

VERSION 2.6
DATE Mai 2011
Copyright SysFera

Table des matières

1	Deploying a DIET platform	5
1.1	GoDIET	5
1.1.1	Installing GoDIET	5
1.1.2	Running Diet	7
1.1.3	Godiet shell	10

Chapitre 1

Deploying a DIET platform

Deployment is the process of launching a DIET platform including agents and servers. For DIET, this process includes writing configuration files for each element and launching the elements in the correct hierarchical order. There are three primary ways to deploy DIET.

Launching **by hand** is a reasonable way to deploy DIET for small-scale testing and verification. This chapter explains the necessary services, how to write DIET configuration files, and in what order DIET elements should be launched. See Section ?? for details.

Using GODIET , a Java-based tool for automatic DIET deployment that manages the creation of configuration file, the staging of files, the launch of elements, the monitoring and reporting upon successful launch, and the cleanup process when the DIET deployment is no longer needed. See Section 1.1 for details.

Writing your own scripts is a surprisingly popular approach. This approach often looks easy initially, but can sometimes take much, much longer than expected, as there are many complexities to manage. Learn how to use GODIET– it will save you time !

1.1 GODIET

GODIET is a cross-platform tool that helps you automate ad-hoc deployment and management procedures for a DIET platform. It manages the creation of configuration files, the staging of files, the launch of software components, the monitoring and the reporting. GODIET is extremely useful for large deployments on a complex physical infrastructure. The main features are :

- Complete running application customization. Infrastructure agnostic approach ;
- command-line interface ;
- distributed command execution via SSH ;
- real-time monitoring of applications' states ;
- handling of complex physical infrastructure, with firewalls and multiple local- and wide-area networks.

1.1.1 Installing GODIET

Requirement

The following operating systems are known to support GoDiet :

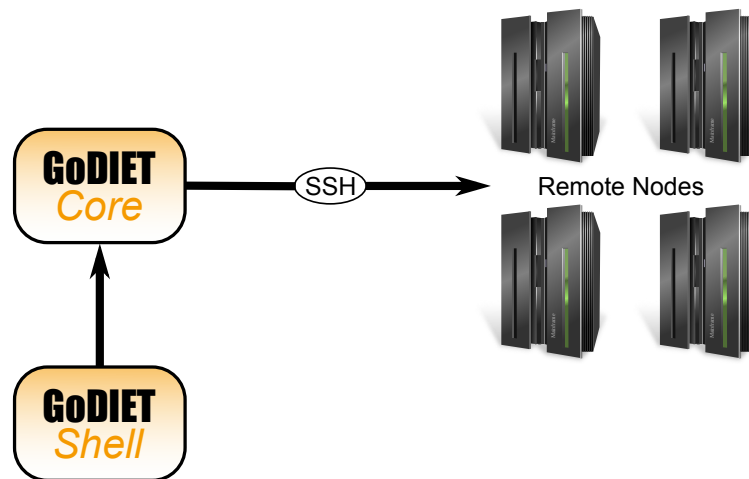


FIGURE 1.1 – Design principle of GODIET.

- Linux : the most recent distributions are likely to work ;
- Mac OS X 10.4 or later.

You need to have Sun Java 6 or OpenJDK6 installed. Download GODIET core and shell on the project's website¹. Check that the run.sh script works : it should display something similar to figure 1.3.

Quickstart

The steps for people in a hurry :

- running the server : execute run-server.sh
- running the client : execute run-client.sh
- create your GODIET configuration file in `${HOME}/.godiet/configuration.xml` 1.1.2. It contains information for remote connection ;
- create your infrastructure description file 1.1.2. It describes your compute nodes, gateways and storage nodes ;
- create your DIET platform description file 1.1.2. It describes all the DIET elements (agents and SeDs) that you want to deploy and manage ;
- run the GODIET shell ??.

And that's it !

GODIET server setup

distribution.layout RMI configuration run-server.sh read configuration.xml

GODIET client setup

distribution.layout RMI configuration run-client.sh

1. <http://graal.ens-lyon.fr/DIET/godiet.html>

1.1.2 Running Diet

Before using GODIET, you need to create three files : one to describe GODIET's configuration, one to describe your infrastructure and one that contains your DIET platform's description. These files use an XML-format and conform to the Configuration.xsd, Infrastructure.xsd and Diet.xsd grammar files (provided with GODIET), respectively. Sample files are provided in the **examples** directory.

Configuration.xml

Ce fichier regroupe les informations indispensables pour que le serveur puisse se connecter et executer les applications sur les machines distantes. This file aggregates information about the local node from where GODIET was launched. It also contains information about user authentication. By default, GODIET looks in the `${HOME}/.godiet/configuration.xml` directory.

The mains elements are :

- **localNode** : the name of the node on which GODIET was launched. This name must be present in the infrastructure description file ;
- **localScratch** : the working directory where GODIET stores its temporary files ;
- **keys** : the paths of your private ssh keys, which are loaded at GODIET's startup. You can give the public key's path, too. GODIET tries to load a file with the same name as the private key and ending with **.pub**. See the **ssh initkeys** command [1.1.3](#) to initialize passwords if your keys are encrypted (i.e. need a passphrase).

General layout of the configuration description (some parts are omitted) :

```
<godiet:configuration localNode="local" schema="Configuration.xsd">

  <localscratch dir="/tmp/scratch_godiet" />
  <user>
    <ssh>
      <key path="/home/.ssh/id_dsa"/>
      <key path="/home/.ssh/admin_cluster2"/>
    </ssh>
  </user>
</godiet:configuration>
```

Infrastructure.xml

GODIET needs to have the description of the infrastructure on which DIET will be running.

You can find the full list of options in the **Infrastructure.xsd** grammar file. You can also look in the **examples** directory.

The fields are :

Domain : aggregates a set of infrastructure elements (nodes, gateways, storage) which are able to communicate with DIET's exchange protocol (i.e., CORBA). Typically, elements separated by a firewall and/or a router must be described in separate domains;

Node : describe a computing node where agents or SeDs will be running. It could be either a physical or virtual machine;

Link : defines a directional link par lequel les communications SSH sont possibles. Il existe deux types de liens

- Entre un domaine et une machine : typiquement des machines derrière un nat (en violet sur la figure).
- Entre deux machines : la source est un routeur.

General layout of the infrastructure description (details are omitted) :

```
<godiet:infrastructure schema="Infrastructure.xsd">
  <domain id="idDomain1"/>
  <domain id="idDomain2"/>
  ..
  <node id="idNode">
    <ssh id="idInterface" domain="idDomainRef" login="login"
      server="ip/hostname" port="2022" />
    <scratch dir="/tmp/scratch_runtime/" />
    <ssh id="idInterface" domain="idDomainRef" login="login"
      server="ip/hostname" />
    <scratch dir="/tmp/scratch_runtime/" />
  </node>

  ..
  <link fromDomain="DomainSysferaLB" to="graal" accessref="graalinterface1"/>
  <link from="graal" to="testbedVM" accessref="testbedVMinterface1" />
  ..
</godiet:infrastructure>
```

Platform.xml

A GoDIET user can describe the desired deployment in an XML file including all the external services needed (e.g., omniNames and LogService). The desired hierarchical organization of agents and servers is expressed directly using the hierarchical organization of XML.

The most important fields are :

- **dietServices** : lists your DIET services, such as omniNames and LogCentral;
- **dietArchitecture** : lists your DIET hierarchy, with all your agents and SeDs.

General layout of the DIET platform description (details are omitted) :

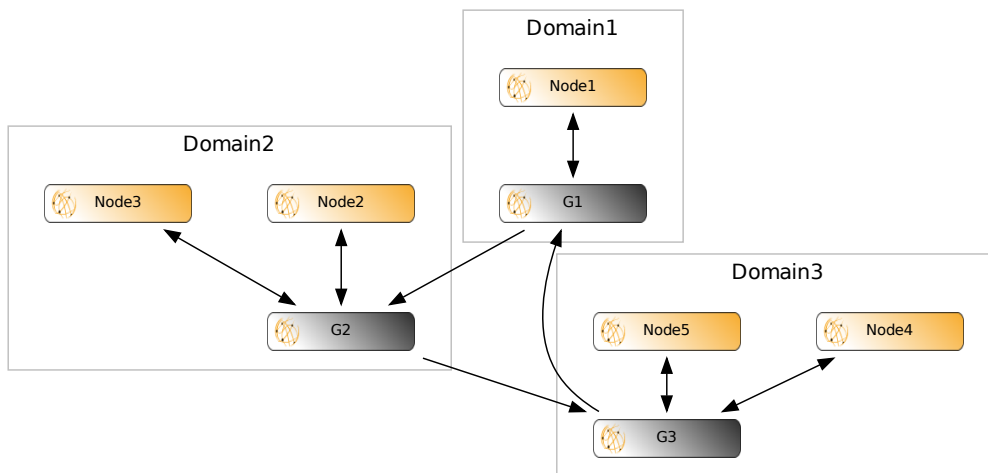


FIGURE 1.2 – Example of the representation of a three-domain infrastructure.

```

<godiet:dietPlatform schema="Diet.xsd">
  <services>

    <omniNames id="omniNames1" domain="DomainId1">
<config server="refIdNode"/>
    </omniNames>
    <omniNames id="omniNames1" domain="DomainId2">
<config server="refIdNode"/>
    </omniNames>
    <!-- Inter-connect Domain1 and Domain2 -->
    <forwarders>
      <client id="client1" type="CLIENT">
        <config server="refIdNode" />
      </client>
      <server id="server1" type="SERVER">
        <config server="refIdNode" />
      </server>
    </forwarders>
  </services>

  <hierarchy>
    <ma id="ma1">
      <config server="refIdNode"/>
    </ma>
    <la id="la1" server="refIdNode">
      <config server="refIdNode"/>
    </la>
    ..
  </hierarchy>
</godiet:dietPlatform>

```

```
        <sed id="sed1">
    <config server="refIdNode"/>
    <file id="sed1_config">
        <template name="sed_template.config" />
    </file>
    <binary name="dmat_manips_server">
        <commandLine>
            <parameter string="${this.configurationFiles.sed1_config.absolutePath}" />
            <parameter string="T" />
        </commandLine>
    </binary>
</sed>
    ..
</la>
    ...
</ma>
    <ma id="ma2" server="refIdNode3">..</ma>
</hierarchy>

</godiet:dietPlatform>
```

1.1.3 Godiet shell

GoDIET shell is the interface to manage and monitor your DIET platform. It includes features such as syntax highlighting, command completion and command history. To use it, simply execute the script `run.sh`. You will obtain a prompt like in figure 1.3

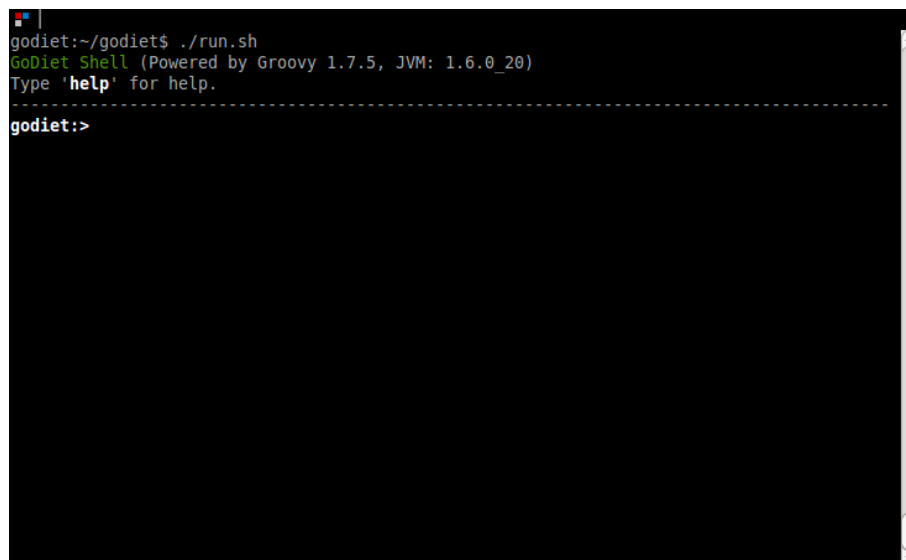
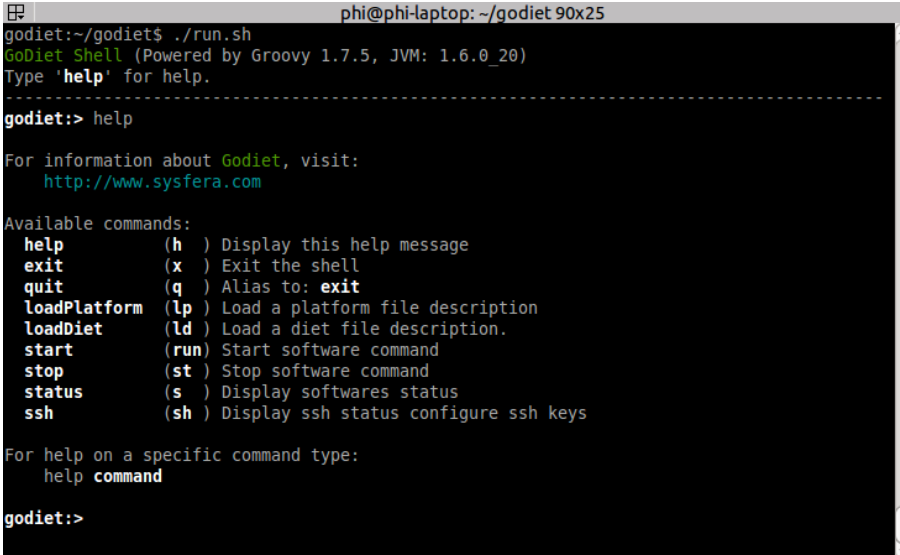


FIGURE 1.3 – GoDIET shell prompt.

The help command

Displays the help message. By default, lists all the commands and short information about them. It gives contextual help when followed by a command name.

- **command** : gives help on the command.



```
phi@phi-laptop: ~/godiet 90x25
godiet:~/godiet$ ./run.sh
GoDiet Shell (Powered by Groovy 1.7.5, JVM: 1.6.0_20)
Type 'help' for help.
-----
godiet:> help

For information about Godiet, visit:
  http://www.sysfera.com

Available commands:
  help      (h ) Display this help message
  exit      (x ) Exit the shell
  quit      (q ) Alias to: exit
  loadPlatform (lp ) Load a platform file description
  loadDiet   (ld ) Load a diet file description.
  start      (run) Start software command
  stop       (st ) Stop software command
  status     (s ) Display softwares status
  ssh        (sh ) Display ssh status configure ssh keys

For help on a specific command type:
  help command

godiet:>
```

FIGURE 1.4 – The help command.

The ssh command

The **ssh** command takes the following arguments :

- **initpasswords** : to initialize the SSH key's passphrase loaded from the configuration file ;
- **addkey** : to register a new key ;
- **modifykey n** : to modify a key that has already been registered ;
- **status** : to display the key's status. Could be PASSWORDNOTSET, PRIVATEKEYERROR, PUBKEYERROR or LOADED.

The loadInfrastructure command

Loads an infrastructure description file [1.1.2](#).

- **filename** : loads an infrastructure from the specified XML file.

The loadDiet command

Loads the DIET platform description [1.1.2](#). When the loading is complete, GODIET automatically computes and creates all the DIET forwarders needed. These DIET forwarders are mandatory when your platform is deployed on several domains. For more details, please refer to the DIET documentation.

- **filename** : loads a DIET platform from the specified XML file.

The start & stop commands

The **start** and **stop** commands both take the same arguments :

- **software** *name* : starts/stops software *name*.
- **services** : starts/stops all services ;
- **agents** : starts/stops all agents ;
- **sed**s : starts/stops all sed
s ;
- **all** : starts/stops all software components ;

The status command

The command displays the status of the DIET platform’s software components. It takes the following arguments :

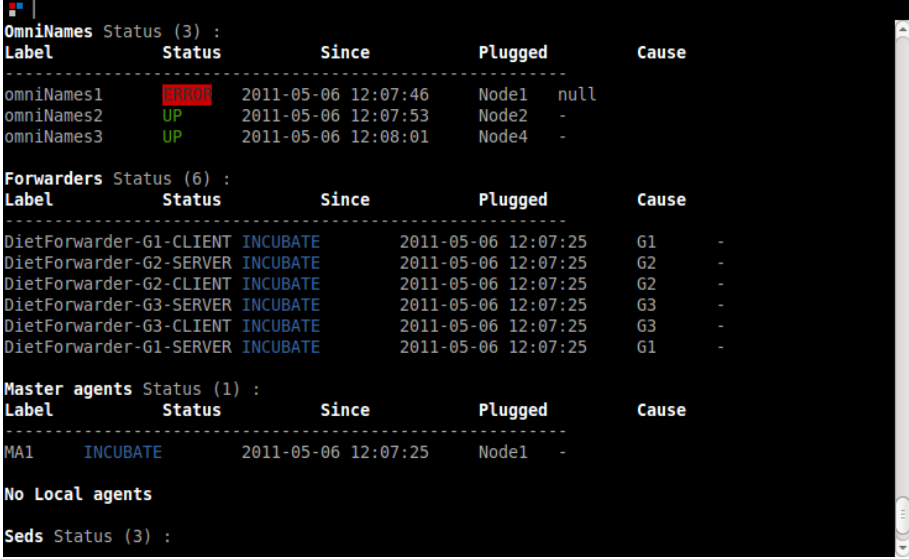
- **ma** : displays the status of the master agents ;
- **la** : displays the status of the local agents ;
- **sed**s : displays the status of the server daemons ;
- **all** : displays the status of all the managed software components.

The **status** command displays the software components managed by GODIET as a table. Figure 1.5 shows an output exemple of this command’s execution. The information is :

- **Label** : the name of the component, as described in the DIET platform description ;
- **Status** : the status of the component ;
- **Since** : the date since when the component has been in this status ;
- **Plugged** : the infrastructure resource it is or will be running on (depending on its state) ;
- **Cause** : an information message, in case of error.

The possible states of a component are :

- **Incubate** : the component is correctly loaded in GODIET ;
- **Ready** : the configuration files are created in the local scratch directory and the component is ready to start ;
- **Up** : the component is running. Its state is periodically checked ;
- **Down** : the component is down. Typically, if the GODIET user has used the **stop** command on this component ;
- **Error** : appears when the component is not running (unable to launch or crashed) or if it is unreachable. The cause is displayed in the Cause row of the information table.



```

OmniNames Status (3) :
Label      Status      Since      Plugged      Cause
-----
omniNames1 ERROR      2011-05-06 12:07:46 Node1 null
omniNames2 UP        2011-05-06 12:07:53 Node2 -
omniNames3 UP        2011-05-06 12:08:01 Node4 -

Forwarders Status (6) :
Label      Status      Since      Plugged      Cause
-----
DietForwarder-G1-CLIENT INCUBATE      2011-05-06 12:07:25 G1 -
DietForwarder-G2-SERVER INCUBATE      2011-05-06 12:07:25 G2 -
DietForwarder-G2-CLIENT INCUBATE      2011-05-06 12:07:25 G2 -
DietForwarder-G3-SERVER INCUBATE      2011-05-06 12:07:25 G3 -
DietForwarder-G3-CLIENT INCUBATE      2011-05-06 12:07:25 G3 -
DietForwarder-G1-SERVER INCUBATE      2011-05-06 12:07:25 G1 -

Master agents Status (1) :
Label      Status      Since      Plugged      Cause
-----
MA1        INCUBATE      2011-05-06 12:07:25 Node1 -

No Local agents

Seds Status (3) :

```

FIGURE 1.5 – The status command.