Contents

1 Introduction

2 Installation
  2.1 Installing the DisnixWebService in NixOS ......................... 2
  2.2 Installing the DisnixWebService manually ....................... 3
    2.2.1 Prerequisites ................................................. 3
    2.2.2 Compiling from source code ................................. 3
    2.2.3 Deploying the web service interface on Apache Tomcat ...... 3
      2.2.3.1 Configuring permissions ................................. 3
      2.2.3.2 Installing shared libraries ........................... 3
      2.2.3.3 Configuring native library access ....................... 4
      2.2.3.4 Deploying the web service interface .................... 4
      2.2.3.5 Securing the web service interface ..................... 4
  2.3 Configuring the Disnix toolset .................................. 5

A Command Reference
  A.1 Utilities ......................................................... 6
    A.1.1 disnix-soap-client ......................................... 6
Chapter 1

Introduction

Disnix is a toolset which main purpose is to deploy service-oriented systems into networks of machines. Disnix is built on top of Nix; a package manager which has some unique features to make deployment safe, reliable, and reproducible.

Disnix contains an interface which exposes remote deployment operations to client tooling. Disnix is designed to support multiple connection protocols by implementing a thin protocol wrapper around the core Disnix service. For example, an SSH interface is included in the main Disnix distribution allowing someone to execute remote deployment steps using an SSH connection.

This package contains the DisnixWebService wrapper and a disnix-soap-client tool that connects to the wrapper. These tools can be combined with the basic Disnix toolset to allow it to use the SOAP protocol to perform remote deployment operations. The main purpose of this package is to show how a custom protocol wrapper can be implemented.
Chapter 2

Installation

This chapter explains how the DisnixWebService can be installed. Firstly, the basic Disnix toolset must be installed. Installation instructions can be found in the Disnix manual. Moreover, a Java servlet container capable of running Apache Axis2 is required for this. The DisnixWebService can be installed on NixOS, which only takes little configuration effort, or manually on other operating systems which require you to perform more configuration steps.

2.1 Installing the DisnixWebService in NixOS

The DisnixWebService can be enabled by setting the following property in your /etc/nixos/configuration.nix configuration file:

```
services.disnix.useWebServiceInterface = true;
```

By enabling this option, the Disnix module automatically configures the Apache Tomcat NixOS service, running the DisnixWebService web application. The web application has been configured with the right properties and permissions so that the deployment operations of the core Disnix service can be accessed.

The web service itself does not have any security facilities whatsoever. If authentication is required, you can put a HTTP reverse proxy in front of it that does basic authentication. In NixOS (using the Apache HTTP server) this can be accomplished by adding the following snippet to the NixOS configuration:

```
services.httpd.enable = true;
services.httpd.adminAddr = "admin@localhost";
services.httpd.hostName = "localhost";
services.httpd.extraConfig = ''
  <Proxy *>
    Order deny,allow
    Allow from all
    AuthType basic
    AuthName "DisnixWebService"
    AuthBasicProvider file
    AuthUserFile ${./htpasswd}
    Require user admin
  </Proxy>
  ProxyRequests off
  ProxyPreserveHost on
  ProxyPass / http://localhost:8080/ retry=5 disablereuse=on
  ProxyPassReverse / http://localhost:8080/
'';
```
The `htpasswd` file (that contains one administrator user) can be generated as follows:

```bash
$ htpasswd -c admin htpasswd
```

## 2.2 Installing the DisnixWebService manually

### 2.2.1 Prerequisites

In order to build the `DisnixWebService` the following dependencies are required. The tooling is implemented in Java. Therefore, the Java Development Kit (at least version 1.5) must be installed. Apache Ant is required to build the package from source code. Apache Axis2 is used to implement the client interface and as hosting environment for the web service front-end. The web service requires the D-Bus Java library (which requires `libmatthew-java`) to connect to the core Disnix service.

### 2.2.2 Compiling from source code

After unpacking or checking out the DisnixWebService source code, it can be compiled by executing the following commands:

```bash
ant
ant install
```

### 2.2.3 Deploying the web service interface on Apache Tomcat

In principle, any Java servlet container or application server compatible with Apache Axis2 should be able to host the web service interface. In this section we explain how the web service interface can be deployed to Apache Tomcat. For other Java servlet containers or application servers you should consult the documentation included with your application server.

#### 2.2.3.1 Configuring permissions

As explained in the Disnix manual, to access operations of the Disnix core service either root permissions are required or a specific user must be a member of the `disnix` user group. On most distributions Apache Tomcat runs as an unprivileged user (such as the `tomcat` user). To grant Apache Tomcat access to the core Disnix service you have to make the `tomcat` user part of the `disnix` group. This can be done by typing:

```bash
$ usermod -a -G disnix tomcat
```

Check your distribution’s configuration to see under which user Apache Tomcat runs.

#### 2.2.3.2 Installing shared libraries

The Disnix web service interface uses the D-Bus system bus to connect to the core Disnix service. D-Bus communicates through Unix domain sockets, requiring the web service to use native libraries. Native libraries can only be invoked by the servlet container itself (not from web application directly) and requires you the install some libraries as shared libraries which can be used by both the servlet container and the web applications.

The first step is to create a shared libs directory inside the Apache Tomcat directory:

```bash
$ mkdir -p shared/lib
```
Then the `DisnixConnection.jar` library from the `DisnixWebService` distribution and the `dbus.jar` from the `dbus-java` package must be copied into the `shared/lib` folder.

The next step is to edit the Apache Tomcat configuration file to enable the shared library loader using `shared/lib` folder. This can be done by editing the `conf/catalina.properties` configuration file in your Apache Tomcat distribution and replacing the line containing: `shared.loader` with the following line:

```
shared.loader=${catalina.base}/shared/lib/*.jar
```

### 2.2.3.3 Configuring native library access

As mentioned earlier, the web service interface use the D-Bus system bus, to access the core Disnix service which requires access to native libraries communicating through Unix domain sockets. The following environment variable must be configured to specify where Apache Tomcat can find native libraries:

```
$ export JAVA_OPTS="-Djava.library.path=/path/to/libmatthew-java/lib/jni"
```

The path above must be replace to the location where `libmatthew-java` has been installed.

### 2.2.3.4 Deploying the web service interface

Finally, the web service front-end itself must be deployed. This can be done by copying the `DisnixWebService.war` file of the `DisnixWebService` distribution into the `webapps/` folder of Apache Tomcat.

### 2.2.3.5 Securing the web service interface

The web service interface has no security facilities whatsoever. To support basic HTTP authentication, a reverse proxy can be put in front of it taking care of it. For the Apache HTTP server you basically need to integrate the following snippet in your configuration:

```
<Proxy *>  
  Order deny,allow  
  Allow from all  
  AuthType basic  
  AuthName "DisnixWebService"  
  AuthBasicProvider file  
  AuthUserFile ./htpasswd  
  Require user admin  
</Proxy>
```

ProxyRequests off
ProxyPreserveHost on
ProxyPass / http://localhost:8080/ retry=5 disablereuse=on
ProxyPassReverse / http://localhost:8080/

The `htpasswd` file (that contains one administrator user) can be generated as follows:

```
$ htpasswd -c admin htpasswd
```
2.3 Configuring the Disnix toolset

In order to make this extension work with basic toolset two things need to be configured. First, the toolset must be configured to use the **disnix-soap-client**. Second, a target property must be configured. By default Disnix uses the **hostname** attribute in the infrastructure model to connect to the target interface. This property does not work with the **DisnixWebService**, because an URL is required in order to connect to the web service.

To let Disnix use the **disnix-soap-client** to communicate with the web service interface, the following environment variable must be configured:

```bash
$ export DISNIX_CLIENT_INTERFACE=disnix-soap-client
```

To let Disnix use a different target property from the infrastructure model the following environment variable must be configured:

```bash
$ export DISNIX_TARGET_PROPERTY=targetEPR
```

By setting this variable, the **targetEPR** attribute is used (which is a convention we use for web services). Alternatively, you can use also use a different attribute name if desired.

If a reverse proxy taking care of HTTP basic authentication has been configured, the following environment variables can be used to provide the credentials:

```bash
$ export DISNIX_SOAP_CLIENT_USERNAME=admin
$ export DISNIX_SOAP_CLIENT_PASSWORD=secret
```
Appendix A

Command Reference

A.1 Utilities

A.1.1 disnix-soap-client

disnix-soap-client — Provides access to the disnix-service through the SOAP protocol

Synopsis

disinx-soap-client --target targetEPR operation [OPTION] paths

DESCRIPTION

The command `disnix-soap-client' provides remote access to a `disnix-service' instance running on a machine in the network by using a SOAP/HTTP connection. This allows the user to perform remote deployment operations on a target machine through SOAP.

In most cases this command is not used directly, but is used by specifying the `--interface' option for a Disnix command-line utility (such as `disnix-env') or by setting the `DISNIX_CLIENT_INTERFACE' environment variable. By using one of those properties, the Disnix tools will use the given interface instead of the standard `disnix-client' which only provides loopback access.

OPTIONS

Operations:

--import Imports a given closure into the Nix store of the target machine. Optionally, transfers the closure from this machine to the target machine

--export Exports the closure of a given Nix store path of the target machine into a file, and optionally downloads it

--print-invalid Prints all the paths that are not valid in the Nix store of the target machine

-r, --realise Realises the given store derivation on the target machine

--set Creates a Disnix profile only containing the given derivation on the target machine
-q, --query-installed  Queries all the installed services on the given target machine

--query-requisites  Queries all the requisites (intra-dependencies) of the given services on the target machine

--collect-garbage  Collects garbage on the given target machine

--activate  Activates the given service on the target machine

--deactivate  Deactivates the given service on the target machine

--lock  Acquires a lock on a Disnix profile of the target machine

--unlock  Release the lock on a Disnix profile of the target machine

--snapshot  Snapshots the logical state of a component on the given target machine

--restore  Restores the logical state of a component on the given target machine

--delete-state  Deletes the state of a component on the given machine

--query-all-snapshots  Queries all available snapshots of a component on the given target machine

--query-latest-snapshot  Queries the latest snapshot of a component on the given target machine

--print-missing-snapshots  Prints the paths of all snapshots not present on the given target machine

--import-snapshots  Imports the specified snapshots into the remote snapshot store

--export-snapshots  Exports the specified snapshot to the local snapshot store

--resolve-snapshots  Converts the relative paths to the snapshots to absolute paths

--clean-snapshots  Removes older snapshots from the snapshot store

--capture-config  Captures the configuration of the machine from the Dysnomia container properties in a Nix expression

--shell  Spawns a Dysnomia shell to run arbitrary maintenance tasks

--help  Shows the usage of this command to the user

--version  Shows the version of this command to the user

General options:

-t, --target=TARGET  Specifies the hostname and optional port number of the SSH server used to connect to the target machine

Import/Export/Import snapshots/Export snapshots/Shell options:

--localfile  Specifies that the given paths are stored locally and must be transferred to the remote machine if needed

--remotefile  Specifies that the given paths are stored remotely and must transferred from the remote machine if needed

Shell options:

--command=COMMAND  Commands to execute in the shell session

Set/Query installed/Lock/Unlock options:

-p, --profile=PROFILE  Name of the Disnix profile. Defaults to: default
Collect garbage options:

-d, --delete-old  Indicates whether all older generations of Nix profiles must be removed as well

Activation/Deactivation/Snapshot/Restore/Delete state options:

--type=TYPE  Specifies the activation module that should be used, such as echo or process.
--arguments=ARGUMENTS  Specifies the arguments passed to the Dysnomia module, which is a string with key=value pairs
--container=CONTAINER  Name of the container in which the component is managed. If omitted it will default to the same value as the type.

Query all snapshots/Query latest snapshot options:

-C, --container=CONTAINER  Name of the container in which the component is managed
-c, --component=COMPONENT  Name of the component hosted in a container

Clean snapshots options:

--keep=NUM  Amount of snapshot generations to keep. Defaults to: 1
-C, --container=CONTAINER  Name of the container to filter on
-c, --component=COMPONENT  Name of the component to filter on

ENVIRONMENT

DISNIX_PROFILE  Sets the name of the profile that stores the manifest on the coordinator machine and the deployed services per machine on each target (Defaults to: default)

DYSNOMIA_STATEDIR  Specifies where the snapshots must be stored on the coordinator machine (defaults to: /var/dysnomia)

COPYRIGHT

Copyright (C) 2008-2018 Sander van der Burg