Projet XXX

Dossier d’Architecture
volet infrastructure

APPROBATION DU DOCUMENT			
Organisation	**Nom (fonction)**	**Date**	**Visa**

DIFFUSION	
Destinataire	**Organisation**

SUIVI DES MODIFICATIONS			
Version	**Date**	**Auteurs**	**Changements**

Sommaire

1. INTRODUCTION

Ce document fourni le point de vue infrastructure de l'application. Se référer aux volets joints pour les points de vue applicatifs, sécurité et développement. Un référentiel joint liste les points à statuer et les hypothèses prises.

Ce point de vue est aussi souvent appelé « point de vue technique » et concerne l'infrastructure : serveurs, réseaux, systèmes d'exploitation, bases de données, intergiciels (middleware), etc. : ce qui permet à l'application de s'exécuter sans être l'application elle-même.

1.1. DOCUMENTATION DE RÉFÉRENCE

Mentionner ici les documents d'architecture de référence (mutualisés). Ce document ne doit en aucun cas reprendre leur contenu sous peine de devenir rapidement obsolète et impossible à maintenir.

N°	Version	Titre/URL du document	Détail
1	*1.0*	*XXX_Regles_sauvegardes*	*Régles de sauvegardes*

1.2. GLOSSAIRE

Note : par souci de concision, nous ne détaillons ici que les termes et acronymes spécifiques à l'application. Pour les définitions générales, veuillez vous référer au glossaire d'entreprise.

Terme	Définition
Replica set	*Cluster actif/actif MongoDB*

2. CONTRAINTES

2.1. DISPONIBILITÉ NOMINALE

Les éléments ici fournis pourront servir de base à l'OLA (Operationnal Level Agreement). Ce chapitre est assez pédagogique car il rappelle la disponibilité plafond envisageable : la disponibilité finale de l'application ne pourra être qu'inférieure.

2.1.1 Présence nominale des exploitants et astreintes

Heures de présence des exploitants, astreintes normales sur le SI...
Comme toute application hébergée au datacenter XXX, l'application disposera de la présence d'exploitants de 7h à 20h jours ouvrés. Aucune astreinte n'est prévue.

2.1.2 Durées d'intervention externes

Lister ici les durées d'intervention des prestataires matériels, logiciels, électricité, telecom...
Le remplacement de support matériel IBM sur les lames BladeCenter est assuré en 4h de 8h à 17h, jours ouvrés uniquement.

2.1.3 Interruptions programmées

Lister ici les interruptions à prévoir pour raison d'exploitation (et non sur incident).
Exemple : suite aux mises à jour de sécurité de certains packages RPM (kernel, libc…), les serveurs RHEL seront redémarrés automatiquement la nuit du mercredi suivant la mise à jour. Ceci entraînera une indisponibilité de 5 mins en moyenne 4 fois par an.

2.1.4 Niveau de service du datacenter

Donner ici le niveau de sécurité du datacenter selon l'échelle Uptime Institute (Tier de I à IV).
La plupart des datacenters sont de niveau I ou II.

Choix	Tier	Caractéristiques	Taux de disponibilité	Indisponibilité statistique annuelle	Maintenance à chaud possible	Tolérance aux pannes
	Tier I	Non redondant	unité 99,671 %	28,8 h	Non	Non
X	Tier II	Redondance partielle	99,749 %	22 h	Non	Non
	Tier III	Maintenabilité	99,982 %	1,6 h	Oui	Non
	Tier IV	Tolérance aux pannes	99,995 %	0,4 h	Oui	Oui

Source : wikipedia

2.1.5 Plan de Reprise ou de Continuité d'Activité (PRA / PCA)

PRA comme PCA répondent à un risque de catastrophe sur le SI (catastrophe naturelle, accident industriel, incendie...).

Un PCA permet de poursuivre les activités critiques de l'organisation (en général dans un mode dégradé) sans interruption notable, voir norme ISO 22301. Le concept est réservé aux organisations très matures car il exige des dispositifs techniques coûteux et complexes (réplication des données au fil de l'eau par exemple).

Un PRA permet de reprendre l'activité suite à une catastrophe après une certaine durée de restauration. Il exige au minium un doublement du datacenter.

Décrire entre autres :

- *Les matériels redondés dans le second datacenter, nombre de serveurs de spare, capacité du datacenter de secours par rapport au datacenter nominal ;*
- *Pour un PRA les dispositifs de restauration (OS, données, applications) prévues ;*
- *Pour un PRA, donner le Recovery Time Objective (durée maximale admissible de rétablissement en h) et Recovery Point Objective (durée maximale admissible de données perdues en h) de l'organisation ;*
- *Pour un PCA les dispositifs de réplication de données (synchrone ? fil de l'eau ? Combien de transactions peuvent-être perdues ?) ;*
- *Présenter la politique de failback (réversibilité) : doit-on rebasculer vers le premier datacenter ? Comment ?*
- *Comment sont organisés les tests de bascule à blanc ? Avec quelle fréquence ?*

2.2. HÉBERGEMENT

- *où sera hébergé cette application ? (datacenter "on premises" ? Cloud ? IaaS ? PaaS ? …)*
- *qui administrera cette application ? Administré en interne ? Sous-traité ? Pas d'administration (PaaS) … ?*

Exemple 1: Cette application sera hébergé en interne dans le datacenter de Nantes (seul à assurer la disponibilité de service exigée) et il sera administré par l'équipe XXX de Lyon.

Exemple 2 : Étant donné le niveau de sécurité très élevé de l'application, la solution devra être exploitée uniquement en interne par des agents assermentés et hors sous-traitance. Pour la même raison, les solutions de cloud sont exclues.

Exemple 3 : Étant donné le nombre d'appels très important de cette application vers le référentiel PERSONNE, elle sera colocalisée avec le composant PERSONNE dans le VLAN XXX.

2.3. COÛTS

Lister les éventuelles limites budgétaires.

Exemple 1 : les frais de services Cloud AWS ne devront pas dépasser 5K€/ an pour ce projet.

3. EXIGENCES

Contrairement aux contraintes qui fixaient le cadre auquel toute application devait se conformer, les exigences non fonctionnelles sont données par les commanditaires du projet (MOA en général). Prévoir des interviews pour les déterminer. Si certaines exigences ne sont pas réalistes, le mentionner dans le document des points non statués.

3.1. PLAGES DE FONCTIONNEMENT

On liste ici les plages de fonctionnement principales (ne pas trop détailler, ce n'est pas un plan de production). Penser aux utilisateurs situés dans d'autres fuseaux horaires.

Les informations données ici serviront d'entrants au SLA de l'application.

No plage	Détail	Intervalle de temps
1	*Ouverture Intranet aux employés de métropole*	*De 8H00-19H30 heure de Paris , 5J/7 jours ouvrés*
2	*Plage batch*	*De 21h00 à 5h00 heure de Paris*
3	*Ouverture Internet aux usagers*	*24 / 7 / 365*
4	*Ouverture Intranet aux employés de Nouvelle Calédonie*	*De 5h30-8h30 heure de Paris, 5J/7 jours ouvrés*

3.2. TEMPS DE RÉPONSE

3.2.1 Temps de Réponse des IHM

Si les clients accèdent au système en WAN (Internet, VPN, LS …), préciser que les exigences de TR sont données hors transit réseau car il est impossible de s'engager sur la latence et le débit de ce type de client. Dans le cas d'accès LAN, il est préférable d'intégrer le temps réseau, d'autant que les outils de test de charge vont déjà le prendre en compte.

Les objectifs de TR sont toujours donnés avec une tolérance statistique (90éme centile par exemple) car la réalité montre que le TR est très fluctuant car affecté par un grand nombre de facteurs.

Inutile de multiplier les types de sollicitations (en fonction de la complexité de l'écran par exemple , ce type de critère n'a plus grand sens aujourd'hui, particulièrement pour une application RIA).

Types de sollicitations :

Type de sollicitation	Bon niveau	Niveau moyen	Niveau insuffisant
Chargement d'une page	*< 0,5 s*	*< 1 s*	*> 2 s*
Opération métier	*< 2 s*	*< 4 s*	*> 6 s*
Édition, Export, Génération	*< 3 s*	*< 6 s*	*> 15 s*

Acceptabilité des TR :

Le niveau de respect des exigences de temps de réponse est :

Bon si :	• *au moins 90 % des temps de réponse sont bons* • *au plus 2% des temps de réponse sont insuffisants*
Acceptable si :	• *au moins 80 % des temps de réponse sont bons* • *au plus 5 % des temps de réponse sont insuffisants*

En dehors de ces intervalles, l'application devra être optimisée et repasser en recette puis être soumise à nouveau aux tests de charge.

3.2.2 Durée d'exécution des batchs

Préciser ici dans quel intervalle de temps les traitements par lot doivent s'exécuter.

Exemple 1 : La fin de l'exécution des batchs étant un pré-requis à l'ouverture du TP, ces premiers doivent impérativement se terminer avant la fin de la plage batch définie en 3.1.

Exemple 2 : le batch mensuel XXX de consolidation des comptes doit s'exécuter en moins de 4 J.

Exemple 3 : les batchs et les IHM pouvant fonctionner en concurrence, il n'y a pas de contrainte stricte sur la durée d'exécution des batchs mais pour assurer une optimisation de l'infrastructure matérielle, on favorisera la nuit pendant laquelle les sollicitations IHM sont moins nombreuses.

3.3. DISPONIBILITÉ

Nous listons ici les exigences de disponibilité. Les mesures techniques permettant de les atteindre seront données dans l'architecture technique de la solution.

Les informations données ici serviront d'entrants au SLA de l'application.

Attention à bien cadrer ces exigences car une MOA a en général tendance à demander une disponibilité très élevée sans toujours se rendre compte des implications. Le coût et la complexité de la solution augmente exponentiellement avec le niveau de disponibilité exigé. L'architecture physique, technique voire logicielle change complètement en fonction du besoin de disponibilité (clusters d'intergiciels voire de bases de données, redondances matériels coûteuses, architecture asynchrone, caches de session, failover ...). Ne pas oublier également les coûts d'astreinte très importants si les exigences sont très élevées. De la pédagogie et un devis permettent en général de modérer les exigences.

A titre d'ordre de grandeur, la haute disponibilité (HA) commence à deux neufs (99%), c'est à dire 87h et 36 mins d'indisponibilité par an ;

3.3.1 Disponibilité exigée par plage de fonctionnement

La liste des plages de fonctionnement est disponible en 3.1.

La disponibilité exigée ici devra être en cohérence avec les contraintes de disponibilités du SI données en 2.1.

No Plage	Disponibilité attendue	Indisponibilité programmée	Indisponibilité non programmée
1	*99.72 %*	*0 %*	*0.28% (2 h/mois)*
2	*94.72 %*	*5% d'interruption programmée - (8,2 h / semaine pour sauvegarde à froid) - 0.2 h / semaine en moyenne pour mise à jour système*	*0.28% (2 h/mois) d'interruption non programmée*

3.3.2 Mode dégradé acceptable

Préciser l'impact maximal accepté sur les temps de réponse lors d'une panne

Exemple 1 (perte d'un nœud d'un cluster) : Les serveurs devront être dimensionnés pour être chacun en mesure d'assurer le fonctionnement de l'application et de limiter l'augmentation des temps de réponse à 20 %.

Préciser les modes dégradés applicatifs envisagés.

Exemple2 (perte d'un service) : Le site `monsite.com` devra pouvoir continuer à prendre les commandes en l'absence du service de gestion de l'historique des commandes.

3.4. ROBUSTESSE

La robustesse du système indique sa capacité à ne pas produire d'erreurs lors d'événements exceptionnels comme une surcharge ou la panne d'un des composants.

Cette robustesse s'exprime en valeur absolue par unité de temps : nombre d'erreurs (techniques) par mois, nombre de messages perdus par an...

Attention à ne pas être trop exigent sur ce point car une grande robustesse peut impliquer la mise en place de système à tolérance de panne complexes, coûteux et pouvant aller à l'encontre des capacités de montée en charge voire même de la disponibilité (pour plus de détail, voir 5).

Exemple 1 : pas plus de 0.001% de requêtes en erreur

Exemple 2 : l'utilisateur ne devra pas perdre son panier d'achat même en cas de panne
-> attention, ce type d'exigence impacte l'architecture en profondeur, voir 5.

Exemple 3 : le système devra pouvoir tenir une charge trois fois supérieure à la charge moyenne (voir 3.5.2) avec un temps de réponse de moins de 10 secondes au 95éme centile.

3.5. VOLUMÉTRIE

La volumétrie ici décrite permettra le dimensionnement initial de la solution. Il est crucial de récupérer un maximum d'informations issues de la production plutôt que des estimations car ces dernières se révèlent souvent loin de la réalité. C'est d'autant plus difficile s'il s'agit d'un projet sans précédent, prévoir alors une marge importante.

Les informations données ici serviront d'entrants au SLA de l'application.

3.5.1 Volumétrie statique

Lister ici les besoins en stockage de chaque composant une fois l'application arrivée à pleine charge (volumétrie à deux ans par exemple).

Prendre en compte :

- *la taille des bases de données;*
- *la taille des fichiers produits;*
- *la taille des files;*
- *la taille des logs ;*
- *...*

Ne pas prendre en compte :

- *le volume lié à la sauvegarde : elle est gérée par les opérationnels ;*
- *le volume des binaires (OS, intergiciels...) qui est à considérer par les opérationnels comme une volumétrie de base d'un serveur (le ticket d'entrée) et qui est de leur ressort ;*
- *les données archivées qui ne sont donc plus en ligne.*

Fournir également une estimation de l'augmentation annuelle en % du volume pour permettre aux opérationnels de commander ou réserver suffisamment de disque de façon proactive.

Pour les calculs de volumétrie, penser à prendre en compte les spécificités de l'encodage (nombre d'octets par caractère, par date, par valeur numérique...). Pour une base de donnée, prévoir l'espace occupé par les index et qui est très spécifique à chaque application. Une (très piètre) estimation préliminaire est de doubler l'espace disque (à affiner ensuite).

N'estimer que les données dont la taille est non négligeable (plusieurs centaines de Mo minimum).

Exemple : Volumétrie statique du composant C :

Donnée	**Description**	**Taille unitaire**	**Nombre d'éléments à 2 ans**	**Taille totale**	**Augmentation annuelle**
Table `Article`	*Les articles du catalogue*	*2Ko*	*100K*	*200 Mo*	*5 %*
Table `Commande`	*Les commandes clients*	*10Ko*	*3M*	*26.6 Go*	*10 %*
Logs	*Les logs applicatifs (niveau INFO)*	*200 o*	*300M*	*56 Go*	*0 % (archivage)*

3.5.2 Volumétrie dynamique

Il s'agit ici d'estimer le nombre d'appels aux composants et donc le débit cible (en Tps = Transactions par seconde) que devra absorber chacun d'entre eux. Un système bien dimensionné devra présenter des temps de réponse moyen à peu près similaires en charge nominale et en pic.

Toujours estimer le "pic du pic", c'est à dire le moment où la charge sera maximale suite au cumul de tous les facteurs (par exemple pour un système de comptabilité : entre 14 et 15h un jour de semaine de fin décembre). Ne pas considérer que la charge est constante mais prendre en compte :

- *les variations journalières. Pour une application de gestion avec des utilisateurs travaillant sur des heures de bureau, on observe en général des pics du double de la charge moyenne à 8h-9h, 11h-12h et 14h-15h. Pour une application Internet grand public, ce sera plutôt en fin de soirée. Encore une fois, se baser sur des mesures d'applications similaires quand c'est possible plutôt que sur des estimations.*
- *les éléments de saisonnalité. La plupart des métiers en possèdent : Noël pour l'industrie du chocolat, le samedi soir pour les admissions aux urgences, juin pour les centrales de réservation de séjours etc. La charge peut alors doubler ou bien plus. Il ne faut donc pas négliger cette estimation.*

Si le calcul du pic pour un composant en bout de chaîne de liaison est complexe (par exemple, un service central du SI exposant des données référentiel et appelé par de nombreux composants qui ont chacun leur pic), on découpera la journée en intervalles de temps suffisamment fins (une heure par exemple) et on calculera sur chaque intervalle la somme mesurée ou estimée des appels de chaque appelant (batch ou transactionnel) pour ainsi déterminer la sollicitation cumulée la plus élevée.

Si l'application tourne sur un cloud de type PaaS, la charge sera absorbée dynamiquement mais veiller à estimer le surcoût et à fixer des limites de consommation cohérentes pour respecter le budget tout en assurant un bon niveau de service.

Exemple d'estimation de charge dynamique pour un composant applicatif d'un site de e-commerce:

Opération REST GET `DetailArticle`	***Estimation***
Nombre d'utilisateurs potentiels	*1M*
Éléments de saisonnalité	*- pic journalier de 20h à 21h00* *- pic annuel de décembre*
Taux maximal d'utilisateurs connectés en même temps de 20h00 à 21h00 en décembre *Nombre maximal d'utilisateurs connectés concurrents*	*5%* *50K*
Durée moyenne d'une session utilisateur	*15 mins*
Nombre d'appel moyen du service par session	*10*
Charge (Transaction / seconde)	*50K * 10/15 / 60 = 556 Tps*

Pour un composant technique (comme une instance de base de donnée) en bout de chaîne et sollicité par de nombreux services, il convient d'estimer le nombre de requêtes en pic en cumulant les appels de toutes les clients et de préciser le ratio lecture /écriture quand cette information est pertinente (elle est très importante pour une base de donnée).

Le niveau de détail de l'estimation dépend de l'avancement de la conception de l'application et de la fiabilité des hypothèses. Dans l'exemple plus bas, nous avons déjà une idée du nombre de requêtes pour chaque opération. Dans d'autres cas, on devra se contenter d'une estimation très large sur le nombre de requêtes total à la base de données et un ratio lecture /écriture basée sur des abaques d'applications similaires : dans ce cas, inutile de détailler plus à ce stade.

Enfin, garder en tête qu'il s'agit simplement d'estimation à valider lors de campagnes de performances puis en production. Prévoir un ajustement du dimensionnement peu après la MEP (relativement aisé si les ressources matérielles sont virtualisées et/ou si l'architecture est scalable horizontalement).

Exemple : la base de donnée Oracle BD01 est utilisée en lecture par les appels REST GET sur `DetailArticle` mais également en mise à jour par les appels POST et PUT sur `DetailArticle` issus du batch d'alimentation B03 la nuit entre 01:00 et 02:00.

Nombre de requêtes SQL en pic vers l'instance BD01 de 01:00 à 02:00 en décembre	**Estimation**
Taux maximal d'utilisateurs connectés en même temps *Nombre maximal d'utilisateurs connectés concurrents*	*0.5%* *5K*
Durée moyenne d'une session utilisateur	*15 mins*
Nombre d'appel moyen du service par session	*10*
Charge usagers GET DetailArticle (Transaction / seconde)	*(10/15)*5K / 60 = 55 Tps*
Nombre de requête en lecture / écriture par appel de service	*2 / 0*
Nombre journalier d'articles créés par le batch B03 *Nombre de requêtes INSERT / SELECT*	*4K* *3 / 2*
Nombre journalier d'articles modifiés par le batch B03 *Nombre de requêtes SELECT / UPDATE*	*10K* *1 / 3*
Nombre de requêtes par secondes : *SELECT* *INSERT* *UPDATE*	 *55*2 + 2*4K/3600 + 1* 10K/3600= 115 Tps* *0 + 3*4K/3600 = 3.4 Tps* *0 + 3*10K/3600 = 8.3 Tps*

Coupe-circuits

Dans certains rares cas, le pic est extrême et imprévisible (effet 'slashdot'). Si ce risque est identifié, prévoir un système de fusible avec déport de toute ou partie de la charge sur un site Web statique avec message d'erreur par exemple. Ce dispositif peut également servir en cas d'attaque de type DDOS. Ceci permet de montrer qu'on gère le problème sans le subir (on évite les erreurs HTTP 404 ou 50x ou bout d'un long timeout) et permet d'assurer un bon fonctionnement à une partie des utilisateurs.

Qualité de Service

Il est également utile de prévoir des systèmes de régulation applicatifs dynamiques, par exemple :

- *via du throttling (limite de nombre de requêtes par origine et unité de temps). A mettre en amont de la chaîne de liaison ;*
- *des systèmes de jetons (qui permettent en outre de favoriser tel ou tel client en leur accordant un quota de jetons différents).*

Exemple : Le nombre total de jetons d'appels aux opérations REST sur la ressource DetailArticle sera de 1000. Au delà de 1000 appels simultanés, les appelants obtiendront une erreur d'indisponibilité 429 qu'ils devront gérer (et faire éventuellement des rejeux à écarter progressivement dans le temps). La répartition des jetons sera la suivante par défaut :

Opération sur DetailArticle	**Proportion des jetons**
GET	*80%*
POST	*5%*
PUT	*15%*

Exemple 2 : un throttling de 100 requêtes par source et par minute sera mis en place au niveau du reverse proxy.

Augmentation prévisionnelle de la charge

Pour faciliter le dimensionnement et éviter d'avoir à redimensionner ses serveurs trop souvent, il est important de donner une estimation de l'augmentation annuelle de la charge. Certaines

organisations estiment la charge cible à cinq ans et choisissent le matériel en fonction puisqu'il s'agit de sa durée de vie moyenne.

3.6. SAUVEGARDES

Donner ici le Recovery Point Objective (RPO) de l'application. Il peut être utile de restaurer suite à :

- *une perte de données matérielle (peu probable avec des systèmes de redondance type RAID) ;*
- *une fausse manipulation d'un power-user ou d'un administrateur ;*
- *un bug applicatif ;*
- *une destruction de donnée volontaire (attaque de type ransomware comme wannacry par exemple)...*

Exemple : on ne doit pas pouvoir perdre plus d'une journée (de 08h à 18h00) de données applicatives

3.7. DURÉE DE RÉTENTION ET ARCHIVAGE

L'archivage concerne les sauvegardes au delà d'un an en général pour raison légales. Les archives sont souvent conservées trente ans ou plus.

Préciser si des données de l'application doivent être conservées à long terme. Préciser les raisons de cet archivage (légales le plus souvent, voir https://www.service-public.fr/professionnels-entreprises/vosdroits/F10029).

Exemple : comme exigé par l'article L.123-22 du code de commerce, les données comptables devra être conservées dix ans.

Il est néanmoins crucial de prévoir des purges régulières pour éviter une dérive continue des performances et de l'utilisation disque (par exemple liée à un volume de base de données trop important).

Il est souvent judicieux d'attendre la MEP voire plusieurs mois d'exploitation pour déterminer précisément les durées de rétention (âge ou volume maximal par exemple) mais il convient de prévoir le principe même de l'existence de purges dès la définition de l'architecture de l'application. Cela peut avoir des conséquences importantes y compris sur le fonctionnel (exemple : s'il n'y a pas de rétention ad vitam aeternam de l'historique, certains patterns à base de listes chaînées ne sont pas envisageables).

Le RGPD apporte depuis mai 2018 de nouvelles contraintes sur le droit à l'oubli pouvant affecter la durée de rétention des informations personnelles.

Exemple 1 : Les pièces comptables doivent être conservées en ligne (en base) au moins deux ans puis peuvent être archivées pour conservation au moins dix ans de plus.

3.8. CONTRAINTE DE DÉPLOIEMENTS ET DE MISE À JOUR

3.8.1 Coté serveurs

Préciser ici comment l'application devra être déployée coté serveur et avec quelles contraintes.

Coté serveur, cela peut être :

- *Doit-on prévoir l'installation d'une image complète de l'OS par boot réseau (solution type Cobbler sous Linux ou Windows Deployment Toolkit sous Windows) ?*
- *Comment sont déployés les composants ? Sous forme de paquets ? Utilise-t-on un dépôt de paquets (type yum ou apt) ? Utilise-t-on des containeurs (Docker, Kubernetes…) ?*
- *Comment sont appliquées les mises jour ?*

3.8.2 Coté clients

Coté client, pour une application Web, cela peut être :

- *Prévoit-on un déploiement de type blue/green ? Si oui, préciser les proportions et la trajectoire visée*
- *si l'application est volumineuse (beaucoup de JS ou d'images par exemple), risque-t-on un impact sur le réseau ?*
- *Une mise en cache de proxy locaux est-elle à prévoir ?*
- *Des règles de firewall ou QoS sont-elles à prévoir ?*

Coté client, pour une application Java :

- *Quel version du JRE est nécessaire sur les clients ?*
- *S'il s'agit d'une application JavaWebStart, prévoit-on une mise à jour incrémentale des jars ?*

Coté client, pour une application client lourd :

- *Quel version de l'OS est supportée ?*
- *Si l'OS est Windows, l'installation passe-t-elle par un outil de déploiement (Novell ZENWorks par exemple) ? l'application vient-elle avec un installeur type Nullsoft ? Affecte-t-elle le système (variables d'environnements, base de registre…) ou est-elle en mode portable (simple zip) ?*
- *Si l'OS est Linux, l'application doit-elle fournie en tant que package ? Ce package est-il dans un dépôt d'entreprise pour être installé par un gestionnaire de paquet type dnf ou apt ?*
- *Comment sont appliquées les mises jour ?*

3.9. CONCURRENCE

Préciser ici les composants internes ou externes pouvant interférer avec l'application.

Exemple 1 : Tous les composants de cette application peuvent fonctionner en concurrence. En particulier, la concurrence batch/TP doit toujours être possible car les batchs devront pouvoir tourner de jour en cas de besoin de rattrapage

Exemple 2 : le batch X ne devra être lancé que si le batch Y s'est terminé correctement sous peine de corruption de données.

3.10. ÉCO-CONCEPTION

L'écoconception consiste à limiter l'impact environnemental des logiciels et matériels utilisés par l'application. Les exigences dans ce domaine s'expriment généralement en WH ou équivalent CO2.

Selon l'ADEME (estimation 2014), les émissions équivalent CO2 d'un KWH en France continentale pour le tertiaire est de 50g/KWH[1].

Exemple 1 : La consommation électrique moyenne causée par l'affichage d'une page Web ne devra pas dépasser 10mWH, soit pour 10K utilisateurs qui affichent en moyenne 100 pages 200 J par an : 50 g/KWH x 10mWH x 100 x 10K x 200 = 100 Kg équivalent CO2 / an.

Exemple 2 : La classe énergétique WEA[2] du site devra être de C ou mieux.

4. PRINCIPES DE L'ARCHITECTURE TECHNIQUE

Quels sont les grands principes techniques de notre application ?

Exemple :

- *Les composants applicatifs exposés à Internet dans une DMZ protégée derrière un pare-feu puis un reverse-proxy et sur un VLAN isolé.*
- *Concernant les interactions entre la DMZ et l'intranet, un pare-feu PIX ne permet les communications que depuis l'intranet vers la DMZ*
- *Comme décrit au chapitre 5, les clusters actifs/actifs seront exposés derrière un LVS + Keepalived avec direct routing pour le retour.*

5. DISPONIBILITÉ

Donner ici les dispositifs permettant d'atteindre la disponibilité exigée au chapitre 3.3.

Les mesures permettant d'atteindre la disponibilité exigée sont très nombreuses et devront être choisies par l'architecte en fonction de leur apport et de leur coût (financier, en complexité, …).

Nous regroupons les dispositifs de disponibilité en quatre grandes catégories :

- *dispositifs de supervision (technique et applicative) permettant de détecter au plus tôt les pannes et donc de limiter le MTDT (temps moyen de détection) ;*
- *dispositifs organisationnels :*

---

1 *http://www.bilans-ges.ademe.fr/static/documents/[Base%20Carbone]%20Documentation%20g%C3%A9n%C3%A9rale%20v11.0.pdf*

2 *voir http://webenergyarchive.com*

- *la présence humaine (astreintes, heures de support étendues, CDD dédiées à l'application...) qui permet d'améliorer le MTTR (temps moyen de résolution) et sans laquelle la supervision est inefficiente ;*
- *la qualité de la gestion des incidents (voir les bonnes pratiques ITIL), par exemple un workflow de résolution d'incident est-il prévu ? si oui, quel est sa complexité ? sa durée de mise en œuvre ? si elle nécessite par exemple plusieurs validations hiérarchiques, la présence de nombreux exploitants n'améliore pas forcement le MTTR ;*

- *dispositifs de redondance technique (clusters, RAID...) qu'il ne faut pas surestimer si les dispositifs précédents sont insuffisants ;*
- *dispositifs de restauration de données (sauvegardes, procédures de restauration) : la procédure de restauration est-t-elle bien définie ? testée ? d'une durée compatible avec les exigences de disponibilité ? C'est typiquement utile dans le cas de perte de données causée par une fausse manipulation ou bug dans le code : il faut alors arrêter l'application et dans cette situation, pouvoir restaurer rapidement la dernière sauvegarde améliore grandement la disponibilité de l'application.*

Rappels sur les principes de disponibilité :

- *la disponibilité d'un ensemble de composants en série : D = D1 * D2 * … * Dn, exemple : la disponibilité d'une application utilisant un serveur Tomcat à 98 % et une base Oracle à 99 % sera de 97.02 %;*
- *la disponibilité d'un ensemble de composants en parallèle : D = 1 – (1-D1) * (1- D2) * ..* (1-Dn). Exemple : la disponibilité de trois serveurs Nginx en cluster dont chacun possède une disponibilité de 98 % est de 99.999 % ;*
- *il convient d'être cohérent sur la disponibilité de chaque maillon de la chaîne de liaison : rien ne sert d'avoir un cluster actif/actif de serveurs Websphere si tous ces serveurs attaquent une base de donnée Oracle localisée sur un unique serveur physique avec disques sans RAID.*
- *on estime un système comme hautement disponible (HA) à partir de 99 % de disponibilité ;*
- *on désigne par «spare» un dispositif (serveur, disque, carte électronique...) de rechange qui est dédié au besoin de disponibilité mais qui n'est pas activé en dehors des pannes. En fonction du niveau de disponibilité recherché, il peut être dédié à l'application ou mutualisé au niveau SI.*
- *le niveau de redondance d'un dispositif peut s'exprimer avec la notion suivante si N est le nombre de dispositifs assurant un fonctionnement correct en charge :*

N	*aucune redondance (ex : il faut deux alimentation pour le serveur, si une tombe, le serveur s'arrête)*
N+1	*un composant de rechange est disponible (mais pas forcement actif), on peut supporter la panne d'un matériel (ex : on a une alimentation de spare disponible).*
2N	*le système est entièrement redondé (mais les composants de remplacement ne sont pas forcement actifs) et peut supporter la perte de la moitié des composants (ex : on dispose de quatre alimentations)*

Clustering

- *un cluster est un ensemble de nœuds (machines) hébergeant la même application ;*

- *le failover (bascule) est la capacité d'un cluster de s'assurer qu'en cas de panne, les requêtes ne sont plus envoyées vers le nœud tombé mais vers un nœud opérationnel ;*
- *en fonction du niveau de disponibilité recherché, chaque nœud peut être :*
  - *actif : le nœud traite les requêtes (ex : un serveur Apache parmi dix et derrière un répartiteur de charge). Temps de failover : 0.*
  - *passif en mode «hot standby» : le nœud est installé et démarré mais ne traite pas les requêtes (ex : une base MySql slave qui devient master en cas de panne de ce dernier via l'outil mysqlfailover). Temps de failover : de l'ordre de quelques secondes (temps de la détection de la panne).*
  - *passif en mode «warm standby» : le nœud est démarré et l'application est installée mais n'est pas démarrée (ex : un serveur avec une instance Tomcat hébergeant notre application en mode inactif en plus d'une autre application - elle - démarrée). En cas de panne, notre application est démarrée automatiquement. Temps de failover : de l'ordre de la minute (temps de la détection de la panne et de démarrage de l'application) ;*
  - *passif en mode «cold standby» : le nœud est un simple spare. Pour l'utiliser, il faut installer l'application et la démarrer. Temps de failover : de l'ordre de dizaines de minutes avec solutions de virtualisation (ex : KVM live migration) et/ou de containers (Docker) à une journée lorsqu'il faut installer/restaurer et démarrer l'application.*
- *Il existe deux architectures de clusters actif/actif :*
  - *les clusters actifs/actifs à couplage faible dans lesquels un nœud est totalement indépendant des autres, soit parce que l'applicatif est stateless (le meilleur cas), soit parce que les données de contexte (typiquement une session HTTP) sont gérées isolément par chaque nœud. Dans le dernier cas, le répartiteur de charge devra assurer une affinité de session, c'est à dire toujours router les requêtes d'un client vers le même nœud et en cas de panne de ce nœud, les utilisateurs qui y sont routés perdent leurs données de session et doivent se reconnecter. Bien entendu, les nœuds partagent tous les mêmes données persistées en base, les données de contexte sont uniquement des données transitoires en mémoire.*
  - *les clusters actifs/actifs à couplage fort (clusters à tolérance de panne) dans lesquels tous les nœuds forment en quelque sorte une super-machine logique partageant la même mémoire. Dans cette architecture, toute donnée de contexte doit être répliquée dans tous les nœuds (ex : cache de sessions HTTP répliqué avec JGroups).*

Failover

Le failover (bascule) est la capacité d'un cluster à basculer un flux de requêtes d'un nœud vers un autre en cas de panne.

Sans failover, c'est au client de détecter la panne et de rejouer sa requête sur un autre nœud. Dans les faits, ceci est rarement praticable et les clusters disposent presque toujours de dispositifs de failover.

Une solution de failover peut être décrite par les attributs suivants :

- *automatique ou manuel ? (dans une solution HA, le failover est en général automatique à moins de disposer d'astreintes 24/7/365 et d'une exploitation extrêmement organisée) ;*
- *quelle stratégie de failover et de failback ?*
  - *dans un cluster dit "N+1", on bascule vers un nœud passif qui devient actif et le restera (le nœud en panne une fois réparé pourra devenir le nouveau serveur de secours). Si un serveur cible ne tiendrait pas seul la charge, on prévoit plusieurs serveurs passifs (cluster dit "N+M") ;*

    - *dans un cluster "N-to-1", on rebasculera (failback) sur le serveur qui était tombé en panne une fois réparé et le serveur basculé redeviendra le serveur de secours ;*
    - *dans un cluster N-to-N (architecture en voie de démocratisation avec le cloud de type PaaS comme App-Engine ou CaaS comme Kubernetes ou Rancher) : on distribue les applications du nœud en panne vers d'autres nœuds actifs et qui auront été dimensionnés en prévision de cette éventuelle surcharge.*
- *transparent via à vis de l'appelant ou pas ? En général, les requêtes pointant vers un serveur en panne (non encore détectée) tombent en erreur (en timeout la plupart du temps). Certains dispositifs ou architectures de FT (tolérance de panne) permettent d'assurer la transparence ;*
- *quelle solution de détection de panne ?*
    - *les répartiteurs de charge utilise des health checkers (tests de santé) très variés (requêtes bouchonnées, analyse du CPU, des logs à distance, etc...) vers les nœuds qu'ils contrôlent ;*
    - *les détections de panne des clusters actifs/passifs fonctionnent la plupart du temps par écoute des palpitations (heartbeat) du serveur passif vers le serveur actif, par exemple via des requêtes multicast UDP dans le protocole VRRP utilisé par keepalived.*
- *quelle durée de détection de la panne ? il convient de paramétrer correctement (le plus court possible sans dégradation de performance) les solutions de détection de panne pour limiter la durée de failover ;*
- *quelle pertinence de la détection ? le serveur en panne est-il *vraiment* en panne ? un mauvais paramétrage peut provoquer une indisponibilité totale d'un cluster alors que les nœuds sont sains.*

Quelques mots sur les répartiteurs de charge :

- *un répartiteur de charge (Load Balancer = LB) est une brique obligatoire pour un cluster actif/actif.*
- *dans le cas des clusters, une erreur classique est de créer un SPOF au niveau du répartiteur de charge. On va alors diminuer la disponibilité totale du système au lieu de l'améliorer. Dans la plupart des clusters à vocation de disponibilité (et pas seulement de performance), il faut redonder le répartiteur lui-même en mode actif/passif (et évidemment pas actif/actif sinon, il faudrait un "répartiteur de répartiteurs"). Le répartiteur passif doit surveiller à fréquence élevée le répartiteur actif et le replacer dès qu'il tombe (les requêtes arrivant au LB en panne avant la bascule sont en erreur) ;*
- *il est crucial de configurer correctement et à fréquence suffisante les tests de vie (heathcheck) des nœuds vers lesquels le répartiteur distribue la charge car sinon, le répartiteur va continuer à envoyer des requêtes vers des nœuds tombés ou en surcharge ;*
- *certains LB avancés (ex : option redispatch de HAProxy) permettent de configurer des rejeux vers d'autres nœuds en cas d'erreur ou timeout et donc d'améliorer la tolérance de panne puisqu'on évite de retourner une erreur à l'appelant pendant la période de pré-détection de la panne.*
- *lisser la charge entre les nœuds et ne pas forcement se contenter de round robin. Un algorithme simple est le LC (Least Connection) permettant au répartiteur de privilégier les nœuds les moins chargés, mais il existe bien d'autres algorithmes plus ou moins complexes (systèmes de poids par nœud ou de combinaison charge + poids par exemple) ;*
- *dans le monde Open Source, voir LVS + keepalived ou HAProxy + keepalived ;*

La tolérance de panne

La tolérance de panne (FT = Fault Tolerance) ne doit pas être confondue avec la disponibilité, elle concerne la capacité d'un système à passer outre les pannes sans perte de données. Par exemple, un disque RAID 1 assure une tolérance de panne transparente ; en cas de panne, le processus écrit ou lit sans erreur après le failover automatique vers le disque sain.

Pour permettre la tolérance de panne d'un cluster, il faut obligatoirement partir sur un cluster actif/ actif avec fort couplage dans lequel les données de contexte sont répliquées à tout moment. Une autre solution (la meilleure) est d'éviter tout simplement les données de contexte (en gardant les données de session dans la navigateur via un client JavaScript par exemple) ou de les stocker en base (SQL/NoSQL) ou en cache distribué (mais attention aux performances). Pour disposer d'une tolérance de panne transparente (le niveau de disponibilité le plus élevé), il faut en plus prévoir un répartiteur de charge assurant les rejeux.

Attention à bien qualifier les exigences avant de construire une architecture FT car en général ces solutions :

1. *complexifient l'architecture et la rendent donc moins robuste et plus coûteuse à construire, tester, exploiter ;*
2. *peuvent dégrader les performances : les solutions de disponibilité et de performance vont en général dans le même sens (par exemple, un cluster de machines stateless va diviser la charge par le nombre de nœuds et dans le même temps, la disponibilité augmente), mais quelque fois, disponibilité et performance peuvent être antagonistes : dans le cas d'une architecture stateful, typiquement gérant les sessions HTTP avec un cache distribué (type Infinispan répliqué en mode synchrone ou un REDIS avec persistance sur le master), toute mise à jour transactionnelle de la session ajoute un surcoût lié à la mise à jour et la réplication des caches, ceci pour assurer le failover. En cas de plantage d'un des nœuds, l'utilisateur conserve sa session à la requête suivante et n'a pas à se reconnecter, mais à quel coût ? ;*
3. *elles peuvent même dégrader la disponibilité car tous les nœuds sont fortement couplés. Une mise à jour logicielle par exemple peut imposer l'arrêt de l'ensemble du cluster.*

Quelques solutions de disponibilité (hors disponibilité du datacenter) :

Solution	Coût	Complexité de mise en œuvre	Amélioration de la disponibilité
Disques en RAID 1 ou RAID 1+0	XXX	X	XXX
Disques en RAID 5	X	X	XX
Redondance des alimentations et autres composants,	XX	X	XX
Bonding des cartes Ethernet	XX	X	X
Cluster actif/passif	XX	XX	XX
Cluster actif/actif (donc avec LB)	XXX	XXX	XXX
Serveurs de spare	XX	X	X
Bonne supervision système	X	X	XX
Bonne supervision applicative	XX	XX	XX
Systèmes de test de vie depuis un site distant	X	X	XX
Astreintes dédiées à l'application, 24/7/365	XXX	XX	XXX
Copie du backup du dernier dump de base métier sur baie SAN (pour restauration express)	XX	X	XX

Exemple 1 : Pour atteindre la disponibilité de 98 % exigée, les dispositifs de disponibilité envisagés sont les suivants :

- *tous les serveurs en RAID 5 + alimentations redondées ;*
- *répartiteur HAProxy + keepalived actif/passif mutualisé avec les autres applications ;*
- *cluster actif /actif de deux serveurs Apache + mod_php ;*
- *serveur de spare pouvant servir à remonter la base MariaDB depuis le backup de la veille en moins de 2h.*

Exemple 2 : Pour atteindre la disponibilité de 99.97% exigée, les dispositifs de disponibilité envisagés sont les suivants (pour rappel, l'application sera hébergée dans un datacenter de niveau tiers III, voir §2.1.4) :

- *tous les serveurs en RAID 1+0 + alimentations redondées + interfaces en bonding ;*
- *répartiteur HAProxy + keepalived actif/passif dédié à l'application ;*
- *cluster actif /actif de 4 serveurs (soit une redondance 2N) Apache + mod_php ;*
- *instance Oracle en RAC sur deux machines (avec interconnexion FC dédiée).*

6. DÉPLOIEMENT EN PRODUCTION

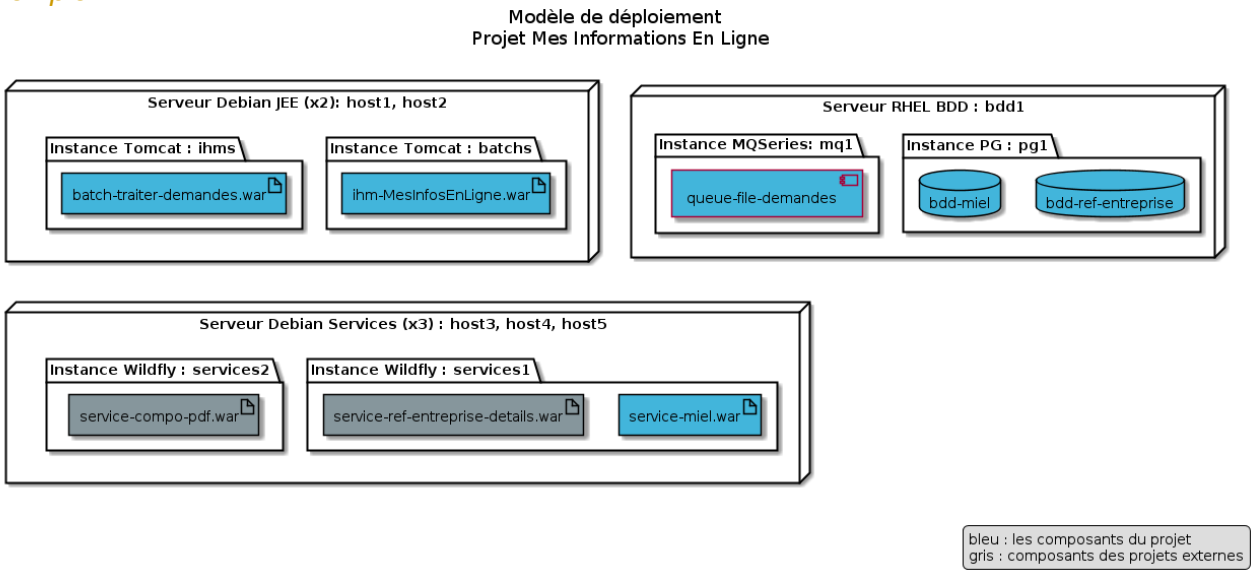
Fournir ici le modèle de déploiement des composants (ici représentés en tant qu'artefacts) sur les différents intergiciels et nœuds physiques (serveurs). Attention : ne pas détailler ici les composants d'infrastructure. Cette section doit permettre de répondre à la question « où tourne le composant x ? ».

Tout naturellement, on le documentera de préférence avec un diagramme de déploiement UML2.

Pour les clusters, donner le facteur d'instanciation en attribut du nœud.

Mentionner optionnellement les composants qui tournent sur le même serveur ou le même serveur d'application pour en rappeler la proximité, en particulier s'ils sont des dépendances.

Exemple :



7. SCHÉMA GLOBAL

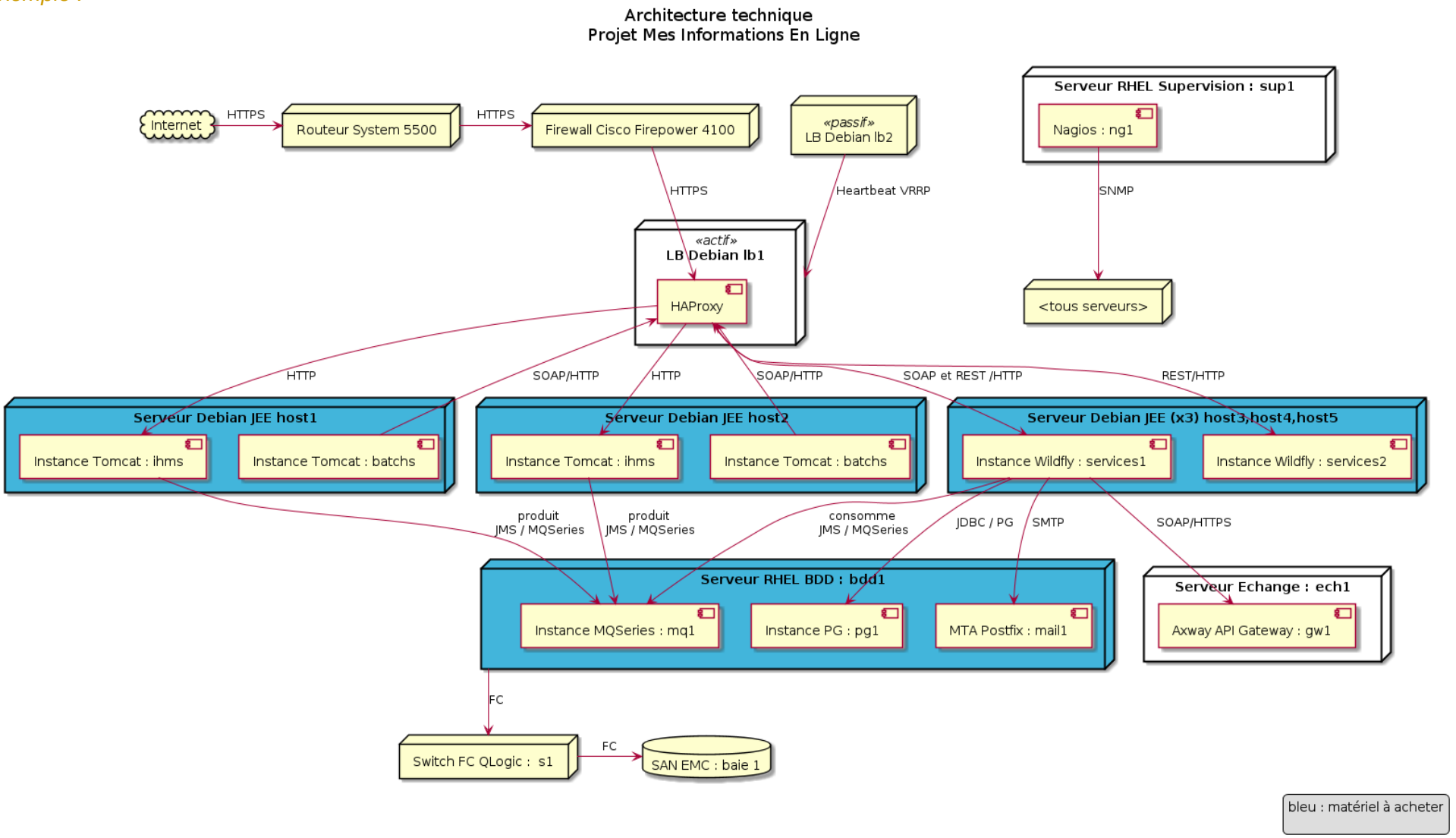
Ce schéma d'architecture technique global détaille les OS, les composants d'infrastructure logiciels (serveurs d'application, bases de données...) voire réseau (pare-feu, appliances, routeurs...) uniquement quand ils aident à la compréhension. Attention néanmoins à ne pas aller trop dans le détail, surtout concernant les briques systèmes ou réseau de niveau SI au risque de dupliquer la documentation entre applications et de rendre le document impossible à maintenir.

Identifier clairement le matériel dédié à l'application (et éventuellement à acheter).

Ce schéma sera de préférence un diagramme de déploiement UML2.

On donne ici les véritables hostnames ou IP des machines (sauf si cela pose des problèmes de sécurité).

Exemple :


Architecture technique
Projet Mes Informations En Ligne
Internet
HTTPS
Routeur System 5500
HTTPS
Firewall Cisco Firepower 4100
«passif»
LB Debian lb2
Serveur RHEL Supervision : sup1
Nagios : ng1
HTTPS
Heartbeat VRRP
SNMP
«actif»
LB Debian lb1
HAProxy
<tous serveurs>
HTTP
SOAP/HTTP
HTTP
SOAP/HTTP
SOAP et REST /HTTP
REST/HTTP
Serveur Debian JEE host1
Instance Tomcat : ihms
Instance Tomcat : batchs
Serveur Debian JEE host2
Instance Tomcat : ihms
Instance Tomcat : batchs
Serveur Debian JEE (x3) host3,host4,host5
Instance Wildfly : services1
Instance Wildfly : services2
produit
JMS / MQSeries
produit
JMS / MQSeries
consomme
JMS / MQSeries
JDBC / PG
SMTP
SOAP/HTTPS
Serveur RHEL BDD : bdd1
Instance MQSeries : mq1
Instance PG : pg1
MTA Postfix : mail1
Serveur Echange : ech1
Axway API Gateway : gw1
FC
Switch FC QLogic : s1
FC
SAN EMC : baie 1
bleu : matériel à acheter

8. VERSIONS DES COMPOSANTS D'INFRASTRUCTURE

OS, bases de données, MOM, serveurs d'application, etc...

Composant	Rôle	Version	Environnement technique
CFT	Transfert de fichiers sécurisé	X.Y.Z	RHEL 6
Wildfly	Serveur d'application JEE	9	Debian 8, OpenJDK 1.8.0_144
Tomcat	Container Web pour les IHM	7.0.3	CentOS 7, Sun JDK 1.8.0_144
Nginx	Serveur Web	1.11.4	Debian 8
PHP + `php5-fpm`	Pages dynamiques de l'IHM YYY	5.6.29	nginx
PostgreSQL	SGBDR	9.3.15	CentOS 7

9. MATRICE DES FLUX TECHNIQUES

On liste ici l'intégralité des flux techniques utilisés par l'application. Les ports d'écoute sont précisés. On détaille aussi les protocoles d'exploitation (JMX ou SNMP par exemple). Dans certaines organisions, cette matrice sera trop détaillée pour un dossier d'architecture et sera maintenue dans un document géré par les intégrateurs ou les exploitants.

Il n'est pas nécessaire de faire référence aux flux applicatifs tels que décrits dans le volet applicatif car les lecteurs ne recherchent pas les mêmes informations. Ici, les exploitants ou les intégrateurs recherchent l'exhaustivité des flux à fin d'installation et de configuration des pare-feu par exemple.

Les types de réseaux incluent les informations utiles sur le réseau utilisé afin d'apprécier les performances (TR, latence) et la sécurité: LAN, VLAN, Internet, LS, WAN,...)

Exemple partiel :

ID	Source	Destination	Type de	Protocole	Port d'écoute

			réseau		
1	lb2	lb1	LAN	VRRP sur UDP	IP multicast 224.0.0.18
2	lb1	host1, host2	LAN	HTTP	80
3	host3, host4, host5	bdd1	LAN	PG	5432
4	sup1	host[1-6]	LAN	SNMP	199

10. DIMENSIONNEMENT DES MACHINES

Mémoire, disque et CPU de chaque nœud. A affiner après campagne de performance ou MEP.

Pour les VM, on considère le disque des partitions système comme interne même si elles sont physiquement sur un SAN.

Le disque SAN concerne des partitions montées sur SAN depuis un serveur physique ou virtuel.

Exemple :

Machines virtuelles lb1, host2 (Reverse proxy)

CPU	**Mémoire**	**Disque interne**	**Disque SAN**
2 VCPU	4 Go	20 Go	-

Machine physique bdd1 (BDD PostgreSQL)

CPU	**Mémoire**	**Disque interne**	**Disque SAN**
16 cœurs Xeon 3Ghz	24 Go	20 Go	3 To

11. ÉCO-CONCEPTION

Lister ici les mesures d'infrastructure permettant de répondre aux exigences d'écoconception listée en 3.10. Les réponses à ses problématiques sont souvent les mêmes que celles aux exigences de performance (temps de réponse en particulier) et à celles des coûts (achat de matériel). Dans ce cas, y faire simplement référence. Néanmoins, les analyses et solutions d'écoconception peuvent être spécifiques à ce thème.

Quelques pistes d'amélioration de la performance énergétique :

- *mesurer la consommation électrique des systèmes avec les sondes PowerAPI[3] (développé par l'INRIA et l'université Lille 1) ;*
- *utiliser des caches (cache d'opcode, caches mémoire, caches HTTP...) ;*
- *pour des grands projets ou dans le cadre de l'utilisation d'un cloud CaaS, l'utilisation de cluster de containers (solution type Swarm, Mesos ou Kubernete) permet d'optimiser l'utilisation des VM ou machines physiques en les démarrant / arrêtant à la volée de façon élastique ;*
- *héberger ses serveurs dans un datacenter performant. Les fournisseurs de cloud proposent en général des datacenters plus performants que du on-premises. L'unité de comparaison est ici le PUE (Power Usage Effectiveness), ratio entre l'énergie consommée par le datacenter et l'énergie effectivement utilisée par les serveurs (donc hors refroidissement et dispositifs externes). OVH propose par exemple des datacenter avec un PUE de 1.2 en 2017 contre 2.5 en moyenne. Néanmoins :*
    - *vérifier l'origine de l'énergie (voir par exemple les analyses de Greenpeace en 2017 sur l'utilisation d'énergie issue du charbon et du nucléaire par Amazon pour son cloud AWS[4]) ;*
    - *garder en tête que l'énergie consommée par l'application coté client et réseau est très supérieure à celle utilisée coté serveur (par exemple, on peut estimer qu'un serveur consommant à peine plus qu'une station de travail suffit à plusieurs milliers voire dizaines de milliers d'utilisateurs). La réduction énergétique passe aussi par un allongement de la durée de vie des terminaux et l'utilisation de matériel plus sobre.*

Exemple 1 : la mise en place d'un cache Varnish devant notre CMS reduira de 50% le nombre de construction de pages dynamiques PHP et permettra l'économie de deux serveurs.

Exemple 2 : L'application sera hébergée sur un cloud avec un PUE de 1.2 et une origine à 80 % renouvelable de l'énergie électrique.

12. EXPLOITATION ET SUPERVISION

Lister ici les grands principes d'exploitation de la solution, pour plus de détail (filesystems sauvegardés, plan de production, planification des traitements...), prévoir un DEX (Dossier d'EXploitation). Si cette application reste dans le standard de l'organisation, ce référer à un dossier commun.

3 *http://www.powerapi.org/*

4 *http://www.clickclean.org*

12.1. ORDRE D'ARRÊT/DÉMARRAGE

Préciser ici l'ordre de démarrage des machines et composants entre eux ainsi que l'ordre d'arrêt. En fonction des situations, on peut faire figurer les composants externes ou non.

Le DEX contiendra une version plus précise de ce chapitre (notamment avec un numéro d'ordre SystemV ou un "Wants" SystemD précis), ce sont surtout les principes généraux des ordres d'arrêt et de démarrage qui doivent ici être décrits.

Le démarrage se fait en général dans le sens inverse des chaînes de liaison et l'arrêt dans le sens de la chaîne de liaison.

Préciser d'éventuelles problématiques en cas de démarrage partiel (par exemple, le pool de connexions du serveur d'application va-t-il retenter de se connecter à la base de donnée si elle n'est pas démarrée ? combien de fois ? quel est le degré de robustesse de la chaîne de liaison ?)

Exemple :

Démarrage

Ordre	**Composant**
1	pg1 sur serveur bdd1
2	mq1 sur bdd1
3	services1 sur serveurs host3, host4 et host5
4	services2 sur serveurs host3, host4 et host5
5	batchs sur serveurs host1, host2
5	ihms sur serveurs host1, host2

Arrêt
Inverse exact du démarrage

12.2. OPÉRATIONS PROGRAMMÉES

Lister de façon macro (le DEX détaillera le plan de production précis) :

- *les batchs ou famille de batchs et leurs éventuelles inter-dépendances. Préciser si un ordonnanceur sera utilisé.*
- *les traitements internes (tâches de nettoyage / bonne santé) du système qui ne remplissent uniquement des rôles techniques (purges, reconstruction d'index, suppression de données temporaires...)*

Exemple 1 : le batch traiter-demande fonctionnera au fil de l'eau. Il sera lancé toutes les 5 mins depuis l'ordonnanceur JobScheduler.

Exemple 2 : le traitement interne ti_index est une classe Java appelant des commandes REINDEX en JDBC lancées depuis un scheduler Quartz une fois par mois.

12.3. SAUVEGARDES ET RESTAURATIONS

Donner la politique générale de sauvegarde. Elle doit répondre aux exigences du chapitre 3.6. De même les dispositifs de restauration doivent être compatibles avec les exigences de disponibilité exprimés en 3.3 :

- *Quels sont les backups à chaud ? à froid ?*
- *Que sauvegarde-t-on ? (bien sélectionner les données à sauvegarder car le volume total du jeu de sauvegardes peut facilement atteindre dix fois le volume sauvegardé).*
  - *des images/snapshots systèmes pour restauration de serveur ou de VM ?*
  - *des systèmes de fichiers ou répertoires ?*
  - *des bases de données sous forme de dump ? sous forme binaire ?*
  - *le contenu de files ?*
  - *les logs ? les traces ?*
- *Les sauvegardent sont-elles chiffrées ? si oui, préciser l'algorithme de chiffrement symétrique utilisé et comment sera gérée la clé ;*
- *Les sauvegardes sont-elles compressées ? si oui, avec quel algorithme ? (gzip, bz2, lzma ? xv ? ...) quel paramétrage (indice de compression) ? attention à trouver le compromis entre durée de compression / décompression et gain de stockage ;*
- *Quel outillage est mis en œuvre ? (peut aller de l'outil « backup-manager » à IBM TSM)*
- *Quelle technologie est utilisée pour les sauvegardes ? (bandes magnétiques type LTO ou DLT ? disques externes ? cartouches RDX ? cloud de stockage comme Amazon S3 ? support optique ? NAS ? ...)*
- *Quelle est la périodicité de chaque type de sauvegarde ? (ne pas trop détailler ici, ceci sera dans le DEX)*
- *Quelle est la stratégie de sauvegarde ?*
  - *complètes ? incrémentales ? différentielles ? (prendre en compte les exigences en disponibilité. La restauration d'une sauvegarde incrémentale sera plus longue qu'une restauration de sauvegarde différentielle, elle même plus longue qu'une restauration de sauvegarde complète)*
  - *quel roulement ? (si les supports de sauvegarde sont écrasés périodiquement)*
- *Comment se fait le bilan de la sauvegarde ? par courriel ? où sont les logs ?*
- *Où sont stockées les sauvegardes ? (idéalement le plus loin possible du système sauvegardé tout en permettant une restauration dans un temps compatible avec les exigences de disponibilité) ;*
- *Qui accède physiquement aux sauvegardes ? à la clé de chiffrement ? (penser aux exigences de confidentialité)*

- *Des procédures de contrôle de sauvegarde et de test de restauration sont-il prévus ? (prévoir un test de restauration une fois par an minimum)*

Exemple de roulement : jeu de 21 sauvegardes sur un an :

- *6 sauvegardes journalières incrémentales ;*
- *1 sauvegarde complète le dimanche et qui sert de sauvegarde hebdomadaire ;*
- *3 sauvegardes hebdomadaires correspondant aux 3 autres dimanches. Le support du dernier dimanche du mois devient le backup mensuel ;*
- *11 sauvegardes mensuelles correspondant aux 11 derniers mois.*

12.4. ARCHIVAGE

Décrire ici les dispositifs permettant de répondre aux besoins d'archivage exprimés au chapitre 3.7

La plupart du temps, la dernière sauvegarde mensuelle de l'année est archivée avec les modalités de stockage suivantes :

- *la technologie : idéalement, on dupliquera par sécurité l'archive sur plusieurs supports de technologies différentes (bande + disque dur par exemple);*
- *un lieu de stockage spécifique et distinct des sauvegardes classiques (coffre en banque par exemple);*

Exemple : la sauvegarde annuelle RDX sera archivée sur bande LTO. Les deux supports seront stockés en coffre dans deux banques différentes.

12.5. PURGES

Donner ici les dispositifs techniques répondant aux exigences du chapitre 3.7.

Exemple : l'historique des consultations sera archivé par un dump avec une requête SQL de la forme `COPY (SELECT * FROM matable WHERE ...) TO '/tmp/dump.tsv'` *puis purgé par une requete SQL DELETE après validation par l'exploitant de la complétude du dump.*

12.6. LOGS

Sans être exhaustif sur les fichiers de logs (à prévoir dans le DEX), présenter la politique générale de production et de gestion des logs :

- *Quelles sont les politiques de roulement des logs ? le roulement est-il applicatif (via un* `DailyRollingFileAppender` *log4j par exemple) ou système (typiquement par le démon logrotate) ?*
- *Une centralisation de logs est-elle prévue ? (presque indispensable pour les architectures SOA ou micro-services). Voir par exemple la stack ELK ;*
- *Quel est le niveau de prolixité prévu par type de composant ? le débat en production est en général entre les niveaux WARN et INFO. Si les développeurs ont bien utilisé le niveau*

INFO pour des informations pertinentes (environnement au démarrage par exemple) et pas du DEBUG, fixer le niveau INFO.

- *Des mesures anti-injection de logs sont-elles prévues (échappement XSS) ?*
- *Penser aux sauvegardes des logs au chapitre 12.3.*

Exemple 1 : les logs applicatifs du composant service-miel seront en production de niveau INFO avec roulement journalier et conservation deux mois.

Exemple 2 : les logs seront échappés à leur création via la méthode `StringEscapeUtils.escapeHtml()` de Jakarta commons-lang.

12.7. SUPERVISION

Comme évoqué au chapitre 5, la supervision est un pilier central de la disponibilité en faisant diminuer drastiquement le MTTD (temps moyen de détection de la panne). Idéalement, elle ne sera pas uniquement réactive mais proactive.

Les métriques sont des mesures brutes (% CPU, taille FS, taille d'un pool...) issues de sondes système, middleware ou applicatives. Les indicateurs sont des combinaisons logiques de plusieurs métriques et sont porteurs de sens (ex : niveau critique si l'utilisation de CPU sur le serveur xxx reste au delà de 95% pendant plus de 5 minutes).

12.7.1 Supervision technique

Lister les métriques :

- *Système (% d'utilisation de file system, load, volume de swap in/out, nombre de threads total ...)*
- *Middleware (% de Used HEAP sur une JVM, nb de threads sur la JVM, % utilisation d'un pool de threads ou de connexions JDBC ..)*

Exemple : on mesura le % de wait io.

12.7.2 Supervision applicative

Lister les métriques applicatives (développés en interne). Ils peuvent être techniques ou fonctionnels :

- *Nombre de requêtes d'accès à un écran ;*
- *Nombre de contrats traités dans l'heure ;*
- *...*

Il est également possible de mettre en place des outils de BAM (Business Activity Monitoring) basées sur ces métriques pour suivre des indicateurs orientés processus.

Exemple : l'API REST de supervision applicative proposera une ressource `Metrique` contenant les métriques métier principaux : nombre de colis à envoyer, nombre de préparateurs actifs ...

12.7.3 Outil de pilotage de la supervision

Un tel outil (comme Nagios, Hyperic HQ dans le monde Open Source) :

- *collecte les métriques (en SNMP, JMX, HTTP ...) de façon périodique ;*
- *persiste les métriques dans un type de base de données de séries chronologiques (comme RRD) ;*
- *consolide les indicateurs depuis les métriques ;*
- *affiche les tendances dans le temps de ces indicateurs ;*
- *permet de fixer des seuils d'alerte basés sur les indicateurs et de notifier les exploitants en cas de dépassement.*

Exemple : la pilotage de la supervision se basera sur la plate-forme Nagios.

12.7.4 Suivi des opérations programmées

Indiquer l'ordonnanceur ou le planificateur utilisé pour piloter les batchs et consolider le plan de production (exemple : VTOM, JobScheduler, Dollar Universe, Control-M,...). Détailler les éventuelles spécificités de l'application :

- *degré de parallélisme des batchs*
- *plages de temps obligatoires*
- *rejeux en cas d'erreur*
- *...*

Exemple : les batchs seront ordonnancés par l'instance JobScheduler de l'organisation.

- *les batchs ne devront jamais tourner les jours féries ;*
- *leur exécution sera bornée aux périodes 23h00 - 06h00. Leur planification devra donc figurer dans cette plage ou ils ne seront pas lancés ;*
- *on ne lancera pas plus de cinq instances du batch XXX en parallèle.*

12.7.5 Supervision des performances

Suit-on les performances de l'application en production ? Cela permet :

- *de disposer d'un retour factuel sur les performances in vivo et d'améliorer la qualité des décisions d'éventuelles redimensionnement de la plate-forme matérielle ;*
- *de détecter les pannes de façon proactive (suite à une chute brutale du nombre de requêtes par exemple) ;*

- *de faire de l'analyse statistique sur l'utilisation des composants ou des services afin de favoriser la prise de décision (pour le décommissionnement d'une application par exemple).*

Il existe trois grandes familles de solutions :

- *les APM (Application Performance Monitoring) : outils qui injectent des sondes sans impact applicatifs, qui les collectent et les restituent (certains reconstituent même les chaînes de liaison complètes via des identifiants de corrélations injectés lors des appels distribués). Exemple : Oracle Enterprise Manager, Oracle Mission Control, Radware, BMC APM, Dynatrace , Pinpoint en OpenSource ...). Vérifier que l'overhead de ces solutions est négligeable ou limité et qu'on ne met en péril la stabilité de l'application.*
- *la métrologie «maison» par logs si le besoin est modeste ;*
- *les sites de requêtage externes (voir aussi les tests de vie en 12.7.6) qui appellent périodiquement l'application et produisent des dashboards. Ils ont l'avantage de prendre en compte les temps WAN non disponibles via les outils internes. A utiliser couplés aux tests de vie (voir plus loin).*

Exemple : les performances du site seront supervisées en continu par pingdom.com. Des analyses de performances plus poussées seront mises en œuvre par Pinpoint en fonction des besoins.

12.7.6 Tests de vie

Il est également fortement souhaitable et peu coûteux de prévoir un système de tests de vie (via des scénarios déroulés automatiquement).

En général, ces tests sont simples (requêtes HTTP depuis un curl croné par exemple). Ils doivent être lancés depuis un ou plusieurs sites distants pour détecter les coupures réseaux.

Il est rarement nécessaire qu'ils effectuent des actions de mise à jour. Si tel est le cas, il faudra être en mesure d'identifier dans tous les composants les données issues de ce type de requêtes pour ne pas polluer les données métier et les systèmes décisionnels.

Exemple pour un site Internet : des tests de vie seront mis en œuvre via des requêtes HTTP lancées via l'outil uptrends.com. En cas de panne, un mail est envoyé aux exploitants.

13. DÉCOMMISSIONNEMENT

Ce chapitre est à remplir quand l'application arrive en fin de vie et doit être supprimé ou remplacé. Il décrit entre autres :

- *les données à archiver ou au contraire à détruire avec un haut niveau de confiance ;*
- *les composants physiques à évacuer ou à détruire ;*

- *les procédures de désinstallation coté serveur et/ou client (il est courant de voir des composants obsolètes toujours s'exécuter sur des serveurs et occasionner des problèmes de performance et de sécurité passant sous le radar)*
- *les contraintes de sécurité associées au décommissionnement (c'est une étape sensible souvent négligée, on peut retrouver par exemple des disques durs remplis de données très sensibles suite à un don à une association par exemple) ;*

Exemple 1 : Le composant X sera remplacé par les services Y. Ensuite les données Oracle Z du silo seront migrées en one-shot via un script PL/SQL + DBLink vers l'instance XX avec le nouveau format de base du composant T.

Exemple 2 : en cas de problème sur le nouveau composant, un retour arrière sera prévu : les anciennes données seront restaurées dans les deux heures et les nouvelles données depuis la bascule seront reprise par le script S1 qui...

Exemple 3 : Les serveurs X, Y et Z seront transmis au service d'action sociale pour don caritatif après avoir effacé intégralement les disques durs via la commande `shred`, 3 passes.