Università di Modena e Reggio Emilia

Zcluster: a simple yet scalable active-passive failover and monitoring system developed on FreeBSD

Candidate:
Andrea Corsini

Supervisor:
Nicola Bicocchi

Co-Supervisor:
Andrea Valsania

December 2017

Acknowledge

I wish to thank my supervisor, Nicola Bicocchi, for his guidance and patience during this last year of work. It has been a long trip into a branch of computer science, full of difficult things to explore and learn for me, but professor Bicocchi backed me up along the entire journey, inspiring me for overcoming the most complicated phase of this project.

I would also like to express my sincere gratitude to Andrea Valsania, who brought me to the discover of new systems and taught me plenty of technical details about them. Despite his countless appointments and his enormous mole of work, he had always found an hour for helping me to solve my numerous troubles.

Table of contexts

List of acronyms

ARP:	Address Resolution Protocol
BSD:	Berkeley Software Distribution
CARP:	Common Address Redundancy Protocol
CDB:	Command Descriptor Block
CSRG:	Computer Science Research Group
CSS:	Cascading Style Sheets
DBMS:	DataBase Management System
DHCP:	Dynamic Host Configuration Protocol
HTML:	HyperText Markup Language
IPC:	Inter-Process Communication
iSCSI:	Internet Small Computer Systems Interface
IT:	Information Technology
JDBC:	Java DataBase Connectivity
JSP:	JavaServer Pages
NFS:	Network File System
OOP:	Object Oriented Programming
PPP:	Point-to-Point Protocol
RAID:	Redundant Array of Independent Disks
SAN:	Storage Area Network
SRS:	Software Requirement Specification
SSH:	Secure Shell
VDEV:	Virtual Device
VHID:	Virtual Host Identifier
ZFS:	Zettabyte File System
ZI:	Zcluster Inspector
ZVS:	Zcluster Virtual Server

Chapter 1

Introduction

The wide majority of technologies were born to sustain all the business activities and, the Information Technology infrastructure discipline, it is surely not an exception.
The IT discipline may be defined as a set of physical devices and software applications that are required to support enterprise processes, decisions making and competitive strategies. Furthermore, the services a firm is capable of providing to its customers, supplier and employees are a direct consequence of its IT infrastructure, thus, nowadays this discipline must be considered as one of the keys for a successful firm.
A component of an existing IT infrastructure, developed in collaboration with Andrea Valsania of Valsania Information Technology Consulting, is zcluster. This company has the purpose of providing dedicated IT system, in a marketplace polluted by severe lack of competences and full of unreliable commercial solutions. Systems as zcluster are the example of Valsania's IT products, projected for meeting the customers requirements along with the efficiency and reliability they really deserve.
Zcluster need was born from a local company, which was unsatisfied of the availability level of its infrastructure and was then seeking for a more efficient and cost effective solution. Even if there were various alternatives on the market, relaying on complex and adaptive softwares, no ones were really suitable for such company.
Thus, a new project started with the leading idea of a light and low-cost system, capable of responding to the market deficiencies, levering exclusively on a set of native resources existing on the open-source OS, FreeBSD.
The result of this creation process is precisely zcluster, a composition of several shell scripts which have as primary goal of suppling an availability service in a low complexity environment.
The IT complexity is a crucial point indeed, as Andrea Valsania specifies in the introduction of "zcluster: simple clustering for FreeBSD on ZFS", if an OS-level high availability solution is more complex than the service we need to make "more reliable", the solution misses its goal.
In the end, the aim of this work is to discover and explain the technical architecture behind zcluster, focusing on the most characterizing details, and to understand how such kind of solution may be improved. Thus, this document is organized in two chapters: the first one, called Zcluster, explores zcluster's architecture, instead, the second one describes the plug-in application created to improve this IT component.

Chapter 2

Zcluster

This chapter presents the whole architecture of zcluster and its basic commands, by enlightening the principal resources building the system up. The chapter also gives a brief introduction to FreeBSD and it explains the relationships between the OS and the IT functionalities, motivating all the various choices.

2.1 Zcluster goals

Basically, zcluster is an arrangement of nodes and storage disks that cooperates to keep always available the services of a firm, in as low as possible complexity context. This is simply done by relying on the characterizing "active-passive" mechanism of zcluster, that automatically moves workloads from a master (or running) node to a failover (or standby) node, just when the following kind of failures are detected:

1. the running node looses the network connectivity.
2. the running node unexpectedly crashes.
3. the shared storage becomes unavailable on the running node.

However, in other situations, i.e. the running node both shuts down or restarts gracefully or again when someone wants to manually move workloads, the system must be able to invoke a failover to the standby node, making a consistent behavior for all the real-life situations.
It is also important to understand that only in these listed cases, zcluster have to move the workloads among its servers, and neither once the crashed node will be again up a failback is wished. This design choice is due to the fact that an entire automated failback-failover structure, implying an automatic failback when the node is up again, would have increased the solution complexity with no remarkable benefits, except being in contrast with the project guidelines.

2.2 FreeBSD overview

From an implementation point of view, zcluster has been developed in shell scripting for FreeBSD machines and nowadays, it is enclosed in a Valsania's port for a fast and simple installation. Since zcluster relies on FreeBSD, a brief introduction of this OS is yet necessary.
FreeBSD operative system is based on the 4.4BSD-Lite release from Computer Systems Research Group at the University of California, Berkeley. FreeBSD Project has spent thousands of hours improving such system for maximum performance and reliability in real-life load situations. Furthermore, the BSD OS benefits significantly from plenty of high quality applications, developed by research centres and universities around the world, as well as a large online documentation. This makes FreeBSD one of the most suitable OS for developing new application like zcluster.
The following list presents some FreeBSD futures necessary to accomplish the zcluster structure.

- **File System structure:** One of the major advantages of FreeBSD is its lightness and cleanliness. The System startup is controlled by rc scripts. At the boot /etc/rc reads the /etc/rc.conf and /etc/defaults/rc.conf to determine which services have to be loaded. The specified services in these files are then started by running the corresponding initialization scripts, located in /etc/rc.d/ and /usr/local/etc/rc.d/. The scripts found in /etc/rc.d/ are for applications part of the "base" system, instead, scripts in /usr/local/etc/rc.d/ are for user-installed applications, such as Apache or zcluster. Since FreeBSD is developed as a complete OS, user-installed applications are not considered to be part of the "base" system. Thus, in order to keep such applications, installed using packages or ports, separate from the "base" system, they are all placed under /usr/local/. Therefore, user-installed binaries reside in /usr/local/bin/, configuration files are in /usr/local/etc/, and so on. Finally, both base services and user services are enabled by adding an entry in /etc/rc.conf, an example of such entries can be found in the figure 2.4.2.

- **Packages and Ports:** FreeBSD provides two complementary technologies for installing third-party software: the FreeBSD Ports collection, for installing from source, and Packages, for installing from pre-built binaries. Either method may be used to install software from local media or from the network. A FreeBSD port is a collection of files designed to automate the downloading, extracting, patching, compiling, and installing applications from source code. On the other hand, a FreeBSD package englobes pre-compiled copies of all the commands, as well as any configuration files and documentation for establishing an application. The main difference between these two ways of installing softwares is that a package may be manipulated with FreeBSD management commands, i.e. pkg install and pkg

delete, but the building options are fixed. Meanwhile, with a port installation, the source code is directly compiled on the machine, and, the building options are manageable.

- **Jails:** They are a light-weight alternative to virtualization featured by FreeBSD, that allow system administrators to partition a computer into several independent mini-systems, called jails. Processes created under a jail can not access files or resources outside of it, for that reason, compromising a service running in a namespace should not let the attacker to compromise the entire system. Jails are also hierarchical, allowing jails-within-jails. This type of virtualization is not used in zcluster but it is one of the major characteristics of FreeBSD, then some words should be mentioned for the BSD know-how.

- **ZFS:** It is a combined file system and logical volume manager originally designed by Sun Microsystems and installed on FreeBSD since 2007. The advantages brought by ZFS include protection against data corruption, support for a pooled storage and the possibility of taking either snapshots or copy-on-write clones. The FreeBSD combination of the volume manager and the file system makes the OS aware of the underlying disks structure, allowing the creation of many file systems all sharing the pool of available storage. Another advantage, related to the physical disks layout awareness, is that existing file systems can grow automatically when additional disks are added to the pool.

- **TCP/IP:** The networking stack of BSD supports many industry standards like DHCP, NFS, PPP and IPv6. This means, FreeBSD machines may interoperate easily with other systems as well as acting like enterprise server, providing vital functions such as NFS, email services, FTP, routing and firewall security services and also put an organization on the WWW. Furthermore, the BSD networking stack has an historical relevance too. Indeed, the Computer Science Research Group at Barkley chose the TCP/IP as the first network to incorporate into the socket IPC framework, due to the fact that the 4.1BSD implementation was publicly available. That influential choice is one of the main reason for the extremely widespread adoption of such protocol suite nowadays.

2.3 Zcluster structure

The principal elements of zcluster are the nodes. A node is a physical servers with installed the latest version of FreeBSD (actually the release 11.1) and a persistent connection to the **Storage Area Network**. The "active-passive" zcluster functionality is achieved by identifying a tuple of resources (CARP vhids, ZFS pools, and service) and grouping them as an entity, called **Zcluster Virtual Server**. In this way, the services inside the ZVSs may be switched between the system nodes in case of failures or other requests, see section 2.1.
The core technologies behind the high availability of the ZVSs are instead **CARP** and the daemon **devd(8)**. By reacting to CARP status changes, monitored by devd**(8)**, the system is able to start an automated failover procedure when a ZVS goes down on a cluster server.
The roles of CARP and devd(8) in this automated procedure are both explained in the following sections.
Furthermore, every node of zcluster uses the ZFS functionalities to menage the storage disks of the SAN, by mirroring them in pools and partitioning these pools in volumes. The disks within a SAN are collected in storage servers and they are exposed on the network as iSCSI LUNs. It is indeed, via the iSCSI protocol itself , that the Z File System of a node is able to reach and interact with the SAN disks. More information about iSCSI and ZFS pool follows.

2.3.1 Zcluster Virtual Server

A Zcluster Virtual Server is not a physical elements of zcluster, but just a way of visualizing the CARP and ZFS resources with the service they are bound to. CARP and ZFS are indeed used to run the firm′s services, such as MariaDB or Apache, in the steady availability mode.
Every ZVS of zcluster must be running only on one node at time, while the other backup servers are ready to take over if something happens to the master one. This design choice has been taken to avoid split-brain condition, a storage inconsistency originated from a data overlap, due to a wrong communication or a bed synchronization of the nodes acting on the ZVS data set. Split-brain condition would have affected the shared storage in an architecture where both nodes have been working together for the ZVS availability.
On the other hand, a cluster node may run various ZVSs at once, where their number is limited by the cluster servers capacity. In fact, a node must be able to run all the services it supports without problems or limitations of any kind, even when every ZVSs choose that node as master one. In this way, the Zcluster Virtual Servers might be independently started, stopped, failed over and failed back on both the known by servers.
The figure 2.3.1 shows one of the major advantages of ZVS architecture that is to allow multiple workloads at the same time on physical cluster nodes. This characteristic maximizes the hardware investment by removing completely the need of idle servers from zcluster.

The figure 2.3.1 describes an architecture of five ZVS spread over four nodes, and, it also underlines that each ZVS is supported at least by two different nodes.

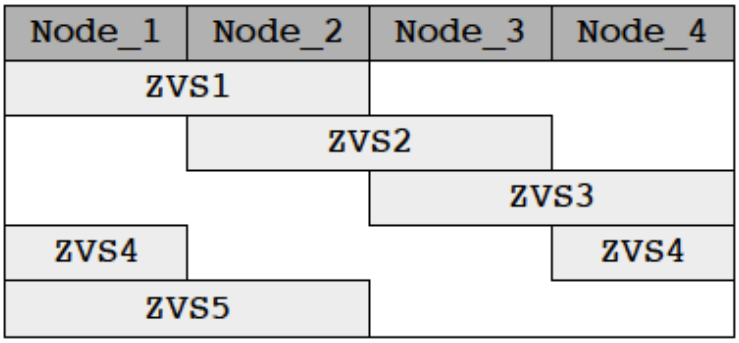


Figure 2.3.1: feasible zcluster architecture of five ZVSs

2.3.2 Common Address Redundancy Protocol

CARP is a redundancy protocol introduced by OpenBSD, that allows multiple hosts on the same network to share an IP address in order to provide high availability for services. A set of hosts is referred to as a redundancy group within such protocol and, each group in a domain owns both a common IP address and a Virtual Host ID. The **VHID** is a unique identifier across the broadcast area and it allows the group members to identify at which redundancy group they belong to, hosts may be part of many different groups indeed. Within the group, one host is designated as the master one and the rest as backup hosts. The master host is the owner of the shared IP address and it responds to any traffic or **ARP** requests directed towards such IP. The master sends regular advertisements on the local network so that the backup hosts know it is still alive. In the event that the master suffers a failure or is taken offline, the backup hosts do not hear the advertisement for a set period of time, thus, one of them will take over the duties of master.
Such protocol is used in zcluster to group the nodes which support a service, by giving them the CARP VHID of its ZVS. In this way, if the running node unexpectedly crashes for whatever reason, the backup server becomes the master of the service.

2.3.3 Daemon devd(8)

The devd(8) utility is a system daemon that is always started at the boot of FreeBSD and it runs in background all the time. For zcluster purposes, devd(8) is configured at the application bootstrap for reacting to the prefixed events listed under /usr/local/etc/devd/. The primary scope of this daemon is to recognize when the CARP mechanism changes the master node. When this happens, devd calls the necessary scripts for either starting a failover or stopping the ZVS, depending on the situation met. Despite CARP has the task to switch the master node inside the redundancy group, it is devd daemon that physically restarts the ZVS

guaranteeing the service availability. Here follows an example of the devd configuration file loaded al the ZVS starting.

```
root@NODE02: ~ # cat /usr/local/etc/devd/zvs_jailer.conf
#
# This file is maintained by the zcluster utility.
# PLEASE DO NOT EDIT IT MANUALLY!

notify 30 {
 match "system"      "CARP";
 match "subsystem"   "1@em0|3@em3";
 match "type"        "(MASTER|BACKUP|INIT)";
 action "/usr/local/libexec/zvs jailer CARP $subsystem $type";
};
notify 30 {
 match "system"      "ZFS";
 match "subsystem"   "ZFS";
 match "type"        "ereport.fs.zfs.io_failure";
 match "pool"        "zsan1";
 action "/usr/local/libexec/zvs jailer ZFS $subsystem $pool";
};
```

Figure 2.3.2: example of devd configuration file

2.3.4 iSCSI and SAN

Internet Small Computer Systems Interface is an IP-based storage networking standard for linking data facilities with networking clients, at the block-device level. iSCSI protocol allows the hosts to send SCSI commands, called **CDB**, to storage devices on remote machines in order to exchange data. iSCSI does it by taking a popular high-performance storage bus and emulating it over a wide range of networks, creating a Storage Area Network.
A **SAN** is primarily used to enhance the storage devices accessible by the system nodes, so that this devices appear to be locally attached SCSI disks. The actors of an iSCSI communication are two. The first actor is the client, called **SCSI initiators,** which initiates a new session for accessing the storage, and the second one is the communication endpoint, the **SCSI target**. From the initiator point of view, the storage available through the iSCSI connection appears as a raw and unformatted disks, known as Logical Unit Numbers, used to identify units on the SAN.

On the other hand, the SCSI target does not initiate any session, but it waits for initiator commands and, when received, it provides the required input/output data.
Zcluster relies on iSCSI to interface the nodes with the storage disks of the SAN, however, the management of such disks is an assignment performed by the ZFS of the cluster server.

2.3.5 ZFS pool

The Z File System nodes is used to ensure that data stored on disks cannot be lost due to physical error or misprocessing by either the hardware or the operative system. This is done by relying on a redundant ZFS storage pool, or **zpool**. Basically, a pool is made up of one or more **vdevs**, virtual devices, which themselves may be composed of a single disk or a group of disks, such as HDDs or SSDs, contacted via the iSCSI protocol. The ZFS exposes the individual disks within the system to the running processes, but it manages the data capacity at the level of vdevs. Again, each vdev may act as an independent unit of a redundant storage volume, organized for example in mirrored or a RaidZ disks configuration, in order to guarantee the data resiliency. ZFS storage pool is a better alternative to other storage solution, like RAID levels, for its unique features and its flexibility, explained in section 2.2.

2.4 Commands

The zcluster command are quite simple due to the fact that, after its first configuration, it does not need any other maintenance. This system logs automatic failover and started/stopped events of any ZVSs in the node file /var/log/messages. Thus, any unexpected events might be found there for further inspections.
Another important thing to remember is that when the first node starts, while the backup one is still down, all the ZVSs are brought online on the first and, they will be running on it until an automatic or manual failover is triggered.

2.4.1 Manual failover/failback

In order to move all or a specific ZVS clustered resource to the backup node, the command to be invoked, on the server where the service is currently running, is:

- *zcluster failover [ZVSNAME]* ← When the command is issued without the ZVS name all the running ZVSs are moved to secondary node.

Instead, to "move back" all or a specific ZVS resource from another node to the backup host, practically the same as before but by issuing the command on the backup server, type the following command on the desired console node:

- *zcluster failback [ZVSNAME]* ← If the command is inserted with the ZVS name, only this ZVS is set running, otherwise all the ZVSs are set running.

2.4.2 ZVSs resources status

The command used to check the ZVSs resources status on a node of the system is the following one:

- *zcluster status [ZVSNAME]* ← Without the name, all the ZVSs status are listed. Since the ZVS name is issued too, only its status is shown.

The expected output of the status command, in the case the [ZVSNAME] is specified, is the tuple, CARP, ZFS and Service composing the Zcluster Virtual Server:

```
root@NODE01:~ # zcluster status mysql
 Status of ZVS "mysql"          ¦ ENABLED
 ---------------- Resources ----------------
 CARP vhid   ¦ 2@em0             ¦ UP
 ZFS pool    ¦ zsan2             ¦ UP
 Service     ¦ mysql-server      ¦ UP
 ------------------------------------------
```

Figure 2.4.1: expected output of a ZVS status command

When the ZVS name is not inserted the output is similar to the previous one, where the name is present, but it will list the entire number of ZVSs, one under the other.
The figure 2.4.1 shows how to insert the command and its related output as well as two other important ZVS details: first is that a ZVS may be ENABLED, when is started, or DISABLED, when is stopped. The second is the resources body structure, that is: resource kind (CARP, ZFS, Service), corresponding resource name and in last position, the resource status (UP or DOWN).

2.4.3 Enable or disable auto failover

The first way, to permanently disable an automatic failover of a ZVS, requires the exclusion of such ZVS from the supported ones on the involved node. This is done by deleting the ZVS entry from the zcluster_zvslist contained in /etc/rc.conf of the server, figure 2.4.2.
The other way to disable the auto failover, without deleting the services from the hosting list, is to set the zcluster_enable variable to “no” in the file /etc/rc.conf too. This file is really important for the node configuration indeed, it outlines many parameters of the node such as the VHIDs, the network IP and the iSCSI habilitation, how can be seen from the image below:

```
hostname="NODE01.zcluster.lab"
ifconfig_em0="inet 192.168.1.111 netmask 255.255.255.0"
ifconfig_em1="inet 192.0.2.101 netmask 255.255.255.0"
ifconfig_em2="inet 10.32.11.217 netmask 255.255.254.0"
defaultrouter="192.168.1.1"
zfs_enable="YES"
vboxguest_enable="YES"
vboxservice_enable="YES"
sshd_enable="YES"

# zcluster pre-requisites
iscsid_enable="YES"
iscsictl_enable="YES"
ifconfig_em0_alias0="inet vhid 1 pass zvs1 alias 192.168.1.200/32"
ifconfig_em0_alias1="inet vhid 2 pass zvs2 alias 192.168.1.210/32"

# zcluster configuration
zcluster_enable="YES"
zcluster_debug="NO"
zcluster_zvslist="www mysql"
```

Figure 2.4.2: rc.conf file of a zcluster node

In some situations could be useful to act directly on the system nodes at runtime, by starting and stopping them. The command for starting and stopping one or all the ZVSs of a node are respectively:

- *zcluster start [ZVSNAME]*
- *zcluster stop [ZVSNAME]*

Chapter 3

Zcluster Inspector

Zcluster Inspector, or ZI, is a plug-in web application created to cooperate with the bigger IT component, zcluster. The ZI rule inside this cooperation is basically to supply a simple management infrastructure aligned with the principles of zcluster. Zcluster Inspector tasks, organized from the primaries to the secondaries, are:

1. Show in real-time the ZVSs status of all zcluster nodes.
2. Give the possibility of changing at runtime the running node of the ZVSs.
3. Keep a monthly history of the ZVSs for further inspections.

In order to make such application many software solutions are been tested during the development phase, with no satisfying results, until reliable and performing resources are been discovered and correctly employed. Java and HTML/JSP are the unmovable pillars behind Zcluster Inspector, each one lays the foundations of half the project. Contrary the implementations of the two halves are the result of several experiments, meaning in future better solutions might be employed for implementing theses two halves.
The logical ZI division is just not the outcome of two different languages, but it is used to split the user world of the application from the zcluster world. The main reason for this separation is that native FreeBSD zcluster must be absolutely free of errors and neither affect by external failures. Since the Java half is divided from the user/HTML half, or Web servlet, it is possible to isolate the more failure-prone user part from the Java base, that interacts directly with the zcluster core. Another important advantage coming from the division is certainly having two independent interfaces, which allow modifications on one half without consequence on the other one. Due to this given structure, ZI also needs a vehicle between its building blocks in order to make it runs, and this carrier is a database. A relational database is the most appropriate way to realize the communication between the Java base and the Web servlet for two causes: the first is that it places an additional safe barrier between the two halves, and, the second cause is it allows asynchronous operations between either sides.

3.1 Java base

The supporting platform of the entire Zcluster Inspector is Java base, a program written in Java code. The choice of Java language is certainly bound to its several features, but among them the one which stands out for its importance, is the Java object-oriented characteristic. OOP allows indeed to make code structures that are suitable to shape the unusual retrieved information of zcluster.
Java base is basically a servlet running on Apache Tomcat v8.5, with the goals of collecting information from the various zcluster nodes and organizing them, after some operations, in tables of a database. Both these actions are extremely important for ZI, indeed without the cluster data there would be nothing to show, therefore, nothing to manage for the final user. This application achieves its goals via a sequence of tasks carried out by different classes, necessary to interface the principal elements of zcluster in the most reliable way.
In particular, these tasks are:

- The creation of a secure communication channel with the system nodes, performed with a SSH connection, capable of establishing a stable and fast shell pipes for exchanging commands and data.
- A refinement of the rough information retrieved with the shell-base channel, realized via an ad-hoc parser that transforms the pulled data into useful objects.
- A storing operations sequence that places the created objects into the appropriate table of the database relying on a JDBC.

Despite Java base seems only a simple process of three actions (SSH-Parser-Database), it hides many complications related to the way it acts. First of all, the procedure of gathering, parsing and storing the data must be repeated after an auto adaptive period of time, that involves an adjustment of the repetition interval on the base of how fast the connection is. Moreover, other complications might arise from the program exceptions which are the result of unexpected internal or external events, i.e. wrong codes flows or mistaken node responds. By combining the frequently cyclic interrogation with the probability of exceptions on the repetition, the implementation complexity of Java base grows very fast, and it gets even worst, if it is considered that the application on its own must be able of managing all these situations, without affecting zcluster.

3.1.1 SSH connection

Secure Shell is a software package that enables safe system administration and file transfers over insecure networks. The SSH protocol uses encryption to secure the connection between a client and a server, in this way user authentication, commands, output, and file transfers are protected against attacks in the network.

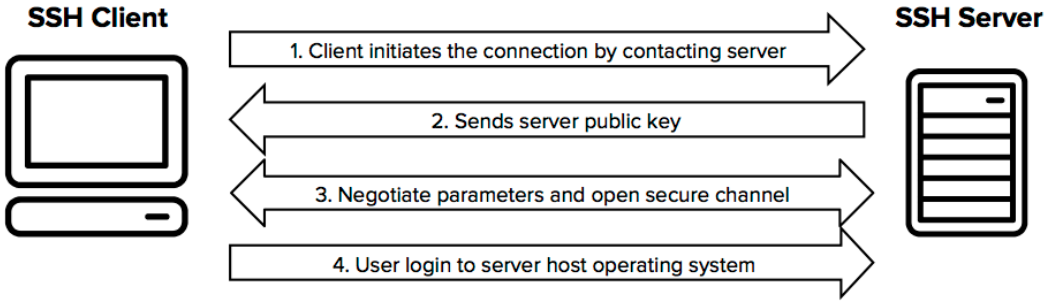


Figure 3.1.1: description of the SSH communication establishment

Java base first task is performed by the SSHConnection class, that exploits the mechanism of public-private key to avoid the insertion of passwords during the SSH login phase. The picture above, figure 3.1.1, describes how the communication takes place and what are the roles in it. The first participant of the communication is the server hosting ZI, illustrated in the picture as SSH client, and the second participant, known as SSH Server, is any cluster nodes of zcluster approached by the connection. A standard communication between the client and a server starts when the first participant asks for a connection on a node. Once the remote request is correctly received, the SSH server replies with its public key, used to identify a private key on the SSH client folder /.ssh/authorized_keys. This private key is then sent to the node for the final login, followed by the command to run on the remote OS, that for the first task of Java base is "zcluster status". If everything goes fine, the output of such command flows through the secure channel back to the SSH client, and finally, it is collected and packed within a map, as a large string identified by the hosting-node name.
In ZI, the SSH protocol is directly used by the environment in which the application runs, through the Java Runtime class. This implementation choice is the outcome of the bad performance and poor reliability experiments did in using SSH, which have led towards the simple Runtime solution.
SSH package is almost present in most modern operative system but in some versions of Window desktop/server OS it is not included by default, therefore, it is necessary a check before the system installation. In addition, the public-private mechanism must be configured, and for doing that, a simple procedure have to be followed. The steps to set up this mechanism are described in details here: https://www.digitalocean.com/community/tutorials/how-to-set-up-ssh-keys--2

3.1.2 Parser

Once the data are correctly stored in a Java map to keep the association node-ZVSs, this structure is passed to the Parser class for being processed and refined. The Parser reads each instance of the map and, by using a matcher with regualr-expressions, it extracts the needed information from the status strings. Every entry of the map is composed of a node name, as the key, and a list of ZVSs status outputs in form of a unique string, show in the figure 3.1.2.

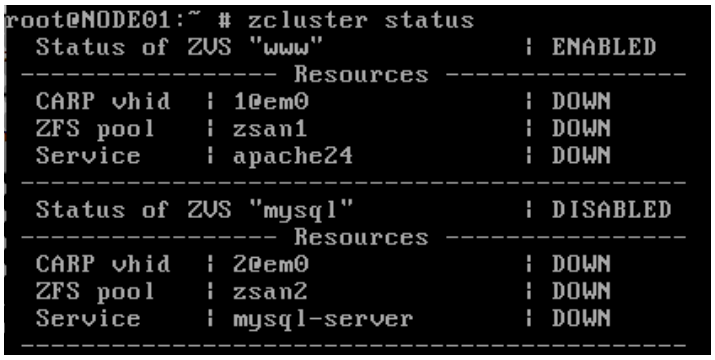


Figure 3.1.2: expected output of a node status command

Due to the presence of these strings in the map, which englobe all the services status per nodes, the parser must be able to recognize every ZVSs output of each node and split them in substrings, before it can extract the needed information. The only way to do both the splitting and extracting operations is indeed via the regular-expressions, contained in the ParserFlag class of Java base. These patterns used by the matcher are essentially pieces of string with special functions which characterize the interested part of the ZVS status, i.e. the dashed line that indicates the ZVS output termination , or the resources names (CARP, ZFS and Service) at the beginning of the lines.

The code complexity of the Java base is essentially concentrated in this class, due to the flow of possible data coming form the cluster servers. Although the ZVS status output should be standardized as shown, there are cases where it is affected by communication errors or by malfunctions on the node. Then, even these situations must be correctly managed in Parser class for preventing undesired application crashes. Normally, the initial string splitting is not a problem but not the same can be said about the data extraction. This is clear from the figures 2.4.1 and 3.1.2, which show that the surrounding structure of the ZVS outputs are well defined, therefore, the possibility of corruption for these limiting characters is extremely low. Instead, in the status extraction case, in addition to the corruption that might be even more misleading, there is the likelihood of ZVS malfunctions, which can return completely unintelligible output. Thus, if a problem arises within the status extraction of a ZVS, the corresponding created object is marked as unreliable and, the assignment of signaling the trouble is given to the Web servlet.

In the end, the normal result of the Parser is a collection of Node-objects, used for keeping trace of which ZVSs are supported from what node, and a collection of ZVS-objects, containing the resources status that builds the ZVSs.

3.1.3 JDBC and database

Another important class of the Java ZI half, that carries out the third task by involving the two Parser-made collections and the database, is the DBCommands. This class acts as an interface for linking the database to the Java base by relying on the JDBC. It has been explained the practical need for the database, at the beginning of this chapter, but nothing has been said about the real motive that determined its employment yet. The database has been projected not just for holding the current status of the ZVSs and nodes, but also for maintaining an monthly history of the services status. Indeed, the ZVS table of the database has an extra column used for keeping the date of the status readings, that is part of the table primary key too. Thanks to this key, the table is able to have two or more rows differing only for the date (Time attribute in the figure 3.1.2), making thus a ZVS history. The database is realized on the Oracle Corporation DBMS Mysql and it is composed of four simple tables: Address, Nodes, Users and ZVSs. The database schema is:

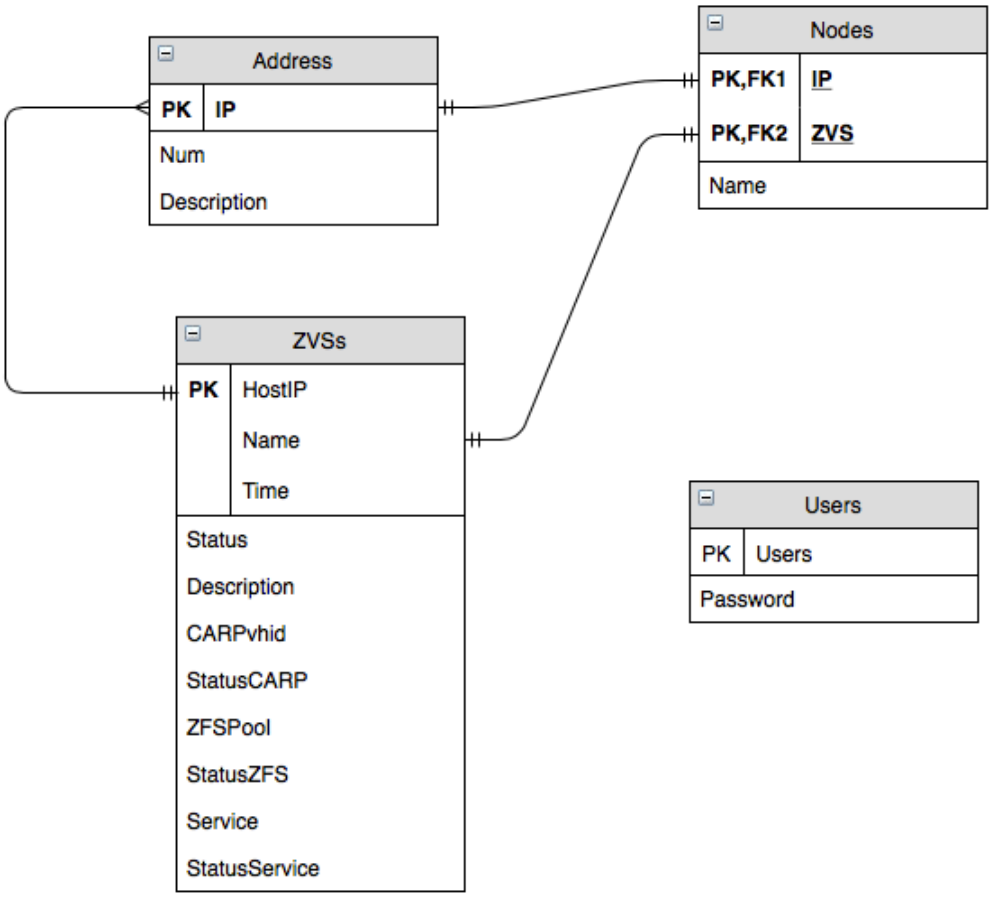


Figure 3.1.2: implemented database schema of Java base

By starting from the Users table, it holds the credential of the authorized users which can access the web page of the Zcluster Inspector. Like the figure 3.1.2 shows, the Users table does not participate in any relation, indeed, it is just used as a safe users credential bank. The Address table along with the ZVSs and Nodes tables build instead the relational database of the application.
The Address contains all the IPs of the existing zcluster nodes and it is the only source of address, not just for the database but even for the SSH communication. Indeed, in the gathering phase the SSH class accesses the DB to collect all the IPs that must be contacted , for this reasons, the table must be always updated with the real cluster server addresses. The Address table participates in a one-to-many relationship with the ZVSs table, each ZVS is supported by al least two different nodes, and in a one-to-one relationship with the Nodes table, each IP is owned by one and only one node.
About the ZVSs there is not much to say, every instance of such table has a key composed of an hosting IP, a unique name and a status reading date, followed by all the resources which make the ZVS. The only thing to underline about this table is the one-to-one relation with the Nodes table. A ZVS is normally supported by two Nodes, but these cases are treated as independently tuples in the DB, meaning that a ZVS called "Apache" has two rows in ZVSs table and each one matches a distinct node IP.
The remained Nodes table keeps the information about the nodes, such as the name and the IP address, and it is particularly useful for the query joins with the ZVSs table in the Web Servlet half of ZI.
The last point to deal with in this section is how the database is used by Java base. The DBCommands class has the task to provide all the necessary commands to read and write the database, using the standard Java library and the Mysql connector, or JDBC. When any part of Java base needs an access on the database, it exploits a method of DBCommands which, by relying on prepared statements and queries, acts on the DB in the defined way.

3.1.4 Cyclic interrogation

One of the ZI tasks is to show real-time status data of the zcluster nodes, therefore, the process of acquiring data must be repeated at certain intervals to give semblance of real-time. This cyclic interrogation of the cluster servers is performed by an apart class, called ZTimer.
The whole sequence of operations in Java base, starting from the SSH communication, passing through the parsing and ending in the database, is managed by the ZTimer, that acts as main class of this program. Basically, ZTimer uses the composition over the other process classes in order to accomplish the Java servlet duties and meanwhile, it keeps also trace of the amount of time spent in the entire cycle with a Java timer.

Although the repetition of the acquiring actions is scheduled at three minutes by default, if the process takes more than this time the interval is automatically adjusted, increasing it of one minute. There may be dozen of reasons which slow down the gathering process, i.e. slow network bit rate or failure on nodes, and most of them cannot be predicted unless at runtime, however, in cases, such as large number of machines, the initial period of repetition may be manually modified, before the suite installation. The manual modification comports the variation of the static variable "sec" inside the ZTimer class.

3.2 Web servlet

The second half of Zcluster Inspector is the Web servlet, a graphical web application written partly in HTML and partly in JavaServer Pages, a technology that allows dynamical generation of web pages. This servlet is designed to be the terminal element of ZI, used by the customer of zcluster to check the proper working of the cluster servers and to simply interact with them. Web servlet allows a set of operations on the system, described in the use case diagram reported in the figure 3.2.1.

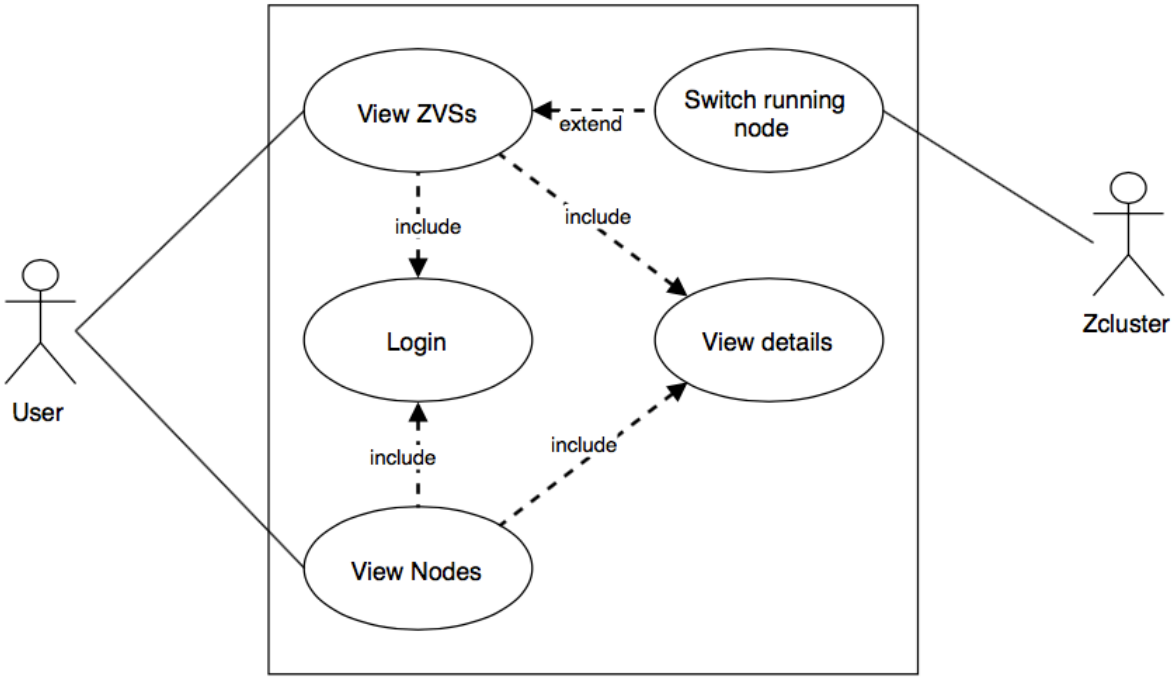


Figure 3.2.1: use case diagram of Web servlet

Generally, apart for the login, that is mandatory for every new session, the operations shown in the diagram do not follow a precise sequence and, it is the user who decides what to do inside the web application.

The Web servlet structure is composed of two main pages: ZVS view and Node view. These pages respectively permits to see the ZVSs and Nodes configuration, other than the details of the resources characterizing the views. It must be remembered that both the pages interact directly with the database, and, the data shown in them are just a graphical representation of what the DB contains. However, there is one case in which the Web servlet does not vehicle its attention on the database, and that is the switching running node operation, an action possible only when the user is focused on the ZVS page. The switching running node operation, when it is triggers, send directly a failover command to the referred node, without passing through the DB or Java base. This is done to maximize the velocity of such action.

3.2.1 Server web

The principal element of this servlet is certainly the web server, Apache Tomcat version 8.5. How has been said yet at the beginning on the Java base section, the web server is the technology on which the entire ZI is build on, indeed, it does not only host the Web servlet but even the Java base servlet. Thus, it is important to identify the major characteristics necessary to run ZI, in case an alternative solution to Apache Tomcat is required. First of all, the employed server must embed a JSP compiler for automatically transforming JSPs into executable Java servlet. Consequently to the previous point, the web server must be a Java Application Server with integrated a servlet container. Despite the surrounding platform of ZI is written in Java language, the unique reason to choose this kind of server is for the adoption of JSP in the Web servlet. As the result of the decoupling of the ZI building halves, Java base is not forcefully bound to a web server, but it can be run in any environment that hosts zcluster with a simple Java complier.
Regarding Apache Tomcat, its adoption is related to the large online documentation and to the great reputation into be one of the most remarkable Java web server. Tomcat server is an open-source software originally developed by Apache Software Foundation and nowadays maintained by an open developers community. As consequence of almost twenty years of history, Tomcat implements several Java EE specifications, which have made it widely used but meanwhile have enlarged its size. Thus, due to the high complexity and the various dependencies, Tomcat is not really appropriate for all the situations, especially when the hosting machine requires a light installation. Moreover, in order to make the Web servlet running, there is no need for a particular Tomcat configuration. Indeed, it is sufficient a standard installation of the Tomcat package from the Apache portal on the desire machine, followed by the deployment of the servlet on it. The only external thing that must be included in the environment is the JDBC suitable for Mysql or for the other DBMS used to design the DB.

3.2.2 Login

One of the most important principle of the Web servlet is the protection against misuses or external attacks in order to guarantee, as usual for an enterprise environment, a safe application. It has already been previously mentioned that the database places an additional safe barrier to protect the zcluster system, but this barrier is pointless if the web half of ZI is not well protected too. Thus, an encapsulation has been places, in form of filter, all around the Web servlet in order to catch the HTTP requests which would access the web application. The employed filter is a java class, called LoginFilter, that checks if it is present a session for the users asking the resource.

Then, if the user session is present, the request may continue and the response is sent back, while if no session exists for that user, the request is redirected to the login form.
This mechanism is an implementation of the Filter provided by the Javax servlet library and it is enabled by adding some lines of code in the web.xml file of the servlet. These code lines specify the location and the name of the filter, other than what resources have to be protected, that in Web servlet case are the ones under the Secured folder. The servlet encapsulation relies on another element, the login form. A login form is an HTML page used to ask the credentials needed for the user authentication. In ZI, this form uses a post method for collecting the username and the password which are verified via the CheckLogin JSP file. Once the credentials are inserted, the CheckLogin gets the parameters from the body message of the form response, and, it uses them in prepared statements for the verification with the Users table of the database. Prepared statements are basically a way to build queries preventing injections on databases. The necessity for acting in this way is bound to the fact that credentials verification done with queries directly on DB tables might be bypassed without a proper handling of the parameters,letting an unauthorized user gains the access to the application anyway.

3.2.3 ZVS view

The principal page of the Web servlet is surely the ZVS view. After a successful login, by default the user is redirected to this page for starting his working session. The main objectives of ZVS view are:

- Show the most recent status of the ZVSs.
- Allow the user to change the running node for any ZVS.

Furthermore, other than thees two primary objectives, the page gives the possibility of watching the details of each ZVS and exploring the status chronology of the services running on zcluster. These functionalities are mapped into a JSP file, called as the page itself. This file uses the JavaServer Pages features, along with the Java classes of such servlet, in order to make the most simple and meanwhile useful GUI for carrying out the four cited objectives. ZVS view page cooperates with DBCommands class to interface itself with the DB, and, with a SSHConnection class to directly send failovers on the ZVS servers, whenever a switching event is triggered. Despite these two classes have the same name of the other two contained in the Java base half, their implementations are completely different. The only reason for the same names of such classes is that they have equivalent roles in the two servlets making ZI. Indeed, like in Java base, DBCommands provides the whole set of tools for reading from and writing to the database, and again, the SSHConnection exploits the secure shell channel to transmit commands on the remote nodes, without caring about the output.
The ZVS view layout is composed of: a menu bar, at the top, a ZVS boxes area, in the middle, and a footer at the bottom of the page.

The menu bar, in turn, is formed by the page title, a searching pane and a swiping button to change the page.
Although the searching button can be useful if there are many ZVS in the page, the really important elements of this bar are the swiping button and the page title. The figure 3.2.2 shows the layout of the menu bar.

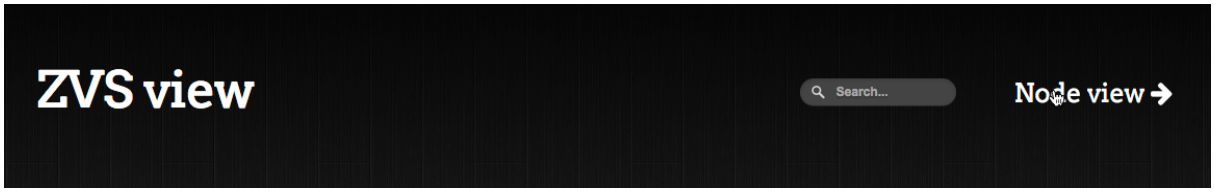


Figure 3.2.2: ZVS view menu bar

The core of ZVS view is however the boxes area that composes the body of the page. This section makes the user able to see on which nodes each ZVS is either running or backup, and furthermore, it gives the possibility of changing the running server for every ZVS, just by clicking on the buttons schematizing the nodes. The figure 3.2.3 illustrates a typical box that groups the two supporting nodes for "www", the ZVS, and the details button. In a typical situation where many ZVSs are running on the system, this body area will be composed of as many boxes as the number of ZVSs placed one beneath the other.

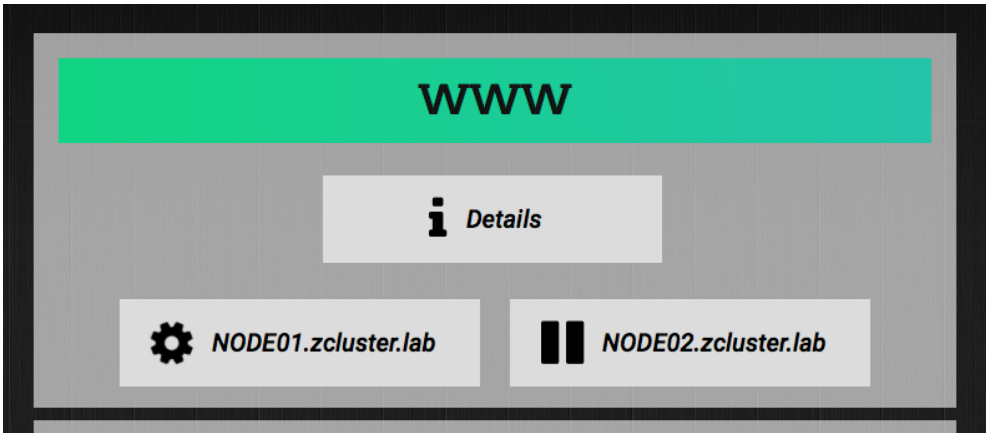


Figure 3.2.3: structure of a ZVS box

Also the details button may trigger an event when the user clicks on it, and if it happens, a new popup window is opened on the ZVS view for giving more information about the ZVS and the nodes that support it. Finally, every box of the body has a background color on the title that indicates the status of the ZVS. if the background color is green then the ZVS is properly working, while if the color is red a problem is affecting the service.
The last part of the page is the footer that contains the history table of the ZVS and the logout button preceded by the legal information about the software.
Like it is shown in the below figure 3.2.4, the ZVS history is hidden by default to make more accessible all the elements of the area, but a simple click on the title will appear a table enclosing the old status of the ZVSs.

This history is obtained with a join of the Nodes and ZVSs tables for giving also information about the supporting servers, and since it is monthly, the result of the query is filtered by date.

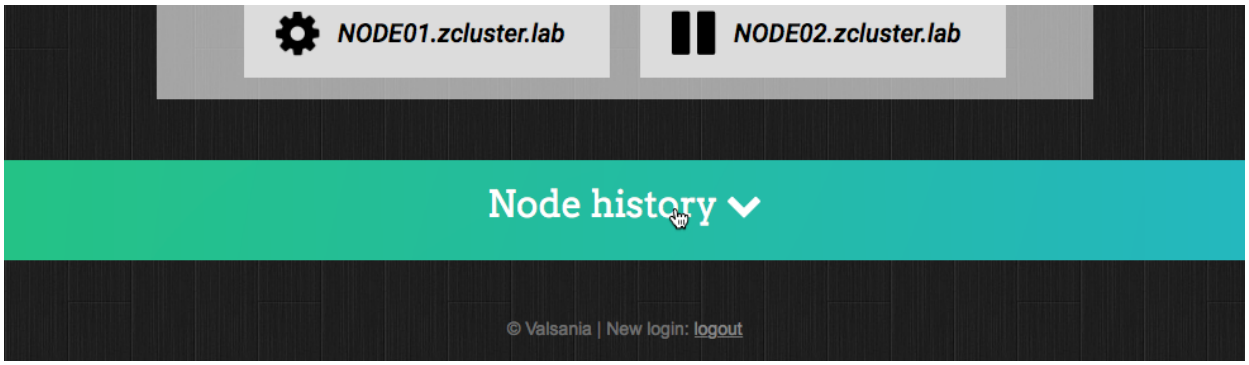


Figure 3.2.4: footer of the ZVS view

ZVSView.jsp file is located under the folder of the servlet /WebContent/Secured, meaning it is a protected resource. It has already been said that such file uses two Java classes in order to perform its objectives, but theses two external resources are not the only ones on which the ZVS view relies on. Indeed, the look of all the elements populating the view are managed via a CSS file called style.css, stored in the /WebContent/Secured/CSS folder. This file is organized per macro-elements and it employs HTML identifiers to map them with the ZVS view elements, as it is usual for the style shaping with CSS. The other external resource on which the ZVSView.jsp relies on is a Java Script file contained in /WebContent/Secured/JS. All the moving effects and some functionalities of the pages are the result of functions made in such script.

3.2.4 Node view

The Node view is the secondary page as well as the second and last view building the Web servlet up. This page has a complete distinct task than the ZVS one, it displays the ZVS of zcluster from a nodes point of view. The structure of the Node view is really similar to the other one, indeed, it is composed of a menu bar,a boxes area and a footer like the primary view. What makes really different the Node view from the ZVS view are the boxes. Each box of this page does not represent a Zcluster Virtual Server but instead a node, and furthermore, the elements inside the boxes schematize the ZVSs hosted by the nodes. Unlike the previous page, that supplies interactive functions for the user, the Node view has just a purpose of describing the cluster node scenario of zcluster.
Sometimes may be useful to see which ZVSs are running and which are backup on a node for balancing the workload, implying switching actions in ZVS view, or for discovering which ZVSs have to move away in case of server maintenance.
The Node view menu bar is exactly the same of the ZVS one, except for the title, that is “Node view”, and the swiping button, that has the “ZVS view” caption.

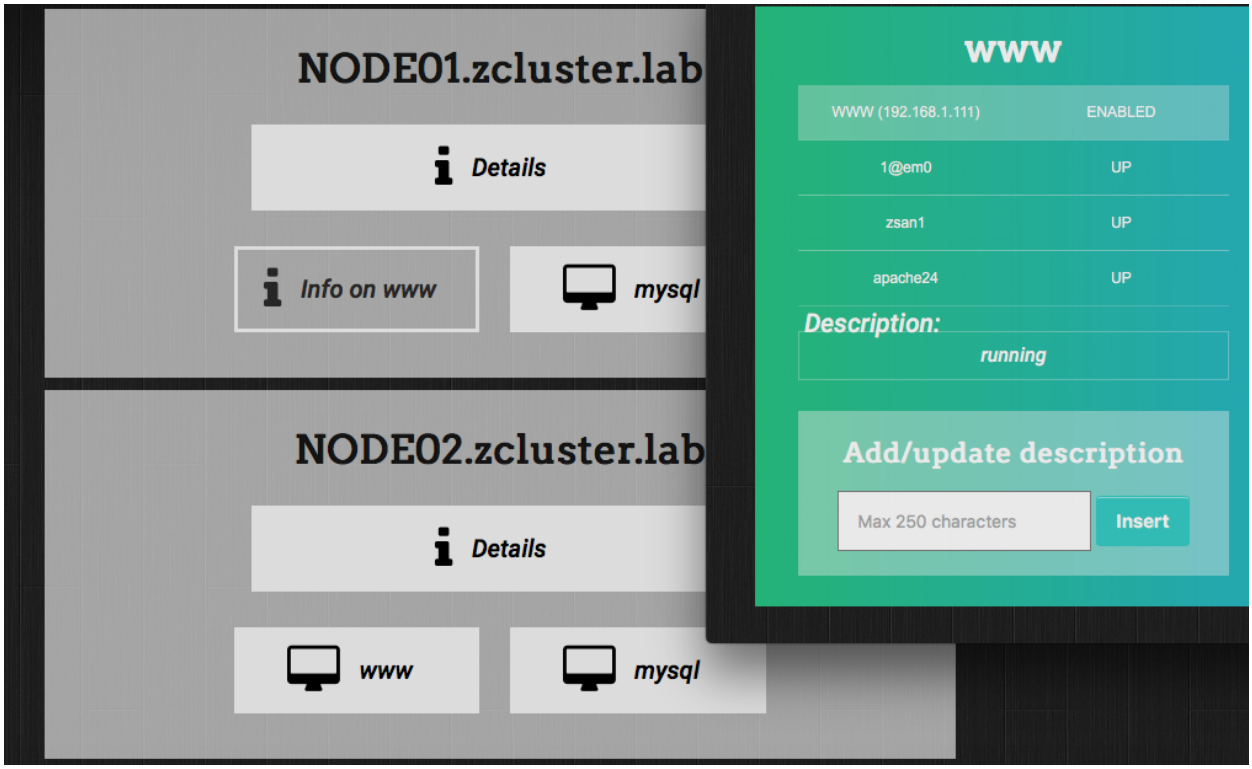


Figure 3.2.5: boxes area of the node view with a details popup window opened

The figure 3.2.5 shows the body area of a Node view, where are present two cluster servers each running two ZVSs, “www” and “mysql”. In addition the image illustrates also what kind of description window is opened after a click on a button schematizing a ZVS. By starting from the boxes, their structure are the same of the ones in ZVS view, but with two slight differences: the first is no background color is necessary under the box title and the second is that the number of parallel ZVS buttons is fixed by the amount of ZVSs on the node. Even in this view a general details button is part of each box, and its basic function is to show the ZVS resources status along with some information about the node. Like the primary view of the Web servlet, every ZVS button is clickable, but instead of switching the running node, they open a new popup, where is visualized the status of the resource tuple. Furthermore this popup window gives the possibility of adding a description to the ZVS row of the database that holds the information about it, like the figure 3.2.5 points out.

The last section of the page is the footer again. It is composed of a node history table and a logout button, preceded by the same software information of the ZVS page. The node footer has a chronologic table different from the ZVS one, that shows only the essential data about the cluster servers, such as the IP address and the ZVS status (enabled, disabled).

Even if the ZVS history is not shown in any images, it collects and displays all the information about the ZVS in the same manner of the node table. The following figure 3.2.6 illustrates the structure of the Node view history table when it is unhidden. Like in the ZVS footer, the history is by default hidden, thus it looks like the one of figure 3.2.4, and only with a click on the title it appears.

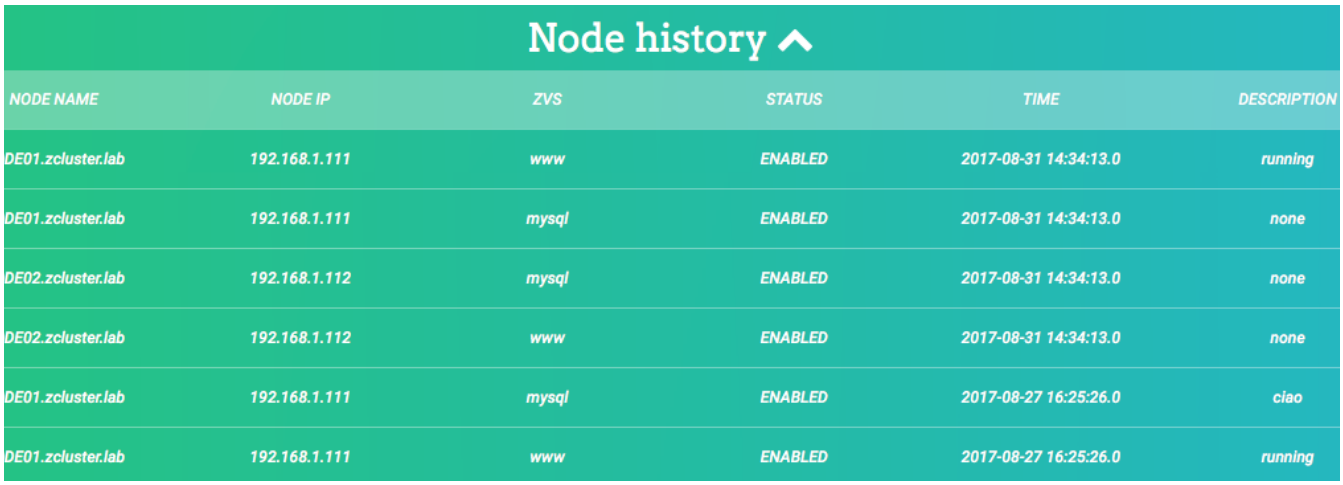


Figure 3.2.6: structure of the Node view history table

All the code necessary to realize the Node view is in the NodeView.jsp file, under the folder / WebContent/Secured of the Web servlet, implying also this page is protected from misuses by the servlet filter. The NodeView file, in order to collect all the data, relays only on one external Java class, the DBCommands. This imported class is the one used also in the ZVS view, but for such page it provides district methods for accessing and extracting information from the DB. SSHConnection is not needed in this view, due to the fact that this page does not implement any communication directly with the cluster server, but just with the database. Moreover, the Node view exploits the same style.css file to shape the appearance of its elements, still via the HTML identifiers, and the same Java Script file for the graphical effects of the page.

References

Valsania, A. (2003-2017). Valsania Information Technology Consulting. Available at: http://www.valsania.it

The FreeBSD Foundation (since 1995). The FreeBSD Project. Available at: https://www.freebsd.org

Appendix A: Zcluster Inspector SRS

INDEX

1.1 Purpose

The aim of this SRS is to point out and describe all the main functionalities of Zcluster Inspector, an extension application of the IT component zcluster.

It is also intended to give a general brief introduction of zcluster structure to understand properly how the Inspector will cooperate with it.

The audience of such report is any person who has both a freeBSD background and a proper knowledge of web server application, thus, whoever is used to the basic computer science concepts.

This dossier complies with the standard IEEE Std 830™-1998 revision of IEEE Std 830-1993.

1.2 Scope

Zcluster Inspector is intended to be a web application that allows users to keep under control and interact with all the basic aspects of zcluster, a command line program.

Among these aspects are the detection and the reporting of problems which affect the many blocks composing zcluster and, maybe not in the first release, ZI should also provide procedures to fix them directly.

Another important goal of ZI, related also to the problems detection, is to keep a small zcluster status history, where stakeholders may find information about old services.

Last but not least, this program have to achieve its management behaviors via a simplified and constrained application interface, suitable even for an unskilled user.

It is important to know Zcluster Inspector is a real-time application. Therefore each point must be implemented with special care for performance and reliability. Again, it must be said ZI is also a graphical application, developed for an enterprise environment. Thus, it should be realized in respect to all the unwritten norms for accomplishing a well styled software in every area.

1.5 Overview

The following section explains the environment in which the application is established, in order to understand how the Zcluster Inspector's interface should be realized.

In particular, in section two, the focus is going to be put on the principal blocks of ZVS system and on the interactions among them. Only few technical details are given about the implementation of zcluster but nothing else except its necessary information are specified.

Furthermore in Overall description, a general description of the user is supplied with particular emphasis on his skills. This should provide a general idea about the complexity of the ZI GUI layout and how much it can be technical.

Thus, at the end of this section, all the general constraints and dependences are specified with what might be added in future releases of Zcluster Inspector.

The third section, on the other hand, places the attention on the specific requirements defined by both the employer and the requirement team in the gathering phase.They are already been categorized as functional or nonfunctional on the base of their priority and their relevance, with the company manager agreement too.

The last section is the diagrams one, that is used as container for all the materials generated during the software negotiation and for any kind of useful diagrams.

2.1 Product perspective

Zcluster Inspector is a management application that should make zcluster more intelligible to any customer of the system. This implies that the software is going to be designed as part of the ZVS suite.

This suite is mostly composed of zcluster, a composition of several shell scripts each of which does a precise task to serve the superior goal of providing a steady availability storage system. These shell scripts are installed on FreeBSD machines via a Valsania Inc. ports. Therefore, due to the established environment, the whole service is supplied on a CLI.

Requirement: Zcluster Inspector should be a graphic application.

In order to run the scripts no external resources are needed, indeed the whole set of instruments is provided by the OS. However, a simple installation of zcluster is not enough to set the system up. In fact a strictly configuration procedure, which involves the modification of plenty of files, must be followed.

Requirement: Future ZI upgrade will make the application able of installing the whole system.

Terminated the installation, zcluster is composed of many nodes and storage disks, whose number depends on the company dimension. These nodes are basically server computers or onboard-server virtualization belonging to the same LAN.

Requirement: Zcluster Inspector should implement a LAN communication with nodes and disks.

In the end, what zcluster really does, once installed, is to automatically failover ZVS every time network problems or node crashes are detected. CARP makes this possible. In fact, if something goes wrong, CARP changes its internal status on the involved ZVS. Meanwhile, the daemon devd(8), which is always searching for these changes, switches the ZVS on the backup node(the standby machine).

Requirement: Zcluster Inspector should also allow manual switching of ZVS running node.

Less important for ZI is how the storage block works. Therefore, it is enough to say the shared storage availability is realized with a ZFS pools built on top of iSCSI LUNs, which are always connected to all the cluster nodes.

2.1.1 System interface

Zcluster must be installed on FreeBSD 10.0 or newer. Instead Zcluster Inspector needs both Apache Tomcat 8.0 or newer and a java compiler in order to work.

No particular specifications are required to the OS which runs ZI but, due to the nature of this application, at least a browser with HTML 5.0 support should be present. All the other dependencies related to the project implementation must be resolved during the package installation.

2.1.2 User interface

The web application is composed of two pages with the addition of a simple login form at the beginning of a new user session.

The main page is called ZVS View and it encloses all the information about the ZVSs outlined in boxes. In particular, each box should contain the two nodes on which the ZVS relies on and an informative menu for service data.

The Figure 2.1.2 shows an arranged layout of how the ZVS boxes might be structured.

The other page, instead, has the task to show which ZVS is running on each node, therefore it is named Nodes View. The structure of this second page is also organized in boxes which represent the Zcluster machines. Each box contains the supported ZVSs and a details button.

All the boxes elements are buttons and on click they should be able to trigger different events. The set of events which may be invoked it is explained later.

- *Page Layout: The page is essentially formed of a list of boxes and a simple header containing the page title and the navigation button to swipe among the views. The figure on the right shows a typical ZVS box with the two nodes and the details button. As the image points out the box layout must be as simple as possible in both the structure and the design. The header's arrange is negotiable but it should be clear and simple as well. The only difference between the two views is in the number of square buttons per box. Obviously in the Node page the box name is a node and the buttons names are ZVSs.*

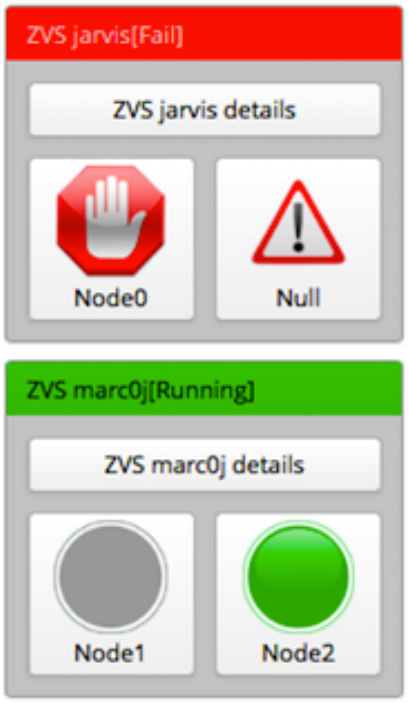


Figure 2.1.2

- *Usability: the servlet should be simple to understand and easy to use. Those characteristics are reached by keeping all the pages extremely clean. The view core consists of boxes as said. Thus, just there shocking colours and communicative images must be used in order to catch the user's attention.*
- *Scalability: due to the enterprise environment in which Zcluster Inspector will operate the scalability is one of the most important aspect. Every building-technologies used for the implementation should be able to work on thousands of services. In this way the system might be capable of manage either large and complex scenario or small and simple ones.*

In addition the entire suite may grow or evolve in future therefore the whole system should be as clear and well documented as possible.

2.1.3 Software interfaces and memory specifications

It has been already specified something about the requirements for installing the complete system but in this section more details and advice are given about the entire suite.

By starting from the core application, zcluster runs on FreeBSD 10.0 or newer that supports a SCSI persistent reservation with iSCSI targets names chosen among the ones in ZFS pools. Due to the light nature of FreeBSD, in the developing phase the OS may be simply virtualized with VirtualBox or similar. Every resources needed for zcluster is already installed in FreeBSD. Therefore the virtualization does not require a huge amount of RAM or storage on disk, so 256 MB of RAM and some dozens GB of disk are enough. The only problem with zcluster is its configuration. Indeed a skilled user is necessary to set up a working node. But once a virtual machines is configured it can be easily duplicated for creating a virtualized system.

Instead Zcluster Inspector is more demanding than ZVS System in term of external resources and dependencies. It requires a relational database management system with its JDBC driver and a web server that supports servlets (java web server). Furthermore two others important software interfaces, JDK and SSH, must be installed on the hosting machine. All these resources are essential to fulfill the requirements listed in the third section. This application does not need huge memory requirement as well, just the space to install all its building resources.

2.1.4 Communications interfaces

Zcluster Inspector is designed to be a widely portable application and for such reason, it should use largely supported protocols and languages. Even if for the moment it has to work only on a defined LAN network, all the unconstrained choices must be oriented to promote future communication updates.

ZI should be written half in java and half in HTML/JSP to satisfy the customer demand. The LAN protocol that has actually been chosen to implement the system interconnection is SSH.

*Finally, in agreement with all the developers, the project will be split in two part, the **web servlet**(HTML/JSP half) and the **java base**(java half), in order to simplify the implementation.*

2.2 Product functions

The major functions of Zcluster Inspector are:

- *Handle the communication with every nodes which compose zcluster.*
- *Retrieve information from all the system nodes.*
- *Process and then write the retrieved information in the database.*
- *Read the history of either ZVSs or nodes from the database.*
- *Organize in a simple graphical layout the current status of ZVSs and nodes.*
- *Allow the user to interact with ZVS in a graphical way.*
- *Send warnings and outline information about problems.*
- *Send commands to nodes for changing their configurations.*
- *Protect the system with user credentials to prevent misuses.*

2.3 User characteristics

The normal user of Zcluster Inspector is any person who works in the company and has a vague idea of how the system is formed. No technical skills are then necessary to use this software, and in ordinary condition, it should be useful just as check that everything works properly. Indeed, ZI is thought to steady display the status of zcluster and, whenever a problem arises, it should provide simple understandable information about the trouble.

Therefore a basic instructions manual must be integrated in ZI with two main aims: the first is to explain how the software can be used along with its characteristics, and the second is to list the possible problems of the application as well as a way to deal with them.

2.4 Constraints

- *Security: Zcluster Inspector is a web application that cooperates with another system on a LAN. Although it works on a protected local server, it may suffer of external attacks and misuses. Thus, it must be well secured to hide the network topology and other sensible information, such as IPs and passwords, and moreover, it must allow the access on the application to only authorised personal.*

- *Criticality: the application is tightly bound to zcluster, then a failure or a problem in ZI should not affect the entire system causing general breakdowns. It is therefore essential to keep any kind of error isolated from the zcluster.*

- *Parallelism: there are two operations which should be carried out at the same time: the first is the cyclic interrogation of the nodes with the corresponding status annotation in the database, and, the second one is the interaction user-GUI. Both these operations must be always operative and responding.*

- *Other Application: it has been largely introduced the relation between this application and zcluster in the previous steps, thus no other words are spent here.*

2.5 Assumption and dependencies

The biggest assumption of this project is certainly that zcluster will not change in future and this can not certainly be real. Indeed it will surely evolve in future, at least to be up to date with the new FreeBSD releases, and Zcluster Inspector must consequently vary.

All the other dependencies of this application are bound to the used components, like the web server and the java libraries. Even in this case, further modifications of such components will surely cause a necessary update of ZI implementation.

2.6 Apportioning of requirements

In the first release of the software is not necessary a complete support for the whole set of problems that may affect zcluster. Just brief information about major issues are enough.

Instead in future versions of Zcluster Inspector a more complete and structured architecture, which must offer solutions to the commonest problems, should be developed to realize instantly corrections.

A further desire of the customer is an upgrade of such application into another one that does not control only the zcluster but helps its installation too.

3.1 Functional requirements

The following section lists the functional requirements organized in tables where they are analyzed and explained. The requirements are presented from the bottom, the rough data acquisition, to the top, the graphic presentation.

3.1.1 Machines cyclic communication

Introduction	*Actors:* *Server Web and Nodes*
	Overview of the functionality: *Exchange of information between the server web and nodes.* ***Java base requirement****.*
Input	*Generic description:* *The trigger event of the communication is a timer that after an interval calls the whole nodes interrogation.*
Process	*Data Validation:* …
	Sequence process: *The web server should establish a SSH connection via the terminal with a public-private RSA key mechanism on the referred node. Thus, it sends the command to ask the status of the machines, that is "zcluster status", and it waits for the response. This procedure should be repeated for any nodes of zcluster and it is the key point of the application, if it doesn't work properly ZI have no reason to exist. The issue with this manner of act is that the pulled information arrives at the database only when the entire cycle is terminated. Therefore, the data are shown on the Web servlet older than the period of time needed to contact all the nodes, that means really far from a real-time data. Unless this timing problem not serious for the first releases, at the end of the cycle, the retrieved data should be packed into a map for being given at the Parser.*
	Handling errors and messages: *This communication might throw bunch of exceptions and warning massages bind to various causes, i.e. timer exceptions and SSH warnings. Therefore an errors management must be really well implemented since the beginning of this section and, a scrupulous verification has to be carried out during the test phase.*
	Parameters with effects on the output: *The number of nodes which compose zcluster affects both the scheduled emission of the data list and the entire communication cycle.*

Output	*In a normal scenario the output coming from the SSH back pipe is a sequence of chars which should be englobed in strings and then inserted in a map, by paying attention to not lose the association ZVS strings-node. Obviously these rough information must be treated to gain a useful meaning for the application. This last task is done by another section of the program (requirement 3.1.2).*

3.1.2 Parser

Introduction	*Actors:* *…*
	Overview of the functionality: *Parse the rough information retrieved in the nodes communication.* ***Java base requirement****.*
Input	*Generic description:* *The parser receives a list of strings, each from a node, and extract the necessary information.*
Process	*Data Validation:* *The input data should be controlled by the program to check if they are parsable strings.*
	Sequence process: *When the cyclic interrogation of the machines is correctly terminated, it produces a Java map that is given to the parser. Then the parser should start by splitting each node string in substring, in order to subdivide the data of the ZVS hosted on the involved node. At this point, the parser should use the meta-words that permit it to discover the status of the ZVS resources. Finally, all these refined information will be temporary stored in Java object,Node and ZVS object, ready to be written in the database.*
	Handling errors and messages: *If the communication goes right no particulars problems should arise. However in some rarely cases the received string might be corrupted and this may cause errors. Such kind of problems must be managed, only for the first realises, by just marking the data as unreliable without any other complication. This makes sense due to the frequently cyclic interrogation that will solve the corruptions bound to the interferences. Meanwhile the unreliable marked data are not used in the following steps.*
	Parameters with effects on the output: *…*
Output	*The expected outputs of this section are two: a java structure which hold the ZVS resources and a node structure that keeps the membership of the referred ZVSs.*

3.1.3 The database history

Introduction	*Actors:* *…*
	Overview of the functionality: *A database has the task to keep, for a period of time, all the information about Zcluster history.* ***Java base requirement****.*
Input	*Generic description:* *Generally, the Input data are the objects made by the java base parser that must be converted into a suitable format for the database. However in some case, like during the login, the data may be given by the Web servlet in form of string.*
Process	*Data Validation:* *Although the vast majority of data comes from the parser, some of them may be directly given by the user and therefore those information must be verified before their use in the DB.*
	Sequence process: *Once the System data produced by the parser are ready to use, they should be inserted into queries to store them in the correct table of the database. The data division among the tables is an automatic process done by this section, according to the type and name of the structure generated in the parsing phase. When the queries are been correctly generated, the last step until the writing is the safe login in the database.*
	Handling errors and messages: *The only exceptions of such stage are the ones thrown by the JDBC and the normal libraries of java. Thus, a usual structure to handle the errors is enough.*
	Parameters with effects on the output: *…*
Output	*This requirement is thought to be devoid of visual output. Indeed, the database role is to decouple the java base side from the servlet side in order to hide the software part which directly cooperates with the zcluster core.*

3.1.4 Login

Introduction	*Actors:* *Users*
	Overview of the functionality: *Open a new user session in the application.* ***Web servlet requirement****.*
Input	*Generic description:* *The user credentials, asked via a web form, are the input of this requirement.*
Process	*Data Validation:* *Due to the way the login is asked, it is important to handle and check the credentials properly. This may prevent problems such as SQL injections during the authentication.*
	Sequence process: *The user credentials are asked at the opening of the web application via a post method in a form. The login view must be shown also when a user, who does not have a valid session active, tries to open any URLs of the application. To achieve this safe behaviour a login filter has to be set in protection of all the Zcluster Inspector pages and the others resources. Then the credentials must be checked with the ones stored in the Users tables of the database. If they are correct a new user session will be opened otherwise the form will be reloaded with some warning messages.*
	Handling errors and messages: *The only useful messages of the current stage are those which specify what goes wrong with the login, in case it fails. In future versions could be useful that an error is generated when a user tries dozens of times to login without success in a sort period of time.*
	Parameters with effects on the output: *The output completely depends on the username and the password given.*
Output	*When the login goes right the main page of the application must be loaded, while, if it goes wrong the output is the page itself with an additional warning.*

3.1.5 ZVS view (main page)

Introduction	*Actors:* *…*
	Overview of the functionality: *Show the status of all the ZVSs in the simplest way.* ***Web servlet requirement****.*
Input	*Generic description:* *The input data are the information stored in the database.*
Process	*Data Validation:* *No data validation is necessary.*
	Sequence process and page information: *When the page is invoked the web server has to collect all the more recent information about the ZVSs from the database. Then, these data should be organized in boxes to present the services status, like it is described in section 2.1.2 User Interface. It is shortly repeated that each box is composed of two square buttons, which represent the ZVS nodes, and a details button. All these buttons are clickable and, depending on the type, they are able to trigger different events. The event thrown by the details button should open a popup window that displays all the general information about the involved ZVS. Whereas a click on the square button asks the java base program to switch the ZVS running node. Finally, in the box caption must be displayed the ZVS name and its general status with the related colour. The general status of a ZVS are three: RUNNING, STOP and FAIL, the respective meanings are: both the nodes are enabled and one is running, the service is manually stopped and both the nodes are disabled. The last important actions, that cyclically affects the page to keep it up to date, is a boxes reload.*
	Handling errors and messages: *The main page is a complex requirement due to the large amount of operations that have to be done. Therefore various errors may arise during the normal use, i.e. database errors, server errors or servlet errors. In these case the desired behaviour is to elicit and print the information about the problems directly on the page, replacing the usual ZVS view.*
	Parameters with effects on the output: *…*
Output	*The output should be a web page divided in two sections. The top section which is formed by the view name and a button for shifting the pages. Indeed, the second section contains the already described ZVS boxes.*

3.1.6 Node view (secondary page)

Introduction	*Actors:* *…*
	Overview of the functionality: *Show which ZVSs are supported by each node.* ***Web servlet requirement.***
Input	*Generic description:* *The input data are the information stored in the database.*
Process	*Data Validation:* *No data validation is necessary.*
	Sequence process: *This page will be open only when the user clicks the shifting button on ZVS view. Then on such request the servlet should access the database for collecting the information to be shown about the nodes. Unlike the main page, just the more recent nodes data are stored in the DB, therefore after the gathering they are ready to use. The page should look like the previous one, on top there is a header with the name and the shifting button and the body is composed of boxes. Each box has the same structure of the ZVS one, even if it does not have only two buttons but as many as the ZVSs are on that node. The details button is also present in the body box and it gives more information about the node on click.*
	Handling errors and messages: *Also the secondary page may encounter various errors like database errors or server errors. Then the behaviour should be the same of the ZVS view which is to print the error information on the page. In such manner a technician can understand and fix the occurred problem.*
	Parameters with effects on the output: *…*
Output	*The output is a simple page composed of a title, a shifting button and a list of boxes. This secondary page has just a presentation purpose, thus the triggered events on clicks should solely show more information about the represented ZVS. Only the details button shows general node information such as the machine IP or type of machine.*

3.1.7 Switch the ZVS running node

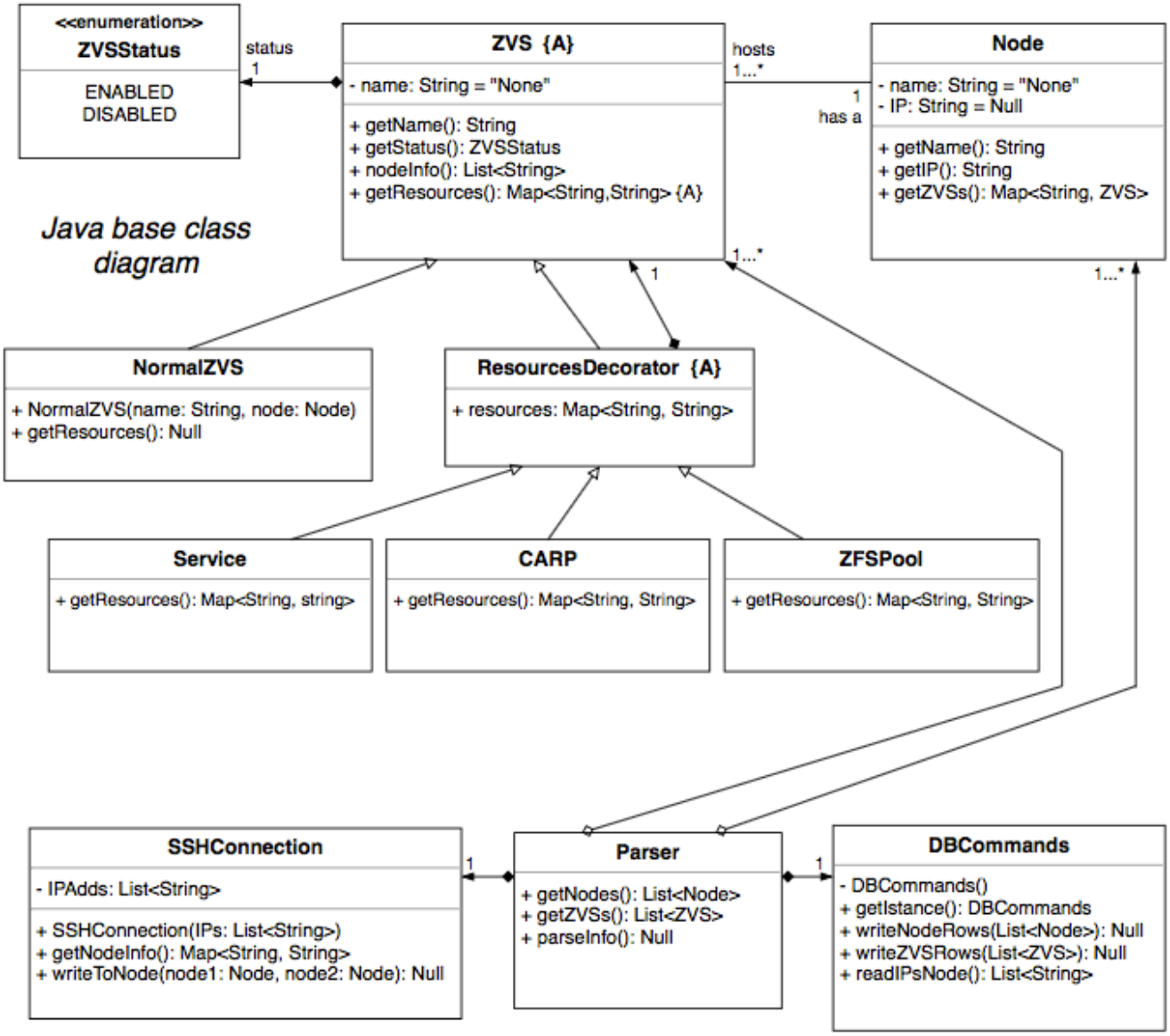
Introduction	*Actors:* *User and ZVS view*
	Overview of the functionality: *Change the running node of a ZVS.* ***Web servlet requirement****.*
Input	*Generic description:* *A click on the square buttons of the ZVS box causes the switching event.*
Process	*Data Validation:* *No data validation.*
	Sequence process: *When the user clicks the node button the switching event should be thrown. This means the event dispatcher calls the java base program to tell it which nodes have to be contacted. Then, a new SSH connection with these machines is opened by the system, after that ZI sends either a failback or a failover command to one of respectively the stopped node and the running node. The last step that completes the process, it is to wait until a new SSH interrogation will parse the nodes.*
	Handling errors and messages: *Like it is already been said in others steps, the SSH communication is the brittlest point of the application and, at the same time, the most important function for the project. Thus, the system should react to a failed situation by getting itself on the default page status and, it must report a detailed message of what went wrong in the related ZVS box.*
	Parameters with effects on the output: *…*
Output	*No particular output is expected after this transaction, except a change in the icons that specify which node is running and which is stopped for the concerned ZVS.*

3.2 Nonfunctional requirements

NFR1-2	Performance requirements
Quickness	*This software is designed to work in a real-time context with a large amount of machines. Thus, even if it is not a main requirement, the application quickness is a really important factor for reaching the usability and the real-time objective.*
Light	*Relate to the previous requirement, the software size should be extremely limited. In fact this application is just a marginal component of an already working package, then it must not overload the entire suite in any way.*

NFR3-4	Software system attributes
Safety	*Like in every enterprise application, the software safety is extremely relevant and the same is for this one. Although the program should work just on a local network, it is not devoid of dangers and in addition, it should not introduce risks on zcluster. Therefore, Zcluster Inspector must be strongly protected from both improper users and external attacks.*
Maintainable	*The whole project implementation should follow all the basics concepts of the software engineering, such as layering, concurrency and design patterns orientation. In this way, the proposed application should be the most performing and efficiency and, at the same time, the future versions will be easier to work out.*

4.1 Class Diagram Java Base



4.2 Activity diagram Servlet