Registry Installation - High Availability Considerations

COmanage Registry Architecture Basics

COmanage Registry is a stateful PHP web application developed using the CakePHP 2.x framework. Currently the only web server supported is Apache HTTP Server (Apache) version 2.2 or 2.4.The application state is saved in a relational database. CakePHP 2.x supports several relational database server implementations.

Users authenticate to COmanage Registry using a federated identity. The Registry itself does not provide nor directly manage any particular federated identity authentication protocol but simply reads the Apache REMOTE_USER CGI environment variable for a user identifier. REMOTE_USER is expected to be populated using some federated identity technology. Common deployment patterns use the Shibboleth Native SP for Apache HTTP Server, SimpleSA MLphp, and mod_auth_openidc. Other user identifiers and user information may be consumed from other CGI environment variables.

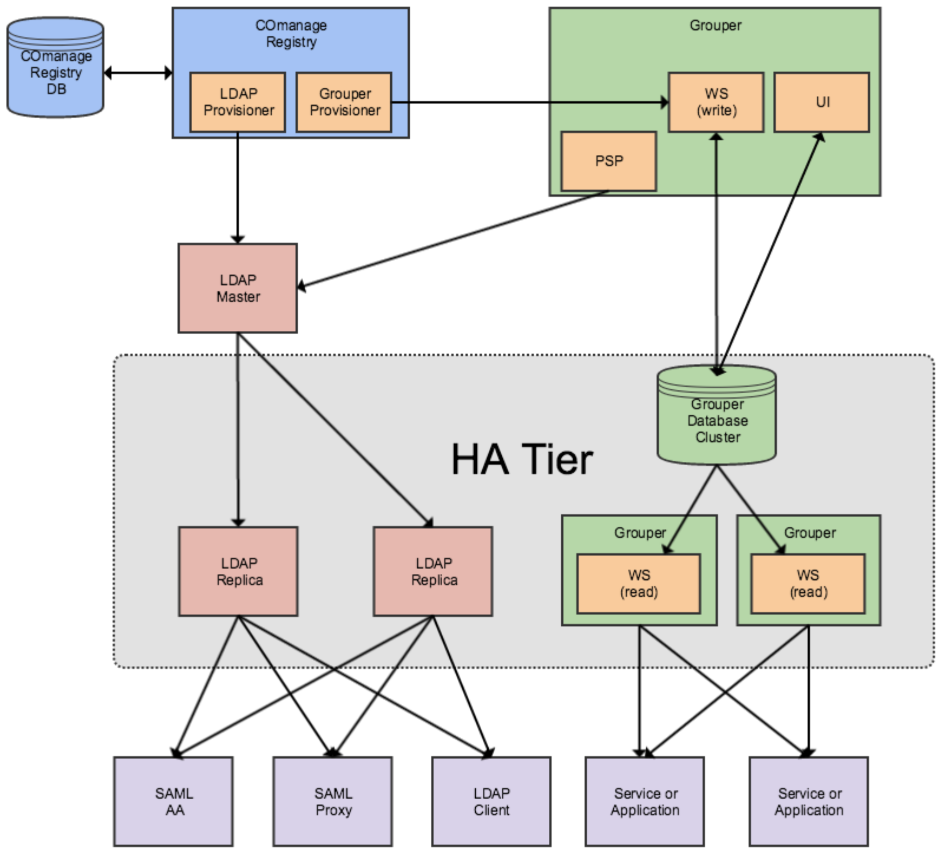
After COmanage Registry authenticates a user by consuming REMOTE_USER the user identity is persisted across requests to the application using stand ard PHP session handling functionality. Specifically any session provided by the federated identity integration (eg. Shibboleth SP) is not required after the COmanage Registry PHP session is established.

COmanage Registry supports a REST interface. The REST client is authenticated via a simple user/password pair transmitted over HTTPS as part of a bas ic auth transaction. More sophisticated authentication mechanisms, such as delegated SAML assertions, may be supported in the future. No persistent session exists between WS calls or invocations.

Common Deployment Pattern

Deployers commonly configure COmanage Registry to provision person and group records to a LDAP directory. Other services such as a SAML attribute authority (AA) or SAML proxy then use the LDAP directory as an authoritative source for user identifiers and attributes including group memberships, which are then often used for access control to services and applications. Some services directly query the LDAP directory for user or group memberships. Deployers also commonly configure COmanage Registry to provision group memberships to Grouper which in turn then provisions the group memberships to the LDAP directory or other systems.

The diagram below shows the common COmanage Registry deployment pattern.



High Availability Approaches

COmanage Registry provides enrollment flows, identifier management, group management, and lifecycle management for members of organizations. Many organizations feel that those capabilities do NOT require high availability (HA). Instead the identifiers, group memberships, and other details about users are consumed by services and used for authorization and access control and those services, tools, and applications DO require HA. As such there are two primary approaches to HA in a COmanage Registry deployment architecture.

HA Only for Downstream Components

Since many organizations only require HA for the services, tools, and applications that consume the information provisioned by COmanage Registry the most common HA approach is to focus on HA for access to LDAP directories and for some deployments Grouper web services (WS) (usually in a read-only mode for HA), and to not deploy COmanage Registry itself in a HA configuration. The diagram above depicts a common deployment scenario that includes a HA tier for LDAP replica servers and Grouper WS servers (and the HA database cluster on which Grouper WS depends). Services such as a SAML AA, SAML proxy, or clients that directly consume from LDAP are configured to rely on the set of LDAP replicas (which may or may not be served from a single logical service endpoint). In this scenario the COmanage Registry is not itself deployed in a HA configuration.

HA for COmanage Registry

When truly required or necessary, COmanage Registry can be deployed in a HA configuration using standard approaches:

- The relational database COmanage Registry uses to persist state must itself be deployed in a HA configuration.
- User PHP sessions must be consistent across the HA configuration (nodes) by either using session affinity (sticky sessions) or properly replicating sessions. Two commons approaches are to use Redis or memcached (not withstanding issues with memcached around its ability to recover from node failure). Writing sessions to a relational database that is itself configured for HA is also a common approach though for performance reasons it does not scale as well horizontally.
- Deployers must take into account any HA needs of the federated identity authentication protocol.