Installation Manual

This is a manual for installing the software in a distributed environment, including how to use the deploy software which makes it easy to configure and install the software. It requires some technical background to understand and use this manual.

This manual describes how to install the NetarchiveSuite web archive software package.

We first describe how to use the included deploy software to configure and install a distributed NetarchiveSuite installlation. The deploy software offers a way to gather settings for multiple machines in one configuration file, which eases the job of configuration and installation. This software generates the installation and start/stop scripts for a multiserver NetarchiveSuite system.

If you are hampered by any limitations in the deploy software, it is of course possible to make your own custom made installation scripts. An inspection of the scripts generated by the deploy software will probably help you in this respect.

For description of the configurations used for installation, please refer to the Configuration Manual

Contents

Search manual

Download as pdf

Installation Overview

Contents

The first part describes the functionality of the deploy software and how it can be used. This involves a description of how to run this module mentioning the required and optional arguments, and the functionality of the scripts generated.

The second part describes the configuration file used by the deploy software, both in structure, content and examples. This also describes the requirements and limitations of Deploy.

The third part describes the different possible installation scenarios.

The fourth part describes the means of deployment, which includes description of how to obtain and install required libraries, how to install the software on separate machines. Finally, the starting, stopping and monitoring of the system is described. This part is useful for those who want to go beyond the limitations inherent in the deploy software.

Some parts of NetarchiveSuite requires external software to run. This is described in appendix A.

This manual does not explain how to configure the applications themselves (see the Configuration Manual for this), how to extend the functionality of the system (see the development project for this) or how to use the running system (see the User Manual for this).

Audience

The intended audience of this manual is system administrators who will be responsible for the actual installation of NetarchiveSuite as well as technical personnel responsible for proper operation of NetarchiveSuite. Knowledge of Unix system administration is expected, and some familiarity with XML and Java is an advantage.

Limitations

Even though the NetarchiveSuite software is developed in Java, and therefore is mostly platform independent, we do have a couple of external calls to the Unix *sort* command. The parts of our software using this external command therefore only run on Linux/Unix, or Windows with Cygwin installed. The parts in question are:

- The `dk.netarkivet.common.GUIApplication`, if the sitesection `dk.netarkivet.viewerproxy.webinterface.QASiteSection` is used
- The `dk.netarkivet.archive.indexserver.IndexServerApplication`

Specifically the following methods all use an external call to the Unix sort() command:

- FileUtils#sortCrawlLog
  - Used in
    - dk.netarkivet.archive.indexserver.CrawlLogIndexCache,
    - dk.netarkivet.viewerproxy.webinterface.Reporting
- FileUtils#sortCDX() (only used in dk.netarkivet.archive.indexserver.CrawlLogIndexCache)
- dk.netarkivet.archive.indexserver.CDXIndexCache#sortFile()
- dk.netarkivet.viewerproxy.LocalCDXCache#getIndex()

The Software is mainly tested on a Linux platform, but with some of the BitarchiveApplication's installed on a Windows platform.

Installation Overview

Using NetarchiveSuite's Deploy utility, the steps required to configure and start a webarchive are

1. Determine the required architecture - ie how many machines you will be using, their locations, their operating systems and which applications should run on each machine
2. Configure the required machines, the required external software (see Appendices) and any relevant firewalls
3. Unpack NetarchiveSuite.zip in a directory on a linux machine
4. Create the config.xml file which describes the architecture and any custom settings. This will also specify your environmentName (e.g. MY_WEBARCHIVE).

5. Modify the other configuration files (logging and security properties) if necessary.
6. Run the Deploy utility. This will create a sub-directory MY_WEBARCHIVE with all the deploy scripts and configuration files you need.
7. Run the install scripts, then the start scripts. You should now have a running netarchivesuite installation.

Choose an Installation Scenario

Contents

Choose a platform

NetarchiveSuite can be installed in a number of different ways, with varying numbers of machines on different sites. There are a number of separate applications in play, most of which can be put on separate machines as needed. To keep clear what is necessary for which setups, we will consider the following types of setup:

- A. **Single-machine setup**. This corresponds to the setup used in the Quick Start Manual, where all applications run on the same machine, and file transfer are done by simply copying files locally. It is the simplest setup, but does not scale very well.
- B. **Single-site setup**. In this scenario, multiple machines are involved, necessitating file transfer between machines and multiple installations of the code. However, the machines are expected to be within the same firewall, so port setup should be no problem.
- C. **Single-site setup with duplicate archive**. This expands on the single-site set-up in that more than one copy of the archived files are used, using the concept of separate "Replica" to indicate the duplicates.
- D. **Multi-site setup**. When more than one site (physical location) is involved, separated by firewalls, extra issues of opening ports and specifying the correct site come into play. This is the most complex scenario, but also more secure against systematic errors, hacking, and other disasters.

Choose Repository

Scenario A and B from section *Choose a platform* involve having a local arcrepository without means of bitarchive replicas. This is configured by a plug-in (please refer to Configure PlugIns in the Configuration Manual).

Scenarios C and D from section *Choose a platform* involve having distributed bitarchive replicas. In these scenarios we have at least two bitarchive replicas. The Replica information must be configured before deployment either in the local settings file or included in the deploy configuration file for your system (please refer to Configure Repository in the Configuration Manual).

Choose the type of database

The NetarchiveSuite can use three types of database:

- Derby database (default)
- PostgreSQL
- MySQL database

By default, the NetarchiveSuite uses an external Derby. Note that from release 3.14.* the choice of an embedded Derby database has been removed to allow several applications to access the database simultaneously. The choice of the database is further described in the section on Plugins.

In fact NetarchiveSuite uses up to three distinct databases depending on which modules of NetarchiveSuite are deployed - a harvest database, an arcrepository administration database, and a wayback indexing database.

The wayback database is configured through hibernate and should be neutral as to which flavour of database is used. The harvest and admin databases are configured via NetarchiveSuites settings file or by the use of deployment settings if the NetarchiveSuite deploy-application is used.

Derby is used in the QUICKSTART installation, but from version 3.20.0 Postgresql is fully supported and is recommended for large archives where it should have superior performance and better support from external tools.

Derby Database

If you want to use a Derby database, you have to run it as a separate process. If the deploy utility is used, then setting the elements <deployHarvestDatabaseDir>harvestDatabase</deployHarvestDatabaseDir> and <deployArchiveDatabaseDir>adminDB</deployArchiveDatabaseDir> will automatically result in the deployment and start of databases in the specified directories. If you prefer to configure the databases by hand you should

1. Start Derby separately
2. `cd "directory with the extracted database" (e.g. <deployInstallDir>/<deployHarvestDatabaseDir>)`
3. `export CLASSPATH=<deployInstallDir>/lib/db/derbynet-10.4.2.0.jar:<deployInstallDir>/lib/db/derby-10.4.2.0.jar`
4. `java org.apache.derby.drda.NetworkServerControl start -p port`

The default port is 1527. Similarly set up a derby for the admin database on its own port.

For the NetarchiveSuite to use this kind of external database, you need to

- Set the setting `settings.common.database.class` to `dk.netarkivet.harvester.datamodel.DerbyServerSpecifics`.
- Set the setting `settings.common.database.url` to `jdbc:derby://<deployMachine>:1527/fullhddb` (substitute the server host for <deployMachine> and 1527 for correct port)
- Set dk.netarkivet.archive.arcrepositoryadmin.DatabaseAdmin to dk.netarkivet.archive.arcrepositoryadmin.DerbyServerSpecifics.

Need to add a permission to the policy file used by your installation, if you use security (see below). The following will allow NetarchiveSuite to access a Derby database on port 1527.

```
grant {
  permission java.net.SocketPermission
"127.0.0.1:1527",
    "connect, resolve";
};
```

Firewall note: You will need to allow the GUIApplication and the HarvestTemplateApplication to be able to access port 1527 on the server where you run the database.

More details on using Derby as a server are available on http://db.apache.org/derby/docs/dev/adminguide/cadminov825266.html the derby pages.

MySQL Database

If you want to use a MySQL database, you have to:

- Set the setting `settings.common.database.class` to `dk.netarkivet.harvester.datamodel.MySQLSpecifics`
- Set the setting `settings.common.database.url` correctly: `jdbc:mysql://localhost/fullhddb?user=root&password=secret` (substitute the server host for localhost, and username/password for root/secret)
- Install the MySQL database (v. 5.0.X) on a machine of your choice
- Download a mysql-connector-java-5.0.X-bin.jar from http://dev.mysql.com/downloads/connector/j/5.0.html
- Add a permission to the policy file used by your installation, if you use security. The following will allow NetarchiveSuite to access MySQL on localhost on the default port 3306.

```
grant {
  permission java.net.SocketPermission "127.0.0.1:3306",
    "connect, resolve";
};
```

Firewall note: You will need to allow the GUIApplication and the HarvestTemplateApplication to be able to access port 3306 on the server where you run the database.

This jar must then be added to the classpath for the applications, that accesses the database: GUIApplication and HarvestTemplateApplication

You can do this manually, when starting these applications. Alternatively, you can add the mysql-connector-java-5.0.X-bin.jar to the lib/db directory, and modify build.xml accordingly:

- Add a line `db/mysql-connector-java-5.0.X-bin.jar` to the property `jarclasspath` just below the line `db/derby-10.1.1.0.jar`.
- Add a line {<include name="db/mysql-connector-java-5.0.X-bin.jar"/> below `include name="db/derby-10.1.1.0.jar/>`

You can then generate a new NetarchiveSuite zipball with

```
ant releasezipball
```

.

This assumes, that you have downloaded the source distribution of the NetarchiveSuite.

PostgreSQL Database

NetarchiveSuite comes with scripts to initialise postgresql databases for both the harvest database and the admin database. These are in

```
scripts/sql/createHarvestDB.pgsql
scripts/sql/createAdminDB.pgsql
```

Read the header of createHarvestDB.pgsql carefully. It describes how to create a separate tablespace foe indexes. If these instructions are not followed, no such tablespace will be created and *this will have a disastrous effect on NetarchiveSuites performance*.

The settings for the two database connections look something like

```
<settings>
                <archive>
                    <admin>

<class>dk.netarkivet.archive.arcrepositoryad
min.DatabaseAdmin</class>
                        <database>

<class>dk.netarkivet.archive.arcrepositoryad
min.PostgreSQLSpecifics</class>

<baseUrl>jdbc:postgresql</baseUrl>
```

```
<machine>localhost</machine>

<port>5432</port>

<dir>admindb</dir>

<username>test</username>

<password>test123</password>
                            </database>
                        </admin>
                    </archive>
                    <common>
                        <database>

<class>dk.netarkivet.harvester.datamodel.Pos
tgreSQLSpecifics</class>

<baseUrl>jdbc:postgresql</baseUrl>

<machine>localhost</machine>
                                <port>5432</port>
                                <dir>harvestdb</dir>

<username>test</username>

<password>test123</password>
```

```
            </database>
        </common>
    </settings>
```

The meaning of the various settings should be fairly obvious: they specify the machine and port-number of the postgresql server, the names of the two databases and the credentials for accessing the databases.

Choose a JMS broker

NetarchiveSuite requires a JMS broker to run. The only type of JMS broker supported at this time is the SunMQ broker and its open source counterpart Open Message Queue.

The installation and start-up of a JMS broker is described in Appendix A.

For description of how to configure the JMS broker, please refer to the Configure JMS Broker.

Firewall note: The machine that runs the JMS broker must be accessible from all machines in the installation on not only port 7676, but also port 33700 (from RMI).

Java

All machines must run Java version 1.6.0 or higher.

Choose the set of machines taking part in the installation/deployment

When you have chosen a scenario, you must decide on the number of machines, you want to use in the deployment of the NetarchiveSuite. For scenario A, the answer is of course one. For the scenarios B, C, and D, the answer is more complicated.

An extra complication is added by installing the system at two different physical location (here referred as EAST and WEST). The distinction between different physical location are relevant if the system is installed at two different institutions with firewalls between them.

At the Danish installation, we operate with 4 kinds of machines:

- Admin machine (one server): Here we deploy one or more BitarchiveMonitorApplications (one for each bitarchive Replica), one ArcrepositoryApplication, one GUIApplication, and a JobManagerApplication, which takes care of job scheduling.
- Harvester machines (one or more): Here we deploy the HarvesterControllerApplications.
- Bitarchive machines (one or more): These machines only run one BitarchiveApplication each (there must be at least one for each bitarchive Replica).
- Access servers (one or more): On these machines, we have the ViewerproxyApplication enabling us to browse in already stored webpages, and the IndexServerApplication. The latter must only be installed on one of the access-servers, as there can only be one in the system.

Apart from the HarvestControllerApplications, there is no requirement that the applications are placed like this, but we will use it as an example throughout the rest of the manual. In the standard set-up used in our test-environment, we have 9 machines:

- 1 bitarchive server (on physical location WEST)
- 2 bitarchive servers (on physical location EAST)

- 1 admin machine (placed on physical location EAST)
- 1 harvester-machine (placed on physical location WEST)
- 2 harvester-machines (placed on physical location EAST)
- 1 access server (placed on physical location WEST)
- 1 access server (placed on physical location EAST)

Choose other plug-ins

Except from the plug-ins described in this section, the installation of plug-ins consists only of the configuration of them.

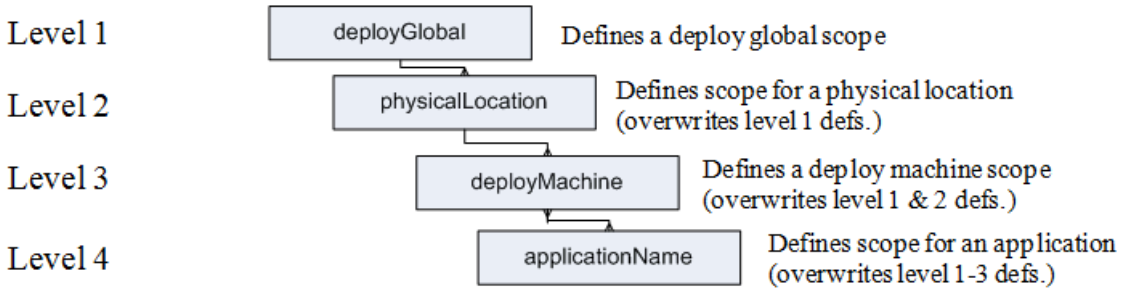
Functionality of the Deploy Software

Contents

Functionality of the Deploy Software

The main function of deploy is to install and configure NetarchiveSuite on a distributed system. This is done through scripts to install, start and stop the applications of NetarchiveSuite based on a configuration file for the system. A sample file is provided with NetarchiveSuite in the file examples/deploy_distributed_example.xml.

The figure below shows the hierarchy of the instances in the deploy configuration file.



Terminology

- environmentName: The required value in the deploy configuration file.
- machineUser: The login for the machine.
- installDir: The directory on a machine where the installation is done. This is the directory environmentName from the ssh initial directory. Linux path: `/home/machineUser/environmentName/`, and most versions of Windows uses the path: `C:\Documents and Settings\machineUser\environmentName`, except Windows Vista (and newest equivalent server) which has the path: `C:\Users\machineUser\environmentName`.

Performing a deploy

The Deploy module has to be run from a Linux/Unix machine, since the scripts for handling the physical locations only works on this platform. Some of the application are supported on Windows, and therefore some machines with Windows as operating system can be used in the distributed system. Just not the machine where the deployment takes place, since the deployment is done through the scripting language Bash which only works on Linux/Unix. The figure below shows what happens when the deploy application is run.

Deploy arguments

Deploy takes the following arguments:

- -C - The configuration file for deploy, has to have the '.xml' suffix.
  - The required structure of this file is described in the Configuration file section. It has to be XML parseable.
- -Z - The NetarchiveSuite file, has to be '.zip'.
  - This is the NetarchiveSuite package file, which is unzipped on all the machines during installation. This contains the libraries which is used when applications are run. The NetarchiveSuite package file is copied to the output directory when deploy is run.
- -L - The log property file, has to be '.prop'.
  - This file contains the basic properties for logging. A copy of this file is made for each machine, where it is changed to fit purposes of the machine. See the Log property file section under Files.
- -S - The security policy file, has to be '.policy'.
  - The security policy file defines where the applications are allowed to operate. A copy of this file is made for each machine, where the required security properties for the applications are granted. See the Security Policy file section under Files.
- -O [OPTIONAL] - The output directory.
  - This is the directory on the root machine (the machine where deploy is run from) where the scripts and setting files are created by deploy (the environmentName is used as default name for the output directory).
- -D [OPTIONAL] - The database, has to be either '.zip' or '.jar'.
  - The database where the harvesting informations are to be located. If the database is not given as an argument, the default database in NetarchiveSuite package file is used. The database has to be placed in an unzippable file ('.zip' or '.jar'), and it is only unzipped on machines where a database

directory has been defined. Currently databases are only supported on Linux machines.

- -R [OPTIONAL] - Whether the temporary file directory should be reset. Any argument different from 'y' or 'yes' will be considered a 'no'.
    - During installation some directories are created, if they do not already exists. This argument defines whether the temporary directory should be cleared during installation (or reinstallation).
- -T [OPTIONAL] - For creating a test instance.
    - The argument is required to have the following format: 'HttpOffsetPort,HttpPort,EnvironmentName,MailReceivers' (no spaces between them). A new config file is created based on these inputs and the given config file (this file has the same name, just with the extension '_test.xml' instead of '.xml'). See the Test instance section.
- -E [OPTIONAL] - For evaluating the config file. Any argument different from 'y' or 'yes' will be considered a 'no'.
    - This evaluates whether the settings in the deploy configuration file is compatible with the standard settings. See the Evaluation section below.
- -A [OPTIONAL] - The archive database, has to be either '.zip' or '.jar'.
    - This database will be used for both the ArcRepository and the DatabaseBasedActiveBitPreservation. If the database is not given as an argument, a default empty archive database in the NetarchiveSuite package file is used. The database has to be placed in an unzippable file ('.zip' or '.jar'), and it is only unzipped on machines where the <globalArchiveDatabaseDir> parameter is defined in the configuration. This is currently only supported on Linux machines.

Other dependencies

Deploy requires the following libraries in the classpath:

- `dk.netarkivet.deploy.jar`
- `dk.netarkivet.archive.jar`
- `dk.netarkivet.common.jar`
- `dk.netarkivet.harvester.jar`
- `dk.netarkivet.monitor.jar`
- `dk.netarkivet.viewerproxy.jar`
- `dom4j-1.6.1.jar` (or newer)
- `commons-logging-1.0.4.jar` (or newer)
- `commons-cli-1.0.jar` (or newer)
- `jaxen-1.1.jar` (or newer)

Deploy uses Java 1.6 and therefore this has to be put in the path before calling the java application.

Note that you only need to mention the dk.netarkivet.deploy.jar explicitly in the classpath, because the others are referenced inside the dk.netarkivet.deploy.jar

Example

The complete call (without optionals) for running deploy will therefore be the following (with `lib/` being the directory for the libraries):

```
export JAVA_HOME=/usr/java/jdk1.6.0_07
export PATH=$JAVA_HOME/bin:$PATH
java -cp lib/dk.netarkivet.deploy.jar
dk.netarkivet.deploy.DeployApplication
-Cdeploy_config.xml -ZNetarchiveSuite.zip
-Ssecurity.policy -Llog.prop
```

where `deploy_config.xml` is the name and path to the configuration file, `NetarchiveSuite.zip` is the path of the NetarchiveSuite package, `security.policy` is the path of the security policy file and `log.prop` is the path of the property file for logging. Java version 1.6.0_07 is specifically called here, though any Java version above 1.6.0 should be usable.

Files

When deploy is run a number of files are created in the output directory. These includes scripts to install, start and kill the applications on the distributed platform. Also the NetarchiveSuite package file is copied to this location (unless it already exists in the output directory).

In addition to a NetarchiveSuite settings file, the following configuration files are also created on a per-machine or per-application basis:

Jmxremote password file

This file is created from scratch for each machine. A large instructional header for the use of the `jmxremote.password` is initially created for the file, then the jmx username and jmx password for the monitor and for heritrix is appended. It is only the jmx logins (username and password), which is used by the applications.

The login variables for the monitor are found through the paths in the settings for any of the applications: `settings.monitor.jmxUsername` and `settings.monitor.jmxPassword`.

The login variables for heritrix are found through the paths in any of the application settings: `settings.harvester.harvesting.heritrix.jmxUsername` and `settings.harvester.harvesting.heritrix.jmxPassword`.

If any application has a monitor defined in the settings file, the monitor must have a jmx login defined. The monitor jmx logins has to be the same for all applications on a machine. This also applies for heritrix jmx logins, though the monitor jmx login and heritrix jmx login does not have to be the same.

Log property file

A log property file for each application is created. This file is given as input and it is changed to fit the application.

The only change in the log property file is changing the tag `APPID` to the identification of the application (`applicationName + " " + applicationInstanceId`). Where the `" " + applicationInstanceId` only is appended to the `applicationName` if the application has an `applicationInstanceId` defined.

The name of this application specific log property file is: `"log_" + applicationIdentification + ".prop"`. Where the `applicationIdentification` is given as `applicationName + " " + applicationInstanceId`, as described above.

Security policy file

The security policy file for a machine is initially a copy of the security policy file given as argument. This machine specific security policy file is then modified to suit the needs of the machine and it's applications.

The tag ROLE is replaced by the monitor.jmxUsername for the machine. This has to be defined on the machine level in the deploy configuration file.

Permission to read the baseFileDir under bitarchive for all applications is granted. The path to these directories are changed to fit the language in security policy.

Evaluate

It is possible to evaluate the content of the configuration file when deploying, by giving the '-E' parameter with argument either 'y' or 'yes'. This is a tool for finding bugs within a configuration file (e.g. a mispelled name or wrongly placed branch).

This checks if the all the branches in the configuration file can be found within the default settings, and makes a warning for those it cannot find. It does not check if the content of these branches are correct (e.g. http-port = -1), it only checks whether the branches also exists in the default settings.

Deploy does not abort the program when unknown branches are found. It only generates warnings about each unknown branch and then continues with the deployment.

Some module have plugins which uses some values within the settings, which is not part of the default settings, and they will therefore be noted as unknown. Such plugin specific branches should not be considered errors, even though warnings are issued for these.

Test instance

In the case where test argument are given a new configuration file is created, with the _test appended to the name (e.g. deploy_config.xml will have the test instance configuration file: deploy_config_test.xml).

The following test arguments are given: `test_HttpOffsetPort`, `test_HttpPort`, `test_EnvironmentName`, and `test_Mailreceivers`. These arguments are given without spaces between them in the above order. An `Offset` variable is calculate as the difference between the `test_HttpPort` and the `test_HttpOffsetPort` (e.g. `Offset` = `test_HttpPort` - `test_HttpOffsetPort`). The value of this `Offset` must be between 0 and 9 .

The test argument is applied to deploy_config_test file, where the following changes are made:

- The environtmentName is changed to `test_EnvironmentName`.
- For every level the `test_HttpPort` replaces the value in the settings path: settings.common.http.port.
- For every level the `test_Mailreceiver` replaces the value in the settings path: settings.common.notification.receiver.
- For every level the `Offset` replaces a single digit in some four-digit ports under settings. This is seen in the table below.

Path	index
settings.common.jmx.port	3

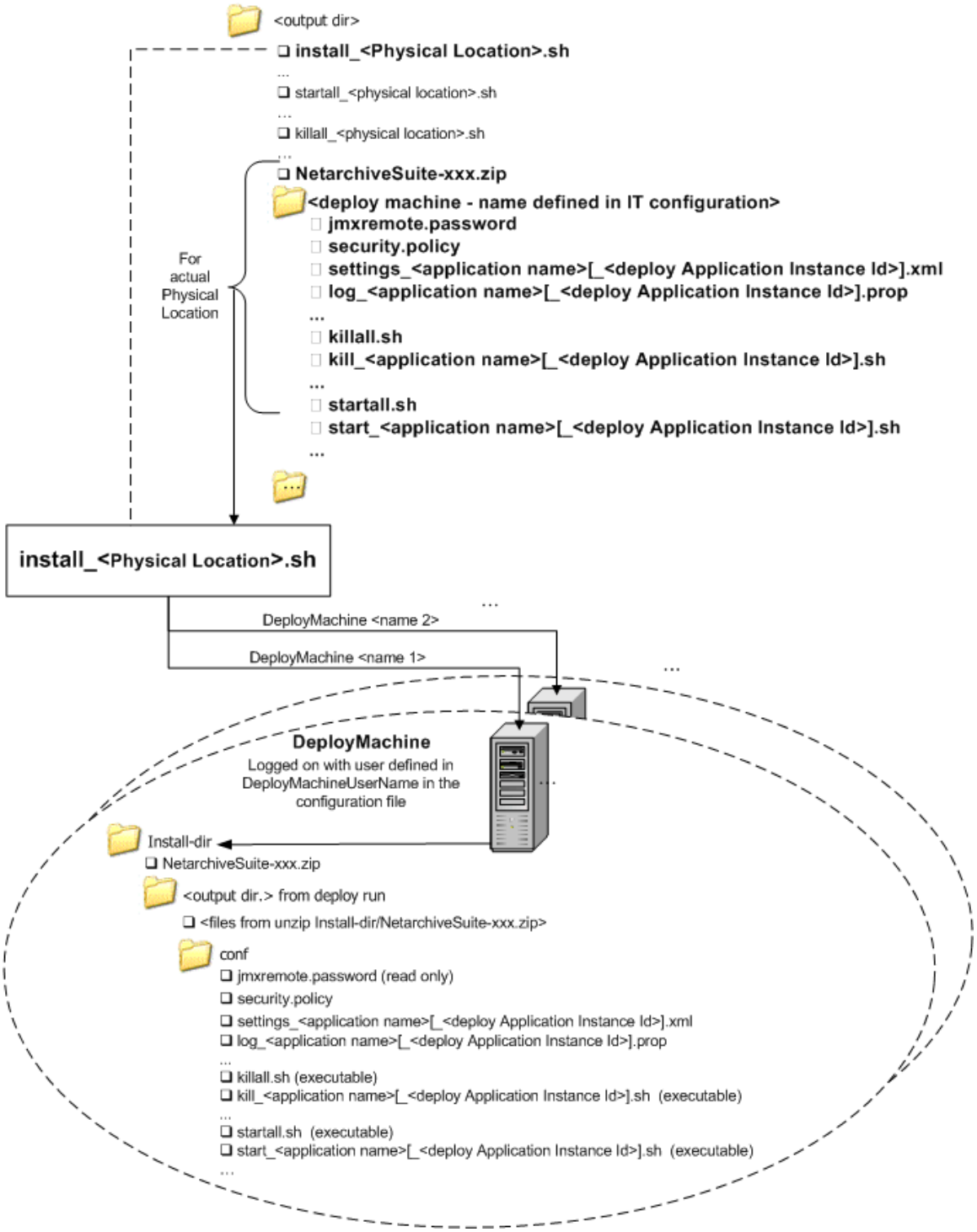
settings.common.jmx.rmiPort	3
settings.harvester.harvesting.heritrix.guiPort	2
settings.harvester.harvesting.heritrix.jmxPort	2

E.g. `Offset` = 7 and a settings.common.jmx.port = 1234 will yield a new settings.common.jmx.port = 1274 for the test instance, whereas a `settings.harvester.harvesting.heritrix.jmxPort = 1234` will yield a new `settings.harvester.harvesting.heritrix.jmxPort = 1734`.

Install

An installation script is created for each physical location. This script contains the commands for making the installation on all the machine of the physical location as described in the pseudo code.

The figure below shows the pattern of installation.



Install script pseudo code

. The install script for a physical location has the following procedure:

- for each machine do the following.

1. Install the NetarchiveSuite file.
2. Install the necessary directories.
3. Install scripts, settings and database.

Install the NetarchiveSuite file

The NetarchiveSuite file is copied to the machine using scp (secure copy). Then file is unzipped in the installation directory, which is created as a subdirectory in the local user directory.

Install necessary directories

In the config file a number of directories are defined, and these directories have to be created during the installation on a machine. The following table show which directories are created based on the main branch where they are defined, and their path from this branch. The branch level represents where the applications have to be defined before they can be applied. They can easily be defined in a prior instance, and then be inherited to the given branch level.

Path	Directory	Branch level
settings.harvester.harvesting.serverDir	$/	applicationName
settings.archive.bitarchive.baseFileDir	$/	applicationName
settings.archive.bitarchive.baseFileDir	$/filedir/	applicationName
settings.archive.bitarchive.baseFileDir	$/tmpdir/	applicationName
settings.archive.bitarchive.baseFileDir	$/atticdir/	applicationName
settings.viewerproxy.baseDir	$/	applicationName
settings.archive.bitpreservation.baseDir	$/	deployMachine
settings.archive.arcrepository.baseDir	$/	deployMachine
settings.tempDir	$/	applicationName

where $/ in Directory is the value of the path. All the directories along this path will be created, if they do not exists already. A directory is only created if the path is defined under settings for the branch level (or inherited to the branch level) and it contains a not empty value.

The installation of the directories will be executed from the installDir. The directories will only be installed if they do not already exist, with the optional exception of the tempDir, which will be removed before creation if the `-R` argument is set to 'yes'. It is only the directory at the end of the path, which has its content removed, not all the directories along the path. E.g. a tempDir with the path `myPath/myEndDir` will only clean the directory '`myEndDir`', and not the directory '`myPath`'.

On Linux/Unix machines directories are created directly through `ssh`, while Windows machines use a batch program, which is installed, run and then deleted.

This is because only a single command line can be run through `ssh`, and this command line is run as `bash` on Linux/Unix and as `batch` on Windows. Since `bash` can take many commands on a single command line, it is possible to install all the directories through `ssh` on Linux/Unix. `batch` on the other hand can only handle a single command per command line, and the directories can therefore not be installed through a single `ssh` call. The `batch` commands to install the directories are therefore combined in a `batch` program, which is installed on the windows machine, then run and afterwards deleted.

Install scripts, settings and database

The jmxremote.password file has to be not-writable when the applications are running, which means that a reinstallation of this file cannot happen before it is made writable again.

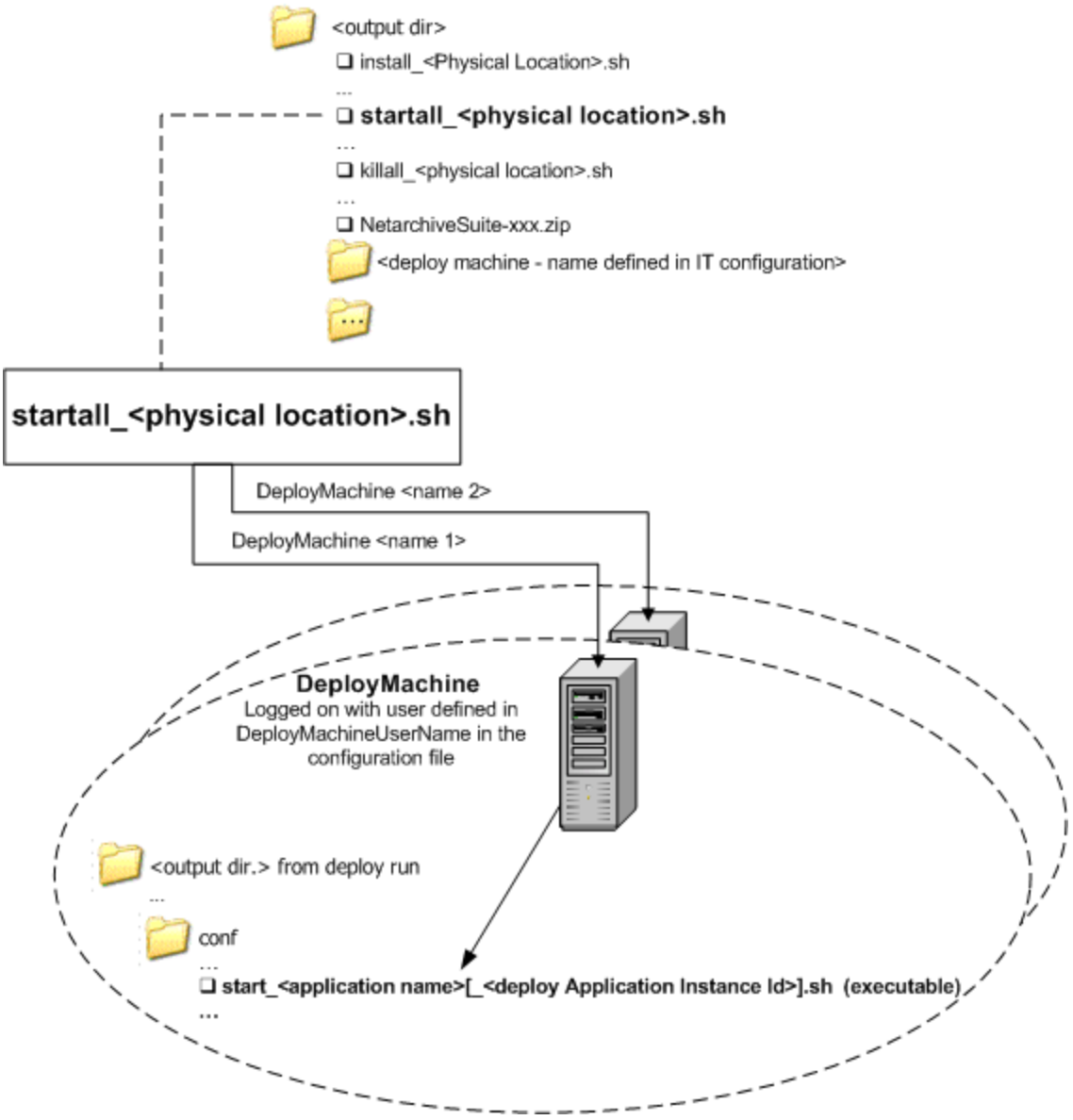
Then all the script and setting files are copied from the local directory with the machine name to the 'conf/' directory in the installation directory on the machine.

Then the optional database is handled, though only on the machines with a specified database directory. This database overrides the existing standard database in the NetarchiveSuite package. The database is then unzipped to the database directory, but only if it is empty.

Then the scripts are made executable and the jmxremote.password is made read-only.

Start, Restart and Kill

The figure below shows how the applications are started, and the same pattern are used for killing the applications again (replace start with kill in the figure).



Note that an application cannot be started if it is already running, and how this is checked is different on the two supported platforms: Linux and Windows platforms, as we will see below.

The restart script can be used for restarting the running applications. It starts by calling the killall script, then waits 5 seconds for the applications to terminate completely, and finally runs the startall script. This script can be used for Windows Services (automatic execution during startup).

Linux

On the Linux platform an application is only started if no instances of this application be found among the running processes. Likewise an application is only killed if it can be found in the process list.

The way an instance of a specific application can be found amongst the list of running processes, is by looking for any process with the same name, and which is using the same settings file.

When killing the an application of the instance `dk.netarkivet.harvester.harvesting.HarvestControllerApplication`, then the Heritrix application is also killed.

Windows

It requires several files on windows to run the application, and making sure that maximum one instance of the application is running. Two scripts for killing it, two scripts for starting it and one temporary file for telling whether it an instance is running.

The application can only be started if the temporary run-file does not exist. It is done by calling a VBS script for running the application. This script starts the application as a process and saves method for killing this process in a kill-process file.

The application can only be killed if the temporary run-file exists. The kill-process file is called for killing the process of the application. Then the temporary run-file is removed, thus telling that the application is not running and can be started again.

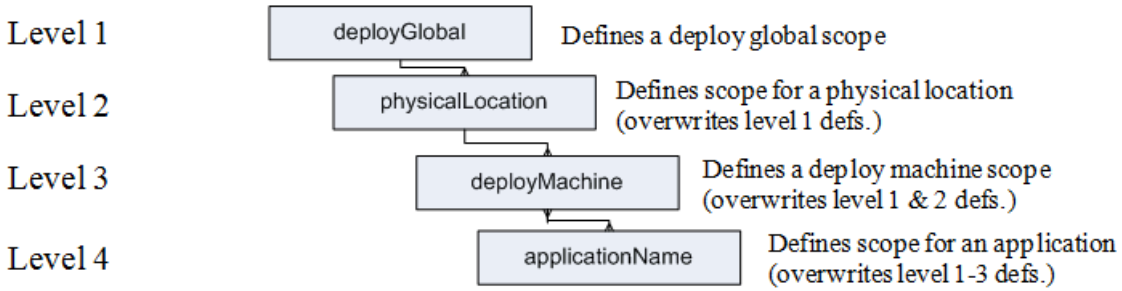
The Heritrix application is not killed when an application of the instance `dk.netarkivet.harvester.harvesting.HarvestControllerApplication` is killed. This is because a Heritrix is not throughly tested on Windows, and might not be supported.

The Deploy Configuration File

Contents

The deploy configuration file contains the definitions for the installation and distribution of !NetarchiveSuite. This involves the scopes for the levels in the figure below, and their settings.

This figure also shows the pattern of inheritance of the settings (`physicalLocation` inherits settings and parameters from `deployGlobal`, `deployMachine` inherits from `physicalLocation`, etc.).



These levels can have several instances of the levels below them.

Settings scope

The settings scope is described in the Configuration Manual for NetarchiveSuite. It is no longer required that every variable within the settings scope is explicitly defined for an application, since the undefined variables are replaced by the default settings, when the application is run.

Each level (in the figure at the beginning of this section) inherits the settings from the level above it (until deployGlobal), though only the variables which is not explicitly defined at the current level. The content of the settings scope at the application level (level 4) is printed into an application specific settings file, which is used for running the application.

Some parts within the settings scope is used by deploy, and they will be described in the following section.

Deploy scope

The levels in the figure can have an instance of the settings scope defined. These settings are inherited through the hierarchy.

The scope levels of Deploy:

- <deployGlobal>
  . Defines a deploy global 1. level scope where settings can be set to overwrite setting defaults.
- <thisPhysicalLocation name="...">
  . Defines the 2. level scope for a physical location. The settings for this scope will overwrite the settings for the 1. level scope (deployGlobal). The attribute 'name' for thisPhysicalLocation overwrites settings.common.thisPhysicalLocation.
- <deployMachine name="..." os="...">
  . Defines a deploy machine 3. level scope where common settings for the machine and the applications running in the machine can be set. These settings will overwrite 1. and 2. level settings. The attribute 'name' for the machine is the network name the machine, and will be used for communicating with the machine. The attribute 'os' is optional and defines the operating system on the machine. If 'os' is not set or has value different from 'windows' (not case sensitive), then the default 'Linux/Unix' is used.
- <applicationName name="...">
  . Defines the 4. level scope where the application specific settings are placed. These settings will overwrite the inherited 1., 2. and 3. level settings. The attribute 'name' for applicationName is used for calling the application. Only the last part of the name is used for all purposes (except running the application) and it overwrites settings.common.applicationName
  (e.g. the application `dk.netarkivet.archive.bitarchive.BitarchiveApplication` will have the name BitarchiveApplication). If the application has an specific applicationInstanceId, it is specified under settings.
  One level can have several instances of a lower level (e.g. a deployMachine can have several applicationName, and not vice versa).

This will look like the following:

```
<deployGlobal>
    <thisPhysicalLocation
name="myPhysicalLocation">
        <deployMachine name="myMachine"
os="linux">
            <applicationName
name="myApplication">
            </applicationName>
            <applicationName
name="myOtherApplication">
            </applicationName>
        </deployMachine>
        <deployMachine name="myOtherMachine"
os="windows">
            <applicationName
name="myApplication">
            </applicationName>
        </deployMachine>
    </thisPhysicalLocation>
</deployGlobal>
```

This configuration has one physical location with two machines, one with Linux/Unix and one with Windows. The Linux/Unix machine has two applications, 'myApplication' and 'myOtherApplication', while the Windows machine has only one application, 'myApplication'.

Parameters

Each of the above scopes can have several of the following parameters defined. These parameters can be applied to each of the above scopes, and they are inherited from the parent scope in the same way as settings.

The parameter scopes the levels can have:

- <deployClassPath>
  . Defines a class path to be added for running an application. Note: several additional class paths can be specified within a scope, but new definitions in inner scopes will overwrite outer scopes.

- <deployJavaOpt>
  . Defines a Java option for an application. Note: several additional java options can be specified within a scope, but new definitions in inner scopes will overwrite all outer scopes.
- <deployInstallDir>
  . Defines the installation directory for a deployMachine, can only handle one deployInstallDir. Note: only one install directory is supported (if several, a warning is placed in the log and the first install directory is used).
- <deployMachineUserName>
  . Defines the user name for a deployMachine. This is used when communicating with the machine. Note: only one machine user name is supported (if several, a warning is placed in the log and the first machine user name is used).
- <deployDatabaseDir>
  . Defines the directory for the database to unzipped. This directory can be full path or path relative to install directory. It is an optional parameter for defining where a machine should have the database unpacked, and if the machine does not include this parameter it will not have the database unpacked. Also it requires the settings.common.database.url set. Note: This must be set on the machines where the database are to be unpacked. Only one database directory is supported (if several, a warning is placed in the log and the first database directory is used).
- <deployBitpreservationDatabaseDir>
  . Defines the directory for the bitpreservation database to be unzipped. This directory can be full path or path relative to the installation directory. It is an optional parameter for defining where a machine should have the bitpreservation database unpacked, and if a machine does not have this parameter it will not have the database unpacked.
  An example of how this works is given below.

```
<deployGlobal>

<deployClassPath>lib/dk.netarkivet.common.ja
r</deployClassPath>

<deployClassPath>lib/dk.netarkivet.archive.j
ar</deployClassPath>
    <deployJavaOpt>-Xmx1536m</deployJavaOpt>
    <thisPhysicalLocation
name="myPhysicalLocation">

<deployMachineUserName>myUserName</deployMac
hineUserName>
        <deployMachine
name="myLinuxMachine">
```

```
<deployInstallDir>/home/myUserName/myInstall
ationDirectory</deployInstallDir>

<deployDatabaseDir>myDatabaseDir</deployData
baseDir>
            <settings>
                <common>
                    <database>

<url>jdbc:derby:myDatabaseDir/fullhddb</url>
                    </database>
                </common>
            </settings>
            <applicationName
name="myLinuxApplication">
            </applicationName>
        </deployMachine>
        <deployMachine
name="myWindowsMachine" os="windows">

<deployInstallDir>C:\myInstallationDirectory
</deployInstallDir>

<deployJavaOpt>-Xmx1150m</deployJavaOpt>
            <applicationName
name="myWindowsApplication">

<deployClassPath>lib/dk.netarkivet.common.ja
r</deployClassPath>

<deployClassPath>lib/dk.netarkivet.harvester
```

```
.jar</deployClassPath>

<deployClassPath>lib/dk.netarkivet.viewerpro
xy.jar</deployClassPath>
            </applicationName>
```

```
        </deployMachine>
    </thisPhysicalLocation>
<deployGlobal>
```

This defines two different machines each with a single application. These machines have different operating systems (one with windows and one with linux), and therefore they have different installation directories and Java options.

The Linux machine inherits the Java option `-Xmx1536m` from the physical location, which inherits it from deployGlobal. The Windows machine has a Java option specified and does therefore not inherit deployGlobal Java option.

The deployDatabaseDir is only specified on the Linux machine, and the database will therefore be unpacked only on this machine. It is specified in settings.common.database.url what type the database is, and where the it is found after it is unpacked. If a specific database is not given as parameter when calling deploy the default Derby database 'fullhddb.jar' is used.

The application myLinuxApplication on the Linux machine does not have any class paths specified, and does therefore inherit the `lib/dk.netarkivet.common.jar` and `lib/dk.netarkivet.archive.jar` all the way from deployGlobal (through thisPhysicalLocation and deployMachine).

On the other hand does myWindowsApplication on the Windows machine not inherit these libraries, since it has its own class paths specified. It has the libraries `lib/dk.netarkivet.common.jar`, `lib/dk.netarkivet.harvester.jar` and `lib/dk.netarkivet.viewerproxy.jar` in the class path, and does therefore not have the `lib/dk.netarkivet.archive.jar` since it is neither specified nor inherited.

The myLinuxApplication will be called with the following command:

```
java -Xmx1536m -cp
lib/dk.netarkivet.common.jar:lib/dk.netarkivet.archive.jar myLinuxApplication
```

The myWindowsApplication will be called with the following command:

```
java -Xmx1150m -cp
lib/dk.netarkivet.common.jar;lib/dk.netarkivet.harvester.jar;lib/dk.netarkivet.viewerproxy.jar myWindowsApplication
```

The class paths are separated with ':' on Linux/Unix and with ';' on Windows.

Application Instance Id

The scope settings.common.applicationInstanceId defines identification of a single application instance (e.g. suffix for application specific scripts, suffix for directory to place files etc.). This is needed in cases where there are more instances of the same application are placed on the same machine (e.g. BitarchiveMonitors)

An example of two identical applications with different application instance id on the same machine is given below:

```
<deployGlobal>
    <thisPhysicalLocation
name="myPhysicalLocation">
        <deployMachine name="myMachine">
            <applicationName
name="dk.netarkivet.archive.bitarchive.Bitar
chiveApplication">
                <settings>
                    <common>

<applicationInstanceId>myFirstInstance</appl
icationInstanceId>
                    </common>
                </settings>
            </applicationName>
            <applicationName
name="dk.netarkivet.archive.bitarchive.Bitar
chiveApplication">
                <settings>
                    <common>

<applicationInstanceId>mySecondInstance</app
licationInstanceId>
                    </common>
                </settings>
            </applicationName>
        </deployMachine>
    </thisPhysicalLocation>
</deployGlobal>
```

These application will be called !BitarchiveApplication_myFirstInstance and !BitarchiveApplication_mySecondInstance respectivly.

Limitations and Requirements

And deploy has the following requirements:

- The environmentName (settings.common.environmentName) has to be set in settings on the global level.
- The environmentName (settings.common.environmentName) must be a combination of digits (0-9) and the letters (a-z, lower or upper case). Deploy fails if the environmentName contains other characters.
- Different environmentNames between physical location level, machine level and application level is not supported (or meaningful).
- Databases are not supported on Windows.
- The GUIApplication and the !ArcRepositoryApplication must be placed on the same machine.
- The install directory on Windows must be "C:\Documents and Settings\user\", where user is the username on the machine. Except Windows Vista (or equivalent server os), where the directory must be `C:\Users\user`, where user is the username on the machine.
- All applications on the same machine with jmx login for monitor must have identical login.
- All applications on the same machine with jmx login for heritrix must have identical login.
- When creating a test instance, the arguments 'http-port' and 'offset' is only supported as 4 digit numbers.
- Every physical location, machine and application must have the name attribute defined.
- Deploy does not handle network connection permissions. E.g. if there is a firewall, it has to be setup to allow the applications in NetarchiveSuite to communicate with each other.
- Permission to create the wanted directories is required.
- The unzip command (or program) has to be accessible through 'ssh'.
- Two instances of the same application on the same machine must have different applicationInstanceIds.
- Several instances of the same setting cannot extend one setting. E.g. a physical location with several instances of the remoteFile defined need to have each remoteFile setting completely defined, since they are not extended by a single remoteFile in the global settings.

The deploy configuration has the following limitations in comparison to the manual installation.

- Only embedded Derby databases have been tested with the new Deploy, and other databases have to be installed manually.
  The limitations and requirements for the configuration of the applications can be found in the Configuration Manual. Specific for deploy are the following:

- Every application must have a jmx-port and rmi-port, and they must be unique for the machine where the application is running.
- dk.netarkivet.harvester.harvesting.!HarvestControllerApplication does not run on Windows machines.
- A dk.netarkivet.archive.bitarchive.!BitarchiveApplication must have at least one settings.archive.bitarchive.baseFileDir defined.
- Only the dk.netarkivet.archive.bitarchive.!BitarchiveApplication is properly tested on the Windows platform. Some of the other applications should work, though they have not been tested enough to say for certain.
- If a machine has several instances of dk.netarkivet.archive.bitarchive.!BitarchiveApplication, then each application must have a unique temporary file directory defined (settings.common.tempDir).

Configuration example

Here is an example of a configuration file for deploy.

Example of deploy configuration file

The following part of this section describes how to change this configuration file template to fit your specific system.

This describes how to make the changes, scope for scope, to fit a system with the same structure,
and it describes how to expand the scopes with new machines and applications.

Deploy Global

The `deployGlobal` scope contains two parts: the parameters and the settings.

Just leave the `<deployClassPath` parameters, since they will be overwritten for the applications which need other libraries.
The `<deployJavaOpt>-Xmx1536m</deployJavaOpt>` parameter just sets the maximum heap size to 1.5 GB (1536 MB).
This value should not be larger than the amount of accessible memory on a machine.

Within the settings scope of `deployGlobal` the following needs to be done.

The environment name is not required to be changed for the system to work, though it is usually a good idea to change this to a more appropriately name for the installation or system.
This is the settings at 'settings.common.environmentName'.

```
<settings>
        <common>

<environmentName>test</environmentName>
        <common>
    <settings>
```

The replicas should be changed to fit the system.
A replica will generally be connected to a specific physical location, though a physical location can have several replicas.
These settings can be found under 'settings.common.replicas'.

```
<settings>
        <common>
            <replicas>
                <replica>
                    <replicaId>A</replicaId>

<replicaName>ReplicaA</replicaName>

<replicaType>bitArchive</replicaType>
                </replica>
                <replica>
                    <replicaId>B</replicaId>

<replicaName>ReplicaB</replicaName>

<replicaType>bitArchive</replicaType>
                </replica>
            </replicas>
        <common>
    <settings>
```

The JMS-broker is defined at the global level, and it should be set to the administation machine, e.g. the machine with the `dk.netarkivet.common.webinterface.GUIApplication`, the `dk.netarkivet.archive.arcrepository.ArcRepositoryApplication` and the instances of `dk.netarkivet.archive.bitarchive.BitarchiveMonitorApplication` should be run.
This is defined in the settings: 'settings.common.jms.broker'.

```
<settings>
            <common>

<broker>kb-test-adm-001.kb.dk</broker>
            <common>
        <settings>
```

If more replicas are wanted, they have to be defined in the settings at the `deployGlobal` level.
Each replica needs a unique `replicaId` and `replicaName`, and it also needs the following applications: `dk.netarkivet.archive.bitarchive.BitarchiveApplication`, and `dk.netarkivet.archive.bitarchive.BitarchiveMonitorApplication`.

Physical Locations

The configuration example file has two physical locations: EAST and WEST.
Every physical location need to have a unique name.

```
<thisPhysicalLocation name="EAST">
        ...
    </thisPhysicalLocation>
    <thisPhysicalLocation name="WEST">
        ...
    </thisPhysicalLocation>
```

For the settings of a physical location the following need to be done.
A physical location needs to know which replica it uses.
This replicaId has to be amongst the replicas defined in the `deployGlobal` scope.
It has the path: 'settings.common.useReplicaId'.

```
<settings>
            <common>

<useReplicaId>A</useReplicaId>
            </common>
        </settings>
```

If using FTPRemoteFile, it is necessary to specify a machine on which an ftp server is running, together with valid login credentials, for example

```
<remoteFile>

<serverName>kb-test-har-001.kb.dk</serverNam
e>

<userName>ftptestuser</userName>

<userPassword>ftptestpasswd</userPassword>
                </remoteFile>
```

The notifications settings should be setup to tell where mails should be sent.
The receiver should be changed to the mail of the administrator of the system.

```
<notifications>

<sender>example@netarkivet.dk</sender>

<receiver>example@netarkivet.dk</receiver>
                </notifications>
```

It is currently not possible to have more than two physical locations, but this problem will be dealt with, and it will be possible in a future release.

Machine

The name of a machine has to be change to the network ID, e.g. either network name or IP address.
The 'os' attribute should only be set for the windows machines, which can only run applications of the instance `dk.netarkivet.archive.bitarchive.BitarchiveApplication`.

```
<deployMachine os="windows"
name="kb-dev-bar-011.bitarkiv.kb.dk">
```

Change the following parameters to fit to the machine definition:
A machine needs to have the following parameters defined (they can also be defined at the physicalLocation level, and then just be inherited).

```
<deployMachineUserName>test</deployMachineUserName>

<deployInstallDir>/home/test</deployInstallDir>
```

There are no specific settings required at the machine level, which is not inherited by the outer scopes.
And therefore no settings to change to fit to your system.

A new machine has to be created within a physical location scope.
It requires the name attribute, and the parameters `deployMachineUserName` and `deployInstallDir` has to be defined or inherited.
The parameter `deployDatabaseDir` is required, if the machine runs an application which requires a database.

Application

All applications need the following settings defined under `settings.common.jmx`:

```
<port>8100</port>

<rmiPort>8300</rmiPort>
```

These port values must be unique for the machine, where the application should run.

A new application needs the name attribute to be defined as the name in the classpath for the application. E.g:

```
<applicationName
name="dk.netarkivet.common.webinterface.GUIA
pplication">
```

It is important to notify that when a new application is added to a machine, which already has an application of the same instance, these applications must have the `settings.common.applicationInstanceId` defined with different values.

Some of the applications require some specific settings to be defined.
This is described in the following specifically

BitarchiveApplication

The `dk.netarkivet.archive.bitarchive.BitarchiveApplication` requires the settings `settings.archive.bitarchive.baseFileDir` to be defined.
This path should be changed, and it has to be changed if the drive/partition in the path does not exist on the machine.

HarvestControllerApplication

For the `dk.netarkivet.harvester.harvesting.HarvestControllerApplication` the following settings defined under `settings.harvester.harvesting.heritrix` should be changed to fit your system: `guiPort` and `jmxPort`.

A new instance of the `dk.netarkivet.harvester.harvesting.HarvestControllerApplication` requires the settings `settings.harvester.harvesting.queuePriority` to be defined to either `LOWPRIORITY` or `HIGHPRIORITY`.
A system requires at least one !HarvestControllerApplication with each priority.

How to add a harvester more on the same machine and set all to HIGHPRIORITY selective harvesting

Using eg deploy_examplexml

- Duplicate the existing harvester <applicationName> definition within <deployMachine>.

In the new duplicate harvester config, change all following duplicate values to new unique values within <deployMachine>:

- <applicationInstanceId>
- <common><jmx><port> and <rmiPort>
- <heritrix><guiport> and <jmxPort>
- <serverDir>harvester_high_2</serverDir>

and set

- <queuePriority>HIGHPRIORITY</queuePriority>

eg:

```
<applicationName
name="dknetarkivetharvesterharvestingHarvest
ControllerApplication">
   <settings>
    <common>

<applicationInstanceId>high2</applicationIns
tanceId>
     <jmx>
      <port>8112</port>
      <rmiPort>8212</rmiPort>
    </jmx>
   </common>
    <harvester>
     <harvesting>

<queuePriority>HIGHPRIORITY</queuePriority>
<heritrix>
       <guiPort>8192</guiPort> <!-- T:
jmxPort to be modified by test (was 8093)
--> <jmxPort>8193</jmxPort>

<jmxUsername>controlRole</jmxUsername>
        <jmxPassword>R_D</jmxPassword>
     </heritrix>

<serverDir>harvester_high_2</serverDir>
    </harvesting>
   </harvester>
  </settings>
 </applicationName>
```

IndexServerApplication and ViewerProxyApplication=

Both the `dk.netarkivet.archive.indexserver.IndexServerApplication` and `dk.netarkivet.viewerproxy.ViewerProxyApplication` should have the `settings.common.http.port` and the `settings.viewerproxy.baseDir'` changed to fit your system.

BitarchiveMonitorApplication

All the instances of `dk.netarkivet.archive.bitarchive.BitarchiveMonitorApplication` should be placed on the same machine as the `dk.netarkivet.common.webinterface.GUIApplication`.
These applications monitors the BitarchiveApplications at a given replica, though they do not have to be on the same physical location.
They should therefore have the `settings.common.useReplicaId` defined.

Manual installation of the NetarchiveSuite

Contents

If the deploy software is not adequate for the installation needed, this section will give some hints on how to distribute and install the NetarchiveSuite software on a number of machines.

In the examples below, we assume that `$deployInstallDir` is set to the directory in which the NetarchiveSuite code is to be installed.

We assume that all machines in the chosen scenario are unix/linux servers. The procedure below may not work on

other platforms. After having created the new settings to be used in the deployment of the software, zip together the NetarchiveSuite files including the new settings and copy the modified NetarchiveSuite.zip to all machines taking part in the deployment:

```
export USER=test
export MACHINES="machine1.domain1,
machine2.domain1, .. machine1.domain2,
machine2.domain2"
for MACHINE in $MACHINES; do
  scp NetarchiveSuite.zip
$USER@$MACHINE:$deployInstallDir
  ssh $USER@$MACHINE "cd $deployInstallDir
&& unzip NetarchiveSuite.zip"
done
```

NetarchiveSuite settings

The NetarchiveSuite settings can be set for applications in three different ways:

- use default setting
- in a setting file
- on command line

Using NetarchiveSuite default settings

If no settings are set, the default setting is used. Please refer to the [Configuration Manual 3.16#DefaultSettings] for more information on these.

Setting NetarchiveSuite settings on the command line

To set the value of a setting on the command line, add "-Dkey=value" to your java command line, for instance:

```
java -Dsettings.common.http.port=8076
dk.netarkivet.common.webinterface.GUIApplica
tion
```

will override the setting for the http port to be 8076.

Setting NetarchiveSuite settings with settings files

To set the values using a configuration file, save the settings in an XML file as described above. By default, NetarchiveSuite will look for the settings file in `conf/settings.xml`, that is: the file `settings.xml` under the directory `conf` from the current working directory. You can override this, by specifying `-Ddk.netarkivet.settings.file=path/to/settings.file.xml` on the commandline, for instance:

```
java
-Ddk.netarkivet.settings.file=/home/netarchive/guisettings.xml
dk.netarkivet.common.webinterface.GUIApplication
```

will read settings from the file `/home/netarchive/guisettings.xml` .

You can even specify multiple configuration files, if you wish. You do this by separating the paths with ':' on unix/linux/MacOS or ';' on windows. For instance:

```
java
-Ddk.netarkivet.settings.file=guisettings.xml:basicsettings.xml
dk.netarkivet.common.webinterface.GUIApplication
```

will read settings from both `guisettings.xml` and `basicsettings.xml` in the current directory.

The order of resolving NetarchiveSuite settings

If a setting is set on both command line and in settings files, or if it is set in multiple settings files, the setting is resolved as follows:

- If the setting is set with system properties (i.e. set on the command line), use these
- Else if the setting is specified in configuration files, use the '"first"' specified value
- Else use default value

As an example, consider the resulting value for http-port (knowing that the default value is empty) when using the following two configuration files:

settings1.xml

```
<settings>
  <common>
    <http>
      <port>8076</port>
    </http>
  </common>
</settings>
```

settings2.xml

```
<settings>
  <common>
    <http>
      <port>8077</port>
    </http>
  </common>
</settings>
```

The following command will use the value empty string as http-port:

```
java
dk.netarkivet.common.webinterface.GUIApplica
tion
```

The following command will use the value 8078 as http-port:

```
java
-Ddk.netarkivet.settings.file=settings1.xml:
settings2.xml
-Dsettings.common.http.port=8078
dk.netarkivet.common.webinterface.GUIApplica
tion
```

The following command will use the value 8076 as http-port:

```
java
-Ddk.netarkivet.settings.file=settings1.xml:
settings2.xml
dk.netarkivet.common.webinterface.GUIApplica
tion
```

The following command will use the value 8077 as http-port:

```
java
-Ddk.netarkivet.settings.file=settings2.xml:
settings1.xml
dk.netarkivet.common.webinterface.GUIApplica
tion
```

Standard commandline settings

The CLASSPATH

The CLASSPATH needed to start and run the java applications in NetarchiveSuite consists of 5 jarfiles, `dk.netarkivet.harvester.jar`, `dk.netarkivet.archive.jar`, `dk.netarkivet.viewerproxy.jar`, `dk.netarkivet.wayback.jar`, and `dk.netarkivet.monitor.jar`. The dk.netarkivet.common.jar and all our 3rd party dependencies need not be added explicitly to the CLASSPATH, as they are referenced indirectly in the jar-files.

```
export
deployInstallDir=/path/to/netarchiveSuite
export
CLASSPATH=$CLASSPATH:$deployInstallDir/lib/d
k.netarkivet.harvester.jar
export
CLASSPATH=$CLASSPATH:$deployInstallDir/lib/d
k.netarkivet.archive.jar
export
CLASSPATH=$CLASSPATH:$deployInstallDir/lib/d
k.netarkivet.viewerproxy.jar
export
CLASSPATH=$CLASSPATH:$deployInstallDir/lib/d
k.netarkivet.wayback.jar
export
CLASSPATH=$CLASSPATH:$deployInstallDir/lib/d
k.netarkivet.monitor.jar
```

<<Anchor(CommandLineLogging)>>

Logging

We use the apache.commons.logging.framework, so we need to point to the wanted logger-class (eg. org.apache.commons.logging.impl.Jdk14Logger) as well as to the logging configuration file. You may want to use different logging properties for different applications, especially when more than one application logs to the same logging directory. E.g. you want the change line `java.util.logging.FileHandler.pattern=./log/APPID%u.log` in the `conf/log.prop` file to something different.

```
export
LOG_SETTINGS="-Dorg.apache.commons.logging.Log=org.apache.commons.logging.impl.Jdk14Logger \
 -Djava.util.logging.config.file=$deployInstallDir/conf/log.prop"
```

Note that if you use the MonitorSiteSection, your logging properties file must contain the handler `dk.netarkivet.monitor.logging.CachingLogHandler`

```
handlers=java.util.logging.FileHandler,java.util.logging.ConsoleHandler, \
dk.netarkivet.monitor.logging.CachingLogHandler
```

JMX settings

Each application instance has its own JMX- and RMI port. For example the JMX port could be 8100 and the associated RMI port 8200, as in the example below, for the first application instance on the machine , then 8101/8201 for the second application instance, and so on. JMX also uses a password-file, which is the same throughout the installation ($deployInstallDir/conf/jmxremote.password)

```
export
JMX_SETTINGS="-Dsettings.common.jmx.port=8100 -Dsettings.common.jmx.rmiPort=8200"
```

Note: For the StatusSiteSection to work, your logging must be configured to use java.util.logging with the `dk.netarkivet.monitor.logging.CachingLogHandler` enabled, see Command Line Logging section (This is done automatically, if the NetarchiveSuite deploy software is used to configure and install your NetarchiveSuite installation).

Select the appropriate settings.file for the application

The conf/settings.xml (the new one configured to your environment) is probably OK for most applications. But you may need to use special purpose settings-files for some applications, e.g. BitarchiveApplications (since you can't

allocate more than one `baseFileDir` on the commandline). The settings file used in an application can be specified by:

```
export
SETTING=-Ddk.netarkivet.settings.file=$deplo
yInstallDir/conf/settings.xml
```

JVM options

We need to set the maximum Java heap size to 1.5 Gbytes. You may use this to change that or add other JVM options.

```
export JAVA_OPTS=-Xmx1536m
```

Admin machine

On the admin machine, we have to start the following 5 applications:

- 1 GUIApplication.
- 1 HarvestJobManagerApplication (handles the scheduling of jobs)
- 2 instances of BitarchiveMonitorApplication (Controlling the access to a single bitarchive replica), one for each bitarchive replicas (e.g. EAST and WEST).
- 1 ARCRepositoryApplication (this application handles access to the bitarchive replicas).

Starting the GUIApplication

Before, we can start the GUIApplication, the external database needs to started in advance (The deploy software does for you if the external database is a derby database).

We also need to prepare the JSP-pages. You can unzip the war-files in the webpages directory as below:

```
cd $deployInstallDir/webpages
rm -rf BitPreservation
unzip -o BitPreservation.war -d
BitPreservation
rm -rf HarvestDefinition
unzip -o HarvestDefinition.war -d
HarvestDefinition
rm -rf History
unzip -o History.war -d History
rm -rf QA
unzip -o QA.war -d QA
rm -rf Status
unzip -o Status.war -d Status
```

Or you can update your settings.xml file to refer to the war-files instead of the unpacked directories, for instance

```
<common>
        ...
        <webinterface>
            ...
            <siteSection>
                <!-- A subclass of
SiteSection that defines this part of the
                     web interface. -->

<class>dk.netarkivet.harvester.webinterface.
DefinitionsSiteSection</class>
                <!-- The directory or
war-file containing the web application
                     for this site
section.-->

<webapplication>webpages/HarvestDefinition.w
ar</webapplication>
            </siteSection>
            ...
        </webinterface>
        ...
    </common>
```

and similar for other sitesections.

Now we are ready to start the application:

```
cd $deployInstallDir
export
APP=dk.netarkivet.common.webinterface.GUIApp
lication
java $JAVA_OPTS $SETTING $LOG_SETTINGS
$JMX_SETTINGS $APP
```

Starting the BitarchiveMonitorApplication instances

In the general set-up with two distributed bitarchive replicas, we have a BitarchiveMonitorApplication associated with each replica. Here the replicas are `ReplicaOne` (with replicaId `ONE`) and `ReplicaTwo` (with replicaId `TWO`).

To distinguish the two instances from each other, we use the '"settings.common.applicationInstanceId"' setting, which is used as a identifier (here we use BMONE and BMTWO) as the two identifiers.

Start the monitor for bitarchive at `ReplicaOne` using `BMONE` as identifier thus:

```
cd $deployInstallDir
export
APP_OPTIONS="-Dsettings.common.archive.bitar
chive.useReplicaId=ONE  \
 -Dsettings.common.applicationInstanceId=BMO
NE"
export
APP=dk.netarkivet.archive.bitarchive.Bitarch
iveMonitorApplication
java $JAVA_OPTS $SETTING $LOG_SETTINGS
$JMX_SETTINGS $APP_OPTIONS $APP
```

Start the monitor for the bitarchive at `ReplicaTwo` using `BMTWO` as identifier thus:

```
cd $deployInstallDir
export
APP_OPTIONS="-Dsettings.common.archive.bitar
chive.useReplicaId=TWO \
 -Dsettings.common.applicationInstanceId=BMT
WO"
export
APP=dk.netarkivet.archive.bitarchive.Bitarch
iveMonitorApplication
java $JAVA_OPTS $SETTING $LOG_SETTINGS
$JMX_SETTINGS $APP_OPTIONS $APP
```

- one ARCRepository (this application handles all access to the bitarchives).

```
cd $deployInstallDir
export
APP=dk.netarkivet.archive.arcrepository.ArcR
epositoryApplication
java $JAVA_OPTS $SETTING $LOG_SETTINGS
$JMX_SETTINGS $APP
```

Harvester machines

On each harvester machine, we have one or more HarvestControllerApplications. Settings related to the HarvestControllerApplication are

- setting.common.applicationInstanceId (to distinguish between HarvestControllerApplications running on same machine)
- settings.harvester.harvesting.queuePriority (to select which of two queues to accept jobs from: HIGHPRIORITY (jobs part of a selective harvest), or LOWPRIORITY (jobs part of a snapshotharvest)
- settings.harvester.harvesting.minSpaceLeft (how many bytes "must" be available in the serverdir to accept crawljobs). The default is 400000000 (~400 Mbytes).

In the following, a low-priority HarvestControllerApplication is started with application instance id=SEL

```
cd $deployInstallDir
 export
APP_OPTIONS="-Dsettings.harvester.harvesting
.queuePriority=LOWPRIORITY
-Dsettings.common.applicationInstanceId=SEL"
 export
APP=dk.netarkivet.harvester.harvesting.Harve
stControllerApplication
 java $JAVA_OPTS $SETTING $LOG_SETTINGS
$JMX_SETTINGS $APP_OPTIONS $APP
```

Bitarchive machines

For each Replica, you can have BitarchiveServer's installed on one or more machines. We suggest using just one BitarchiveServer for each machine, though it is possible to use more than one.

Each BitarchiveServer can have storage on several filesystems, so if archive-storage is spread over more than one filesystem, you need to modify the settings file like this

```
<settings>
  ..
  <archive>
    ...
    <bitarchive>
      ...

<baseFileDir>/home/fileSys1/</baseFileDir>

<baseFileDir>/home/fileSys2/</baseFileDir>
      ...
    </bitarchive>
  </archive>
  ..
</settings>
```

Starting a BitarchiveServer requires knowing what Replica it resides on, and the credentials required for correcting the data stored in the bitarchive, for `ReplicaOne` with id `ONE` this would be:

```
cd $deployInstallDir
 export
APP_OPTIONS="-Dsettings.archive.bitarchive.u
seReplicaId=ONE \

-Dsettings.archive.bitarchive.thisCredential
s=CREDENTIALS"
 export
APP=dk.netarkivet.archive.bitarchive.Bitarch
iveApplication
 java $JAVA_OPTS $SETTING $LOG_SETTINGS
$JMX_SETTINGS $APP_OPTIONS $APP
```

Access servers

On the access-servers, we deploy any number of **ViewerProxyApplication** instances, and maybe one **IndexServer Application** (only one in all) used to generate indices needed by the harvesters and the ViewerProxyApplication instances.

```
cd $deployInstallDir
 export
APP=dk.netarkivet.archive.indexserver.IndexS
erverApplication
java $JAVA_OPTS $SETTING $LOG_SETTINGS
$JMX_SETTINGS $APP
```

Each ViewerproxyApplication instance uses a application instance id(settings.common.applicationInstanceId), and its own distinct base directory (settings.viewerproxy.baseDir). They also belong to a Replica(settings.archive.bitarchive.useReplicaId). In the start sample below, the instance uses application instance id "first" and 'viewerproxy_first' as base directory, and belongs to `ReplicaOne` with id `ONE`:

```
cd $deployInstallDir
 export
APP_OPTIONS="-Dsettings.common.applicationIn
stanceId=first \

-Dsettings.viewerproxy.baseDir=viewerproxy_f
irst \

-Dsettings.archive.bitarchive.useReplicaId=O
NE"
 export
APP=dk.netarkivet.viewerproxy.ViewerProxyApp
lication
 java $JAVA_OPTS $SETTING $LOG_SETTINGS
$JMX_SETTINGS $APP_OPTIONS $APP
```

About the NetarchiveSuite support for wayback, see Additional Tools Manual

Starting and stopping the NetarchiveSuite

Contents

This section describes how to start and stop the NetarchiveSuite.

Note that the deploy module can make scripts for this purpose. Please refer to the [Configuration Manual 3.16] for more information on how to use the deploy module.

You need to start and stop the NetarchiveSuite applications in the correct order. The most critical part is that the BitarchiveMonitor must not start before the BitarchiveServers, as it might then initiate batch jobs before all BitarchiveServers are up and running and thus not receive the batch message. The following is a suggested order of

startup:

NetarchiveSuite application startup order

1. Start the databases used by NetarchiveSuite
2. The BitarchiveApplication (one or more) on all bitarchive servers is started:

```
dk.netarkivet.archive.bitarchive.BitarchiveApplication
```

3. The applications on the admin-machine are started:

```
- dk.netarkivet.common.webinterface.GUIApplication
 - dk.netarkivet.archive.arcrepository.ArcRepositoryApplication
 - dk.netarkivet.harvester.scheduler.HarvestJobManagerApplication
 - dk.netarkivet.archive.bitarchive.BitarchiveMonitorApplication for
Replica One
 - dk.netarkivet.archive.bitarchive.BitarchiveMonitorApplication for
Replica Two
```

4. The applications on the harvester machines are started: Start each HarvesterControllerApplication instance deployed on this machine
5. The applications on the access-servers are started by first starting the IndexServer and then one or more ViewerproxyApplication instances.

NetarchiveSuite application stopping order

After locating the process-id of any given process, the actually killing of the process is done on unix-machines with the kill command:

```
kill $PID
```

The killing itself is done in the following order:

1. The applications on the admin-machine are killed:

```
- dk.netarkivet.harvester.scheduler.HarvestJobManagerApplications.
 - dk.netarkivet.common.webinterface.GUIApplication
 - dk.netarkivet.archive.arcrepository.ArcRepositoryApplication
 - dk.netarkivet.archive.bitarchive.BitarchiveMonitorApplication
```

   a. Now you can shutdown the databases, if you like.
2. The BitarchiveApplication on all bitarchive servers are shut down:

```
dk.netarkivet.archive.bitarchive.BitarchiveApplication
```

3. The applications on the harvester machines are shut down in arbitrary order:

4. The applications on the access-servers are shutdown by first killing the IndexServer and then the ViewerproxyApplication instances.

Remember to empty the JMS queues after shutting down the NetarchiveSuite if you are upgrading the system or want to reset the system. If any outstanding JMS messages are around next time the NetarchiveSuite is started, they may cause deserialization errors if the message definitions have changed. To empty the JMS queue, you need to know what JMS environmentName your NetarchiveSuite instance have been using. The details of this are explained in Appendix A.

In the Danish installation, we empty the queues each time the system is restarted, so the effect of leaving messages in the queues over a restart even when not upgrading has not been tested in practice.

Monitoring a running instance of NetarchiveSuite

Contents

The Status component of the NetarchiveSuite GUI that uses JMX to communicate with all running applications makes it easy monitor a running NetarchiveSuite installation.This component gives you access to the 100 latest logmessages from the applications, and a proper errormessage, if any application is off-line.

If you want to get more information about the current status of a particular application, you can use the program "jconsole". You need to know on which machine the the application is running (MACHINE), the JMX port (JMX_PORT) and RMI port (RMI_PORT) assigned to the application instance, and password for the `monitorRole` (set in **jmx.password** file and settings `settings.monitor.jmxUsername` and `settings.monitor.jmxPassword`, see Configure Monitoring). Then you just write jconsole, and click on the 'advanced' tab, enter the URL.

```
service:jmx:rmi://MACHINE:RMI_PORT/jndi/rmi://MACHINE:JMX_PORT/jmxrmi
```

When asked for username, enter **monitorRole** and the password set for the application. Log entries can now be examined for the given application instance by selecting MBeans, and unfolding `dk.netarkivet.common.logging`. Furthermore you can examine the system resources allocated to any given application.

Appendix A - Necessary external software

Contents

The NetarchiveSuite is developed and tested with Sun Java SE (Standard Edition) JDK version 1.6.0_21. In any case a Java 1.6+ JDK will be necessary to compile and run the NetarchiveSuite, and we recommend that all applications use the same JDK.

The following external software is required for running the applications

- JMS
- FTP This is only required, if FTPRemoteFile is the chosen RemoteFile Plugin.
- SSH (Installed as default under Unix/Linux, and WinSSHD by http://www.bitvise.com does the trick on Windows).
- Unzip. "unzip.exe" on Windows, and "unzip" on Linux.

Windows specific

Some application requires the Unix command `sort`, but they should be able to run under Windows if Cygwin is installed. This should only affect the ViewerProxy and the IndexServer.

Installing and configuring a JMS broker

The software have been tested with the free JMS broker from Sun "Open Message Queue 4.4", and the commercial JMSBroker "Sun MQ 3.6 Enterprise Edition".

Obtaining a JMS broker

Sun's Open Message Queue can be obtained from the following site: https://mq.dev.java.net/downloads.html

Go to the section named "Legacy Versions", and click on the Linux link in the subsection "Open MQ 4.4 Binary Downloads". This will give you a jar-file named "mq4_4-binary-Linux_X86-XXXXXXXX.jar". (We have no reason to suppose that NetarchiveSuite will have problems with newer versions but these are still untested with our software.)

Note: We only support installation on the Linux platform here. However, you may want to install your JMS broker on a different platform. Binary versions are available at the site for: Solaris Sparc, Solaris x86, Linux (x86), Windows (x86). If you want to build a binary for another platform, the source can be downloaded from the download-page.

Installing the JMS broker

Select Linux server where you want to install JMS broker, and select an installation directory. Then log on the linux server as root, and do the following:

```
export
INSTALLATION_DIR=/path/to/installationdir
   cd $INSTALLATION_DIR
   unzip mq4_1-binary-Linux_X86-XXXXXXXX.jar
   chmod +x ./mq/bin/imqbrokerd
   ./mq/bin/imqbrokerd -reset store -tty
(tests that the broker can start - CTLR-C to
stop)
```

Check that it starts, and that the last message is

```
Broker <localhost>:7676 ready
```

We are now ready to configure the JMS broker.

Configuring the JMS broker

- Edit the file `$INSTALLATION_DIR/mq/etc/imqenv.conf` to set IMQ_DEFAULT_JAVAHOME to a JDK1.5.0.
- Changing the number of the listening port number 7676 is done by editing the line
  . `imq.portmapper.port=7676`
  . in the file
  . `$INSTALLATION_DIR/mq/lib/props/broker/default.properties`
- Set max listeners any given queue to 20. You need to make sure, that the following line
  . `imq.autocreate.queue.maxNumActiveConsumers=20`
  . is present and not commented out in the file
  . `$INSTALLATION_DIR/mq/var/instances/imqbroker/props/config.properties`
  . (increase the number 20 if you have more than that number of applications of the same kind on the same bitarchive replica, for instance more than 20 bitarchiveapplications)
- Set max producers to 100. You add the following line
  . `imq.autocreate.destination.maxNumProducers=100`
  . in the file
  . {{$INSTALLATION_DIR/mq/var/instances/imqbroker/props/config.properties }}
  . If you get an error like this:
  . {{Producer can not be added to destination PROD_COMMON_MONITOR Queue, limit of 100 producers would be exceeded }}
  . in the JMS broker log, you need to increase this value.

Starting and stopping JMS

The broker is started directly in this way:

```
$INSTALLATION_DIR/mq/bin/imqbrokerd -reset
store -tty &
```

The sysadmin would maybe like to start the broker on machine startup by inserting the statement above into the /etc/rc.d/rc.local

The broker is stopped in this way:

```
logon on machine as root
find processid for the broker (ps auxw |
grep imqbrokerd)
kill -9 $IMQ_PROCESSID
```

Alternatively press Crtl-c, if the terminal where the broker was started, is still available

You can test that JMS broker is alive by telnetting to its port, where it will give some technical information in reply:

```
[user@udvikling kb-dev-adm-001.kb.dk]$
telnet localhost 7676
Trying 127.0.0.1...
Connected to localhost.localdomain
(127.0.0.1).
Escape character is '^]'.
101 imqbroker 4.1
portmapper tcp PORTMAPPER 7676
[sessionid=17296836783035176 96]
cluster_discovery tcp CLUSTER_DISCOVERY
46760
jmxrmi rmi JMX 0
[url=service:jmx:rmi://udvikling.kb.dk/stub/
rO0...Hg=]
admin tcp ADMIN 46763
jms tcp NORMAL 46762
cluster tcp CLUSTER 46764
.
Connection closed by foreign host.
```

To run JMS client applications, include the following jar files in the classpath :

```
$INSTALLATION_DIR/mq/lib/jms.jar
$INSTALLATION_DIR/mq/lib/imq.jar
```

Create a passfile named '.imq_passfile' (used when emptying JMS queues):

```
imq.imqcmd.password=REPLACE_WITH_PASSWORD
```

How to empty queues

log on as root to the server, where the JMS broker is installed. The following assumes that the JMS environmentName is PROD, and that JMS password file resides in ~root/.imq_passfile:

```
export JMS_ENV=PROD
export MQ_HOME=/usr/local
# imqcmd using -u admin -passfile
~/.imq_passfile
$MQ_HOME/bin/imqcmd list dst -t q -u admin
-passfile ~/.imq_passfile | grep
^${JMS_ENV}_ | cut -f1 -d\ |xargs -r -n 1
$MQ_HOME/bin/imqcmd destroy dst -t q -u
admin -passfile ~/.imq_passfile -f -n
$MQ_HOME/bin/imqcmd list dst -t t -u admin
-passfile ~/.imq_passfile | grep
^${JMS_ENV}_ | cut -f1 -d\ |xargs -r -n 1
$MQ_HOME/bin/imqcmd destroy dst -t t -u
admin -passfile
~/.imq_passfile -f -n"
```

How to allocate additional JMS broker memory

```
export MQ_HOME=/usr/local
$MQ_HOME/mq/bin/imqbrokerd -vmargs "-Xms256m
-Xmx512m" -reset store -tty &
#which adds min 256Mb and max 512MB heap
space
```

Installing and configuring FTP

If you decide to use FTPRemote for file transfer in the NetarchiveSuite, you need to install and start one or more FTP servers, before you begin the installation of the NetarchiveSuite. Any brand of FTP-servers will probably do, but we have good experience with Proftpd.

You can download Proftpd from http://www.proftpd.org/. We are using version 1.2.10, but any recent non-beta version will probably do.

The text below shows part of the proftpd.conf needed by NetarchiveSuite. Other parameters in proftpd.conf may be left with their default values.

```
# Port 21 is the standard FTP port.
Port                                21
# Umask 022 is a good standard umask to
prevent new dirs and files
# from being group and world writable.
Umask                               022
# To prevent DoS attacks, set the maximum
number of child processes
# to 30.  If you need to allow more than 30
concurrent connections
# at once, simply increase this value.  Note
that this ONLY works
# in standalone mode, in inetd mode you
should use an inetd server
# that allows you to limit maximum number of
processes per service
# (such as xinetd).
MaxInstances                        250
# Set the user and group under which the
server will run.
User                                nobody
#Group                              nogroup
Group                               nobody
# To cause every FTP user to be "jailed"
(chrooted) into their home
# directory, uncomment this line.
```

```
#DefaultRoot ~
# Normally, we want files to be
overwriteable.
## This is necessary to allow the append
operation
AllowOverwrite          on
AllowStoreRestart on
# Bar use of SITE CHMOD by default
<Limit SITE_CHMOD>
  DenyAll
</Limit>
# This enables or disables the PAM
```

```
authentication module.
# The default is 'on'.
#AuthPAM off
```

If you want to have the FTP-server use a specific directory for uploading files, e.g. ~/ftp, you can use add the configuration

```
DefaultChdir ~/ftp
```

If the /ftp does not exist, the server will fallback to the "".

Starting and stopping a Proftpd server

Log as root on to the server, where Proftpd is installed, and the following command will start the FTP-server

```
/usr/local/sbin/proftpd
```

The following will kill the FTP-server.

```
killall -9 proftpd
```

Appendix B - Starting Netarchivesuite automatically

Contents

This manual contains the description about how to make the applications start automatically when the operating system is starting.

Currently, when a computer is rebooted, the applications has to be started manually. This describes how to make the operating systems start the applications during startup.

Linux

Note: This has been tested with Redhat Enterprise Linux 5, so it probably works on Fedora (Core) as well.

Log in as administrator. Create the following script in '/etc/init.d/' (the name of the script will be referred to as `netarkiv`):

```
#!/bin/bash
# chkconfig: 345 80 20
# description: netarkiv
[ -x
/home/USERNAME/ENV_NAME/conf/startall.sh ]
|| exit 0
case $1 in
    start)
        su - netarkiv -c
'ENV_NAME/conf/startall.sh'
        ;;
    stop)
        su - netarkiv -c
'ENV_NAME/conf/killall.sh'
        ;;
    *)
        echo "Usage: $0 { start | stop }"
        exit 1
esac
```

Where `USERNAME` is the name of the user for the installation, and `ENV_NAME` is the environment name for NetarchiveSuite (defined in the configuration file).

The following command has to be run for the `netarkiv` script to be run during start-up and shut-down of Linux:

```
chkconfig --add netarkiv
```

The script can also be run manually, by the commands:

```
service netarkiv stop
service netarkiv start
```

Windows

This is an example of how to make Windows 2003 Server automatically call a script during start-up. The restart script has to be run, since it might not have closed correctly last time (e.g. power-failure, spontaneous reboot, etc.). This cleans up before the applications are restarted.

Create the service.

- Install Microsoft Resource Kit Windows 2003 Server.
- Run the program `RkTools.exe`, and install with standard settings.
- Open a Command Prompt, and go to the directory where the Resource Kit has been installed (e.g. `C:\Program Files\Windows Resource Kits\Tools`).
- Install a service with the following command `Instsrv <ServiceName> <path to resource kit>\srvany.exe` (e.g. `Instsrv BitApp "C:\Program Files\Windows Resource Kits\Tools\srvany.exe"`).
- Open the registration database with `regedit`, and find the service through the path `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\<SercviceName>`.
- Make sure that the start value is 2 (starting automatically).
- Create a new 'Key' called `Parameters`.
- In this 'Key' create a new 'String Value' called `Application`, which contains the complete path to the bat-script (e.g. `c:\users\USERNAME\ENV_NAME\conf\restart.bat`).
- Also within the 'Key' create another 'String Value' called `AppDirectory`, which should contain a path to the directory where the bat-script is placed (e.g. `c:\users\USERNAME\ENV_NAME\conf`).

Now the application should automatically start during Windows startup.