

MANUEL D'INSTALLATION DE AMDA KERNEL

Rédigé par : F.CASIMIR (AKKA)	Diffusé à : CNES/ IRAP
Approuvé par : Chef de projet CNES : DUFOURG Nicolas	

LISTE DES MODIFICATIONS DU DOCUMENT

Vers.	Date	Paragraphe	Description de la modification
01.0	20/11/12		Création du document
01.01	22/11/12	P6	Corrections mineurs
01.02	19/12/12	P7 P8	GCC 4.7 Installation Compilation mode Release ou mode Debug
01.03	15/01/13		Livraison Version 1.0.0
01.04	05/02/13	P9	Livraison Version 1.1.0
01.05	11/02/13	§2.1	Prérequis complémentaire
01.06	22/02/13		Prise en compte de la Recette du produit V1.1.0 et livraison de V1.1.1
01.07	29/11/13	§4.1	Livraison Version 1.1.2 Contient la correction des bugs US-86, US-87, US-88

SOMMAIRE

1 INTRODUCTION	5
2 PRE-REQUIS.....	6
2.1 Les différentes plateformes.....	6
3 PRE-INSTALLATION.....	8
3.1 Installation à partir des repository CentOS	8
3.1.1 Java (1.6.0).....	8
3.1.2 GCC (4.4.6-4.el6)	8
3.1.3 CMake (2.6.4-5.el6).....	8
3.1.4 libxml2	8
3.2 Installation à partir de tar.gz.....	8
3.2.1 log4cxx	8
3.2.2 GCC (4.7)	9
3.2.3 CSLIM.....	9
4 INSTALLATION	10
4.1 Décompression des fichiers	10
4.2 Configuration Globale	10
4.3 Compilation Mode Debug.....	10
4.3.1 Configuration	10
4.3.2 Compilation.....	10
4.4 Compilation Mode Release	10
4.4.1 Configuration	10
4.4.2 Compilation.....	11
5 POST-INSTALLATION	12
5.1 Passage des tests d'acceptation à partir de FitNesse.....	12
5.1.1 Etape 1 Ouverture de l'outil de test d'acceptation.....	12
5.1.2 Etape 2 Lancement des tests.....	12
6 PROCEDURE DE DESINSTALLATION.....	14
6.1 Etape 1 : Arrêt de FitNesse.....	14
6.2 Etape 2 : Suppression de la livraison.....	14
7 DOCUMENTS APPLICABLES ET DE REFERENCE (A/R)	15
8 GLOSSAIRE ET ABREVIATIONS.....	16
8.1 Glossaire	16
8.2 Abréviations	16
ANNEXE A. INSTALLATION D'UNE PLATEFORME D'INTEGRATION CONTINUE.....	17
8.3 Jenkins	18
8.4 MySql	19
8.5 Sonar.....	19

8.6 Plugin cxx-sonar.....	20
8.6.1 Sonar-runner	20
8.6.2 Doxygen	21
8.6.3 Valgrind	21
8.6.4 CppCheck.....	21
8.6.5 Gcov	21
8.6.6 Les jobs Jenkins de AKKA	21

1 INTRODUCTION

Ce document décrit la procédure d'installation du projet AMDA-Kernel-NG en vue de son développement et de son utilisation. Pour sa mise en exploitation un second manuel devra être écrit lorsque son intégration avec l'IHM sera décidée. Dans ce second document ne sera présenté que le déploiement des binaires et bibliothèques nécessaires à l'exécution de AMDA-Kernel (sans : java, CSlim, CMake, FitNesse).

Ce document indique chronologiquement les étapes d'installation de AMDA_Kernel-NG sur la machine hôte du serveur :

- Pré-requis,
- Pré-install,
- Compilation,
 - Configuration,
 - Test

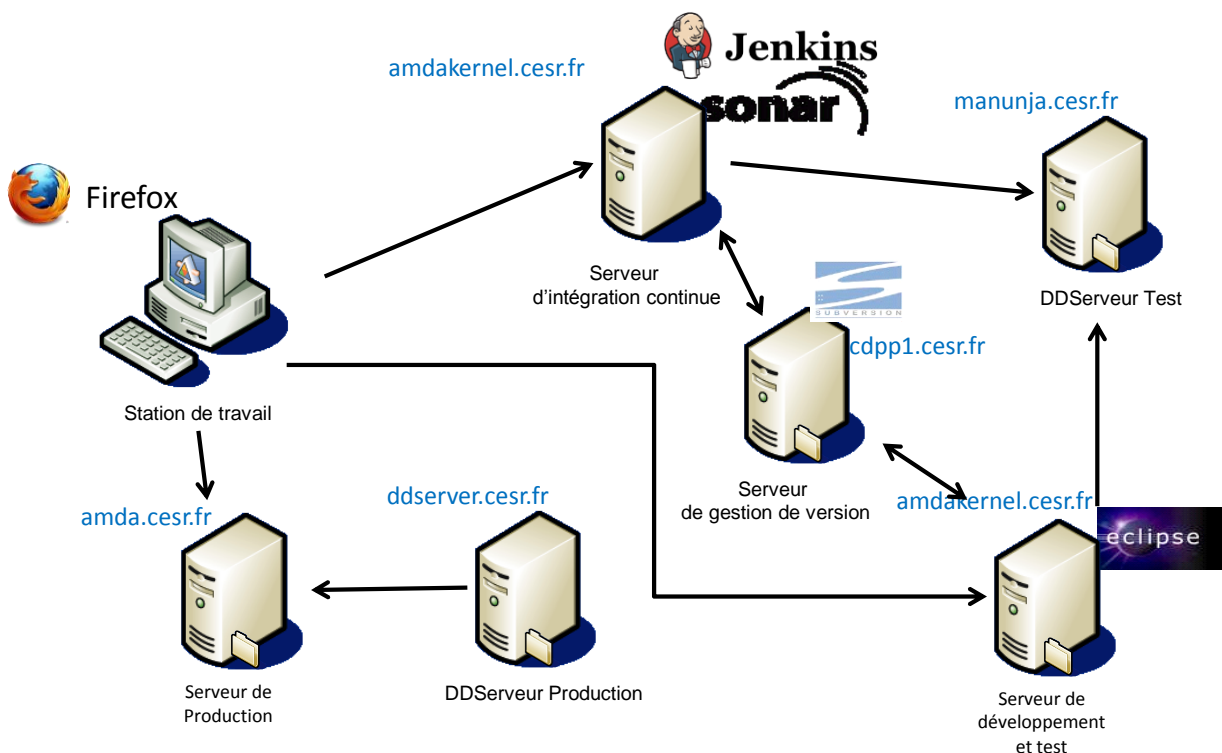
Et en annexe, ce trouve une procédure d'installation d'une plateforme d'intégration continue adaptée à AMDA-Kernel.

2 PRÉ-REQUIS

- Un serveur 64 Bits sous CentOS 6.3
- Le port 8081 de ce serveur doit être accessible depuis la machine de test.
- Ce serveur doit pouvoir atteindre DDServer
- Un utilisateur avec les droits administrateur
- Un utilisateur de développement.
- Fichier /etc/hosts configuré (hostname -i doit retourner quelque chose)

1.1 LES DIFFERENTES PLATEFORMES

Infrastructure AMDA



Plateforme	Server	Pré-requis	Description
------------	--------	------------	-------------

Plateforme	Server	Pré-requis	Description
Développement	La plateforme de développement a été installée sur une machine virtuelle VirtualBox 'amdakernel.cesr.fr', tournant sur le serveur 'cdpp1.cesr.fr' à l'IRAP.	Espace requis : <ul style="list-style-type: none"> • 8Go d'espace disque pour le système • 200Mo pour /boot • 1Go par développeur (/home) • 8Go pour l'installateur (/home) 	Y seront déposés les sources pour compilation.
Intégration	amda-intergration.cesr.fr Idem dev. : amdakernel.cesr.fr'	Espace requis : <ul style="list-style-type: none"> • 8Go d'espace disque pour le système • 200Mo pour /boot • 3Go pour l'installateur (/home) 	Les binaires des prérequis seront récupérés de la plateforme de développement. Les sources des produits à intégrer viendront du serveur de version
Production	amda.cesr.fr	Espace requis : <ul style="list-style-type: none"> • 8Go d'espace disque pour le système • 200Mo pour /boot 	Les binaires des prérequis seront récupérés de la plateforme de développement. Les produits à exploiter seront ceux produits et testés par la plateforme d'intégration continue.
Serveur de gestion de version	cdpp1.cesr.fr (svn://cdpp1.cesr.fr/depot kernel/AMDAKernel)		

De plus (en particulier en production) il faut prévoir suffisamment d'espace disque pour stocker les fichiers utilisateurs (fichier paramètre xml, et .so correspondant (environ 1.5 Mo par paramètre compilé)).

2 PRE-INSTALLATION

2.1 INSTALLATION A PARTIR DES REPOSITORY CENTOS

Avec l'utilisateur d'administration.

2.1.1 Java (1.6.0)

```
yum install java-1.6.0-openjdk
```

2.1.2 GCC (4.4.6-4.el6)

```
yum groupinstall 'Development Tools'
```

2.1.3 CMake (2.6.4-5.el6)

```
yum install cmake
```

2.1.4 libxml2

```
yum install libxml2 # Package libxml2-2.7.6-8.el6_3.3.x86_64 already installed and latest version  
yum install libxml2-devel # libxml2-devel-2.7.6-8.el6_3.3.x86_64 zlib-devel-1.2.3-27.el6.x86_64
```

2.2 INSTALLATION A PARTIR DE TAR.GZ

Nous considérons pour cette étape que les fichiers suivants sont placés dans le répertoire pointé par la variable d'environnement **\$THIRD_PARTY_SOURCE**. Penser à la définir.

- apr-1.4.6.tar.gz
- apr-util-1.5.1.tar.gz
- apache-log4cxx-0.10.0.tgz
- boost_1_51_0.tar.gz
- fitnessse-cpp.tar.gz

L'installation de ces produits se fera sous **/opt/local**.

Si l'utilisateur utilisé n'a pas de droit sudo, passer avec l'utilisateur d'administration (exemple su root).

2.2.1 log4cxx

```
cd $THIRD_PARTY_SOURCE  
tar xvfz $THIRD_PARTY_SOURCE/apr-1.4.6.tar.gz  
tar xvfz $THIRD_PARTY_SOURCE/apr-util-1.5.1.tar.gz  
tar xvfz $THIRD_PARTY_SOURCE/apache-log4cxx-0.10.0.tar.gz  
cd apr-1.4.6/  
./configure --prefix=/opt/local  
make  
sudo make install  
cd ../apr-util-1.5.1/  
./configure --prefix=/opt/local --with-apr=$THIRD_PARTY_SOURCE/apr-1.4.6  
make  
sudo make install  
cd ../apache-log4cxx-0.10.0/  
./configure --prefix=/opt/local --with-apr=$THIRD_PARTY_SOURCE/apr-1.4.6 --with-apr-util=$THIRD_PARTY_SOURCE/apr-util-1.5.1
```



```
export LD_LIBRARY_PATH=/opt/local/lib:$LD_LIBRARY_PATH
make
sudo make install
```

2.2.2 GCC (4.7)

```
# Download and install it.
sudo mkdir -p /opt/tools/gcc/4.7.2
sudo chmod -R 0777 /opt/tools/gcc/4.7.2
cd /opt/tools/gcc/4.7.2
#y mettre bld.sh gcc-4.7.2-archives.tar.gz
chmod a+x bld.sh
tar xvzf gcc-4.7.2-archives.tar.gz

sudo yum install -y glibc-devel.i686
./bld.sh &> bld.log

mkdir -p "/opt/tools/gcc/4.7.2/boost/include"
cd /opt/tools/gcc/4.7.2/boost/
ln -s /opt/tools/gcc/4.7.2/src/boost_1_51_0/stage/lib lib
cd /opt/tools/gcc/4.7.2/boost/include
ln -s /opt/tools/gcc/4.7.2/src/boost_1_51_0/boost boost
```



2.2.3 CSLIM

```
cd $THIRD_PARTY_SOURCE
tar xvzf fitnessse-cpp.tar.gz
cd fitnessse-cpp
./makeAll.sh
#Attention : Cette étape se termine par un corp dump de l'exécutable de test, ce n'est pas grave.
sudo ./install.sh
./cleanAll.sh
```

3 INSTALLATION

3.1 DECOMPRESSION DES FICHIERS

Nous utilisons la variable **\$WORKSPACE** dans tout le document. Pensez à la définir.

Sous **\$WORKSPACE** si les répertoires **AMDA_Kernel** et/ou **AMDA_Kernel_V1.1.2** existent les renommer.

Le fichier compressé suivant **2013-11-29-AMDA_Kernel_V1.1.2.tgz** doit être décompressé dans le répertoire pointé par **\$WORKSPACE**. Le dossier obtenu se nomme : **AMDA_Kernel_V1.1.2**.

Création d'un lien symbolique

```
ln -s AMDA_Kernel_V1.1.2 AMDA_Kernel
```

3.2 CONFIGURATION GLOBALE

```
GXX_HOME="/opt/tools/gcc/4.7.2/rtf"  
export PATH="${GXX_HOME}/bin:/opt/local/bin:${PATH}"  
export LD_LIBRARY_PATH="${GXX_HOME}/lib64:${GXX_HOME}/lib:${LD_LIBRARY_PATH}"
```

Idées :

- le mettre dans le ".profile" pour les développeurs.
- dans le script de lancement pour l'application déployée.

3.3 COMPILATION MODE DEBUG

3.3.1 Configuration

```
cd $WORKSPACE/AMDA_Kernel  
export BUILD_TYPE=Debug  
cmake -E make_directory build  
cmake -E chdir build cmake -DCMAKE_BUILD_TYPE=Debug ..
```

3.3.2 Compilation

```
cd $WORKSPACE/AMDA_Kernel  
cmake --build build  
make -C build install VERBOSE=1
```

LA GENERATION DES BINAIRES SE FERA SOUS « \$WORKSPACE/AMDA_Kernel/build/debug. »

3.4 COMPILATION MODE RELEASE

3.4.1 Configuration

```
cd $WORKSPACE/AMDA_Kernel
```

```
cmake -E make_directory build  
cmake -E chdir build cmake -DCMAKE_BUILD_TYPE=Release ..
```

3.4.2 Compilation

```
cd $WORKSPACE/AMDA_Kernel  
cmake --build build  
make -C build install VERBOSE=1
```

LA GENERATION DES BINAIRES SE FERA SOUS « \$WORKSPACE/AMDA_Kernel/build/Release. »

4 POST-INSTALLATION

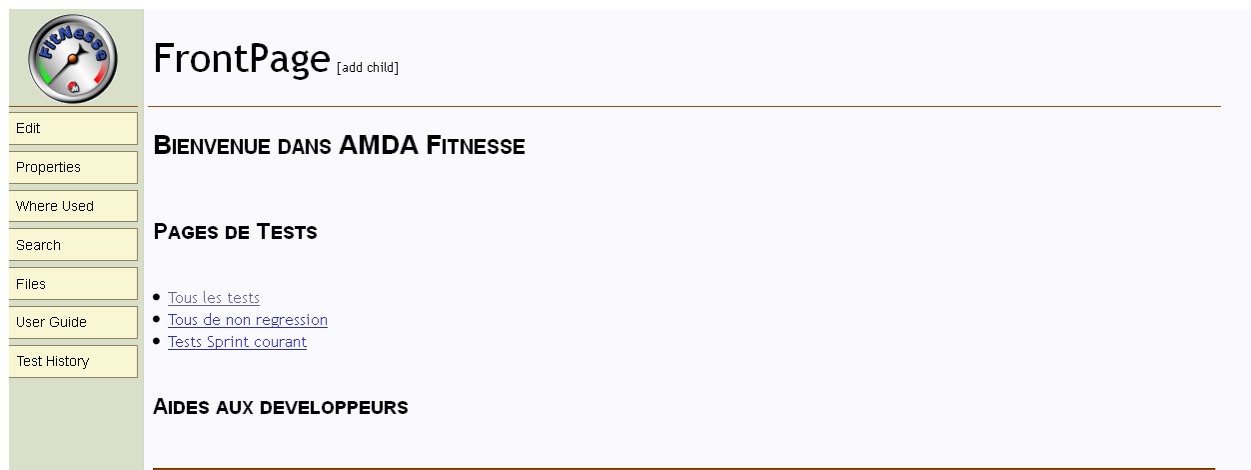
4.1 PASSAGE DES TESTS D'ACCEPTATION A PARTIR DE FITNESSSE

```
cd $WORKSPACE/AMDA_Kernel
nohup java -jar COTS/FitNesse/fitnessse.jar -d test -e 0 -p 8081 &
```

POUR TESTER L'APPLICATION MODE RELEASE METTRE UN FICHIER FLAG DE NOM "Release.flag" DANS LE REPERTOIRE « \$WORKSPACE/AMDA_KERNEL »

4.1.1 Etape 1 Ouverture de l'outil de test d'acceptation

Dans un navigateur rentrer le nom de votre serveur suivi du port 8081. Exemple : <http://amdakernel.cesr.fr:8081>
L'écran suivant doit s'afficher.



The screenshot shows the FitNesse web interface. On the left is a sidebar with a 'FitNesse' logo and a menu containing: Edit, Properties, Where Used, Search, Files, User Guide, and Test History. The main content area has a title 'FrontPage [add child]' and a heading 'BIENVENUE DANS AMDA FITNESSSE'. Below this is a section 'PAGES DE TESTS' with three links: 'Tous les tests', 'Tous de non regression', and 'Tests Sprint courant'. At the bottom of the main area is a section 'AIDES AUX DEVELOPPEURS'.

4.1.2 Etape 2 Lancement des tests

Pour lancer les tests d'acceptation :


- cliquer sur le chapitre tous les tests.



The screenshot shows the 'Releases' page in FitNesse. The sidebar is identical to the previous screenshot. The main content area has a title 'Releases [add child]' and a heading 'RELEASES'. Below this is a section 'LIST DES RELEASES DE AMDA-KERNEL' with a single link: 'Release 1'.


- cliquer sur le menu « Suite » en haut à gauche.

Résultat attendu : tout doit être au vert.



ReleaseS

SUITE RESULTS [\[history\]](#)



Test Pages: 10 right, 0 wrong, 0 ignored, 0 exceptions Assertions: 61 right, 0 wrong, 0 ignored, 0 exceptions (60.324 seconds)

TEST SUMMARIES

SLIM:BUILD/BIN/CSLIMTESTSERVER

2 right, 0 wrong, 0 ignored, 0 exceptions	ReLease1.SprinT1.UserStory31 (0.002 seconds)
15 right, 0 wrong, 0 ignored, 0 exceptions	ReLease1.SprinT2.UserStory32 (10.428 seconds)
4 right, 0 wrong, 0 ignored, 0 exceptions	ReLease1.SprinT2.UserStory34 (2.855 seconds)
13 right, 0 wrong, 0 ignored, 0 exceptions	ReLease1.SprinT2.UserStory36 (0.007 seconds)
4 right, 0 wrong, 0 ignored, 0 exceptions	ReLease1.SprinT3.UserStory04 (1.402 seconds)
2 right, 0 wrong, 0 ignored, 0 exceptions	ReLease1.SprinT3.UserStory12 (0.001 seconds)
13 right, 0 wrong, 0 ignored, 0 exceptions	ReLease1.SprinT3.UserStory35 (9.655 seconds)
3 right, 0 wrong, 0 ignored, 0 exceptions	ReLease1.SprinT3.UserStory49 (4.584 seconds)
2 right, 0 wrong, 0 ignored, 0 exceptions	ReLease1.SprinT3.UserStory54 (0.006 seconds)
3 right, 0 wrong, 0 ignored, 0 exceptions	ReLease1.SprinT4.UserStory46 (0.006 seconds)

TEST OUTPUT

TEST SYSTEM: SLIM:BUILD/BIN/CSLIMTESTSERVER

[ReLease1.SprinT1.UserStory31](#)

5 PROCÉDURE DE DÉSINSTALLATION

5.1 ETAPE 1 : ARRET DE FITNESS

```
#Récupérer le PID du process  
ps -ef | grep FitNesse  
  
#Kill du process à l'aide de la commande kill  
kill ???
```

5.2 ETAPE 2 : SUPPRESSION DE LA LIVRAISON

```
rm -rf $WORKSPACE/AMDA_Kernel
```

6 DOCUMENTS APPLICABLES ET DE RÉFÉRENCE (A/R)

A/R	Référence	Titre
R1	CDPP-AR-32500-382-SI	Dossier d'architecture du noyau d'AMAD-NG
R2	CDPP-CD-32500-436-SI	Dossier de conception du noyau d'AMAD-NG
R3	CDPP-IF-32500-438-SI	Dossier de contrôle des interfaces du noyau AMDA-NG

7 GLOSSAIRE ET ABREVIATIONS

7.1 GLOSSAIRE

Terme	Définition

7.2 ABREVIATIONS

Abréviation	Nom détaillé

ANNEXE A. INSTALLATION D'UNE PLATEFORME D'INTÉGRATION CONTINUE

Le principe de l'intégration continue est d'aller encore plus loin dans l'automatisation des vérifications ; si les tests unitaires se concentrent sur la non-régression du code, l'intégration continue prend en compte tous les aspects d'un développement logiciel. Le but est toujours de détecter les problèmes le plus tôt possible, pour s'assurer qu'ils soient résolus le plus vite possible.

Le but de cette annexe est de vous aider à mettre en place une plateforme d'intégration continue pour le projet AMDA Kernel en vous appuyant sur celle d'AKKA qui utilise l'outil Jenkins.

Une plate-forme d'intégration continue s'exécute régulièrement, avec une fréquence la plus courte possible (au minimum toutes les nuits, au mieux plusieurs fois par jour). À chaque exécution, elle réalise plusieurs actions :

- Récupération du code source, depuis le dépôt qui est la plupart du temps un outil de gestion de source (CVS, SVN, Git, SourceSafe, ...). On peut éventuellement vérifier le nombre de fichiers ou la taille globale d'un projet, pour s'assurer qu'il n'y a pas eu de suppressions intempestives.
- Vérification et optimisation du code. Cette étape n'est pas obligatoire, mais l'exécution automatique d'un outil d'optimisation du code (comme [lint](#), [Jlint](#), [php-sat](#), ...) est une bonne idée, car elle assure un minimum de qualité sur le code écrit par les développeurs.
- Compilation des sources (pour du code en Java ou en C++, par exemple). Si un projet ne peut pas être compilé complètement, il faut remonter rapidement l'information, pour que le code problématique soit corrigé. Un projet qui ne compile pas peut bloquer l'ensemble des développeurs qui travaillent dessus.
- Exécution des tests unitaires. Cette étape a été décrite précédemment, je ne vais pas revenir dessus.
- Packaging de l'application. Cela peut prendre de nombreux aspects, suivant le type de logiciel que l'on développe. Pour un logiciel « lourd », on va tenter d'en générer automatiquement l'archive d'installation ; pour un service Web, on va préparer les fichiers pour leur mise en ligne.
- Déploiement de l'application. Là encore, tout dépend du type de logiciel. Cela peut aller de l'installation automatique du logiciel sur une machine, jusqu'à la mise en ligne d'un site Web. L'idée est qu'un logiciel qu'on ne peut pas installer ne sert à rien. S'il y a un souci avec les procédures d'installation, ou une incompatibilité entre ces procédures et la nouvelle version du logiciel, il faut corriger cela.
- Exécution des tests fonctionnels. En reprenant les mêmes outils que les tests unitaires, il est possible de vérifier un grand nombre de cas d'utilisation.

En plus de tout cela, on ajoute souvent une étape supplémentaire, même si elle ne fait pas partie de l'intégration continue à proprement parler. Il s'agit de la génération automatique de la documentation de développement, qui est faite à partir des informations présentes dans le code source (grâce à des outils comme [JavaDoc](#), [PHPDoc](#), [Doxygen](#), [HeaderBrowser](#), ...). Il est toujours plus facile de développer quand on a sous la main une version à jour de la documentation.

À la fin de l'exécution de toutes ces actions, la plate-forme doit envoyer des messages aux personnes concernées par les problèmes relevés. Ces messages ne doivent pas se transformer en spam, car ils deviendraient inutiles (personne n'y ferait plus attention). Il faut donc faire attention à remonter les vrais problèmes, et ne pas mettre tous les développeurs en copie sauf dans les cas nécessaires. La résolution des bugs remontés doit être la **priorité première** d'une équipe de développement. C'est simple, si on continue à développer en sachant qu'il y a des bugs, on sait pertinemment qu'il faudra encore plus de temps et d'énergie pour les corriger, tout en risquant de devoir refaire les « sur-développements ».

En bout de course, l'application déployée par l'intégration continue doit être accessible à l'équipe de test, qui peut ainsi procéder à ses vérifications complémentaires sans avoir à se soucier des étapes techniques en amont (compilation, packaging, déploiement).

Dans le cadre du projet AMDA Kernel, AKKA utilise SVN et n'exécute pas pour le moment d'étape de déploiement. Les outils utilisés sont SONAR et CppCheck pour une analyse de qualimétrie statique du code et pour l'analyse dynamique Valgrind et gcov. L'installation de ces divers éléments est décrite ci-dessous.

7.3 JENKINS

Procédure à suivre :

- <http://www.andrewzammit.com/blog/installing-jenkins-ci-on-centos-6-x-tomcat-with-an-ajp-proxy/>

Version PDF :



Installing Jenkins CI
on CentOS 6.x (Tomc

Résumé :

```
yum install screen
yum upgrade
wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo
rpm --import http://pkg.jenkins-ci.org/redhat/jenkins-ci.org.key
yum install jenkins
yum install httpd
#TODO emacs /etc/sysconfig/iptables
service iptables restart;
#TODO emacs /etc/httpd/conf.d/vhosts.conf
#TODO emacs /etc/httpd/conf/httpd.conf
service httpd start;
#Starting httpd: httpd: apr_sockaddr_info_get() failed for bas-amda-01
#httpd: Could not reliably determine the server's fully qualified domain name, using
127.0.0.1 for ServerName
#
[ OK ]
service httpd start;
service jenkins start;
```

Post-installation

A partir du menu « Jenkins->Administrer Jenkins->Gestion des plugins », installez si nécessaire les plugins suivants :

- EnvInject Plugin
- Jenkins Subversion Plug-in
- FitNesse Plugin
- Jenkins Sonar Plugin
- Jenkins Doxygen Plug-in

Notes :

- Certains plugins sont déjà installés voir onglet « Installés » : CVS, SVN, ...

- URL de Jenkins à l'IRAP: <http://amdakernel.cesr.fr>

7.4 MYSQL

Procédure à suivre :

- <http://www.howtoforge.com/installing-apache2-with-php5-and-mysql-support-on-centos-6.3-lamp>

Version PDF :



Installing Apache2
With PHP5 And MySQL

Résumé :

```
yum install mysql mysql-server //mysql.x86_64 0:5.1.61-4.el6
chkconfig --levels 235 mysqld on
/etc/init.d/mysqld start
mysql_secure_installation
```

7.5 SONAR

Prérequis :

- Le port 9000 de ce serveur d'intégration continu doit être accessible depuis une machine de consultation.

Procédure à suivre :

- <http://doc.ubuntu-fr.org/sonar>

Version PDF :



sonar -
Documentation Ubuntu

Résumé :

```
cd /opt
mkdir sonar
cd sonar
unzip $THIRD_PARTY_SOURCE/sonar-3.2.zip
sudo ln -s /opt/sonar/sonar-3.2/bin/linux-x86-64/sonar.sh /usr/bin/sonar
mysql -u root -p < $THIRD_PARTY_SOURCE/create_database.sql
```

Edition du fichier /opt/sonar/sonar-3.2/conf/sonar.properties pour utiliser une base mysql

Configuration en tant que service

Créez le fichier /etc/init.d/sonar et placez-y le contenu suivant:

```
#!/bin/sh
/usr/bin/sonar $*
```

puis attribuez-lui les droits d'exécution:

```
sudo chmod 755 /etc/init.d/sonar
```

et configurez-le en tant que service:

```
sudo update-rc.d sonar defaults 98 02 //Si problème /etc/init.d/sonar start
```

Remarques IRAP :

- le lancement du service s'effectue avec la commande: `/etc/init.d/sonar start`
- le port utilisé est le 9000 (configuration de base). Il est nécessaire d'ouvrir ce port dans le service iptables
- dans le fichier de configuration `'/opt/sonar/sonar-3.2/conf/sonar.properties'`, nous avons commenté la ligne `'sonar.embeddedDatabase.port: 9092'`, et décommenté la ligne `'sonar.jdbc.url'` de la section MySQL.
- Sonar est accessible en intranet à l'url suivante: <http://amdakernel.cesr.fr:9000>

7.6 PLUGIN CXX-SONAR

Procédure à suivre :

- <http://docs.codehaus.org/pages/viewpage.action?pagelid=185073817>

Version PDF



C++ Plugin
(Community) - Sonar

Résumé :

```
cp $THIRD_PARTY_SOURCE/sonar-cxx-plugin-0.1.jar /opt/sonar/sonar-3.2/extensions/plugins/
```

7.6.1 Sonar-runner

Elle est téléchargeable à l'url suivante:

<http://repository.codehaus.org/org/codehaus/sonar-plugins/sonar-runner/1.4/sonar-runner-1.4.zip>

Cette archive a été dezippée dans le répertoire : '/opt/sonar/'

7.6.2 Doxygen

```
sudo yum install doxygen
```

7.6.3 Valgrind

```
cd third-party/  
tar xvjf ../install/valgrind-3.8.1.tar.bz2  
cd valgrind-3.8.1/  
./autogen.sh  
./configure --prefix=/opt/local  
make  
sudo make install
```

7.6.4 CppCheck

```
sudo yum install tinyxml-2.6.1-1.el6.x86_64.rpm  
sudo yum install cppcheck-1.56-1.el6.x86_64.rpm
```

7.6.5 Gcov

Site d'origine du package :

- <http://pypi.python.org/pypi/gcovr>

```
cd /opt/  
tar xvzf $THIRD_PARTY_SOURCE/gcovr-2.4.tar.gz
```

7.6.6 Les jobs Jenkins de AKKA

Ces Jobs sont une base pour faire les vôtres. Il faudra sûrement y faire quelques modifications comme les chemins SVN par exemple.

```
cd /var/lib/jenkins  
tar xvzf jenkins-jobs.tgz
```

Après cette étape la prise en compte des jobs par Jenkins peut se faire à chaud via le menu « Jenkins -> Recharger la configuration à partir du disque »

Remarques IRAP, les modifications suivantes ont été apportées aux jobs :

- pour AMDA-Kernel_Integration :
 - Gestion de code source: SVN, avec l'URL du repository:
svn://cdpp1.cesr.fr/depotkernel/AMDAKernel
- pour AMDA-Kernel_NonRegression :
 - Gestion de code source: SVN, avec l'URL du repository:
svn://cdpp1.cesr.fr/depotkernel/AMDAKernel
- pour AMDA-Kernel_SONAR :
 - Gestion de code source: SVN, avec l'URL du repository:
svn://cdpp1.cesr.fr/depotkernel/AMDAKernel
 - Dans le script de la couverture des tests, il faut remplacer la référence locale à un répertoire d'AKKA '/home/g.schneller/gcovr-2.4/scripts/gcovr', par '/opt/gcovr-2.4/scripts/gcovr'