Service Directory Installation and API Guide

Date: March 9, 2016

Americas Headquarters

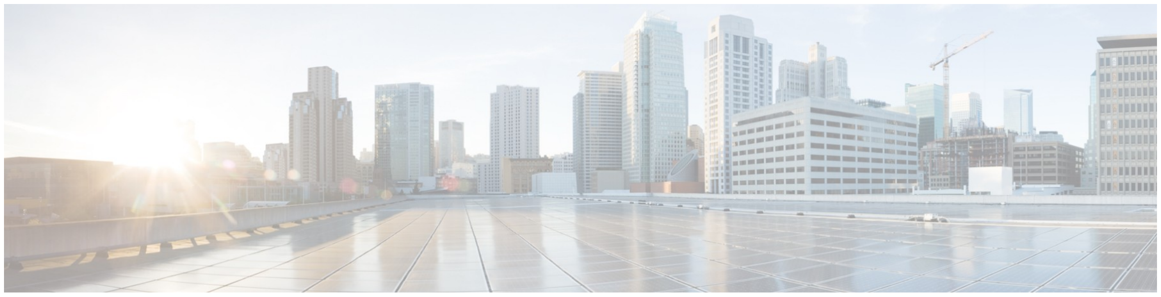
Cisco Systems, Inc.

170 West Tasman Drive

San Jose, CA 95134-1706

USA
http://www.cisco.com
Tel: 408 526-4000

800 553-NETS (6387)
Fax: 408 527-0883



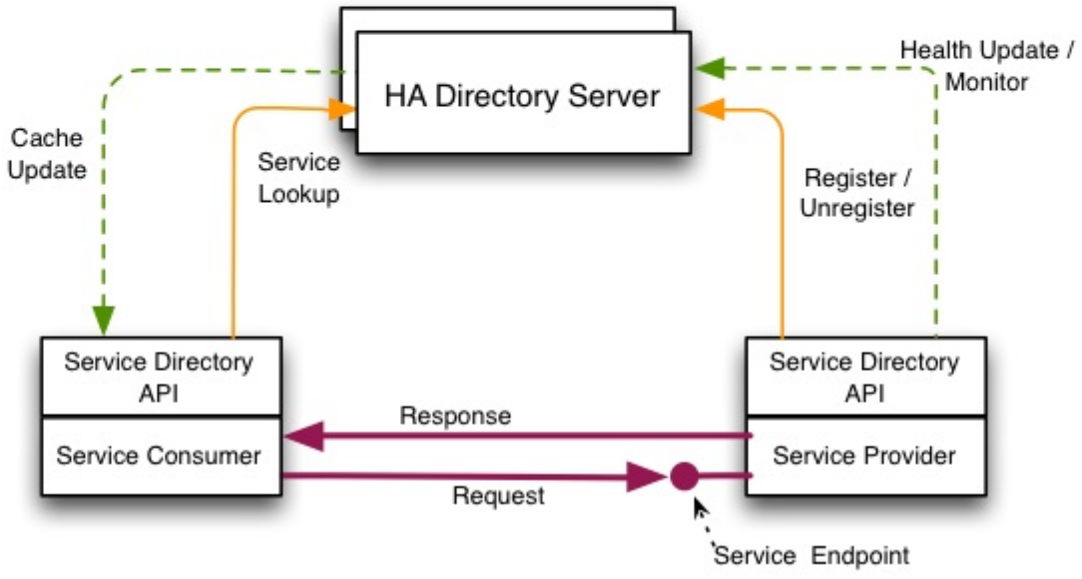
CONTENTS

Service Directory Overview

As the complexity of customer integration grows, so does the number of provisioned service instances. Such complex deployments require that 100's of components and interfaces be created, and then managed per operational requirements including system evolution, resource, and network changes.

Service Directory provides the platform features necessary to support auto-wiring for a large system of many component types and instances. Service Directory is both transparent to the means by which the components are deployed, their organization as Service implementations, components, product families, and so on. It is largely agnostic to the most component implementation concerns, such as language and operating system. Language-specific client libraries may be created, as required. The Service Directory deployment adapts dynamically to changes in the deployed system, requiring no specific user interventions.

Also, the Service Directory facility is an essential requirement of a Service Oriented Architecture (SOA). Providing the basis for a *Service Provider* to register its *Service Endpoints*, such that the attributes required to communicate with the endpoints can be discovered by a *Service Consumer* that needs to use the service. The following diagram provides the pieces that make up the Service Directory deployment.



Architecturally, the Service Directory comprises a highly available (HA) back-end Directory Server, a client API, and a well-defined protocol for clients to communicate with the Directory Server. The following provides a description of the key terms and the components that comprise the Directory Service.

- **Service** – is an abstraction for a logical business process that is uniquely named. The components in a distributed system interact with the service by learning about a specific instance of the service to communicate with. Each instance is implemented by a *Service Provider* application (which may implement many different service instances). The interface to the service instance is called a *Service Endpoint*; its representation is standardly encoded in URI format.
- **Directory Server** – is an application component of the Service Directory facility, which provides distributed persistence for the Service Directory service instance registrations, plus various lookup features, for its clients. The server is responsible for tracking the health of the registered service instances (those intended to be monitored), and to provide a means to make service availability visible to its clients, as required. This may include aggregate service availability, as well as instance availability state.
- **Service Directory Client** – is an application, which interacts with a Directory Server according to one or any of the following roles:
  - Service Directory Provider, which registers one or more services that are implemented by the application itself.
  - Service Directory Consumer, which looks up services by name to obtain information about Service Endpoints that it should communicate with.
  - Service Directory Proxy Provider, a weakened notion of a provider, responsible for registering one or more services, which are implemented externally.
- **Service Directory API** – is a library component of the Service Directory facility that provides a means for a client to fulfill any or all of the above roles.
  Beyond providing a convenient, callable interface, hiding the details of the communications protocols between client and Directory Server, the API includes a performance-boosting service cache (for highly efficient lookups by consumers), and an efficient heartbeat mechanism for monitoring the availability of provided services. A `ServiceDirectory` class (static usage, only) provides the entry point for using the features of the Service Directory API, including: configuration, getting references to the functional interfaces for the main Service Directory client roles, and resetting the API (for unit testing).
- **Service Instance** – is a Service Instance representation of a named Service Endpoint owned by a Service Provider. It is represented in the API by the class `ServiceInstance`. The ServiceInstance contains the name, instance id, URI, and metadata information. Typically, once having looked up a Service and having retrieved a specific ServiceInstance, the client acting as a Service Consumer uses the URI to invoke the function supplied at the specific endpoint of that instance.
  Multiple instances of a named service may exist, each sharing the common service name but with different attributes. Attributes may be intrinsic, modifiable, or provider-defined metadata, and may include:
  - The Service Name that is common to all instances of the service (intrinsic).
  - The identity of the application implementing the service instance (For example, the Service Provider Address (intrinsic)).
  - An attribute to indicate whether the service instance is monitored or not (intrinsic).
  - The *Service Endpoint*, which is a URI specifying the means for a client to communicate with the instance (modifiable).
  - The known status of the instance: Up or Down (modifiable).
  - Additional attributes, in the form of service meta-data, which may be used to refine the lookups to obtain a subset of service instances (modifiable).

- **Registration Manager** – is the portion of the Service Directory API, which supports the roles of the Service Directory Provider (and Proxy Provider). The `RegistrationManager` interface provides methods to register, update, and unregister a ServiceInstance. The API's implementation class is responsible for all communications with the Directory Server in support of these features. This includes an optimized heartbeat model to regularly assert the availability of the provider's registered (and monitorable) instances.
- **Lookup Manager** – is the portion of the Service Directory API, which supports the roles of the Service Directory Consumer. The `LookupManager` interface provides methods for simple ServiceInstance lookups, as well as more refined queries. A change notification interface allows a client application to register for a callback when the state of an instance, of a specified service, has changed.
  The API's implementation class is responsible for all communication, maintaining the cache of instances previously requested by lookups or queries, based on dynamic change notifications from the Directory Service. Single-instance lookup supports a default rotational selection mechanism, providing a default client-side load-balancing feature.

Directory Server Software Installation and Configuration

The Directory Server supports requirements for service registration; monitoring and lookup via RESTful interface, and provides a service facade on top of a persistent data store. This chapter describes how to install and configure the Directory Server.

Installation Requirements

The following, as a minimum are required for the Service Directory software installation:

- RHEL6.4/CentOS6.4 or greater
- Sun Java SDK 1.7.0u5 or greater
- 2 GB RAM or greater

Directory Server Installation

To install the Directory Server software, complete the following:

1. Download and install the server RPM from the following location, using the following command:

   ```
   http://10.84.65.203:8081/nexus/content/repositories/vss-
   releases/com/cisco/oss/foundation/directory/cisco.sd.server/1.2.1-5/
   ```

   ```
   rpm -ivh cisco.sd.server-{version}.noarch.rpm
   ```

2. Install the server package using YUM. The YUM repository can be find at yum_repo_def.

   ```
    yum install cisco.sd.server
   ```

3. Start the server with default configuration.

   ```
   service sdd start
   ```

4. Verify the Directory Server installation status.

   ```
    service sdd status
   ```

   ```
    VCS directory server is running [UP]
   ```

Alternatively the Directory Server can be installed by deploying the foundation services VMware template or QCOW image, which can be downloaded at from the following location:

http://10.84.65.9/VCSF/templates/Foundation-Core/1.6.0/latest-foundation-services/

Directory Server Configuration

To configure the Directory Server a script is used, which is installed after the RPM installation/upgrade. This script can be used for both standalone and HA deployment configurations.

NOTE: This script MUST be executed after the RPM installation/upgrade. This can be done during the deploy/upgrade time.

The Directory server can be configured to run in the following modes:

- Standalone
- Replicated

Standalone Mode

To configure a Directory Server to run in a standalone mode, run the following script:

```
/opt/cisco/sd/bin/configSDServer.sh isHA false
```

Replicated Mode

To configure a Directory Server to run in a replicated mode, run the following script:

```
/opt/cisco/sd/bin/configSDServer.sh isHA true myID id_number quorum “quorum_value”
```

Where:

id_number – Is a number in the range of 1 to 255. This number needs to be different for each server deployed in the same cluster. All the nodes in the cluster must know the other nodes addresses (IP address or hostname), which is specified in quorum argument.

quorum_value – A value formatted as <id1>:<address1>;<id2>:<address2>;<id3>:<address3>

Note that the quorum_value is the same for all nodes in the cluster.

```
In a cluster of 3 nodes where id 1~3 form the cluster, the sample configuration for node
number (=2) is as follows:
/opt/cisco/sd/bin/configSDServer.sh isHA true myID 2 quorum
"1:192.168.1.1;2:192.168.1.2;3:192.168.1.3"

In a cluster of 6 nodes where id 1~3 form the main cluster and 4~6 form the observer
cluster, the sample configuration for node number (=4) is as follows:
/opt/cisco/sd/bin/configSDServer.sh isHA true myID 4 quorum
"1:192.168.1.1;2:192.168.1.2;3:192.168.1.3;4:192.168.1.4:observer;5:192.168.1.5:observer;
6:192.168.1.6:observer"
```

Troubleshooting the Directory Server

In the event of any Directory Server issues, the following can be checked or used to help with the issue:

- Directory Server Failed to Start – if the Directory Server fails to start, refer to the following log for details:

  ```
  /var/log/cisco/sd/runsd.log for details
  ```

- Directory Server Runtime Errors – if there are Directory Server runtime errors, refer to the following log for details:

  ```
  /var/log/cisco/sd/server.log
  ```

- Check, Stop, or Restart Directory Server – to check server status, or to stop/restart the directory server, use one of the following commands (Note that this command should be run by video user not root user):

  ```
  service sdd status|stop|start|restart
  ```

  ```
  /opt/cisco/sd/bin/runds.sh status|stop|start|restart (version 1.1.0-5 or earlier)
  ```

- Configuration and Version Information – the Directory Server configuration and version information are stored in the following directory:

  ```
  /etc/cisco/sd
  ```

Using the Directory Service API

NOTE: This chapter assumes that you have downloaded and configured the Directory Server.

The OSS v1.x SD API requires setting up a v1.x Directory Server. For a client project to use the Service Directory API *V1.x*, the SD API library is specified in the *dependency* section of the project's `pom.xml` file:

```
<dependency>
    <groupId>com.cisco.oss.foundation.directory</groupId>
    <artifactId>sd-api</artifactId>
    <version>1.2.1-5</version>
</dependency>
```

If you are using Ant for your projects, you can download the SD API jars with dependency libraries as a tar ball (sd-api-*version*.tar.gz) from the following location:

http://search.maven.org/#search|ga|1|g%3A%22com.cisco.oss.foundation.directory%22%20AND%20a%3A%22sd-api%22

API Use Cases

Before using the Service Directory API to connect to the directory server, the API needs to be informed of the location of the Directory Service. The following configuration settings are used to communicate this to the API:

The server address:

```
DirectoryServiceRestfulClient.SD_API_SD_SERVER_FQDN_PROPERTY
```

With a default value:

```
vcsdirsvc
```

The server port:

```
DirectoryServiceRestfulClient.SD_API_SD_SERVER_PORT_PROPERTY
```

If the Directory Server is using the default port value (2013), then you only need to ensure that the server address (first property above) works for your environment.

The following methods can be used to make your configuration settings work in your environment:

- Host Alias – Define a host alias `vcsdirsvc` to resolve to the IP address where the Directory Service runs. Some production deployments may use a DNS (especially where HA Directory Server clusters are fronted by a load balancer exposing a virtual IP address). This can also be accomplished by defining the host alias in the /etc/hosts file with the DIR-SVC-IP value, by adding the following line:

  ```
  DIR-SVC-IP vcsdirsvc
  ```

  In this method, the default address setting maps to the DNS, which resolves to the desired IP address.

- Adding a config.properities File – Another method is to add a file named config.properties in your Java classpath (or found using a path which can be specified to the API). This file must have (as a minimum) the following lines:

  ```
  com.cisco.oss.foundation.directory.server.fqdn=SERVER-IP-OR-HOSTNAME
  com.cisco.oss.foundation.directory.server.port=SERVER-PORT
  ```

- Using the Configuration File – Another method is to use the configuration features of Service Directory API to directly set properties for Directory Service IP address or hostname (if it is resolvable), and/or the port number, in code:

  ```
   ServiceDirectory.getServiceDirectoryConfig().setProperty(
     DirectoryServiceRestfulClient.SD_API_SD_SERVER_FQDN_PROPERTY, "SERVER-IP-OR-
     HOSTNAME");
  ServiceDirectory.getServiceDirectoryConfig().setProperty(
     DirectoryServiceRestfulClient.SD_API_SD_SERVER_PORT_PROPERTY, SERVER-PORT);
  ```

  The advantage of this approach is that your application will control where the values for SERVER-IP-OR-HOSTNAME and SERVER-PORT come from (for example, your own property definitions).

  NOTE: A static `shutdown()` method is provided for the `ServiceDirectory` class. If `ServiceDirectory.shutdown()` is called, the SD API will be completely shut down. `ServiceDirectory` is reusable after the `ServiceDirectory.reset()` is called.

Service registration

```
// Get a RegistrationManager instance from ServiceDirectory.
// The ServiceDirectory will load a default ServiceDirectoryConfig and
// instantialize a RegistrationManager instance.
RegistrationManager registrationManager = ServiceDirectory.getRegistrationManager();

// The name of the service
String serviceName = "odrm-setupsession";

// IP or the fully qualified host name of the machine which runs the service.
String address = "10.10.35.7";

// Construct the service instance. serviceName and address together uniquely
// identify a service instance.

ProvidedServiceInstance instance = new ProvidedServiceInstance(serviceName, address);

// Setting the service instance URI.
// URI is defined as tcp:
//address:port for the TCP end point
instance.setUri("http://odrm.cisco.net:8090/ndvr/setupsession");
// By default, the instance status is DOWN instance.setStatus(OperationalStatus.UP);
// Setting the service instance metadata. Optional Map<String, String> meta = new
HashMap<String, String>();
meta.put("version", "2.5.0");
meta.put("datacenter", "datacenter1");
meta.put("region", "east");
instance.setMetadata(meta);

// The port of the service. Optional

int port = 8090;

// The TLS port of the service. Optional

int tls_port = 8443;

// Protocol. Optional
String protocol = "http";
instance.setPort(port);
instance.setTls_port(tls_port);
instance.setProtocol(protocol);

// healthCallback is optional, leave it to be null when registering,

// if the service should not be monitored, e.g. the provider is acting as a proxy.
ServiceInstanceHealth healthCallback = new ServiceInstanceHealth(
         public boolean isHealthy(){
         // implementation, skip here.
         return true;
         }
);

registrationManager.registerService(instance, healthCallback);
OR
registrationManager.registerService(instance);
// Update the OperationalStatus of the instance to DOWN,

// DOWN instance will be removed from lookup.
registrationManager.updateServiceOperationalStatus(serviceName,
instance.getAddress(), OperationalStatus.DOWN);

// Update the instance URI

String uri = "http://odrm.cisco.net:8090/ndvr/ setupsessionnew";
registrationManager.updateServiceUri(serviceName,
instance.getAddress(), uri);

//Update metadata meta.put("solution-node", "odrm");
registrationManager.updateServiceMetadata(serviceName,
instance.getAddress(), meta);

// Unregister the instance.

registrationManager.unregisterService(serviceName, instance.getAddress());
```

Service Lookup

```
// Get the LookupManager from ServiceDirectory.
LookupManager lookupManager = ServiceDirectory.getLookupManager();
String serviceName = "odrm-setupsession";

// Simple service lookup, an available (only **UP**) service instance is returned via
Round-Robin

// policy from the list of the ServiceInstances

ServiceInstance serviceInstance1 = lookupManager.lookupInstance(serviceName);

// Get service endpoint uri
String uri = instance.getUri();

// Look up all ServiceInstances (both UP and DOWN) of the Service.
List<ServiceInstance> allServiceInstances = lookupManager.getInstances(serviceName);

// Filtering the ServiceInstance via some query criteria on the metadata.
ServiceInstanceQuery query = new ServiceInstanceQuery()
.getEqualQueryCriterion("version", "1.0")
.getEqualQueryCriterion("datacenter", "dc01");

// Returns an available ServiceInstance which matches the query criteria.
ServiceInstance versionedServiceInstance = lookupManager.queryInstanceByName(serviceName,
query);

// Query all ServiceInstances which match the query criteria.

List<ServiceInstance> queryedServiceInstances =
lookupManager.queryInstancesByName(serviceName, query);
```

Service Change Callback

```
// A ServiceInstanceChangeListener interface is provided, and user needs to implement the
// onChange method. Example debug messages are provided as examples.

private ServiceInstanceChangeListener sdCallback = new ServiceInstanceChangeListener () {

      public void onChange(ChangeType type, InstanceChange<ServiceInstance> change) {

      switch (type) {
      // An instance is deleted
      case REMOVE:
         LOGGER.debug("Service instance {} has been removed from cache",
                 change.from);
         break;
      // A new instance is added
      case ADD:
         LOGGER.debug("Service instance {} has been added to cache.",
                 change.to);
         break;
      // There is a status change for the instance UP->DOWN or DOWN->UP
     case STATUS:
         LOGGER.debug(
                 "Service instance {} has changed Status from {} to {}",
                 change.from.getAddress(), change.from.getStatus(),
                 change.to.getStatus());
         break;
      // There is a URI change
      case URL:
         LOGGER.debug(
                 "Service instance {} has changed URL from {} to {}",
                 change.from.getAddress(), change.from.getUri(),
                 change.to.getUri());
         break;
      // There is meta data change
     case META:
         Map<String, String> map = new HashMap<String, String>();
         map.putAll(change.to.getMetadata());
         LOGGER.debug(
                 "Service instance {} has changed Metadata from {} to {}",
                 change.from.getAddress(), change.from.getMetadata(),
                 map);
         break;
     default:
         break;
     }
 };

  public void addNotify() throws ServiceException {
     // Add callback to the lookupManager
     LookupManager lookupManager = ServiceDirectory.getLookupManager();
     lookupManager.addInstanceChangeListener(serviceName, sdCallback);
   }
```

Tuning the parameters

The service instance change notification is based on a polling mechanism. The Service Directory API periodically sends polling requests to the server for any instance changes. The default-polling interval is 1 second to allow for any service instance change to be notified timely. The client cache also acts as the service instance change listener, which is updated when there is any service instance change. The Service Directory API provides parameters to allow an application to turn off the polling or adjust the polling interval.

```
// Set polling interval
ServiceDirectory.getServiceDirectoryConfig().setProperty(
   DirectoryLookupService.SD_API_POLLING_DELAY_PROPERTY, 10);
```

```
// Disable client cache
ServiceDirectory.getServiceDirectoryConfig().setProperty(
   ServiceDirectoryConfig.SD_API_CACHE_ENABLED_PROPERTY, false);
```