Configuration in DIT (CiDIT)

> ⚠ **Work in progress**
>
> This site is in the process of being reviewed and updated.

Introduction

This effort aims to store the configuration for ApacheDS within the Directory Information Tree (DIT) to expose, manage and replicate it via LDAP. This includes the configuration for core services without networking as well as protocol service configurations when network services such as LDAP, and Kerberos are enabled.

Advantages

There are several advantages to storing the configuration within the DIT. These are listed below:

- Remote Access and Management Via LDAP: the configuration of the server can be accessed and altered remotely via LDAP
- Partial Replication of Configuration: some portions of the server's configuration which are not specific to an instance can be replicated within a cluster to allow cluster wide configuration management through a single point of access.
- Dynmaic Reconfiguration: the core as well as the server both support persistent search as well as internal notification mechanisms to inform subsystems of changes in the DIT which can be used to dynamically reconfigure subsystems when the configuration is changed via LDAP.
- Simple Command Line or GUI Based Configuration: since the configuration is stored in LDAP it can easily be altered using standard command line LDAP tools or GUI based LDAP tools; even LDIF files can be used to alter the server's configuration on startup.
- Configuration Changelog and Versioning: the changes to the DIT based configuration can be tracked by the server and versioned using snapshoting.
- Configuration Exports: the configuration can easily be exported to an LDIF file and used to setup new servers by simply importing the exported LDIF.
- Leveraging Existing Configuration Formats: the existing mechanisms for configuring the server in standalone and embedded mode still apply since they will be used to override or overwrite the configuration stored in the DIT.

Out of DIT Configuration

Presently configuration for the standalone server is stored in a server.xml file which uses Spring configuration to load configuration beans for both the core and the server. When embedding the configuration beans are assembled programmatically. In both cases the configuration bean hierarchy is used to construct the components in the server to initialize it into the solid state.

Disadvantages

- Dynamic re-configuration is difficult if at all possible
- Applications building on ApacheDS must cope with on disk configuration file
- Configuration changes require editing of XML file which is error prone
- XML file is growing larger as new features are introduced into the server

CiDIT Setup

The apacheds project would be modified to contain a prefabricated system partition module similar to the way the schema partition is managed today with the dynamic schema subsystem. This prefabricated system partition will contain the default configuration we ship with the standalone installers of apacheds within the server-xml project. A new configuration-plugin module will contain a maven plugin which loads the default server.xml file in the server-xml module using Spring. The plugin will use this configuration to create configuration entries in the system partition during the build. Of course a new schema would be needed to support these new configuration entries as well as DAO code to manage CRUD operations on configuration beans. The system partition will be packaged into a jar and un-jarred during the initialization sequence if it is not present in it's respective location.

This brings us to the initialization sequence. The server's core will have it's initialization sequence modified to start up the system partition after starting the schema subsystem. If no overriding configuration is provided either through a server.xml file or programmatically, the system partition will be used to load the configuration store in the DIT using a default PartitionConfiguration for the system partition. Once the configuration is loaded the system partition will be shutdown and restarted using the system PartitionConfiguration that was extracted if that PartitionConfiguration differs from the default. Now if an overriding startup configuration is supplied it will be used to overwrite the startup configuration in the system partition if it differs from the configuration provided. The rest of the initialization sequence will be the same as it was before using the configuration bean hierarchy to drive it.

To support dynamic reconfiguration reconfigurable services can either implement or expose access to a special listener interface that updates them when the configuration changes on the DIT. Each configurable component when instantiated can be checked to see if it supports this interface and if so, it or it's listener, can be registered with a configuration service in the server that dispatches DIT based configuration change events to reconfigurable component listeners. The component itself handles how it should respond to configuration change events. During the first round of implementation we may just have do nothing listeners without event dispatching. When using overwriting configurations loaded either from a server.xml file or provided programmatically then dynamic reconfiguration will be disabled by default: in fact modifications to the configuration will be disabled. Since the DIT configuration will be overwritten anyway using the provided configuration there is no point to support DIT based reconfiguration. In this case we are merely exposing the configuration through LDAP and the server.xml or programmatic configuration becomes the master copy which can only be changed with edits to the server.xml or code changes to the programmatic configuration.

New extensions to ApacheDS can be added easily with CiDIT even if those extensions have specific configuration requirements. The configuration system will use cues in entries to instantiate the configuration bean classes used for configuring components like Interceptors, Partitions, ExtendedOperationHandlers and new protocol services. The schemas for these new configuration beans can be added dynamically to support the storage of the new configuration within the DIT. The configuration system will use these new schema elements to store and load component configurations dynamically while dynamically instantiating and configuring the beans. The now allows us to design mechanisms for dynamically adding new components to the server without requiring a restart.

Once the configuration is managed in the DIT other applications which build on ApacheDS like Triplesec will easily be able to load their own configurations into the DIT during their build processes using the configuration plugin. They in turn can support a server.xml file that contains their specific configuration elements. This application specific server.xml file can be used to build their prefabricated system partition with their respective configuration for apacheds components as well as extension components. Also the schema partition plugin can be used to build the prefabricated schema partition to support the configuration schema needed for non-ApacheDS configuration information. Managing the customization of such applications after an installation will be much easier through DIT based changes. These applications can also use their server.xml files to overwrite their configuration in the DIT as well.

Eventually we should do away with using the server.xml file however for now we can include it in the installation footprint as a default-server.xml file. If renamed to server.xml it can be loaded as the over[writing|riding] configuration. Many people seem to love this XML gunk so we'll keep it for now, but they will not be able to benefit from the dynamic reconfiguration capabilities and this may make them adopt the pure CiDIT approach.

Detailed Changes Needed for CiDIT

The above final state to be reached through this CiDIT effort will take some work and there are many details that need to be ironed out. The following sections discuss various aspects of the effort.

> ⚠ Talk about the changes needed to get to the CiDIT setup described above. Discuss the refactoring required at a module, component, and class level. Discuss the new modules and functionality that is needed.

Configuration Bean to Configuration Entry (Schema) Mapping

Configuration beans exist for the server and its existing components. These and their properties need to be mapped to objectClasses and attributeTypes in a schema. We will also need data access code to perform CRUD operations on these entities. This is simple to do when the beans are already known but we cannot presume this since new component types will emerge and applications based on ApacheDS will extend its functionality with new components like interceptors, partitions, extended operations along with the various other extension points in the server. For this reason the data access code must be generalized and dynamic. This in turn requires a consistent convention for mapping configuration beans to LDAP schema entities without presuming the configuration classes are all known at runtime. This convention will be leveraged by the data access code to implement the require CRUD functionality.

Overall, it would be nice to have a tool that generates schema elements from bean interfaces using annotations to override defaults in this mapping convention to govern the process. When combined with schema annotations, additional annotations could also help control how the data access code performs CRUD operations. Such a tool could be reused all over in our projects at directory as well as by our users. Great idea but do we have the time to implement it or not implement it? It will save time but it will take time so when do we break even?

There are several issues here to confront.

Mapping Java Bean Classes to ObjectClasses

> ⚠ This section is going to be seriously long so we need to break it out into another confluence page.

See Mapping Java Beans to LDAP ObjectClasses.

Configuration Plugin

Configuration Schema

Discuss the way the configuration schema needs to be designed to keep configuration loading dynamic and resilient to missing elements when all configuration beans are not visible on the classpath. For example we will have a stock configuration for things like the Kerberos server but the kerberos jars may not be present and used so we cannot load the bean for it.

The schema must be designed so we can load beans by class and not need to have the code predefined. This way new elements can be added later to extend the server.

Discuss how the DIT hierarchy is to be structured for the configuration elements.

More stuff ...

Add more of the details