DisnixOS User’s Guide

Draft (Version 0.5)
# Contents

1 Introduction 1  
   1.1 Features ................................................................. 1  
   1.1.1 Infrastructure deployment ........................................... 1  
   1.1.2 Infrastructure virtualisation ....................................... 1  
   1.1.3 Automated testing of a service-oriented system ................... 1  
   1.1.4 Infrastructure instantiation and provisioning .................... 2  

2 Background 3  

3 Installation 7  
   3.1 Compiling DisnixOS from source ....................................... 7  
   3.1.1 Prerequisites ......................................................... 7  
   3.1.2 Compiling DisnixOS .................................................. 7  
   3.2 Installing DisnixOS by using the Nix package manager ............... 7  

4 Usage 9  
   4.1 Deploying the infrastructure .......................................... 9  
   4.2 Deploying the services and infrastructure at the same time ........ 9  
   4.3 Deploying services with DisnixOS and infrastructure with NixOps .... 10  
   4.4 Deploying a system in a virtual network ................................ 10  
   4.5 Collecting garbage ..................................................... 10  
   4.6 Building a system on the coordinator machine ....................... 10  
   4.7 Building system components on target machines ..................... 11  
   4.8 Generating an infrastructure model ................................... 11  
   4.9 Snapshotting the state of the infrastructure ....................... 11  
   4.10 Restoring the state of the infrastructure ............................ 11  
   4.11 Deleting obsolete infrastructure state ............................. 12  
   4.12 Automated testing of Disnix deployments ............................ 12
A Command Reference

A.1 Main commands
   A.1.1 disnixos-env
   A.1.2 disnixos-vm-env

A.2 Utilities
   A.2.1 disnix-backdoor-client
   A.2.2 disnix-nixops-client
   A.2.3 disnixos-manifest
   A.2.4 disnixos-instantiate
   A.2.5 disnixos-deploy-network
   A.2.6 disnixos-collect-garbage
   A.2.7 disnixos-clean-snapshots
   A.2.8 disnixos-query
   A.2.9 disnixos-geninfra
   A.2.10 disnixos-gentests
Chapter 1

Introduction

Disnix is a distributed service deployment toolset which main purpose is to deploy service oriented systems (i.e. systems that can be decomposed into "distributable units") into networks of machines having various characteristics (such as operating systems) and is built on top of Nix, a package manager which has some unique features compared to conventional package managers to make deployment safe and reliable.

The purpose of Disnix is to manage the deployment of distributable application components (or services) in a network of machines. Application components can be anything ranging from web services, web applications, databases, and ordinary processes.

Disnix, however, expects the underlying infrastructure, such as machines running certain kinds of operating systems with certain containers (e.g. a DBMS or application server) to be present already. It does not manage the deployment of the infrastructure and their system configurations. Normally, a system administrator must manually install machines in the network having the required characteristics by following the installation steps of the operating system.

Moreover, a system administrator or developer must manually write an infrastructure model, matching the actual configuration of the network in order to correctly deploy a service-oriented system, which may be a lot of work and error prone.

This DisnixOS extension provides complementary infrastructure management tools for Disnix built on top of NixOS, a GNU/Linux distribution built around the Nix package manager. NixOS is a distribution in which complete system configurations can be built from declarative specifications and supports atomic upgrades and rollbacks of complete system configurations.

1.1 Features

DisnixOS adds the following features to the basic Disnix toolset:

1.1.1 Infrastructure deployment

DisnixOS allows you to write a network of NixOS configurations in a model and to automatically deploy the system configurations in a network of machines through their Disnix interfaces.

1.1.2 Infrastructure virtualisation

DisnixOS uses the NixOS test driver to build a network of virtual machines closely matching the actual network and automatically deploys the service-oriented system in the virtual network, so that it can be easily tested by developers.

1.1.3 Automated testing of a service-oriented system

DisnixOS uses the NixOS test driver techniques, used for the NixOS testsuite, to automatically script the deployment and testcases in a distributed environment, so that a service-oriented system can be automatically tested, and run on Hydra, a continuous build and integration server built on top of Nix.
1.1.4 Infrastructure instantiation and provisioning

DisnixOS can also be used in combination with NixOps, a tool that deploys networks of NixOS configurations and automatically instantiate and provisions virtual machines in an IaaS environment, such as Amazon EC2.
Chapter 2

Background

NixOS is a GNU/Linux distribution using the Nix package manager to manage all packages, system components (such as the Linux kernel) and configuration files. In NixOS, system configurations are derived from declarative specifications describing properties of the system, such as kernel modules, partitions, services and system packages.

Example 2.1 A NixOS configuration running a OpenSSH server

```nix
{pkgs, ...}: 1
{
  boot = { 2
    loader = {
      grub = {
        device = "/dev/sda";
      };
    };
  };

  fileSystems = [ 3
    { mountPoint = "/";
      device = "/dev/sda2";
    }
  ];

  swapDevices = [ 4
    { device = "/dev/sda1"; }
  ];

  services = {
    openssh = {
      enable = true;
    };
  };

  environment = {
    systemPackages = {
      pkgs.mc
      pkgs.subversion
      pkgs.lynx
    };
  };
}
```

Example 2.1 shows a Nix expression describing the configuration of a machine.
This line states that the expression is a function, having an argument called \texttt{pkgs}, which refers to \texttt{Nixpkgs}, a collection of 2500 packages that can be deployed by the Nix package manager.

This section of the NixOS specification describes the settings of the GRUB bootloader, such as that the partition containing the MBR is \texttt{/dev/sda}.

This section of the NixOS specification describes the settings of hard drive partitions, such as the root partition and the swap partition.

This section of the NixOS specification describes the services running on the system, such as the OpenSSH server providing remote SSH access.

This section of the NixOS specification describes the system packages, which can be used by all users on the machines, such as the Lynx web browser and Midnight Commander.

The specification in Example 2.1 should be stored in the following location: \texttt{/etc/nixos/configuration.nix}. The complete configuration can be deployed by running:

\begin{lstlisting}
$ nixos-rebuild switch
\end{lstlisting}

NixOS configurations can also be used in an attribute set, describing a network of machines:

\begin{example}
A network of NixOS configurations
\end{example}

\begin{lstlisting}
{  
test1 = import ../configurations/openssh.nix;  
test2 = import ../configurations/tomcat.nix;  
test3 = import ../configurations/httpd.nix;  
}
\end{lstlisting}

Example 2.2 shows a network of NixOS configurations. Basically, this expression is an attribute set in which each attribute refers to a NixOS configuration:

1. This attribute declares that we have a machine with identifier \texttt{test1}, having a configuration running a OpenSSH server, described earlier in Example 2.1.
2. This attribute declares that we have a machine with identifier \texttt{test2}, having a configuration running an Apache Tomcat server.

By typing the following instruction on the command line, the network of machines are built from source code, distributed to the target machines and finally activated:

\begin{lstlisting}
$ disnixos-deploy-network network.nix
\end{lstlisting}

In distributed environments, there are several more advanced features that may be desired to use. For example, each individual machine configuration has properties that are portable among any machine (such as the system services and end-user packages to run) and some properties are machine dependent, such as the partitions and boot loader settings. Moreover, system configurations may also have dependencies on other machines in the network.

NixOS network models also allow to separate machine configuration aspects in separate files and to allow one machine configuration to refer to another. The following example shows a more advanced configuration capturing the logical aspects of machines in a network. Furthermore, the system configurations have references to each other:
Example 2.3 A logical NixOS network model

```nix
{ tomcat = {pkgs, ...}: 

services = { 
    openssh = { 
        enable = true; 
    }; 
    tomcat = { 
        enable = true; 
    }; 
};

httpd = {pkgs, nodes, ...}: 

{ 
    services = { 
        openssh = { 
            enable = true; 
        }; 
    };

httpd = { 
    enable = true; 
    adminAddr = "admin@example.com"; 
    extraModules = [ "proxy_balancer "]; 
    extraConfig = '' 
    <Proxy balancer://cluster> 
        Allow from all 
        BalancerMember http://${nodes.tomcat.config.networking.hostName}:8080 retry=0 
    </Proxy> 

    ProxyStatus full 
    ProxyPass /server-status ! 
    ProxyPass / balancer://cluster/ 
    ProxyPassReverse / balancer://cluster/ 
    '';
};
};
}
```

The model in Example 2.3 is an attribute set in which each value refers to a NixOS configuration:

1. The `tomcat` machine is a simple configuration hosting Apache Tomcat serving an arbitrary number of web applications, and an OpenSSH server.

2. The `httpd` machine runs a reverse proxy that forwards all incoming requests to the Apache Tomcat server. It fetches the hostname of the `tomcat` machine’s configuration through the `nodes` parameter.

The logical network model above only captures machine independent properties and can be used in combination with a physical network model, capturing machine specific properties, such as the partitions and bootloader settings:
Example 2.4 A physical NixOS network model

```nix
{
  tomcat = {pkgs, ...}:
    {
      boot = {
        loader = {
          grub = {
            device = "/dev/sda";
          }
        }
      }
      fileSystems = [
        { mountPoint = "/";
          device = "/dev/sda2";
        }
      ];
      swapDevices = [
        { device = "/dev/sda1"; }
      ];
    }
  httpd = {pkgs, ...}:
    {
      boot = {
        loader = {
          grub = {
            device = "/dev/sda";
          }
        }
      }
      fileSystems = [
        { mountPoint = "/";
          device = "/dev/sda3";
        }
      ];
      swapDevices = [
        { device = "/dev/sda2"; }
      ];
    }
}
```

The network in model shown in Example 2.4 captures machine-specific properties, such as partition and bootloader settings. With the following command the network can be deployed taking both the given logical and physical properties into account:

```
$ disnixos-deploy-network network-logical.nix network-physical.nix
```

Disnix is a toolset supporting the deployment of services. DisnixOS combines the latter feature of using disnixos-deploy-network to deploy networks of NixOS machines, as addition to Disnix to provide infrastructure deployment.

Physical network models can also refer to virtual machine properties hosted in an IaaS environment, such as Amazon EC2. If DisnixOS is used with NixOps, these machines can be automatically instantiated and deployed as well. Consult the NixOps manual for more details on VM instantiation and provisioning.
Chapter 3

Installation

This chapter explains how DisnixOS can be installed. Apart from the basic Disnix toolset, only the coordinator machine needs the DisnixOS extension tools.

3.1 Compiling DisnixOS from source

This section explains how to compile DisnixOS from source code. Usually, this is only needed in special circumstances such as for development, or trying a new platform. A more convenient way to install DisnixOS is to use the Nix package manager.

3.1.1 Prerequisites

In order to build DisnixOS from source code, the following dependencies are required: Since it is an extension to Disnix, the core Disnix toolset must be installed. Have a look for installation instructions in the Disnix manual. Moreover, for network communication with the NixOS test driver, DisnixOS uses socat, which must also be present on the system. For infrastructure instantiation and provisioning, NixOps must be installed. Have a look at the NixOps manual for more information.

3.1.2 Compiling DisnixOS

After unpacking or checking out the DisnixOS sources, it can be compiled by executing the following commands:

```
$ ./configure options...
$ make
$ make install
```

When building from the Git repository, these should be preceded by the command:

```
$ ./bootstrap
```

The installation path can be specified by passing the `--prefix=prefix` to `configure`. The default installation directory is `/usr/local`. You can change this to any location you like. You must have write permission to the `prefix` path.

3.2 Installing DisnixOS by using the Nix package manager

The easiest way to use DisnixOS is by installing the DisnixOS package with the Nix package manager from the Nixpkgs repository by typing:

```
$ nix-env -i disnixos
```
Another way is downloading the DisnixOS source distribution and to compile it manually. Moreover, it is also a good thing to have to basic Disnix toolset in your environment. This can be done by typing:

```
$ nix-env -i disnix
```
Chapter 4

Usage

DisnixOS offers some additional use cases in addition to the ones provided by the basic Disnix toolset, which are shown in this section.

4.1 Deploying the infrastructure

To deploy a network of NixOS configurations of machines, the following instruction can be used:

```
$ disnixos-deploy-network network.nix
```

By executing the command above, all the system configurations are built from the NixOS specifications, then distributed to the machines in the network and finally activated.

The same command can also take multiple network models:

```
$ disnixos-deploy-network network-logical.nix network-physical.nix
```

By executing the command above, the machine configurations in the logical and physical model are merged together by the NixOS module system into a single network configuration. Then the configurations are built, distributed and activated.

4.2 Deploying the services and infrastructure at the same time

In order to automatically deploy NixOS configurations in a network machines and the services of which a distributed system is composed, the following instruction can be used:

```
$ disnixos-env -s services.nix -n network.nix -d distribution.nix
```

This command is essentially a combination of the disnixos-deploy-network (which deploys the infrastructure) and disnix-env (which deploys the services) commands.

Likewise, it can also take multiple network models that get merged together:

```
$ disnixos-env -s services.nix -n network-logical.nix -n network-physical.nix -d distribution.nix
```

The disnixos-env supports most of the options that disnix-env supports. For example, adding the --build-on-targets parameter builds all system configurations on the target machines, instead of the coordinator machine.
4.3 Deploying services with DisnixOS and infrastructure with NixOps

Besides using Disnix for deploying infrastructure, we can also take the facilities of NixOps into account and use its VM instantiation and provisioning properties. To easily force all DisnixOS tools to use NixOps, we can set the following environment variable:

$ export DISNIXOS_USE_NIXOPS=1

When this option is set, we can use NixOps to instantiate and deploy a network (which has the deployment name: test) of NixOS machines in the cloud:

$ nixops create ./network-logical.nix ./network-physical.nix -d test
$ nixops deploy -d test

We can set the following environment variable to make test the default deployment and allowing it to be used by any DisnixOS utility:

$ export NIXOPS_DEPLOYMENT=test

After NixOps is done deploying the system configurations, we can deploy the services into the network by running:

$ disnixos-env -s services.nix -n network-logical.nix -n network-physical.nix -d distribution.nix

DisnixOS automatically uses NixOps’ SSH connection facilities to connect to remote machines and execute remote deployment steps.

4.4 Deploying a system in a virtual network

You can also deploy a system in a network of cheaply instantiated virtual machines provided by the NixOS test driver, which is useful for testing. This can be done by executing the following instruction:

$ disnixos-vm-env -s services.nix -n network.nix -d distribution.nix

The disnixos-vm-env command is almost similar to the disnix-env command, except that it generates and launches a network of virtual machine instances. Moreover, this command cannot be used to upgrade a system.

4.5 Collecting garbage

The DisnixOS toolset provides a wrapper around Disnix’s garbage collector command to take a collection of network models instead of infrastructure model to automatically collect garbage from a network of machines. The following instruction uses two network models to connect and removes old generations of Nix profiles (through the -d option) as well:

$ disnixos-collect-garbage -d network-logical.nix network-physical.nix

4.6 Building a system on the coordinator machine

In order to just build the services on the coordinator machines, without distributing or activating, the following instruction can be used:

$ disnixos-manifest -s services.nix -n network-logical.nix -n network-physical.nix -d distribution.nix
Apart from building services, we can also build the system configurations by using the same command:

```
$ disnixos-manifest -n network-logical.nix -n network-physical.nix
```

Omitting the services and distribution models, causes `disnixos-manifest` to build the NixOS system configurations. The resulting manifest files can be distributed to the target machines and activated by using the `disnix-distribute` and `disnix-activate` commands.

### 4.7 Building system components on target machines

As with the basic Disnix toolset we can also delegate builds to the target machines in the network. The following command generates a distributed derivation file for the services:

```
$ disnixos-instantiate -s services.nix -n network-logical.nix -n network-physical.nix -d distribution.nix
```

Omitting the services and distribution models, causes `disnixos-instantiate` to generate a distributed derivation file for the system configurations (infrastructure):

```
$ disnixos-instantiate -n network-logical.nix -n network-physical.nix
```

The resulting distributed derivation files for the services and infrastructure can be built by running:

```
$ disnix-build ./result
```

### 4.8 Generating an infrastructure model

DisnixOS uses a network model instead of an infrastructure model, which the basic Disnix toolset uses. Essentially, the semantics of these models are quite similar. The `disnixos-geninfra` can be used to generate an infrastructure model from the network model:

```
$ disnixos-geninfra network.nix
```

The output of the tool is the path to the generated infrastructure model. This file can be used by tools in the basic Disnix toolset. For example:

```
$ disnix-collect-garbage $(disnixos-geninfra network.nix)
```

The above command runs the Disnix garbage collector on machines defined in the network model.

### 4.9 Snapshotting the state of the infrastructure

It is also possible to manage the state of mutable components deployed as part of a NixOS configuration for systems that have the Dysnomia NixOS module enabled. To snapshot their state and transfer the composite snapshot to the coordinator run:

```
$ disnixos-snapshot-network network.nix
```

### 4.10 Restoring the state of the infrastructure

Besides snapshotting the state of the infrastructure, it is also possible to restore it:

```
$ disnixos-restore-network network.nix
```
4.11 Deleting obsolete infrastructure state

The obsolete state of the mutable components that have been deployed as part of the infrastructure can be removed as follows:

$ disnixos-delete-network-state network.nix

4.12 Automated testing of Disnix deployments

Another use case of DisnixOS is to automatically package a Disnix project, build the corresponding manifest and run automated tests non-interactively from Nix expressions.

Example 4.1 A Nix expression jobset for testing Disnix projects

```nix
{ nixpkgs = <nixpkgs>, system = builtins.currentSystem }:

let
  pkgs = import nixpkgs { inherit system; };

  disnixos = import "${pkgs.disnixos}/share/disnixos/testing.nix" { #
    inherit nixpkgs system;
  };

  version = "1.0";

in
rec {
  tarball = disnixos.sourceTarball {
    name = "testproject-zip";
    inherit version;
    src = ./.;
    officialRelease = false;
  };

  manifest = disnixos.buildManifest {
    name = "test-project-manifest";
    version = builtins.readFile ./version;
    inherit tarball;
    servicesFile = "deployment/DistributedDeployment/services.nix";
    networkFile = "deployment/DistributedDeployment/network.nix";
    distributionFile = "deployment/DistributedDeployment/distribution.nix";
  };

  tests = disnixos.disnixTest {
    name = "test-project-tests";
    inherit tarball manifest;
    networkFile = "deployment/DistributedDeployment/network.nix";
    dysnomiaStateDir = /var/state/dysnomia;
    testScript = ''
    # Wait until the front-end application is deployed
    $test1->waitForFile("/var/tomcat/webapps/testapp");

    # Wait a little longer and capture the output of the entry page
    my $result = $test1->mustSucceed("sleep 10; curl --fail http://test2:8080/testapp ←
    ");
```
Example 4.1 shows a Nix expression defining three jobs:

1. This line imports the testing function abstractions provided by DisnixOS.
2. This function invocation constructs a bzip2 compressed tarball containing all files belonging to a Disnix project.
3. Here, the three DisnixOS models provided by a source distribution are built from source code and a `manifest.xml` is produced. Basically this function abstraction does the same thing as the `disnixos-manifest` command.
4. This job spawns a network of QEMU machines inside a Nix build, consults DisnixOS to deploy the system and performs some testcases. Basically this function abstraction does mostly what the command `disnixos-vm-env` does, but in a non-interactive manner. Moreover, it also takes a test script as a parameter that is written in Perl. Test scripts use the facilities of the NixOS test driver for execution. Consult the NixOS manual for more details on how to write test scripts. Optionally, it can also refer to a Dysnomia snapshot folder. If a snapshot folder has been provided, then the latest snapshots inside are restored in the VMs after the system has been successfully deployed.
Appendix A

Command Reference

A.1 Main commands

A.1.1 disnixos-env

disnixos-env — Installs or updates the services and infrastructure of a distributed system

Synopsis

disnixos-env -s services_nix -n network.nix [-n network2.nix...] -d distribution_nix [OPTION]
disnixos-env --rollback [OPTION]
disnixos-env --switch-generation NUM [OPTION]

DESCRIPTION

The command `disnixos-env' is used to install, upgrade or roll back the services and infrastructure of distributed system in a given environment. This command requires three Nix expressions as input parameters; A services model capturing the components of a distributed system and its inter-dependencies; A network model capturing the NixOS configurations of machines in the network and its properties and a distribution model which maps services to machines.

By invoking this command it will first build, distribute and activate all the NixOS configurations of the machines in the network. Then it will build all the services that are defined in the distribution model from source code including all its dependencies. If all the services are successfully built, the closures of the services are transferred to the target machines in the network. Finally, the services are activated by traversing the inter-dependency graph of all the services. In case of a failure, a rollback is performed to bring the system back to its previous configuration.

When there is already a distributed system configuration deployed, an upgrade is performed. In this phase only the changed parts of the system are deactivated and activated. In this process we also deal with the inter-dependencies so that no service deployment fails due to a missing inter-dependency.

This command is essentially a composition of the `disnixos-deploy-network' command (which deploys the infrastructure) and `disnix-env' (which deploys the services of which the system is composed).

Optionally this command can use `nixops' instead of `disnixos-deploy-network', by adding the --use-nixops command-line option or by setting DISNIXOS_USE_NIXOPS environment variable.

OPTIONS

-s, --services=services_nix  Services Nix expression which describes all components of the distributed system
-n, --network=network_nix Network Nix expression which declares a NixOS configuration for each machine in the network
-d, --distribution=distribution_nix Distribution Nix expression which maps services to machines in the network
--switch-to-generation=NUM Switches to a specific profile generation
--rollback Switches back to the previously deployed configuration
--target-property=PROP The target property of an infrastructure model, that specifies how to
--deploy-state Indicates whether to globally deploy state (disabled by default)
-p, --profile=PROFILE Name of the profile that is used for this system. Defaults to: default
-m, --max-concurrent-transfers=NUM Maximum amount of concurrent closure transfers. Defaults to: 2
--build-on-targets Build the services on the target machines in the network instead of managing the build by the coordinator
--coordinator-profile-path=PATH Path where to store the coordinator profile generations
--no-upgrade Do not perform an upgrade, but activate all services of the new configuration
--no-lock Do not attempt to acquire and release any locks
--no-coordinator-profile Specifies that the coordinator profile should not be updated
--no-target-profiles Specifies that the target profiles should not be updated
--no-migration Do not migrate the state of services from one machine to another, even if they have been annotated as such
--no-delete-state Do not remove the state of deactivated services
--show-trace Shows a trace of the output
--use-nixops Use NixOps instead of Disnix’s deployment facilities
--disable-disnix Do not enable the Disnix service on the target machines by default
--no-infra-deployment Only deploy the services, not the infrastructure
-h, --help Shows the usage of this command
-v, --version Shows the version of this command

ENVIRONMENT

DISNIX_CLIENT_INTERFACE Sets the client interface (defaults to: disnix-ssh-client)
DISNIX_TARGET_PROPERTY Sets the target property of an infrastructure model, that specifies how to connect to the remote Disnix interface. (Defaults to: hostname)
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).
DISNIX_DEPLOY_STATE If set to 1 it also deploys the state of all components. (defaults to: 0)
DISNIX_NO_DELETE_STATE If set to 1 it does not delete the obsolete state after upgrading. (defaults to: 0)
DYSONONIASTATEDEIR Specifies where the snapshots must be stored on the coordinator machine (defaults to: /var/state/dysnomia)
DISNIXOS_USE_NIXOPS When set to 1, it specifies that NixOps is used to take care of the infrastructure deployment, so that it properly interprets NixOps configurations and uses NixOps tooling instead of Disnix’s deployment facilities.
A.1.2  disnixos-vm-env

disnixos-vm-env — Deploys and runs a distributed system in a network of VMs spawned by the NixOS test driver

Synopsis

disnixos-vm-env -s services_nix -n network_nix [-n network_nix...] -d distribution_nix [OPTION]

DESCRIPTION

The command `disnixos-vm-env` is used to generate and launch a network of virtual machines, closely matching the machine configurations defined in the network model.

This command requires three Nix expressions as input parameters; A services model capturing the components of a distributed system and its inter-dependencies; A network model capturing the NixOS configurations of machines in the network and its properties and a distribution model which maps services to machines.

By invoking this command, it will first build a network of virtual machines, closely matching the NixOS configurations in the network model. Then it will build all the services that are defined in the distribution model from source code including all its dependencies. Finally, it launches the virtual machines and deploys the system into the virtual network.

OPTIONS

- `-s, --services=services_nix` Services Nix expression which describes all components of the distributed system
- `-n, --network=network_nix` Network Nix expression which declares a NixOS configuration for each machine in the network
- `-d, --distribution=distribution_nix` Distribution Nix expression which maps services to machines in the network
- `-deploy-state` Indicates whether to globally deploy state (disabled by default)
- `-show-trace` Shows a trace of the output
- `-h, --help` Shows the usage of this command
- `-v, --version` Shows the version of this command

ENVIRONMENT

`DISNIX_DEPLOY_STATE` If set to 1 it also deploys the state of all components. (defaults to: 0)

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

COPYRIGHT

Copyright (C) 2008-2016 Sander van der Burg
A.2 Utilities

A.2.1 disnix-backdoor-client

disinix-backdoor-client — Provides access to the disnix-service through a socket with public access

Synopsis

disinix-backdoor-client --target socat_address operation [OPTION] [paths]

DESCRIPTION

The command `disnix-backdoor-client' provides remote access to a `disnix-service' instance running on a machine in the network by using an insecure socket with public access. This tool is designed for use with `disnixos-vm-env' to automatically deploy a system in a network of virtual machines.

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

--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 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 be transferred from the remote machine if needed

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
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/state/dysnomia`)

COPYRIGHT

Copyright (C) 2008-2016 Sander van der Burg

A.2.2 disnix-nixops-client

disnix-nixops-client — Provides access to the disnix-service through NixOps SSH interface

Synopsis

disnix-nixops-client --target hostname operation [OPTION] [paths]

DESCRIPTION

The command `disnix-nixops-client` provides remote access to a `disnix-service` instance running on a machine in the network using NixOps. This allows the user to perform remote deployment operations on a target machines that are instantiated and deployed in some IaaS environment.

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

`-s, --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
--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 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

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

ENVIRONMENT

NIXOPS_OPTS  Additional properties which are passed to the `nixops` command

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/state/dysnomia)

COPYRIGHT

Copyright (C) 2008-2016 Sander van der Burg

A.2.3  disnixos-manifest

disnixos-manifest — Generate a manifest file from DisnixOS expressions

Synopsis

disnixos-manifest -s services_nix -n network_nix [-n network2_nix...] -d distribution_nix [OPTION]

DESCRIPTION

The command `disnixos-manifest` generates a manifest file from a service, network and distribution Nix expression, which can be used for the distribution of services to machines in the network and for the activation of services on target machines in the right order.

Since the manifest file contains Nix store paths of every service, a side effect of running this command is that all the services that have to be activated are automatically built from source and stored in the Nix store of the coordinator machine.

Most users and developers don’t need to use this command directly. The command `disnixos-env` performs generation of a manifest automatically. It is mostly used for debugging purposes or to perform certain tasks manually.

OPTIONS

-s, --services=services_nix  Services Nix expression which describes all components of the distributed system
-n, --network=network_nix  Network Nix expression which declares a NixOS configuration for each machine in the network
-d, --distribution=distribution_nix  Distribution Nix expression which maps services to machines in the network
--target-property=PROP The target property of an infrastructure model, that specifies how to connect to the remote Disnix interface. (Defaults to hostname)

--interface=INTERFACE Path to executable that communicates with a Disnix interface. Defaults to: disnix-ssh-client

--deploy-state Indicates whether to globally deploy state (disabled by default)

--no-out-link Do not create a 'result' symlink

--show-trace Shows a trace of the output

--vm Configures a manifest used for the NixOS test driver

--use-nixops Use NixOps instead of Disnix’s deployment facilities

--disable-disnix Do not enable the Disnix service on the target machines by default

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

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

ENVIRONMENT

DISNIX_CLIENT_INTERFACE Sets the client interface (defaults to: disnix-ssh-client)

DISNIX_TARGET_PROPERTY Sets the target property of an infrastructure model, that specifies how to connect to the remote Disnix interface. (Defaults to: hostname)

DISNIX_DEPLOY_STATE If set to 1 it also deploys the state of all components. (defaults to: 0)

DISNIXOS_USE_NIXOPS When set to 1, it specifies that NixOps is used to take care of the infrastructure deployment, so that it properly interprets NixOps configurations and uses NixOps tooling instead of Disnix’s deployment facilities.

COPYRIGHT

Copyright (C) 2008-2016 Sander van der Burg

A.2.4 disnixos-instantiate

disnixos-instantiate — Instantiate a distributed derivation from DisnixOS expressions

Synopsis

disnixos-instantiate -s services_nix -n network_nix [-n network2_nix...] -d distribution_nix [OPTION]
disnixos-instantiate -n network_nix [-n network2_nix...] [OPTION]

DESCRIPTION

The command 'disnixos-instantiate' generates a distributed derivation file from a service, network and distribution Nix expression, which can be used to build the services on the target machines from source code by using the 'disnix-build' command.

If only network expressions are given without a services or distribution expression, then an infrastructure distributed derivation file is generated.

Most users and developers don’t need to use this command directly. The command 'disnixos-env' performs instantiation of a distributed derivation automatically. It is mostly used for debugging purposes or to perform certain tasks manually.
OPTIONS

-s, --services=services_nix  Services Nix expression which describes all components of the distributed system

-n, --network=network_nix  Network Nix expression which declares a NixOS configuration for each machine in the network

-d, --distribution=distribution_nix  Distribution Nix expression which maps services to machines in the network

--target-property=PROP  The target property of an infrastructure model, that specifies how to connect to the remote
  Disnix interface. (Defaults to: hostname)

--interface=INTERFACE  Path to executable that communicates with a Disnix interface. Defaults to: disnix-ssh-client

--no-out-link  Do not create a ‘result’ symlink

--show-trace  Shows a trace of the output

--vm  Configures a manifest used for the NixOS test driver

--use-nixops  Use NixOps instead of Disnix’s deployment facilities

--disable-disnix  Do not enable the Disnix service on the target machines by default

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

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

ENVIRONMENT

DISNIX_CLIENT_INTERFACE  Sets the client interface (defaults to: disnix-ssh-client)

DISNIX_TARGET_PROPERTY  Sets the target property of an infrastructure model, that specifies how to connect to the remote
  Disnix interface. (Defaults to: hostname)

DISNIXOS_USE_NIXOPS  When set to 1, it specifies that NixOps is used to take care of the infrastructure deployment, so that
  it properly interprets NixOps configurations and uses NixOps tooling instead of Disnix’s deployment facilities.

COPYRIGHT

Copyright (C) 2008-2016 Sander van der Burg

A.2.5  disnixos-deploy-network

disnixos-deploy-network — Installs or updates the infrastructure of a distributed system

Synopsis

disnixos-deploy-network [OPTION] network_expr...

DESCRIPTION

The command `disnixos-deploy-network` is used to install or upgrade the NixOS configurations of machines in a network through
the Disnix service.
OPTIONS

--target-property=PROP  The target property of an infrastructure model, that specifies how to connect to the remote Disnix interface. (Defaults to hostname)

--interface=INTERFACE  Process that communicates with the remote disnix service. Defaults to: disnix-ssh-client

-p, --profile=PROFILE  Name of the profile that is used for this system. Defaults to: default

-m, --max-concurrent-transfers=NUM  Maximum amount of concurrent closure transfers. Defaults to: 2

--disable-disnix  Do not enable the Disnix service on the target machines by default

--build-on-targets  Build the services on the target machines in the network instead of managing the build by the coordinator

--show-trace  Shows a trace of the output

-h, --help  Shows the usage of this command

-v, --version  Shows the version of this command

ENVIRONMENT

DISNIX_CLIENT_INTERFACE  Sets the client interface (which defaults to `disnix-ssh-client')

DISNIX_TARGET_PROPERTY  Specifies which property in the infrastructure Nix expression specifies how to connect to the remote interface (defaults to: hostname)

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).

COPYRIGHT

Copyright (C) 2008-2016 Sander van der Burg

A.2.6  disnixos-collect-garbage

disnixos-collect-garbage — Delete garbage from a network of machines

Synopsis

disnixos-collect-garbage [OPTION] network_nix [network2_nix...]

DESCRIPTION

The command ‘disnixos-collect-garbage’ collects all garbage from all the machines defined in a network Nix expression and optionally removes all the older profiles.
OPTIONS

-d, --delete-old  Removes all the old Nix profile generations

--interface=INTERFACE  Path to executable that communicates with a Disnix interface. Defaults to `disnix-ssh-client`

--target-property=PROP  The target property of an infrastructure model, that specifies how to connect to the remote Disnix interface. (Defaults to: hostname)

--use-nixops  Use NixOps instead of Disnix for infrastructure deployment

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

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

ENVIRONMENT

DISNIX_CLIENT_INTERFACE  Sets the client interface (which defaults to `disnix-ssh-client’)

DISNIX_TARGET_PROPERTY  Specifies which property in the infrastructure Nix expression specifies how to connect to the remote Disnix

DISNIXOS_USE_NIXOPS  When set to 1, it specifies that NixOps is used to take care of the infrastructure deployment, so that it properly interprets NixOps configurations and uses NixOps tooling instead of Disnix’s deployment facilities.

COPYRIGHT

Copyright (C) 2008-2016 Sander van der Burg

A.2.7 disnixos-clean-snapshots

disnixos-clean-snapshots — Delete older generation of snapshots from a network of machines

Synopsis

disnixos-clean-snapshots [OPTION] network_nix [network2_nix...]

DESCRIPTION

The command `disnixos-clean-snapshots’ removes all older snapshot generations stored on the machines in the network.

OPTIONS

--interface=INTERFACE  Path to executable that communicates with a Disnix interface. Defaults to `disnix-ssh-client’

--target-property=PROP  The target property of an infrastructure model, that specifies how to connect to the remote Disnix

--keep=NUM  Amount of snapshot generations to keep. Defaults to: 1

--use-nixops  Use NixOps instead of Disnix for infrastructure deployment

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

-v, --version  Shows the version of this command to the user
ENVIRONMENT

DISNIX_CLIENT_INTERFACE  Sets the client interface (which defaults to `disnix-ssh-client')

DISNIX_TARGET_PROPERTY   Specifies which property in the infrastructure Nix expression specifies how to connect to the remote interface (defaults to: hostname)

DISNIXOS_USE_NIXOPS    When set to 1, it specifies that NixOps is used to take care of the infrastructure deployment, so that it properly interprets NixOps configurations and uses NixOps tooling instead of Disnix’s deployment facilities.

COPYRIGHT

Copyright (C) 2008-2016 Sander van der Burg

A.2.8 disnixos-query

disnixos-query — Query the installed services from machines

Synopsis

disnixos-query [OPTION] network_nix [network2_nix...]

DESCRIPTION

The command `disnixos-query’ collects and displays all the installed services from the machines defined in a given infrastructure model.

OPTIONS

-p, --profile=PROFILE Name of the profile in which the services are registered. Defaults to: default

--interface=INTERFACE Path to executable that communicates with a Disnix interface. Defaults to `disnix-ssh-client’

--target-property=PROP The target property of an infrastructure model, that specifies how to connect to the remote Disnix interface. (Defaults to hostname)

--use-nixops Use NixOps instead of Disnix for infrastructure deployment

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

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

ENVIRONMENT

DISNIX_CLIENT_INTERFACE  Sets the client interface (which defaults to `disnix-ssh-client’)

DISNIX_TARGET_PROPERTY   Specifies which property in the infrastructure Nix expression specifies how to connect to the remote interface (defaults to: hostname)

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)
A.2.9 disnixos-geninfra

dihnixos-geninfra — Generate an infrastructure model from network models

Synopsis

dihnixos-geninfra [OPTION] network_nix [network2_nix...]

DESCRIPTION

The command `disnixos-geninfra' generates an infrastructure model from NixOS network models, so that properties of the machines can be used by the basic Disnix toolset.

OPTIONS

--no-out-link Do not create a ’result’ symlink
--use-backdoor Indicates that the backdoor must be enabled so that the VMs can be accessed through a socket
--use-vm-testing Indicates that we generate virtual machines, so that the physical characteristics are automatically added
--use-nixops Use NixOps instead of Disnix’s deployment facilities
--show-trace Shows a trace of the output
-h, --help Shows the usage of this command
-v, --version Shows the version of this command

ENVIRONMENT

DISNIXOS_USE_NIXOPS When set to 1, it specifies that NixOps is used to take care of the infrastructure deployment, so that it properly interprets NixOps configurations and uses NixOps tooling instead of Disnix’s deployment facilities.

COPYRIGHT

Copyright (C) 2008-2016 Sander van der Burg

A.2.10 disnixos-gentests

dihnixos-gentests — Generate initialialization steps of the test suite

Synopsis

dihnixos-gentests [OPTION] -n network_expr [-n network_expr2...] MANIFEST
DESCRIPTION

The command `disnixos-gentests` generates a part of the testsuite that starts the backdoors on the machines in the network and deploys the services in the network of machines, by invoking `disnix-activate`.

OPTIONS

- `-n`, `--network=network_nix` Network Nix expression which declares a NixOS configuration for each machine in the network
- `--no-out-link` Do not create a `result` symlink
- `--show-trace` Shows a trace of the output
- `--use-nixops` Use NixOps instead of Disnix’s deployment facilities
- `-h`, `--help` Shows the usage of this command

ENVIRONMENT

`DISNIXOS_USE_NIXOPS` When set to 1, it specifies that NixOps is used to take care of the infrastructure deployment, so that it properly interprets NixOps configurations and uses NixOps tooling instead of Disnix’s deployment facilities.

COPYRIGHT

Copyright (C) 2008-2016 Sander van der Burg