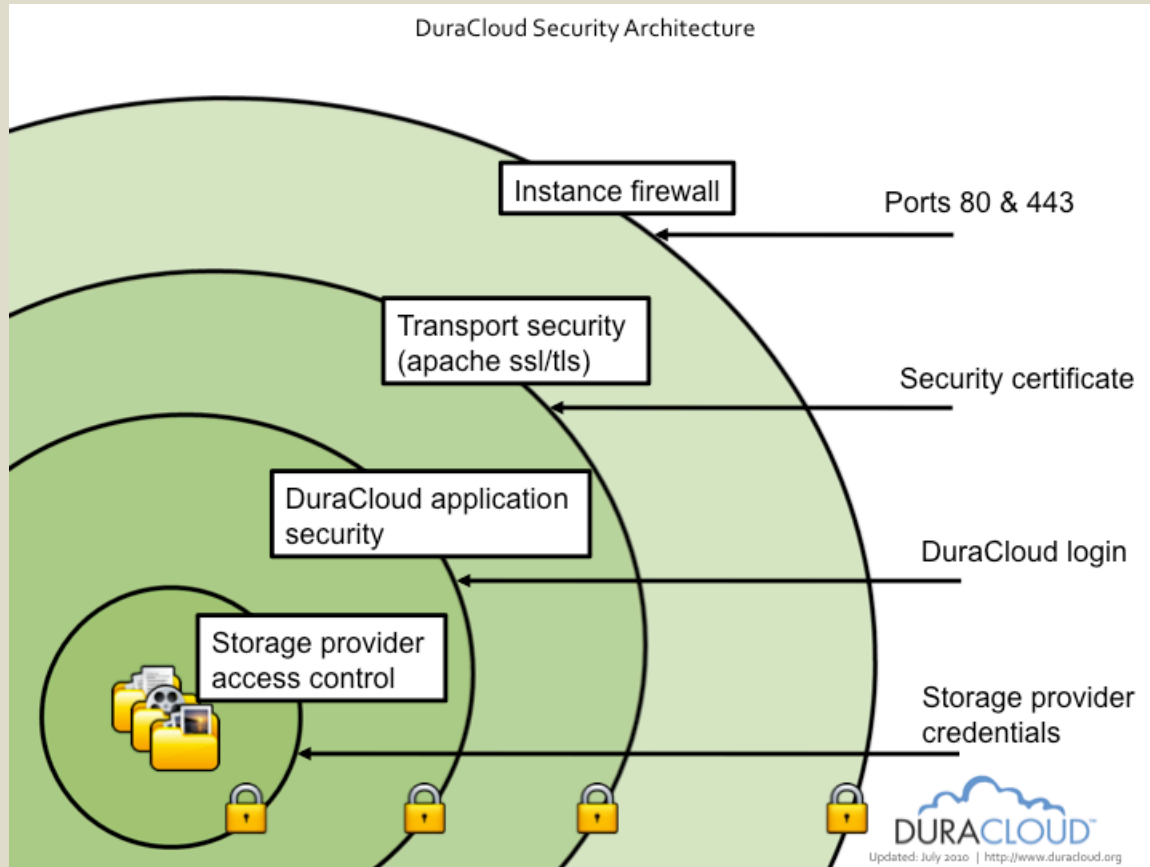
This diagram illustrates the multiple levels of security provided by DuraCloud.

#### Instance Firewall

- The first defense level is the Instance Firewall. DuraCloud instances are surrounded by a firewall that surrounds the *Primary Instance*. This firewall is constructed to allow connections only through the standard HTTP (80) and HTTPS (443) ports. Requests coming in over HTTP are redirected to HTTPS to ensure secure transport.

#### Transport Security

- As noted



above, all communi with DuraClou via HTTP meaning all inform that goes comes or DuraClou encrypted transport ensures t transmiss into or ou DuraClou be read c by the intended recipient.

**DuraCloud Application Security**

- The DuraClou applicatio itself con access to content to ensure th only regis users (wh must be : by an Administrator with the p role, can perform t (such as download 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 conter describ above wi concerns about wh might be occuring underlyin providers

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

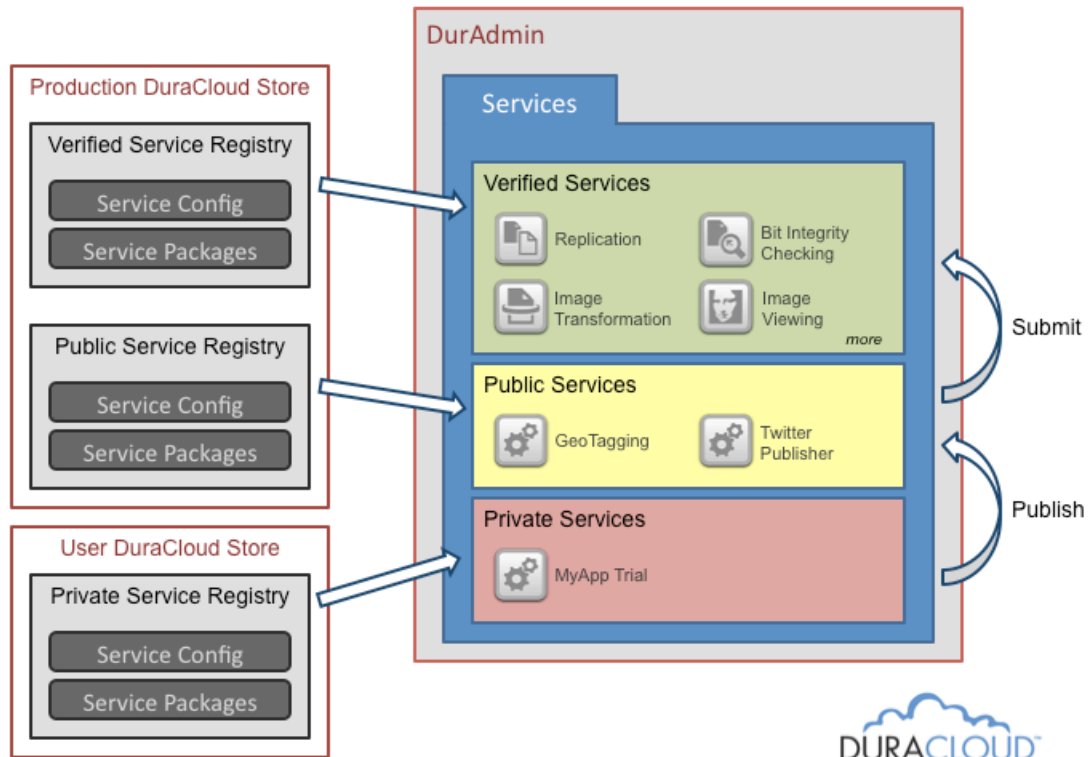
#### **Private Service Registry**

- A private service registry will be where DuraCloud account holders purchase their personal services. These services are likely to have been created by or for a user to fulfill specific content processing needs. No other DuraCloud users will have access to services located in the private service registry. This diagram illustrates the way these services will be stored in a user's personal storage area.

#### **Public Service Registry**



## DuraCloud Service Registries



**DURACLOUD**  
Updated: July 2010 | <http://www.duracloud.org>

- The Public Service Registry is where services private service registries are published to make them available to others. The intent is to allow these services to be available to use for free or for a fee. The determination of the fee would be by the creator of the service. While the services are available to all, the user will be on a "at your own risk" basis. These services in the Public Service Registry be stored in a central DuraCloud storage location.

### Verified Service Registry

- Services in the Verified Service Registry are those that have been submitted and verified by the DuraCloud committee team. These services are tested to ensure that they function properly within the DuraCloud system and perform the actions that are intended to perform. They, like services in the Public Service Registry,

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 backups.
- When the Sync Tool starts up, it will look through all of the files in each of the local sync 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 sync 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 backup 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 sync 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

- Security
  - Version 0.3+ of DuraCloud includes a requirement that all data must be transported over SSL. In order to connect to your DuraCloud instance, the Sync Tool needs to have access to the DuraCloud SSL certificate. See the page on [DuraCloud Security](#) for more information about including the DuraCloud certificate in your local truststore.
  - If you add the DuraCloud certificate to a truststore which is not picked up by your Java installation, you may need to specify its location on the command line. To do this, add the parameter

```
-Djavax.net.ssl.trustStore=<path-to-truststore>
```

prior to the -jar parameter when starting the Sync Tool. You will, of course, have to replace <path-to-truststore> with the path to the truststore in which the DuraCloud certificate is stored.

- An error similar to this:

```
Caused by: javax.net.ssl.SSLHandshakeException:
sun.security.validator.ValidatorException:
PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException:
unable to find valid certification path to requested target
```

when attempting to run the Sync Tool indicates that the you'll need to include the explicit path to your truststore.

- Restarting
  - You can perform a restart of the Sync Tool by using the `-c` command line option to point to the Sync Tool configuration file, which is written into the backup 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 sync directories to DuraCloud) you will need to either point it to a new backup directory, or clear out the existing backup location.
  - The Sync Tool will perform a clean start (not a restart) if the list of sync directories is not the same as the previous run. This is to ensure that all files in all sync 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 sync 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.
- Backup Directory - these files and directories can be found in the backup directory (specified using the `-b` 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 backup directory. This file can be used to restart the Sync Tool, using the `-c` 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 `-c` 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 backup 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 sync directories (`-d` 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-0.6.0-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	
-p	--port	Yes	No	The port of the DuraCloud DuraStore application	443
-s	--space	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	
-w	--password	Yes	Yes	The password necessary to perform writes to DuraStore	
-d	--sync-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.	
-b	--backup-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
-x	--sync-deletes	No	No	Indicates that deletes performed on files within the sync directories should also be performed on those files in DuraCloud; if this option is not included all deletes are ignored	Not set
-e	--exit-on-completion	No	No	Indicates that the sync tool should exit once it has completed a scan of the backup directories and synced all files; if this option is included, the sync tool will not continue to monitor the sync dirs	Not set

- When the Sync Tool runs, it creates a backup of your configuration in the backup 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
-c	--config-file	Yes	Yes	Read configuration from this file (a file containing the most recently used configuration can be found in the backup-dir, named synctool.config)

- An example for running the Sync Tool

```
java -jar synctool-0.6.0-driver.jar -b C:\tools\synctool\backup -d 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, gain status, etc 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 "."
    - between 3 and 63 characters
    - must start with a letter or number
    - 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 all services currently have a "Location" configuration option which is intended to allow for the deployment of services at varying locations. At the moment, however, services can only be deployed on the primary service instance. As this configuration option is consistent across all services it will not be included in the listing for each service.

### Replication Service

#### Description:

The Replication Service provides a way to ensure that the content added to DuraCloud is stored with at least two storage providers. The Replication Service performs on-ingest replication of content. This means that once the Replication 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. Replicate from this store: 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. Replication to this store: The secondary store where content will be copied after it has been added to the primary store.

### JPEG 2000 Image Viewer Service

#### Description:

The JPEG 2000 Image Viewer Service, when deployed, 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 Streaming Service

#### Description:

The Media Streaming service 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 Streaming 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.

## Fixity Service

#### Description:

The Fixity service provides the ability to verify that the content held within DuraCloud has maintained its bit integrity. There are five modes of operation:

1. All-in-one generation of system MD5s for items in provided listing and verification
2. All-in-one generation of system MD5s over entire space(s) and verification to provided listing
3. Generation of system MD5 listing for items in provided listing
4. Generation of system MD5 listing over entire space
5. Comparison of two provided MD5 listings.

#### Configuration Options:

1. #

## Image Conversion Service

#### Description:

The Image Conversion service provides a simple way to convert relatively small numbers of image files from one format to another.

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

#### Configuration Options:

1. Source Space: DuraCloud space where source image files are stored
2. Destination Space: DuraCloud space where converted 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 converted
4. Destination Color Space: The colorspace of the converted 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 converted to sRGB.
5. Source file name prefix: Only files beginning with the value provided here will be converted. 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 converted. For example, you enter .jpg, only files whose names ends with the string .jpg will be processed. This field is optional.

## Bulk Image Conversion Service

#### Description:

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

#### Configuration Options:

1. Source Space: DuraCloud space where source image files are stored
2. Destination Space: DuraCloud space where converted image files will be placed, along with a file which details the results of the conversion process
3. Working Space: DuraCloud space used to store processing information necessary to run the job as well as log files generated as part of the job flow
4. Destination Format: The image format to which the source files will be converted
5. Destination Color Space: The colorspace of the converted 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 converted to sRGB.
6. Source file name prefix: Only files beginning with the value provided here will be converted. For example, if you enter ABC, only files whose names begin with the string ABC will be processed. This field is optional.

7. Source file name suffix: Only files ending with the value provided here will be converted. For example, you enter .jpg, only files whose names ends with the string .jpg will be processed. This field is optional.
8. Number of Server Instances: The number of servers to use to perform the image conversion task.
9. 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](#).

## ImageMagick Service

### Description:

The ImageMagick Service deploys the [ImageMagick](#) application on a DuraCloud service instance, which allows other services to take advantage of its features. The Image Conversion Service requires that this service be deployed in order to operate correctly.

### Configuration Options:

None

## Web App Utility Service

### Description:

The Web App Utility service 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 JPEG 2000 Image Viewer Service requires that this service be deployed in order to operate correctly.

### Configuration Options:

None