Ceptor Configuration Server

The Configuration Server is the first server component to startup - it must be running for the other servers to be able to start.

The Configuration Server supplies configuration values to other servers, and it keeps track of which servers are running, and can route traffic between them.

The configuration is by default kept in a configuration file, called ceptor-configuration.xml

Connections to Clients

Clients open up a persistent TCP/IP socket connection to the configuration server - once established, this works in 2 ways, so configuration 'clients' can request information from the config server, and the config server can either push or pull information from ‘clients’.

A config client is a small communications layer that runs in all servers in the system. This communications layer is responsible for transporting data, sending messages between the config server and all other servers in the system.

Several different types of messages are sent over this interface; statistics messages, management messages, configuration messages, status messages and data messages.

Note: Even the configuration server is a configuration client. This special case is true when the clustering option is enabled. The exact same messages flow between config servers.

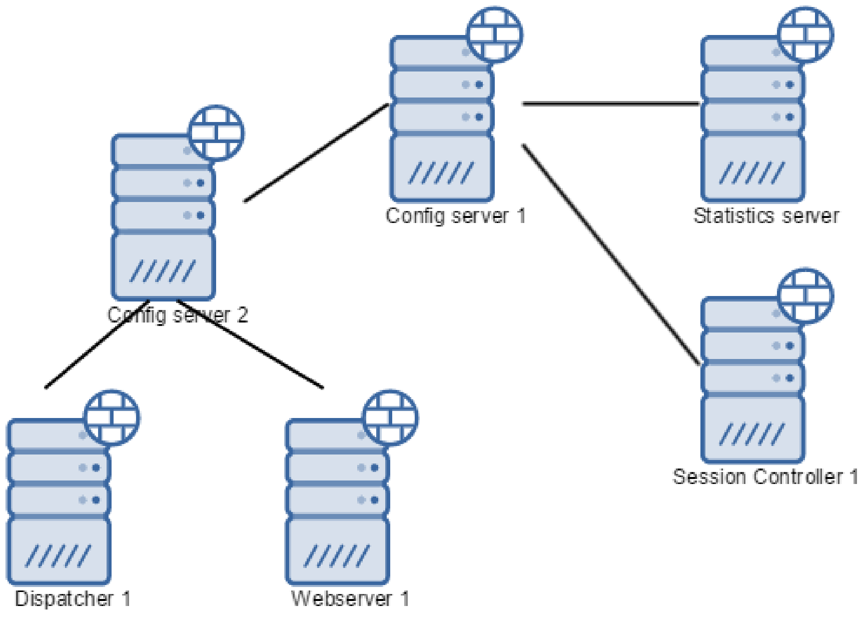
This peer-to-peer connectivity makes the config server a very powerful tool in monitoring and controlling all servers in the system.

The connection between the servers and the config server is designed to be fault tolerant. This means that if the servers for some reason get disconnected, they will immediately try to reconnect to an available config server, while each server will continue its work. When a connection is reestablished, data between the disconnected server and the config server is synchronized.

Clustering

Like most (all but statistics) of the servers in Ceptor PortalProtect, all services are redundant using multiple servers doing the same work. This also applies to the configuration server(s).

Two or more configuration servers can be set up as a cluster, one running as a master and all others as slaves.



As above example shows, dispatcher 1 and web server 1 servers, are connected to config server 2, configured as a slave, and PTS server 1 and statistics server and config server 2 is connected to config server 1, where config server 1 is configured as a master.

The differences between master and slave are:

- A master is not connected (as a client) to other config servers.
- A slave is (normally) connected to a master config server.

- A slave can only “see” clients connected to itself.
- A master can “see” clients connected to itself and those connected to slave config servers. This means that all messages send is “passed through” the config slaves if they are sent from the master.
- When a master server goes online, it’s the master holder of configuration data. This means that all other servers ‘view’, is compared to the master servers view and appropriate messages like “configuration was updated” messages will be sent if configurations are “out-of-sync”.
- When a slave server goes online and it connects to the master server, any configuration changes made on the slave server are overwritten with the newest available configuration on the master server. This will result in that the slave config servers will send “configuration was updated” to all its connected clients.
- The cluster master and slaves, is designed to be fault-tolerant. This means that either config server can go offline for a period of time, and the other (remaining) server(s) will take over. When the server then comes online, internal states are synchronized.
- When acting as a slave, the statistics data received from clients are places on a queue, and data is sent from slaves to master every 30 sec – just like the statistics flusher process. This is described in the section about the statistics server. When acting as a master, the received statistics data is places on a queue, waiting for the statistics server to request it.

The config server clustering option gives the Portal Protect a robust, redundant interface which will give Portal Protect servers high availability and a persistent configuration state at all times.

Config Server Security

When a config server is running, it is accepting requests. When a connection is accepted, the config server will examine the connecter and compare connecters IP/hostname to a “positive list”. If connecter machine is not on the positive list, the config server will close the connection.

This will ensure that no one can install config client software, and connect to a configuration server in the system, getting information from the configuration server.

Configuration

Configuration is stored in a file called ceptor-configuration.xml - the file is structed like this:

```
<system name="Ceptor PortalProtect" version="6.0.0" copyright="(c) 2001-2017, Asseco Denmark">
        <server name="configserver1" type="config" description="master config server" extends="configservers">
                <group name="cluster" description="cluster server configuration">
                        <property name="cluster.server" value="master" valid="master|slave" description="valid 
values=&quot;master&quot; and &quot;slave&quot;"/>
                        <property name="cluster.server.master" value="nios://localhost:21233?
validateservercert=false" description="host and port of master server"/>
                </group>
                <group name="general" description="general configuration">
                        <property name="list.log.directories" value="${portalprotect.home}/logs,PortalProtect 
Logs" description="List of log directories from which logfiles can be downloaded via the admin GUI"/>
                        <property name="server.listenurls" value="local://21233;nios://21233" description="URLs to 
listen for connection"/>
                        <property name="statistics.server" value="statisticsserver1" description="name of the 
statistics server"/>
                        <property name="statistics.server.listenurls" value="local://21112" description="Listen 
URL for connection from statistics server"/>
                </group>
        </server>
        <server name="configservers" type="abstract" description="config servers shared configuration" extends="">
                <group name="consul" description="Consul Service lookup">
                        <property name="consul.alloweduripattern" value="/v1/catalog/service/app*" description="
Used to restrict which services an agent can lookup"/>
                        <property name="consul.pollinterval" value="10" description="Cache lookups in consul for 
10 seconds"/>
                        <property name="consul.servers" value="" description="Consul servers to lookup services 
within - e.g. http://192.168.160.132:8500"/>
                </group>
                <group name="security" description="security configuration">
                        <property name="access.control.configuration" value="${portalprotect.home}/config
/portalprotect-security.xml" description="access control definitions"/>
                        <property name="access.controller" value="dk.itp.security.accesscontrol.
AccessControlListXMLImpl" description="config server access controller implementation"/>
                        <property name="remote.servers" value="localhost,192.168.255.255,10.255.255.255" 
description="allowed remote servers, host:port separated by ,;"/>
                </group>
        </server>
</system>
```

The name and location of **ceptor-configuration.xml** is controlled by configuration in the launcher; ceptor_**launch.xml** and can be changed.

```
<jvm name="cfg" vmargs="-Xmx1024M -Djava.awt.headless=true -Xnoclassgc -XX:+HeapDumpOnOutOfMemoryError -XX:
+ExitOnOutOfMemoryError" systemclasspath="">
        <config servers="loadbalance:nios://localhost:21233?validateservercert=false;nios://localhost:
21234?validateservercert=false" />
        <classloader name="cl">
                <service name="configserver1" launcherclass="dk.itp.managed.service.ConfigServerLauncher">
                        <property name="configuration"
                                value="${portalprotect.home}/config/ceptor-configuration.xml" />
```

Here, you can specify another location or filename instead of the default.

It has <system> tag which contains a number of <server> entries - each server entry has a name, a type, a description and an extends.

Name must be unique to the entire configuration, otherwise an error occurs.

Type indicated of what type the server is. A special type exists which is named **abstract**. This server is not a real server but a pseudo server. The pseudo servers, like logconfiguration and snmpconfiguration, are placeholders for generic properties. Servers can extend other servers or pseudo servers.

Multiple e**xtends** configurations are listed separated by commas. This list defines the configurations that the current server inherits its configuration from. Listing multiple other servers or pseudo servers in the *extends* property, will enable multiple inheritances.

If multiple inherited properties have non-unique keys, the effective property will be the one specified last - meaning the one on the last extend, or the one in specified in the server that inherits and thus overwrites a property existing in the server entry it inherited from.

Updating Configuration

Ceptor has two types of configuration - an active configuration which is the one currently in use, and the passive one that might or might not contains uncommitted changes.
Once changes are committed, they become the active configuration.

Active Configuration

When a Ceptor Configuration Server starts up the first time, it will look for a special filed called `ceptor-configuration.xml.<configservername>.active` - e.g. `ceptor-configuration.xml.configserver1.active` - this file is considered the "active" configuration, and is a binary non-editable copy of the configuration.

If you update the configuration file outside of Ceptor Console or the Administration API, or deploy new versions of Ceptor, you can delete the `.active` configuration file to force Ceptor to reload from the uncommited configuration, thereby activating it automatically.

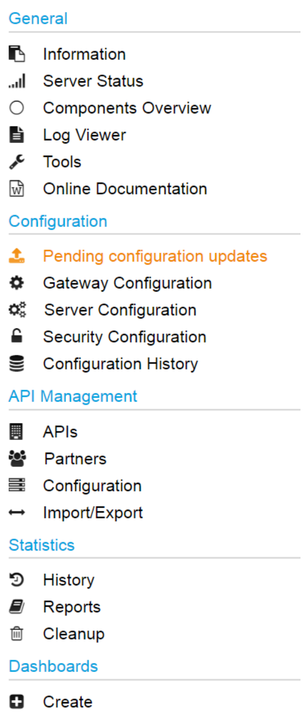
Any changes made to the configuration file either using the console or manually needs to be committed before they are used in practice - any reads and/or modifications done using the Ceptor Console act on non-active configuration.

Uncommitted Configuration

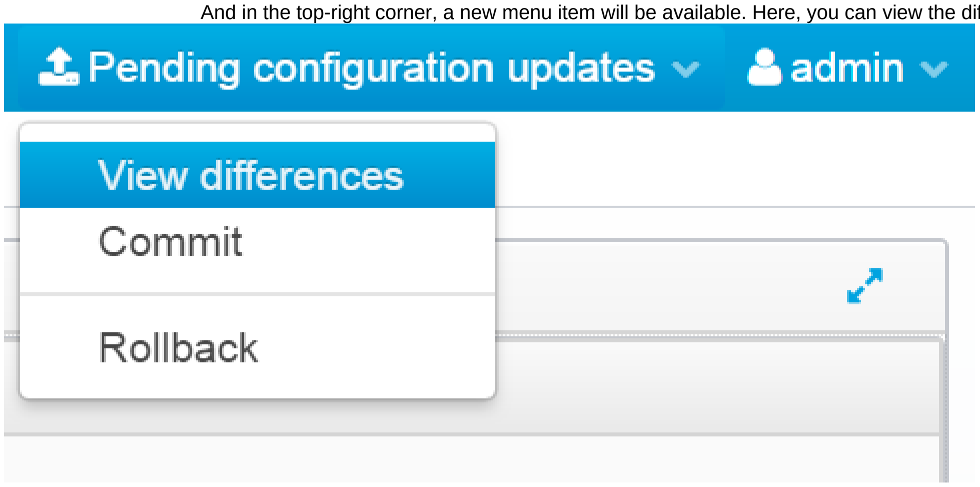
Any changes to ceptor-configuration.xml and any included files needs to be committed to the active configuration before it will take effect. All changes are propagated between Configuration Servers immediately when done, but they will not be activated unless specifically committed.

Using the Administration API (programmatically or the command-line version) or using Ceptor Console you can commit any changes made.

When there is a difference between the active configuration and the uncommitted configuration, You will see the following new menu item with the console:

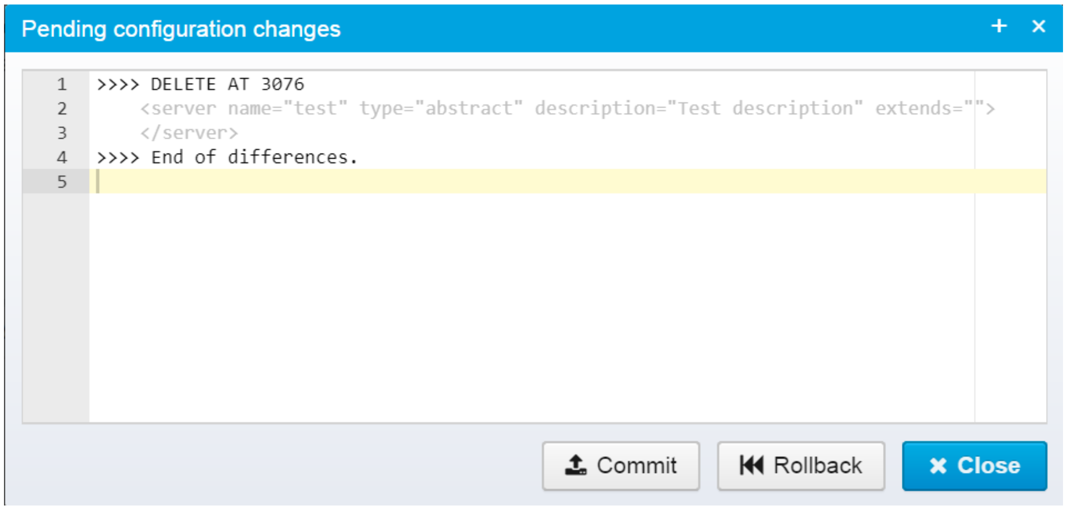


And in the top-right corner, a new menu item will be available. Here, you can view the differences, commit the changes or roll them back.



When committing changes, Ceptor will first send them out to all components giving them a chance to validate it - in case any validation warnings or errors occur, they will be shown to you and you need to confirm you want to override them, or you can choose to cancel the commit and fix any errors. Typical errors could include unresolvable DNS names, missing targets in a destination server definition etc.

Before committing, you can view the differences between the uncommitted and active versions.



```
>>>> DELETE AT 3076
    <server name="test" type="abstract" description="Test description" extends="">
    </server>
>>>> End of differences.
```

This view shows the differences between the two versions - in the example above. we deleted a server definition, and need to commit it to activate the changes.

Once changes are committed, they take effect immediately.

Remember to commit the configuration, otherwise the changes you make are not active.

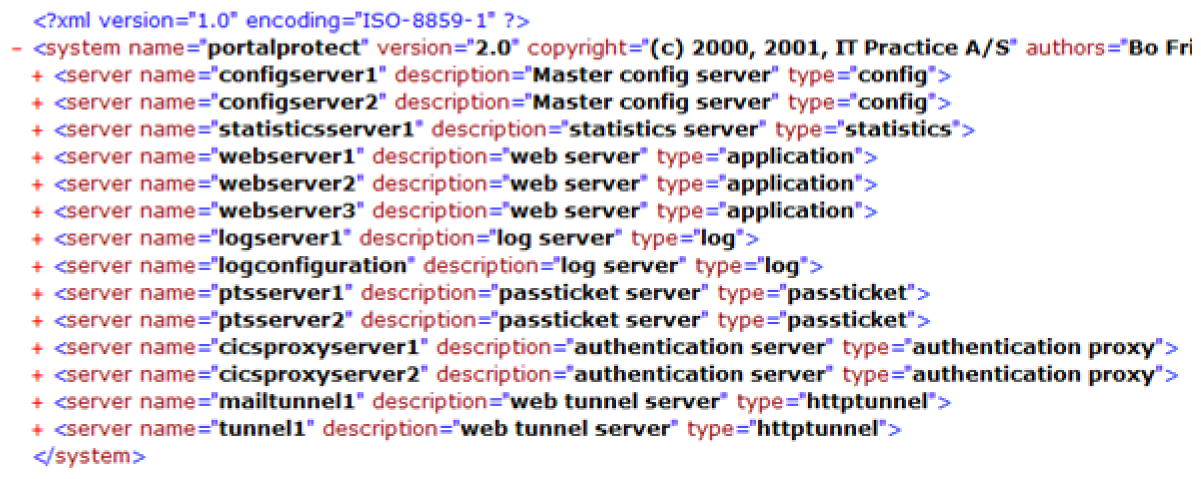
Configuration File Details

As previously mentioned the configuration of PortalProtect happens through the configuration file. If required this file can of course be generated through a build or programmatically. It is not required that all changes happen through the GUI functionality described in the previous chapters.

<PP_HOME>/config/ceptor-configuration.xml.

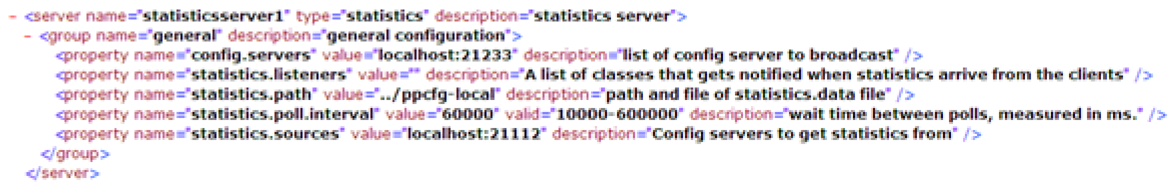
General Structure

The following example shows the content of the configuration file.



```
  <?xml version="1.0" encoding="ISO-8859-1" ?>
- <system name="portalprotect" version="2.0" copyright="(c) 2000, 2001, IT Practice A/S" authors="Bo Fri
  + <server name="configserver1" description="Master config server" type="config">
  + <server name="configserver2" description="Master config server" type="config">
  + <server name="statisticsserver1" description="statistics server" type="statistics">
  + <server name="webserver1" description="web server" type="application">
  + <server name="webserver2" description="web server" type="application">
  + <server name="webserver3" description="web server" type="application">
  + <server name="logserver1" description="log server" type="log">
  + <server name="logconfiguration" description="log server" type="log">
  + <server name="ptsserver1" description="passticket server" type="passticket">
  + <server name="ptsserver2" description="passticket server" type="passticket">
  + <server name="cicsproxyserver1" description="authentication server" type="authentication proxy">
  + <server name="cicsproxyserver2" description="authentication server" type="authentication proxy">
  + <server name="mailtunnel1" description="web tunnel server" type="httptunnel">
  + <server name="tunnel1" description="web tunnel server" type="httptunnel">
  </system>
```

Looking into a server configuration inside – this could be an example.



```
- <server name="statisticsserver1" type="statistics" description="statistics server">
  - <group name="general" description="general configuration">
      <property name="config.servers" value="localhost:21233" description="list of config server to broadcast" />
      <property name="statistics.listeners" value="" description="A list of classes that gets notified when statistics arrive from the clients" />
      <property name="statistics.path" value="../ppcfg-local" description="path and file of statistics.data file" />
      <property name="statistics.poll.interval" value="60000" valid="10000-600000" description="wait time between polls, measured in ms." />
      <property name="statistics.sources" value="localhost:21112" description="Config servers to get statistics from" />
    </group>
  </server>
```

All properties must be defined in a group. A group called “general” could for example be a good place for entries not fitting into other groupings.

Structuring the Configuration File

To structure the ceptor-configuration.xml file several options are available. First of all servers can use the “extends” property as described in section Configuration->New. This allows for common properties to be places in abstract server definitions inherited by servers when extending.

Including Configuration Snippets

It is possible to include part of the configuration file from other files or directories. This makes it possible to split up a large configuration file and also allow for external deliveries directly into own files without the need to merge the content into the ceptor-configuration.xml file itself.

The syntax for including individual files is as follows.

<include file=”***includefile.xml***”/>

When this include statement is present on the same XML level as the server entries, the include file can contain a single server configuration or the file can contain multiple server configurations. In the latter case the server entries must be put inside an <include> tag in the file as shown below. It is also possible to use a server include file inside an application cluster definition. This will result in all the servers in the include entry being part of that particular application cluster.

This is an example of an include file containing a single server entry.

```
<server name=”testapp-server1” type=”appserver”>
  <group name=”generic”>
    <property name=” ......
    ......
    ......
  </group>
</server>
```

This is an example of an include file containing multiple server entries.

```
<include>
  <server name=”testapp-server1” type=”appserver”>
    <group name=”generic”>
      <property name=” ......
      ......
      ......
    </group>
  </server>
  <server name=”testapp-server2” type=”appserver”>
    <group name=”generic”>
      <property name=” ......
      ......
      ......
    </group>
  </server>
  ......
  ......
</include>
```

When the include statement is inside a server statement on the same level as the group statements it can contain one or more group statements – in the same way as the server statements shown above.

It is also possible to include a complete directory containing include files. Such an include statement looks like this.

<include directory=”***project1_includes***”/>

This will result in all files with an *.xml* extension in the *project1_includes* folder to be read and parsed as individual include files.

It is not possible to configure include statements through the management console. This is means for administrators who generate the ceptor-configuration.xml file manually or build it thru some other means. The structure of the ceptor-configuration.xml will be kept if anything is changed through the management console. Includes will be kept and new property/group entries will be saved in appropriate include files.

It is also important to notice that all include information is replicated between clustered configuration servers. This means that any include file and/or directory on the master configuration server will be created and written on the slave server. It is vital that any include file and/or directory must be accessible on both servers (the same path but on different servers). If this is not the case the configuration servers will not start up and/or replicate properly!

Also any error while parsing include files will result in the same start-up errors as when a configuration error is present in the ceptor-configuration.xml file itself. The name of the include file will in these case be written in the log entry.

Using Macros in Configuration Values

You can use a number of macros within the configuration values – these macros are then replaced with the appropriate string before the value is used.

These values can be:

- `${systemproperty:xxxx}` where xxxx is the name of the java system property to replace the macro with.
- `${environment:xxxx}` where xxxx is the name of the environment variable to replace the macro with.
- `${file:xxxx}` where xxxx is the name of a file to load and replace the macro with the contents of - this is especially useful for files containing secrets injected in container environments.
- `${applicationcluster:target:xxxx}` where xxxx is the name of the applicationcluster – meant for use within targetservers or alternateserver.xxxx.targets properties only. Replaces the value there with the macro with a list of hostname:port,serverid
- `${applicationcluster:hostname:xxxx}` where xxxx is the name of the applicationcluster – can be used e.g. in the list of valid IP addresses, will end up containing the hostname of all servers within the cluster.
- `${applicationcluster:servername:xxxx}` where xxxx is the name of the applicationcluster – can be used to list the server names of all servers within the cluster.
- `${consul:xxxx}` where xxxx is a consul URL (see www.consul.io) to query for active services. Can only be used within targetservers or alternateserver.xxxx.targets properties for dispatcher.

With systemproperty, environment and file macros, you can add a default value that is used if the given system property, environment variable or file does not exist - this is done by adding :- (colon and dash) followed by the default value, like this example: `${environment:elasticsearch.enabled:-false}` where the macro is replaced with the value of the `elasticsearch.enabled` environment variable, but if that does not exist, `false` is used instead.

Application Clusters

It is also possible to group application server properties into what is called application clusters.

Within these cluster configurations, individual servers will be defined and for each server the team that knows the physical addresses and ports can provide this information.

These cluster groups can then be accessed through macro expansion in the configuration file (or where accessing configuration) with these three possibilities:

- `${applicationcluster:target:xxx}` -> hostname:port,name;hostname:port,name ....
- `${applicationcluster:hostname:xxx}` -> hostname;hostname ....
- `${applicationcluster:servername:xxx}` -> name;name ....

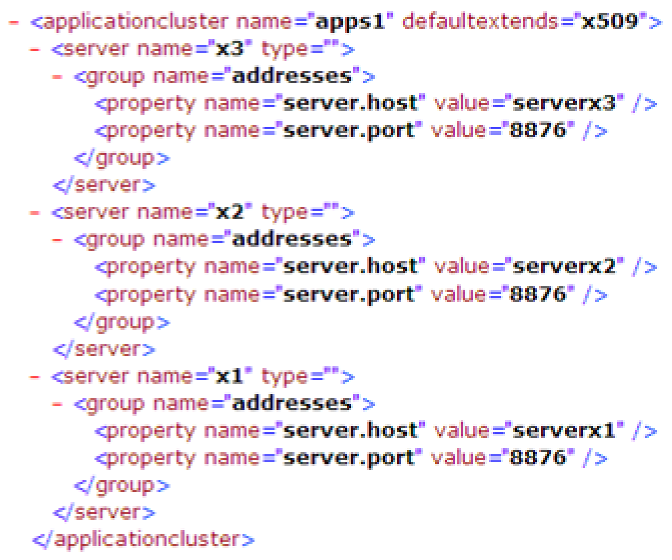
The configuration can be put into an include file and hence the information about DNS names and ports, does not need to be contained in the configuration file which can then be shared across the environments, allowing for all environment specific data regarding application servers to be in an application cluster configuration (see example below).

It is also possible to access the configuration of the servers in the cluster through the administration console or the admin client because the application cluster can be defined with a default extension from an abstract server configuration – so it is possible to create an application cluster – have it extend an abstract server definition, and then all servers within the cluster will automatically extend this configuration.

This allows for other projects to define their specific server parameters and deliver those to PortalProtect administrators who can then add these server configurations to an application cluster, where they will extend common properties used by other servers of the same type.

The example below shows how x509 servers in this configuration are defined within an application cluster configuration and how the configuration for the individual servers could look.

In this example the maintenance team for x509 servers need only deliver XML snippets with name and listen address – the rest of the portal protect configuration file is provided for example by the PortalProtect administration team. The x509 maintenance team does not even need to know which abstract definition their servers are extending.



```
- <applicationcluster name="apps1" defaultextends="x509">
  - <server name="x3" type="">
    - <group name="addresses">
        <property name="server.host" value="serverx3" />
        <property name="server.port" value="8876" />
      </group>
    </server>
  - <server name="x2" type="">
    - <group name="addresses">
        <property name="server.host" value="serverx2" />
        <property name="server.port" value="8876" />
      </group>
    </server>
  - <server name="x1" type="">
    - <group name="addresses">
        <property name="server.host" value="serverx1" />
        <property name="server.port" value="8876" />
      </group>
    </server>
  </applicationcluster>
```