3/12/2009

ARC configuration manual

Zsombor Nagy*
Ferenc Szalai†
Martin Savko‡

*zsombor@niif.hu
†szferi@wsbricks.com
‡martin.savko@gmail.com

Contents

1 Introduction

The arched daemon's configuration file contains general information about the daemon's environment (pid file, log file, log level, library paths, plugins), and it contains one (or more) message chains, which specifies how the different services are connected to the outside world. The daemon's listening interface, certificates used, and the HTTP path of the services are among the things specified in the chain. This part contains also the configuration specifics for each service.

A chain could contain multiple types of components: message chain components (called MCCs) which provide TCP, TLS, HTTP and SOAP support; the plexer, which connects different HTTP paths to different services; and the services themselves with all their specific configuration details.

The configuration file is in XML format, with XML schemas and namespaces given and documented.

For typical deployment scenarios there is an additional mechanism to create an XML configuration on-the-fly based on XML templates (called profiles) and more administrator-friendly INI configuration files.

Configuration related arched command line switches:

```
-c <xml config file>
-i <ini config file>
-d
-s <xml schema>
```

By default the daemon looks for the configuration files in the following order: first /etc/arc/server.ini, then if it is not found: /etc/arc/server.xml. However user can define which configuration should be used with the -c or -i command line argument. If accidentally both of them is defined then the -i has greater priority.

In order the check the generated XML configuration the -d options can be used. If used, arched will print the XML configuration to the standard output and then stop.

2 XML level configuration

- The first part of the config is the Server part, which contains the path of the pid file, and the path of the log file, and the verbosity of the logging.
- The second part of the config is the ModuleManager part, which contains the path to the directory with the ARC libraries.
- The third part is the Plugins part with the list of 'plugins' (e.g. message chain components, services) used in this config.
- The fourth part is the Chain part, which contains the description of the message chain, which connects the services to the network. (It is possible to have more than one chain in one configuration.) A chain could contain the message chain components, the plexer and the services.
  - The TCP message chain component does the listening on the network interface. This is the place to set the IP version to use, the IP and port to listen to.
  - The TLS message chain component implements the security layer. This is the place to set the server-side key and the certificate, and the directory of the trusted CA certificates.
  - The configuration of the HTTP and the SOAP message components most probably don't need to be changed.
  - The Plexer component has a list of all available services, and has a HTTP path for each of them. The path is a regular expression, the incoming HTTP requests will be sent to the service which has a matching path in the Plexer config.
  - Then come the specific configurations of all the services which will be running in this arched.

2.1 The service only gets its own config

It is important to know that the services will not be able to get the whole configuration, they will only get their part, so everything a service should know have to be specified in its configuration part, e.g. a service would not know the IP or the port of the server, the HTTP path to itself or the path of the key and certificate file unless these are specified in the service configuration part too.

Because some services would establish some network connection while running (e.g. to send SOAP messages to other services), and these connections would most probably be secure HTTPS connection, these services may have a part where a client-side key and certificate file have to be specified which can be used to establish connection.

A service most probably would want to advertise its URL by registering itself to an information indexing service (called ISIS), in order to do this, the full URL of the service should be put into the configuration part of the service. This means that if the daemon's interface (port or hostname) would be changed, all these URLs should be modified too.

2.2 Summary of where to change things

- pidfile, logfile, loglevel: the Server part
- path to the ARC libraries/modules/plugins: the ModuleManager part
- listening interface (port, IP, IP version): the TCP Component in the Chain part; and maybe the URLs of the services should be changed also in the configuration of the services.
- path to the server-side key and certificate files and CA dir: the TLS Component in the Chain part
- path to the client-side key and certificate files and CA dir: in the configuration of the individual services in the Chain part
- the HTTP path of the services: the Plexer in the Chain part, the paths are specified in regular expressions; and maybe the URL of the service should be changed also in the configuration of the service itself.
- the URL of the information indexing service (ISIS) where a service should advertise itself: in the config of each services in the Chain part

There are documentations and XML schemas for the possible configuration options of the message chain components (MCCs) and for all the services (TODO: link to some docs and stuff)

3 INI style configuration

Because the parameters which a system administrator would want to change while configuring arched are spread across the whole XML configuration file and sometimes even repeated several times, and because there are lots of things in the XML configuration file which would be always the same for a given deployment-scenario, it is possible to create different configuration profiles for different scenarios, and then configure these profiles with configuration files which are in a more user-friendly INI format containing key-value pairs grouped in sections. Then the XML config is generated from the given profile and the INI config. (A profile is basically a full XML configuration, but has some extra details about which part of the XML should be overwritten with which INI configuration parameter.)

For example there is a profile for a secure Echo service which contains a working set of message chain components and the Echo service with reasonable defaults for the paths of libraries, logfile, pidfile, certificates, the port to listen, the HTTP path of the service, etc. Then the system administrator can create an INI config which only contains the things which need to be non-default, e.g. if the key and certificate files are in a non-standard directory, the correct paths could be specified in the INI config, and these paths then would be put into the profile everywhere where needed (even multiple times).

Or if there is a profile with several services, all of them registering to an information index service (ISIS), which means that each service has the URL of the ISIS service in its config, this URL could be changed by

adding one line to the INI config with the new URL, which makes this URL inserted into the config of each services.

3.1 Issues

Using the INI config is much easier for the system administrator than figuring out where to modify the XML config in order to change some paths or URLs. However using the INI config is much less flexible then using the XML config. Everything depends on the profiles. If the system administrator would need a specific set of services, and there is no profile for that, then the INI config can't be used. Or if there is profile but a given configuration parameter is not given in the profile, then it cannot be set in the INI config.

Another issue is that for each individual profile a separate documentation would be needed, because a profile can use arbitrary INI tags. e.g. the INI tag for setting the log level could be loglevel in one profile and it could be debug in another, or something completely different in the third one. The way to set the path to the directory of the trusted CAs' certificates could be cacert=path or cacertificatesdir=path depending which profile is in use. Currently there are example INI configs for most of the existing profiles, which should give hints about the possible INI tags.

3.2 INI section

The key-value pairs in an INI file are grouped into sections, like this:

```
key1=value1
key2=value2

[section1]
key1=value11
key3=value3

[section2]
key5=value5
```

The key-value pairs before the first section name are in the common section. The sections are used to selectively override arguments. e.g. the profile could say that it needs the value for key1, and then it could specify that it needs the value from the common section only, or from the —section1 section only, or it could say that it needs the value from either of these two sections and the value from section1 overrides the value from the common section, or vice versa.

For a more specific example, if there is a client-side key file given in the common section, then the profile could specify for each service, that it should use this key file unless there is a different key file specified in the section of the given service. This means that if there is an INI section for the service, and it contains a path to the key file, then that path will be used, but if it does not contain a path, then the common key path will be used from the common section.

3.3 Profiles

The profile is a full XML service configuration with some additional XML attributes about which part of the XML should be overwritten by which INI parameter. Let's look at this part of a profile:

```
<infosys:URL inisections='register ahash' initag=\verb!index_server1!/>
```

The initag attribute tells the key of the parameter, the inisections attribute specifies a list of sections with the last one having the highest priority. This XML tag here is empty, which means that it's default value is an empty string.

- If the INI contains a key `index_server1` somewhere in the 'register' or 'ahash' sections, then the value from that key would be put into this XML node.

- If there is an `index_server1` key in both the 'register' and the 'ahash' section, then the value from the 'ahash' section would be used, it has the higher priority.
- If there is no `index_server1` key in none of these two sections in the INI config, then this 'infosys:URL' tag will still be in the generated XML config, but with an empty content.

This part has a non-empty default:

```
<cfg:File inisections='common' initag='logfile'>/var/log/arched.log</cfg:File>
```

This means if that would be a 'logfile' key in the 'common' section of the INI configuration file, then this default value will be overwritten by the value of that key from the INI file. If there is no key called 'logfile' in the 'common' section of the INI file, then the generated XML config will contain the given default value.

3.4 Limitations

Currently overwriting XML attributes are not supported. e.g. the id attribute of:

```
<next id='tls'/>
```

can not be overwritten by the INI config.

Currently there is no support for XML tags with variable number of occurrences. This means that if there would be a part in the XML config like this:

```
<ISISURL>the URL of an ISIS service</ISISURL>
<ISISURL>the URL of another ISIS service</ISISURL>
<ISISURL>the URL of a third ISIS service</ISISURL>
```

It is not possible to control the number of ISISURL tags from the INI config. If there are three such lines in the profile, then there will be three lines in the generated XML config, no matter how many URLs are specified in the INI config. A workaround could be using empty defaults, like this:

```
<lib:ISISURL inisections='register librarian' initag=\verb!index_server1!/>
<lib:ISISURL inisections='register librarian' initag='index_server2'/>
<lib:ISISURL inisections='register librarian' initag='index_server3'/>
```

Then the INI config could be like this:

```
[register]
index_server1=the URL of an ISIS service
index_server2=the URL of another ISIS service
```

Which means that the first and second ISISURL line would be filled with data from the INI, and the third one would be left empty. For this to work, it should be supported by the service itself, e.g. the service should throw out empty ISISURLs.

Here it was needed to use different initags for the three ISISURLs, because if same initags would be used like this:

```
<lib:ISISURL inisections='register librarian' initag='index_server'/>
<lib:ISISURL inisections='register librarian' initag='index_server'/>
<lib:ISISURL inisections='register librarian' initag='index_server'/>
```

Then this INI config would cause unexpected results:

```
[register]
index_server=the URL of an ISIS service
index_server=the URL of another ISIS service
```

This INI config would generate this XML config from that profile:

```
<lib:ISISURL>the URL of another ISIS service</lib:ISISURL>
<lib:ISISURL>the URL of another ISIS service</lib:ISISURL>
<lib:ISISURL>the URL of another ISIS service</lib:ISISURL>
```

Which is the same URL three times, because the last value of the `index_server` tag would be inserted into each places.

There are future plans for adding support for this kind of simple multiplicity, but there are more complex cases which are not trivial to handle. e.g. in order to register to multiple ISIS services, the path to the key and certificate files should be given for each ISIS service even though they most probably would be the same. This part would register to one ISIS service:

```
<infosys:Registrar>
<infosys:URL inisections='register shepherd' initag='index_server'/>
<infosys:KeyPath inisections='common shepherd' initag='client_key'/>
<infosys:CertificatePath inisections='common shepherd' initag='client_cert'/>
<infosys:CACertificatesDir inisections='common shepherd' initag='cacert'/>
</infosys:Registrar>
```

It is possible that we want to register only to one ISIS or two or maybe three or more, then for each of them this section should be repeated. Currently the workaround looks like this:

```
<infosys:Registrar>
<infosys:URL inisections='register shepherd' initag=\verb!index_server1!/>
<infosys:KeyPath inisections='common shepherd' initag='client_key'/>
<infosys:CertificatePath inisections='common shepherd' initag='client_cert'/>
<infosys:CACertificatesDir inisections='common shepherd' initag='cacert'/>
</infosys:Registrar>
<infosys:Registrar>
<infosys:URL inisections='register shepherd' initag='index_server2'/>
<infosys:KeyPath inisections='common shepherd' initag='client_key'/>
<infosys:CertificatePath inisections='common shepherd' initag='client_cert'/>
<infosys:CACertificatesDir inisections='common shepherd' initag='cacert'/>
</infosys:Registrar>
<infosys:Registrar>
<infosys:URL inisections='register shepherd' initag='index_server3'/>
<infosys:KeyPath inisections='common shepherd' initag='client_key'/>
<infosys:CertificatePath inisections='common shepherd' initag='client_cert'/>
<infosys:CACertificatesDir inisections='common shepherd' initag='cacert'/>
</infosys:Registrar>
```

Which allows to register to up to three ISIS services, and has empty defaults in order to be able to configure only one or two or zero ISIS services. However, the ISIS registration part should handle the empty defaults, and throw them away.

4 Examples

The most up to date Profiles can be found in the Nordugrid SVN:

`http://svn.nordugrid.org/trac/nordugrid/browser/arc1/branches/nox/src/hed/profiles`

Upon a successful installation (nordugrid-arc-nox-hed package when installing from binaries), all the example configuration files coming with the release can be found under

`<instllation_directory>/share/arc/`