DUCC Installation and Verification
Excerpt From Complete DUCC Documentation

Written and maintained by the Apache
UIMA™Development Community

Publication date: April 2019

Overview

DUCC is a multi-user, multi-system distributed application. For first-time users a staged installation/verification methodology is recommended, roughly as follows:

- Single system installation - single node - all work runs with the credentials of the installer.
- Optionally add worker nodes to the cluster.
- Enable CGroup containers. CGroups protect DUCC managed processes running on the same machine from inappropriately stealing memory or CPU resources from each other. CGroups are also required for DUCC to measure and report performance metrics on managed processes.
- Enable multiple-user support - processes run with the credentials of the submitting user, while DUCC runs as user *ducc*. This step requires root authority on one or more machines.

Nominally user *ducc* runs the DUCC software. However another user, for example *duccster*, could be chosen. In this document references to user *ducc* will mean to the user that you have chosen, be it *ducc* or *duccster* or some other. The chosen user is the OS user installing DUCC.

When upgrading from an existing installation the *ducc_update* script may be used to replace the system files while leaving the site-specific configuration files in place. For more information see *"ducc_update"* in the Administrative Commands section of the DuccBook.

Since with this release the persistence data about completed work is stored in a database, additional upgrade steps are required to convert the older file-based data in order to preserve information about past work. For more information see *"db_create"* and *"db_loader"* in the Administrative Commands section of the DuccBook.

DUCC is distributed as a compressed tar file. If building from source, this file will be created in your svn trunk/target directory. The distribution file is in the form

```
uima-ducc-[version]-bin.tar.gz
```

where [version] is the DUCC version; for example, *uima-ducc-2.1.0-bin.tar.gz.* This document will refer to the distribution file as the "<distribution.file>".

Software Prerequisites

Single system installation:

- Reasonably current Linux. DUCC has been tested on RHEL 6 & 7, SLES 11 & 12, Ubuntu 14.04 & 16.04 with 64-bit Intel and IBM Power (Big and Little Endian) hardware.

  Note: On some systems the default *user limits* for max user processes (ulimit -u) and nfiles (ulimit -n) are defined too low for DUCC. The shell login profile for user *ducc* should set the soft limit for max user processes to be the same as the hard limit (ulimit -u 'ulimit -Hu'), and the nfiles limit raised above 1024 to at least twice the number of user processes running on the cluster.
- Python 2.x, where 'x' is 4 or greater. DUCC has not been tested on Python 3.x.
- Java JDK 7 or 8. DUCC has been tested and run using IBM and Oracle JDK 1.7 & 1.8. A JDK is required by DUCC's web server for JSP compilations, for which a JRE is insufficient.
- Passwordless ssh for the userid running DUCC

Additional requirements for multiple system installation:

- All systems must have a shared filesystem (such as NFS or GPFS) and common user credentials. The $DUCC_HOME directory must be located on a shared filesystem.

Additional requirements for running multiple user processes with their own credentials.

- A userid *ducc*, and group *ducc*. User *ducc* must be the only member of group *ducc*.

- DUCC installed and run with user *ducc* credentials.

  Note: With these restrictions any userid can be configured to be the DUCC user and group.

- Root access is required to setuid-root the DUCC process launcher.

Additional requirements for CGroup containers:

- A userid *ducc*, and group *ducc*. User *ducc* must be the only member of group *ducc*.
- DUCC installed and run with user *ducc* credentials.

  Note: With these restrictions any userid can be configured to be the DUCC user and group.

- Root access is required to configure and enable a custom cgroup for DUCC.
- A running cgroup subsystem. True by default for newer OS.
- Create a custom cgroup definition that the DUCC user can use to run managed applications in. The custom definition to go into /etc/cgconfig.conf is described in *CGroups Installation and Configuration.*
- The cgroup tools package, e.g. libcgroup-tools on centos and cgroup-tools on ubuntu.
- On some flavors of Linux, the cgroup swap accounting is not enabled and swap is reported as N/A. To enable swap accounting add swapaccount=1 kernel parameter. More information on this step is available here: `http://unix.stackexchange.com/questions/147158/how-to-enable-swap-accounting-for-memory-cgroup-in-archlinux`.

In order to build DUCC from source the following software is also required:

- A Subversion client, from `http://subversion.apache.org/packages.html`. The svn url is `https://svn.apache.org/repos/asf/uima/uima-ducc/trunk`.
- Apache Maven, version 3.3.3+, from `http://maven.apache.org/index.html`

The DUCC webserver server optionally supports direct "jconsole" attach to DUCC job processes. To install this, the following is required:

- Apache Ant, any reasonably current version.

To (optionally) build the documentation, the following is also required:

- Latex, including the *pdflatex* and *htlatex* packages. A good place to start if you need to install it is `https://www.tug.org/texlive/`.

More detailed one-time setup instructions for source-level builds via subversion can be found here: `http://uima.apache.org/one-time-setup.html#svn-setup`

Building from Source

To build from source, ensure you have Subversion and Maven installed. Extract the source from the SVN repository named above.

Then from your extract directory into the root directory (usually current-directory/trunk), and run the command

```
mvn install
```

or

```
mvn install -Pbuild-duccdocs
```

if you have LaTeX installed and wish to do the optional build of documentation. The build-duccdocs profile can also activated if the environment valiable BUILD_DUCCDOCS is set true.

Note: LaTeX output is quite verbose and it is suppressed when run via maven.

An alternative approach for modifying DUCCDOCS is to use the LaTeX commands manually. Not only is the build much faster, it also exposes LaTeX error messages lost by the suppression. To build duccbook.html manually and leave the output in the same directory:

```
cd {path-to-src}/uima-ducc-duccdocs/src/site/tex/duccbook
htlatex duccbook.tex
```

Use the pdflatex command to build the PDF version.

Note: The manual command needs to be run twice if modifications have been made that affect internal book references such as the table of contents. Also, temporary files created by LaTeX will need to be cleaned up manually.

If this is your first Maven build it may take quite a while as Maven downloads all the open-source pre-requisites. (The pre-requisites are stored in the Maven repository, usually your $HOME/.m2).

When build is complete, a tarball is placed in your current-directory/trunk/target directory.

Documentation

After installation the DUCC documentation is found (in both PDF and HTML format) in the directory ducc_runtime/docs. As well, the DUCC webserver contains a link to the full documentation on each major page. The API is documented only via JavaDoc, distributed in the webserver's root directory `$DUCC_HOME/webserver/root/doc/api.`

If building from source, Maven places the documentation in

- `trunk/uima-ducc-duccdocs/target/site` (main documentation), and
- `trunk/target/site/apidocs` (API Javadoc)

Single System Installation and Verification

Any user ID can be used to run a single-system DUCC, but "ducc" userid is recommended. This user will employ cgroups as well as launching and running processes with the credentials of the submitting user.

If multiple nodes are going to be added later, the ducc runtime tree should be installed on a shared filesystem so that it can be mounted on the additional nodes.

Verification submits a very simple UIMA pipeline for execution under DUCC. Once this is shown to be working, one may proceed installing additional features.

Minimal Hardware Requirements for Single System Installation

- One Intel-based or IBM Power-based system (Big or Little Endian). (More systems may be added later.)
- 8GB of memory. 16GB or more is preferable for developing and testing applications beyond the non-trivial.
- 1GB disk space to hold the DUCC runtime, system logs, and job logs. More is usually needed for larger installations.

Please note: DUCC is intended for scaling out memory-intensive UIMA applications over computing clusters consisting of multiple nodes with large (16GB-256GB or more) memory. The minimal requirements are for initial test and evaluation purposes, but will not be sufficient to run actual workloads.

Single System Installation

1. Expand the distribution file with the appropriate umask:

   ```
   (umask 022 && tar -zxf <distribution.file>)
   ```

   This creates a directory with a name of the form "apache-uima-ducc-[version]".

   This directory contains the full DUCC runtime which you may use "in place" but it is highly recommended that you move it into a standard location on a shared filesystem; for example, under ducc's HOME directory:

   ```
   mv apache-uima-ducc-[version] /home/ducc/ducc_runtime
   ```

   We refer to this directory, regardless of its location, as $DUCC_HOME. For simplicity, some of the examples in this document assume it has been moved to /home/ducc/ducc_runtime.

2. Change directories into the admin sub-directory of $DUCC_HOME:

   ```
   cd $DUCC_HOME/admin
   ```

3. Run the post-installation script:

   ```
   ./ducc_post_install
   ```

   If this script fails, correct any problems it identifies and run it again.

   Note that *ducc_post_install* initializes various default parameters which may be changed later by the system administrator. Therefore it usually should be run only during this first installation step.

4. If you wish to install jconsole support from the webserver, make sure Apache Ant is installed, and run

   ```
   ./sign_jconsole_jar
   ```

   This step may be run at any time if you wish to defer it.

That's it, DUCC is installed and ready to run. (If errors were displayed during ducc_post_install they must be corrected before continuing.)

Initial System Verification

Here we verify the system configuration, start DUCC, run a test Job, and then shutdown DUCC.

To run the verification, issue these commands.

1. cd $DUCC_HOME/admin
2. ./check_ducc

   Examine the output of check_ducc. If any errors are shown, correct the errors and rerun check_ducc until there are no errors.

3. Finally, start ducc: ./start_ducc

Start_ducc will first perform a number of consistency checks. It then starts the ActiveMQ broker, the DUCC control processes, and a single DUCC agent on the local node.

You will see some startup messages similar to the following:

```
ENV: Java is configured as: /share/jdk1.7/bin/java
ENV: java full version "1.7.0_40-b43"
ENV: Threading enabled: True
MEM: memory is 15 gB
ENV: system is Linux
allnodes /home/ducc/ducc_runtime/resources/ducc.nodes
Class definition file is ducc.classes
```

```
OK: Class and node definitions validated.
OK: Class configuration checked
Starting broker on ducchead.biz.org
Waiting for broker ..... 0
Waiting for broker ..... 1
ActiveMQ broker is found on configured host and port: ducchead.biz.org:61616
Starting 1 agents
********** Starting agents from file /home/ducc/ducc_runtime/resources/ducc.nodes
Starting warm
Waiting for Completion
ducchead.biz.org Starting rm
     PID 14198
ducchead.biz.org Starting pm
     PID 14223
ducchead.biz.org Starting sm
     PID 14248
ducchead.biz.org Starting or
     PID 14275
ducchead.biz.org Starting ws
     PID 14300
ducchead.biz.org
    ducc_ling OK
    DUCC Agent started PID 14325
All threads returned
```

Now open a browser and go to the DUCC webserver's url, http://<hostname>:42133 where <hostname> is the name of the host where DUCC is started. Navigate to the Reservations page via the links in the upper-left corner. You should see the DUCC JobDriver reservation in state WaitingForResources. In a few minutes this should change to Assigned. Now jobs can be submitted.

Submitting a test job

1. $DUCC_HOME/bin/ducc_submit –specification $DUCC_HOME/examples/simple/1.job

Open the browser in the DUCC jobs page. You should see the job progress through a series of transitions: Waiting For Driver, Waiting For Services, Waiting For Resources, Initializing, and finally, Running. You'll see the number of work items submitted (15) and the number of work items completed grow from 0 to 15. Finally, the job will move into Completing and then Completed..

Since this example does not specify a log directory DUCC will create a log directory in your HOME directory under

```
$HOME/ducc/logs/job-id
```

In this directory, you will find a log for the sample job's JobDriver (JD), JobProcess (JP), and a number of other files relating to the job.

This is a good time to explore the DUCC web pages. Notice that the job id is a link to a set of pages with details about the execution of the job.

Notice also, in the upper-right corner is a link to the full DUCC documentation, the "DuccBook".

Registering a test service

First start an application broker using the UIMA-AS script included in the DUCC installation, see UIMA-AS broker. Then in another shell define the broker URL using the name of the host the broker was started on, and register the service. On your browser select the Services page and wait for the service to become Available.

```
export DefaultBrokerURL=tcp://<broker-host-name>:61616
$DUCC_HOME/bin/ducc_services --register $DUCC_HOME/examples/simple/1.service
```

The service can be tested by sending it CASes that specify a sleep value.

```
export UIMA_HOME=$DUCC_HOME/apache-uima
export UIMA_CLASSPATH=$DUCC_HOME/lib/uima-ducc/examples
export UIMA_JVM_OPTS=-DDUCC_HOME=$DUCC_HOME
$UIMA_HOME/bin/runRemoteAsyncAE.sh -c $DUCC_HOME/examples/simple/resources/randomsleep/FixedSleepCR.xml
```

To remove the service use:

```
$DUCC_HOME/bin/ducc_services --unregister <service-id-number>
```

To stop DUCC

```
cd $DUCC_HOME/admin
./stop_ducc -a
```

Add additional nodes to the DUCC cluster

Additional nodes must meet all *prerequisites* (listed above).

$DUCC_HOME must be on a shared filesystem and mounted at the same location on all DUCC nodes.

If user's home directories are on local filesystems the location for user logfiles should be specified to be on a shared filesystem.

Addional nodes are normally added to a worker node group. Note that the DUCC head node does not have to be a worker node. In addition, the webserver node can be separate from the DUCC head node (see webserver configuration options in ducc.properties).

For worker nodes DUCC needs to know what node group each machine belongs to, and what nodes need an Agent process to be started on.

The configuration shipped with DUCC have all nodes in the same "default" node pool. Worker nodes are listed in the file

`$DUCC_HOME/resources/ducc.nodes.`

During initial installation, this file was initialized with the node DUCC is installed on. Additional nodes may be added to the file using a text editor to increase the size of the DUCC cluster.

Ducc_ling Configuration - Running with credentials of submitting user

DUCC launches user processes through ducc_ling, a small native C application. By default the resultant process runs with the credentials of the user ID of the DUCC application. It is possible for multiple users to submit work to DUCC in this configuration, but it requires that the user ID running DUCC has write access to all directories to which the user process outputs data. By configuring the ducc user ID and ducc_ling correctly, work submitted by all users will run with their own credentials.

Before proceeding with this step, please note:

- The sequence operations consisting of *chown* and *chmod* MUST be performed in the exact order given below. If the *chmod* operation is performed before the *chown* operation, Linux will regress the permissions granted by *chmod* and ducc_ling will be incorrectly installed.

ducc_ling is designed to be a setuid-root program whose function is to run user processes with the identity of the submitting user. This must be installed correctly; incorrect installation can prevent jobs from running as their submitters, and in the worse case, can introduce security problems into the system.

ducc_ling can either be installed on a local disk on every system in the DUCC cluster, or on a shared-filesystem that does not suppress setuid-root permissions on client nodes. The path to ducc_ling must be the same on each DUCC node. The default path configuration is $DUCC_HOME/admin/${os.arch}/ in order to handle clusters with mixed OS platforms. ${os.arch} is the architecture specific value of the Java system property with that name; examples are amd64 and ppc64.

The steps are: build ducc_ling for each node architecture to be added to the cluster, copy ducc_ling to the desired location, and then configure ducc_ling to give user ducc the ability to spawn a process as a different user.

In the example below ducc_ling is left under $DUCC_HOME, where it is built.

As user *ducc*, build ducc_ling for necessary architectures (this is done automatically for the DUCC head machine by the ducc_post_install script). For each unique OS platform:

1. cd $DUCC_HOME/admin
2. ./build_duccling

Then, as user *root* on the shared filesystem, *cd $DUCC_HOME/admin*, and for each unique OS architecture:

1. chown ducc.ducc ${os.arch}
   (set directory ownership to be user ducc, group ducc)
2. chmod 700 ${os.arch}
   (only user ducc can read contents of directory)
3. chown root.ducc ${os.arch}/ducc_ling
   (make root owner of ducc_ling, and let users in group ducc access it)
4. chmod 4750 ${os.arch}/ducc_ling
   (ducc_ling runs as user root when started by users in group ducc)

If these steps are correctly performed, ONLY user *ducc* may use the ducc_ling program in a privileged way. ducc_ling contains checks to prevent even user *root* from using it for privileged operations.

If a different location is chosen for ducc_ling the new path needs to be specified for ducc.agent.launcher.ducc_spawn_path in $DUCC_HOME/resources/site.ducc.properties. For more information see *"Properties merging"* in the DuccBook.

CGroups Installation and Configuration

Note: A key feature of DUCC is to run user processes in CGroups in order to guarantee each process always has the amount of RAM requested. RAM allocated to the managed process (and any child processes) that exceed requested DUCC memory size will be forced into swap space. Without CGroups a process that exceeds its requested memory size by N% is killed (default N=5 in ducc.properties), and memory use by child processes is ignored.

DUCC's CGroup configuration also allocates CPU resources to managed processes based on relative memory size. A process with 50% of a machine's RAM will be guaranteed at least 50% of the machine's CPU resources as well.

The steps in this task must be done as user root and the ducc user.

To install and configure CGroups for DUCC:

1. Install the appropriate libcgroup package at level 0.37 or above (see *Installation Prerequisites*).

2. For newer OS configure /etc/cgconfig.conf as follows:

```
# Define cgroup for the userid you have opted to install and run DUCC, normally ’ducc’ but in this
group duccster {
 perm {
     task {
        uid = duccster;
     }
     admin {
        uid = duccster;
     }
 }
 memory {}
 cpu{}
 cpuacct{}
}
```

For older OS that mount cgroups in /cgroup, configure /etc/cgconfig.conf as follows:

```
# Mount cgroups for older OS (e.g. RHEL v6)
mount {
   cpuset = /cgroup/cpuset;
   cpu = /cgroup/cpu;
   cpuacct = /cgroup/cpuacct;
   memory = /cgroup/memory;
   devices = /cgroup/devices;
   freezer = /cgroup/freezer;
   net_cls = /cgroup/net_cls;
   blkio = /cgroup/blkio;
}
# Define cgroup for the userid you have opted to install and run DUCC, normally ’ducc’ but in this
group duccster {
 perm {
     task {
        uid = duccster;
     }
     admin {
        uid = duccster;
     }
 }
 memory {}
 cpu{}
 cpuacct{}
}
```

3. Restart the cgconfig service as per the specific OS instructions.

4. Verify the custom cgroup configuration is active by getting good output from the following 3 commands:

```
cgget -g memory:ducc | grep swappiness
cgget -g cpu:ducc  | grep cpu.shares
cgget -g cpuacct:ducc | grep cpuacct.stat
```

Note: if CGroups is not installed on a machine the DUCC Agent will detect this and not attempt to use the feature. CGroups can also be disabled for all machines or for individual machines. Use *ducc.agent.launcher.cgroups.enable* and *ducc.agent.exclusion.file* respectively in *site.ducc.properties*.

Full DUCC Verification

This is identical to initial verification, with the one difference that the job “1.job” should be submitted as any user other than ducc. Watch the webserver and check that the job executes under the correct identity. Once this completes, DUCC is installed and verified.

Enable DUCC webserver login

This step is optional. As shipped, the webserver is disabled for logins. This can be seen by hovering over the Login text located in the upper right of most webserver pages:

```
System is configured to disallow logins
```

To enable logins, a Java-based authenticator must be plugged-in and the login feature must be enabled in the ducc.properties file by the DUCC administrator. Also, ducc_ling should be properly deployed (see Ducc_ling Installation section above).

A beta version of a Linux-based authentication plug-in is shipped with DUCC. It can be found in the source tree:

```
org.apache.uima.ducc.ws.authentication.LinuxAuthenticationManager
```

The Linux-based authentication plug-in will attempt to validate webserver login requests by appealing to the host OS. The user who wishes to login provides a userid and password to the webserver via https, which in-turn are handed-off to the OS for a success/failure reply.

To have the webserver employ the beta Linux-based authentication plug-in, the DUCC administrator should perform the following as user ducc:

```
1. edit site.ducc.properties
2. locate: ducc.ws.login.enabled = false
3. modify: ducc.ws.login.enabled = true
4. add:
   ducc.authentication.implementer=org.apache.uima.ducc.ws.authentication.LinuxAuthenticationManager
5. save
```

Note: The beta Linux-based authentication plug-in has limited testing. In particular, it was tested using:

```
Red Hat Enterprise Linux Workstation release 6.4 (Santiago)
```

A beta version of a File-based authentication plug-in is shipped with DUCC. It can be found in the source tree:

```
org.apache.uima.ducc.ws.authentication.SecureFileAuthenticator.java
```

The File-based authentication plug-in will attempt to validate webserver login requests by checking againist a file that only the user can view, either in the security directory specified in ducc.properties, else in the user home directory. The user who wishes to login provides a userid and password to the webserver via https, which in-turn are checked against the file-based password for a success/failure reply.

Each time a login is attempted by the user, a new password is generated in the secure file. Thus, each login password is single use.

To have the webserver employ the beta File-based authentication plug-in, the DUCC administrator should perform the following as user ducc:

```
1. edit site.ducc.properties
2. locate: ducc.ws.login.enabled = false
3. modify: ducc.ws.login.enabled = true
4. add:
   ducc.authentication.implementer=org.apache.uima.ducc.ws.authentication.SecureFileAuthenticator
5. save
```

Note: The beta File-based authentication plug-in has limited testing.

Alternatively, you can provide your own authentication plug-in. To do so:

```
1. author a Java class that implements
   org.apache.uima.ducc.common.authentication.IAuthenticationManager
2. create a jar file comprising your authentication class
3. put the jar file in a location accessible by the DUCC webserver, such as
    $DUCC_HOME/lib/authentication
4. put any authentication dependency jar files there as well
5. edit ducc.properties
6. add the following:
   ducc.local.jars = authentication/*
   ducc.authentication.implementer=<your.authenticator.class.Name>
7. locate: ducc.ws.login.enabled = false
8. modify: ducc.ws.login.enabled = true
9. save
```

DUCC webserver user data restricted access

To configure for user data secure access, use the following settings (port numbers can be any available, but normally as shown below):

```
ducc.ws.port = 42133
ducc.ws.port.ssl = 42155
ducc.ws.login.enabled = true
ducc.ws.user.data.access = encrypted
```

With these settings, users will be required to login to the DUCC Web Server to view user data (e.g. user log files). In addition, user data will be transported over https only.

There is a list of exempted URIs for which transport over http is allowed:

```
$DUCC_HOME/webserver/etc/http-uri-encryption-exemption.list
```

The exemption list allows, for example, the

```
ducc_monitor
```

to fetch status over http:

```
ducc-servlet/proxy-job-monitor-report
```

DUCC daemons monitoring and notification

```
$DUCC_HOME/bin/ducc_watcher is a Python script that, when run, contacts the
DUCC Web Server to fetch data and determine the status of the critical head node daemons.
It can be run as a cron job to detect down daemons and send email notifications
to a list of receipients specified via command invocation option.

Use the --help options for details.

Purpose: send e-mail when a DUCC daemon state changes to not up

Files created by script:
 - /tmp/<user>/ducc_watcher.state
```

```
  + comprises the last recorded state of DUCC daemons

- /tmp/<user>/ducc_watcher.log
  + comprises a log produced by the script
```