

---

# Audit BDGEOS

ÉTAT DES LIEUX ET  
RECOMMANDATIONS



13 MAI 2024

---

**AFIF DHAHRI / KARIM BEKOUCHI**

---

Version du document	Date	Diffusion
V1.0	13/05/2024	Interne uniquement
V1.1	06/06/2024	Interne uniquement

### Auteur

Prénom NOM	Fonction	Société	Date
Karim BEKOUCHI	Directeur de projets	Softeam	13/05/2024
Afif DHAHRI	Architecte solutions	Softeam	13/05/2024

## Sommaire

---

<b>1</b>	<b>Introduction .....</b>	<b>4</b>
<b>2</b>	<b>Méthodologie .....</b>	<b>4</b>
<b>3</b>	<b>Cadre de la mission .....</b>	<b>4</b>
<b>4</b>	<b>Organisation de l’audit.....</b>	<b>5</b>
<b>5</b>	<b>Résultat de l’audit .....</b>	<b>5</b>
<b>5.1</b>	<b>Analyse des choix d’architectures et de technologies.....</b>	<b>5</b>
5.1.1	Architecture globale .....	5
5.1.2	Architecture du POC .....	6
5.1.3	Serveur d’application .....	6
5.1.4	Serveur de cartographie .....	8
5.1.5	Serveur de base de données .....	9
5.1.6	Serveur de fichiers .....	10
<b>5.2</b>	<b>Analyse du code.....</b>	<b>10</b>
5.2.1	Code PHP .....	11
5.2.2	Templates Twig:.....	12
5.2.3	Code C++.....	12
5.2.4	Code SQL .....	13
5.2.5	Les Tests .....	15
<b>5.3</b>	<b>Documentation.....</b>	<b>15</b>
<b>5.4</b>	<b>Analyse d’infrastructure .....</b>	<b>15</b>
<b>6</b>	<b>Recommandations .....</b>	<b>16</b>
<b>6.1</b>	<b>QuickWins .....</b>	<b>16</b>
<b>6.2</b>	<b>Long terme .....</b>	<b>16</b>
<b>7</b>	<b>ROI.....</b>	<b>17</b>

# 1 Introduction

Cet audit a été demandé par Shom sur le projet BDGEOS à la société Softeam afin de faire une étude de l'architecture technique et applicative et déterminer l'obsolescence des composants en vue d'une migration de l'application. Il est demandé aussi de faire une analyse technique du portage de la base ORACLE vers POSTGRES

BDGEOS est une Base de Données spécialisée dans l'archivage des données GÉOphysiques du SHOM.

Elle est composée d'une base de données Oracle contenant les données géophysiques de gravimétrie et magnétisme et d'un IHM permettant la manipulation des données ainsi que d'un serveur de fichiers hébergeant les fichiers d'import/export.

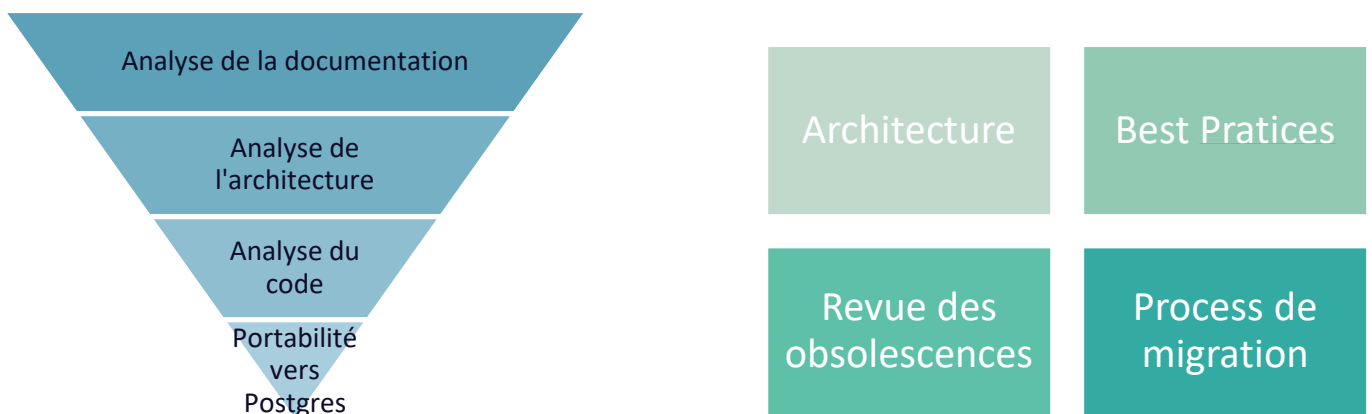
## 2 Méthodologie

Un kick-off a été réalisé afin d'identifier les attentes et présenter la méthodologie.

L'audit a ensuite été réalisé dans un premier temps par une analyse des documents.

Puis l'analyse de l'architecture actuelle afin de mettre en place un prototype à l'image de l'application afin de simuler la future migration et s'assurer de la compatibilité entre les composants proposés.

Par la suite, une analyse du code et particulièrement de la base de données afin d'étudier la faisabilité d'un portage d'ORACLE vers POSTGRES.



## 3 Cadre de la mission

Les objectifs de cet audit sont multiples et résumés comme suit :

- Assurer l'adéquation entre l'architecture du code et les montées de version des composants.
- Etudier la faisabilité du portage de la base de données d'ORACLE vers POSTGRES

- Analyser la qualité du code
- Proposer une méthodologie de migration
- Présenter le ROI attendu
- Etudier la structure de l'infra pour permettre une maintenance simple

## 4 Organisation de l'audit

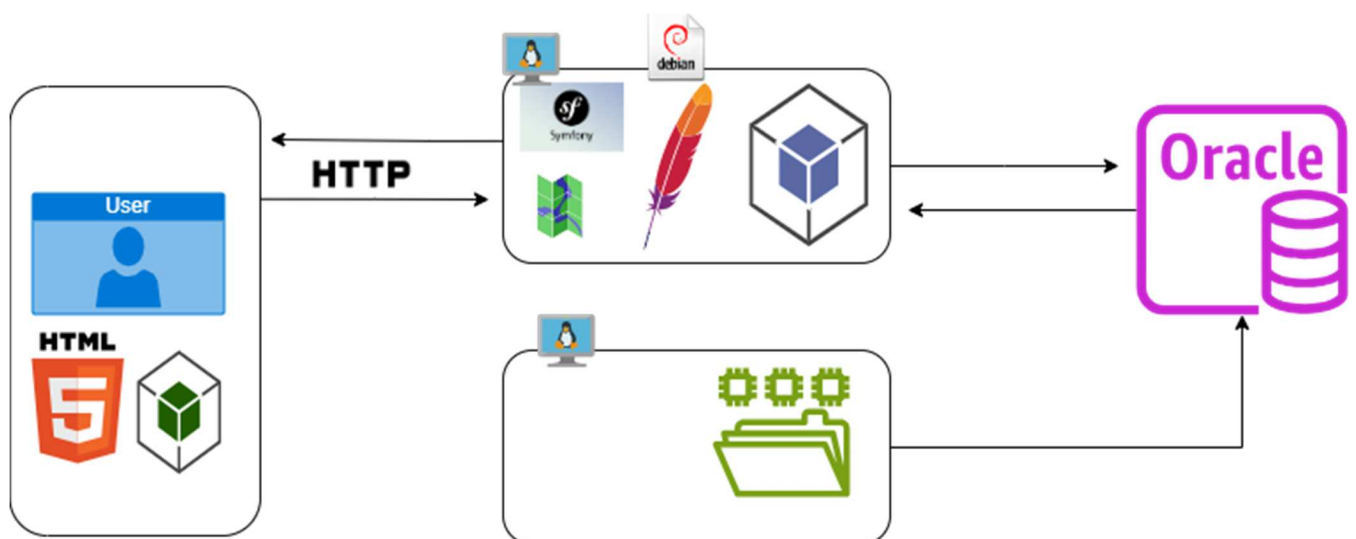
L'audit est organisé en plusieurs phases :

1. Étude de la documentation
2. Étude
  - a. Analyse d'architecture
  - b. Étude du code
  - c. Méthodologie de migration
3. Recommandations
  - a. Quick Win
  - b. Recommandations long terme

## 5 Résultat de l'audit

### 5.1 Analyse des choix d'architectures et de technologies

#### 5.1.1 Architecture globale



L'architecture globale est composée d'un serveur d'application, un serveur de base de données et un serveur de fichiers.

Les différents organes de l'architecture répondent parfaitement au besoin.

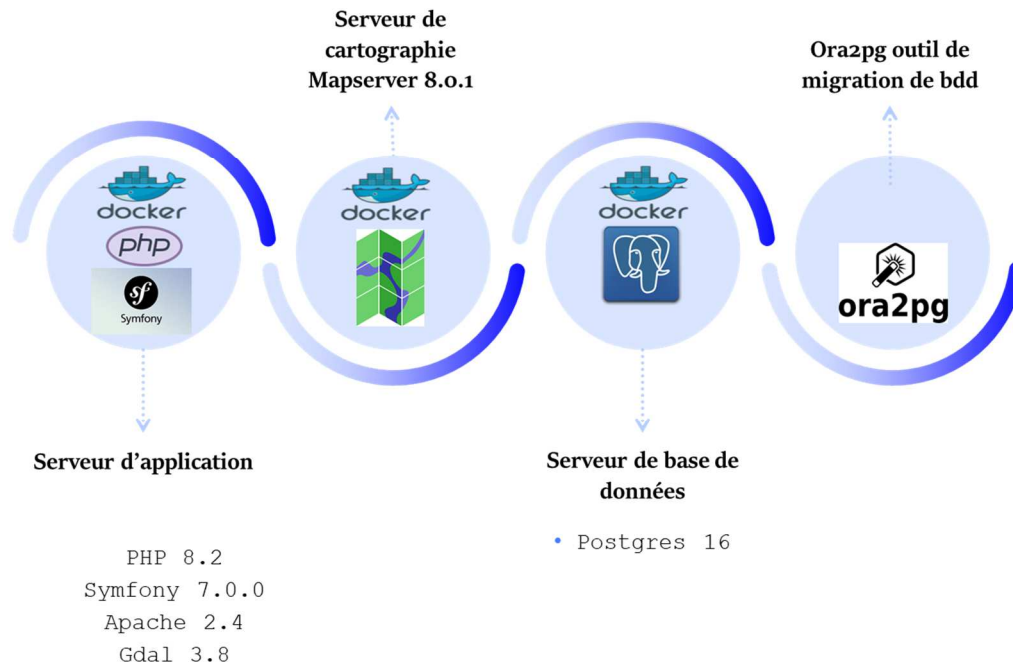
### 5.1.2 Architecture du POC

Nous avons opté pour la conteneurisation des différentes briques de l'architecture applicative afin d'en simplifier l'usage.

Nous avons utilisé docker et avons construit les containers suivants pour le besoin de l'audit :

- Un container qui comporte apache, php, symfony qui va héberger le code php de l'application
- Un container qui contient le serveur de cartographie mapserver
- Un container de base de données oracle 11g
- Un container de base de données Postgres qui est la cible de la migration
- Un container de l'utilitaire de migration ora2pg qui nous servira à étudier la faisabilité du portage d'Oracle vers Postgres.

- Monter un environnement en mode conteneurisé avec docker
- Identifier les versions cibles des frameworks de l'application



### 5.1.3 Serveur d'application

L'application actuelle est hébergée sur une VM Debian squeeze 6.0.1 qui n'est plus supportée depuis le 29/02/2016. Nous cibons la version 12.5 releasedée en février 2024.

Voici les versions actuelles des différents composants du serveur d'application ainsi que la version cible de chaque composant.

Logiciel	Version	Fin de support	Vulnérabilités	Cible
PHP	5.3.3	2018	260	8.2
Symphony	2.0.0	2013		7.0
Apache	2.2.16	2018	57	2.4.58

Le passage d'Apache 2.2 à 2.4 implique une modification du fichier de configuration pour la gestion des autorisations. Voici un tableau récapitulatif des modifications à apporter.

Action	Apache 2.2	Apache 2.4
<b>To deny all the requests</b>	Order deny,allow Deny from all	Require all denied
<b>To allow all the requests</b>	Order allow,deny Allow from all	Require all granted
<b>To deny all the requests from all the hosts except example.com</b>	Order Deny,Allow Deny from all Allow from example.com	Require host example.com

Le passage de Symfony 2 à Symfony 7 implique une revue de la structure du projet.

Par ailleurs l'application actuelle est organisée en 2 bundles un pour la cartographie et un 2eme pour la gestion. On abandonnera cette structure de bundle qui n'est plus supportée par Symfony 7.

Voici la nouvelle structure d'un projet Symfony. A noter le remplacement du répertoire web par public et la suppression du répertoire app.

## ▼ hanamikoji ~/Documents/ht

- ▶ bin
- ▶ config
- ▶ public
- ▶ src
- ▶ templates
- ▶ var
- ▶ vendor
- .env
- .env.dist
- .gitignore
- composer.json
- composer.lock
- symfony.lock

Le répertoire templates doit contenir tous les fichiers twig contenus actuellement dans src\CS\BdgeosCoreBundle\Resources\views

Le répertoire src aura l'arborescence suivante

- ▼ src
  - ▶ Controller
  - ▶ Entity
  - ▶ Migrations
  - ▶ Repository
  - Ⓢ Kernel.php

Le répertoire Repository correspond au répertoire DAO actuel.

Symfony utilise désormais composer pour la gestion des dépendances. Il faudra se baser sur le fichier deps actuel pour construire le composer.json.

### 5.1.4 Serveur de cartographie

MapServer est utilisé en mode CGI (Common Gateway Interface). La CGI est une interface qui va lancer un programme exécutable par l'intermédiaire d'un serveur web HTTP comme Apache.

Pour la réalisation du POC, le choix de séparer le serveur de cartographie du serveur d'application est pour alléger le premier conteneur et améliorer les performances.



Nous proposons de migrer Mapserver de la version 5.6.6 vers la version 8.0.1. Il faudra aussi migrer la librairie GDAL de 1.8 à 3.8 pour la compilation de mapserver.

Logiciel	Version	Fin de support	Vulnérabilités	Cible
Mapserver	5.5.6	2021	5	8.0.1
GDAL	1.8.0	2012	4	3.8.1

Cette migration implique une revue des fichiers MAPFILE. Voici les plus importantes modifications qui concernent BDGEOS :

- Supprimer imagetype, imagequality de l'objet MAP
- Supprimer dump et utiliser METADATA à la place
- Mettre STYLE dans SYMBOL
- Supprimer transparency, maxscale, minscale de l'objet CLASS
- Supprimer COLOR [r] [g] [b] et utiliser CLASS STYLEs à la place
- Remplacer FILTER par PROCESSING NATIVE\_FILTER.
- Rajouter "wms\_enable\_request" "\*" dans metadata pour activer toutes les requêtes wms.

### 5.1.5 Serveur de base de données

Le but de l'audit est d'étudier la faisabilité du portage de la base ORACLE 11g vers POSTGRES 16.

La base actuelle ORACLE 11 g est composée des objets suivants :

- 63 tables, 341 colonnes dont 73 indexées.
- 43 séquences
- 4 vues
- 15 fonctions
- 1 package avec 38 procédures et 5 fonctions

La migration de la plupart des objets se fait avec l'outil ora2pg

Il faut configurer la base oracle dans le fichier de conf ora2pg (chaîne de connexion).

Il faudra un lancement de l'outil ora2pg par type d'objet à migrer (variable TYPE à définir dans le fichier ora2pg.conf).

Voici les différents types utilisés lors de la migration de la base BDGEOS :

- # TABLE Export tables, constraints, indexes, ...
- # PACKAGE Export packages

---

#	INSERT	Export data from table as INSERT statement
#	VIEW	Export views
#	GRANT	Export grants
#	SEQUENCE	Export sequences
#	TRIGGER	Export triggers
#	FUNCTION	Export functions
#	PROCEDURE	Export procedures
#	TABLESPACE	Export tablespace

La base BDGEOS implémente les données géo spatiales d'Oracle à travers des colonnes de type SDO\_GEOMETRY pour permettre la représentation des lots sous forme de lignes et de type SDO\_PC pour la représentation des modèles raster. La base BDGEOS utilise aussi des fonctions du package oracle SDO\_UTIL ainsi que d'autres procédures SDO spécifiques à Oracle.

Il existe aussi des procédures stockées en java dans la base de données (OSCOMMAND\_RUN et HOST) qui permettent de lancer des commandes système sur les VM hébergeant la base de données.

L'import/export des données de la base se fait avec datapump spécifique à Oracle.

Voici les différentes extensions à utiliser dans la future base postgres afin de remplacer toutes les fonctionnalités oracle spécifiques de BDGEOS :

- Postgis
- Pointcloud
- Postgis\_pointcloud
- Pljava ou Plpython pour remplacer les procédures stockées Host et OsCommand\_Run.
- File\_fdw pour remplacer les external table.

### 5.1.6 Serveur de fichiers

Le serveur de fichiers permet de stocker les fichiers d'import/export de la base de données.

Il faudra procéder à la migration de l'OS de la VM de la version 10 à 12 de Debian.

Cette migration se résume à la copie des fichiers et leur arborescence vers la nouvelle VM.

## 5.2 Analyse du code

Le but de cette partie est de relever tous les points d'attention sur le code applicatif de toutes les briques de l'application en vue de leur migration vers les versions cibles.

La liste des recommandations n'est pas exhaustive mais permet d'énumérer les évolutions importantes à apporter au code actuel afin d'assurer la migration de l'application.

### 5.2.1 Code PHP

Le code respecte la structure MVC avec une couche modèle, une couche repository, une couche service et une couche controller.

L'application comporte 114 fichiers php avec 24946 lignes de code.

Le passage de la version 5.3.3 à la version 8.2 de PHP implique une revue importante du code actuel pour prendre en compte toutes les dépréciations et nouveautés entre les 2 versions.

Le passage de Symfony 2 à Symfony 7 implique aussi le remplacement d'un certain nombre de bibliothèques. Voici une liste des composants utilisés dans BDGEOS et qui ont été dépréciés avec les différentes mises à jour de Symfony.

Composant déprécié	Nouveau composant
<b>swiftmailer</b>	Symfony/mailer
<b>Twig-extension</b>	twig/string-extra
<b>SensioFrameworkExtraBundle</b>	Symfony/twig-bundle
<b>JMSSecurityExtraBundle</b>	Symfony/security-bundle
<b>AsseticBundle</b>	Symfony\webpack-encore-bundle

Par ailleurs, des changements structurels de PHP impliquent une revue de tous les fichiers PHP.

PHP utilise désormais les attributs à la place des annotations pour les routes et les templates.

Cela implique une modification de la syntaxe dans tous les controllers de l'application

Ancienne Syntaxe	Nouvelle syntaxe
<b>use Symfony\Component\Routing\Annotation\Route</b>	use Symfony\Component\Routing\Attribute Route
<b>@Route("path", name= "name_route")</b>	<b>#[Route('path', name: 'name_route')]</b>
<b>use Sensio\Bundle\FrameworkExtraBundle\Configuration \Template</b>	use Symfony\Bridge\Twig\Attribute\Templ ate
<b>@Template("CSBdgeosCoreBundle:Bilan:index.html.t wig")</b>	<b>#[Template('Bilan/index.html.twig')]</b>

Les méthodes has et get pour récupérer les services dans les controllers sont dépréciées et Symfony utilise désormais l'injection des objets dans les constructeurs.

---

La session est récupérable à partir de la request et non avec le `get('session')`.

L'entitymanager est récupéré avec l'injection de `EntityManagerInterface` dans les controllers.

ORM n'utilise plus les annotations mais les attributs, il faudra remplacer `@ORM` par des `#[ORM]` et ce dans tous les fichiers entity.

Les méthodes `fetch` ont été dépréciées et il faudra les remplacer avec les nouvelles méthodes. Par exemple, remplacer `fetchAll` par `fetchAllAssociative`, `fetchColumn` par `fetchOne`.

Le merge cascade du `onetomany` a été dépréciée

PHP exige désormais la spécification du type de retour des fonctions. Il faudra donc rajouter le type de retour dans toutes les fonctions héritées au moins, pour éviter les erreurs de compilation.

Certaines classes ont été dépréciées et remplacées par de nouvelles tel que `RoleInterface`, `AdvancedUserInterface`.

Certaines méthodes de classes user ont été dépréciées. `loadUserByUsername` devient `loadUserByIdentifier` et `getByUsername` devient `getUserIdentifier`.

### 5.2.2 Templates Twig:

L'application comporte 109 templates twig. Voici les principales modifications à apporter :

- Les chemins des templates twigs n'utilisent plus : comme séparateur mais plutôt `/`. Il faudra revoir tous les chemins vers les templates dans les controllers ainsi que lors des `include` de sous templates.
- La directive `render` change de syntaxe `render {{` au lieu de `{%`
- La gestion des variables globales dans les templates twig a changé et il faudra utiliser `app_global` à la place(exemple `{app_global()['dateFormat']}` au lieu de `dateFormat`)

### 5.2.3 Code C++

La génération des fichiers raster de type Geotiff se fait avec un programme C++ se basant sur les librairies GDAL. Il lance les exécutables `gdal_translate` et `gdaladdo` de la librairie GDAL.

La montée de version des bibliothèques GDAL de 1.8 à 3.8 n'a pas d'impact sur le code actuel. Il suffira de recompiler le programme avec la nouvelle version de la librairie GDAL.

Le passage d'Oracle vers Postgres nécessite d'abandonner les librairies OCI de connexion à la base oracle et utiliser `libpqxx` à la place pour la connexion à la base postgres.

Par contre, il est à noter que postgres propose un utilitaire `raster2pg` permettant de charger des fichiers rasters en base qui est plus simple à utiliser.

Des warnings sur la conversion implicite dépréciée en C++ sont à noter et nécessitent une modification mineure du code lors de la déclaration des `char *` en rajoutant `const`.

**generationGeoTIFF.c:119:32: warning: ISO C++ forbids converting a string constant to 'char\*' [-Wwrite-strings]**

Un autre warning sur un sprintf nécessite une modification du code avec un rajout de %d dans le format.

**generationGeoTIFF.c:1161:24: warning: too many arguments for format [-Wformat-extra-args]**

#### 5.2.4 Code SQL

L'application contient 3631 lignes de code PL/SQL répartis entre 15 procédures et un package. L'analyse du code était orientée sur la faisabilité du passage d'Oracle vers Postgres.

La plupart des objets de la base ont pu être migrés vers Postgres avec l'outil ora2pg moyennant des corrections manuelles.

Il faut tout de même revoir les types des variables numériques lors du passage d'Oracle vers postgres.

En effet, Oracle ne gère pas les types natifs sql (smallint, integer, bigint, float, real, double precision). Il gère un type Number qui équivaut au type numeric de Postgres.

Les types natifs sont plus rapides dans Postgres et sont donc à privilégier d'où le besoin de la revue manuelle de la migration (par exemple les méthodes f\_leve\_intersect et f\_lot\_intersect il faudra mettre les paramètres correspondant aux longitudes/latitudes en double precision au lieu de bigint qui a été proposé par ora2pg).

Oracle ne gère pas le type boolean, certains champs ont été donc déclarés de type int ce qui pose problème à ORM où les champs ont été déclarées en boolean et lors du passage à postgres ORM s'attend à un type boolean en base.

Oracle a un seul type date contrairement à Postgres qui fait la différence entre une date avec ou sans heure. Dans BDGEOS toutes les dates sont avec heure donc il faut les migrer en timestamp dans postgres.

Des modifications sont à apporter au niveau des requêtes sql post migration :

- from dual à supprimer
- Syntaxe nextval modifié
- sysdate à remplacer avec current\_timestamp

Une étude plus approfondie a été menée sur les types et les fonctions géo spatiales d'oracle utilisés dans BDGEOS. Voici la liste de celles utilisées dans BDGEOS et leur équivalent Postgres.

Oracle	Postgres
<b>SDO_GEOMETRY (Type)</b>	GEOMETRY
<b>SDO_GEOMETRY (constructeur)</b>	ST_MAKELINE

<b>SDO_PC</b>	P cpatch
<b>SDO_UTIL.SIMPLIFY</b>	ST_SimplifyPreserveTopology
<b>SDO_AGGR_CONCAT_LINES</b>	st_union
<b>SDO_RELATE(geom,VarBox,'mask=anyinteract')</b>	st_intersects
<b>CLIP_PC</b>	Pc_intersection
<b>getvertices</b>	Développement spécifique à faire

La migration des données géo spatiales d'Oracle vers Postgres peut se faire avec l'outil OGROGR.

Le prototype utilisé lors de la migration ne comporte pas le module SDO\_PC d'oracle pour des raisons de licence. L'étude de la migration reste donc théorique.

Les graphiques en nuage de points permettent d'interpréter la relation entre plusieurs variables, et de déterminer si l'une des variables peut servir à prévoir une autre ou si les variables changent par elles-mêmes.

L'utilisation de nuages de point permet d'alléger et simplifier le stockage des données géospatiales dans la base de données en réduisant le nombre de lignes dans les tables. En effet, nous ne stockons pas unitairement les points mais par paquet (dans BDGEOS un bloc contient 10000 points).

Dans BDGEOS, les données des modèles sont stockées sous forme de point\_cloud SDO\_PC (type oracle pour nuage de point).

Il existe une colonne de type SDO\_PC dans la table BASE pour stocker les nuages des points des modèles.

La table BLKTAB contient les blocs de point cloud sa création se fait ainsi

```
create table BLKTAB as select * from mdsys.sdo_pc_blk_table NOLOGGING;
```

sdo\_pc\_pkg\_init permet de créer les metadata d'un point cloud et qui sera inséré par la suite dans la table BASE.

sdo\_pc\_pkg.create\_pc permet de créer la table de blocs correspondant au point cloud.

Lors de l'export des modèles en csv, on utilise la méthode PC\_CLIP de Oracle pour récupérer les blocks correspondants à la partie du modèle à exporter.

Dans postgres il faudra utiliser les extensions pointcloud et pointcloud\_postgis.

La création de la table de blocs se fait ainsi

```
CREATE TABLE BASE ( id SERIAL PRIMARY KEY, pc PCPATCH)
```

La création des blocs dans postgres peut se faire manuellement en utilisant ST\_Collect pour créer une collection de PC\_POINT puis de créer le pcpatch en effectuant un group by.

---

Cette méthode permettrait de minimiser l'impact sur le reste du code et garder le même procédé actuel par contre il est probable qu'elle pose des problèmes de performance.

Postgres, recommande d'utiliser le format LAS de fichiers modèles pour les charger et les exporter. L'import/export se fait avec des pipelines PDAL.

Il faudra réétudier le besoin métier sur la partie gestion des modèles (import/export) afin de l'adapter au mieux à Postgres.

### **5.2.5 Les Tests**

Il n'existe pas de tests unitaires dans BDGEOS. Il faudra mettre en place une stratégie de tests afin de simplifier les futures migrations ou évolutions de l'application.

## **5.3 Documentation**

Il faudra mettre à jour le manuel d'installation de l'application ainsi que le dossier de programmation une fois la migration réalisée.

Le manuel Utilisateur n'est pas sujet à modification.

## **5.4 Analyse d'infrastructure**

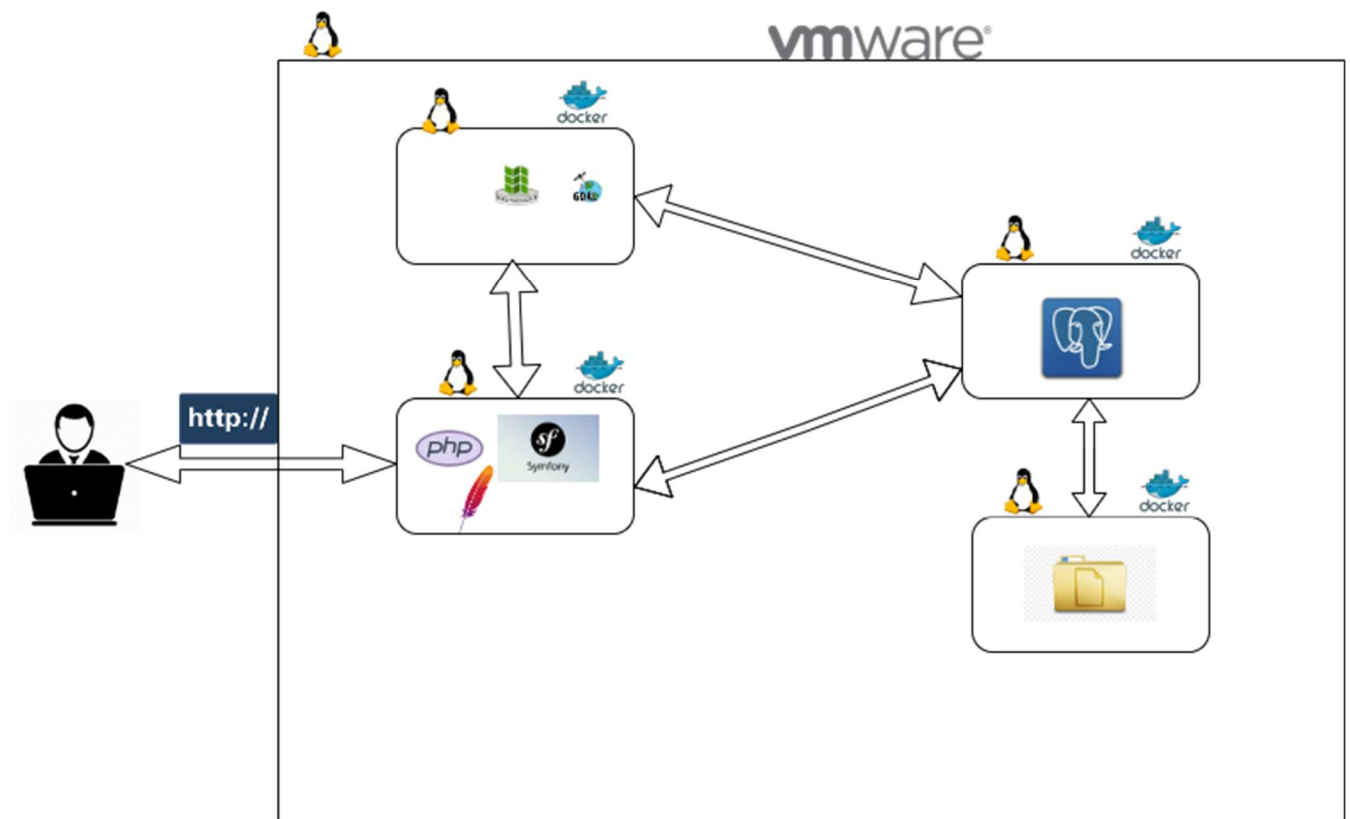
Nous proposons une nouvelle architecture de l'application qui permettra une meilleure maintenabilité ainsi qu'une amélioration des performances.

Nous souhaitons ainsi séparer la couche cartographique de BDGEOS de l'IHM afin d'améliorer les performances et éviter de saturer le serveur d'application.

Par ailleurs, une architecture avec des conteneurs pour chaque brique de BDGEOS faciliterait l'installation de l'application ainsi que les futures mises à jour des OS et framework.

Nous aurons ainsi 4 containers :

- Container avec mapserver et librairie GDAL
- Container avec serveur apache, symfony et php
- Container avec un serveur de base de données postgres
- Container pour héberger les fichiers d'import/export



### Résumé des constatations

- Assurer l'adéquation entre l'architecture du code et le besoin : **BONNE**
- Analyser la qualité du code : **BONNE**
- Valider la portabilité ORACLE vers POSTGRES: **FAISABLE** (moyennant revue de certains modules import/export)
- Analyser la chaine CI/CD : **A Mettre en place**
- Analyser la stratégie de test :(il existe un plan de validation qui peut servir comme base pour le développement des tests unitaires) **A Mettre en place**

## 6 Recommandations

### 6.1 QuickWins

- Revoir les requêtes SQL en base et dans le code PHP pour améliorer la performance
- Séparer la cartographie de l'IHM pour décharger le serveur d'application

### 6.2 Long terme

- Revoir l'architecture et opter pour une conteneurisation des différents modules afin d'en simplifier la maintenance



- 
- Mettre en place une authentification forte à l'application (OAuth2) afin d'améliorer la sécurité de l'application (utiliser l'annuaire LDAP de l'entreprise le cas échéant)

## 7 ROI

Le passage d'un serveur de base de données Oracle à Postgres permet de gagner en terme de coût (licence vs open source), en simplicité et en flexibilité et cela sans perdre en performance et en fonctionnalités compte tenu de l'utilisation de BDGEOS de la base de donnée.

Le coût de cette portabilité n'est pas très important étant donné l'existence d'outils de migration open source et fiables.

La mise à jour des versions des différents composants permet de faire face à de nombreuses vulnérabilités de sécurité sur les versions actuelles datant de 2011.

Le passage à Symfony 7 facilitera désormais les futures migrations grâce aux outils de mise à jour intégrés à Symfony.