L'automatisation au service de la défense et de la sécurité par conception dans un monde déconnecté

Nicolas Loriot [1], Fabien Lebossé[1]

[1] *Airbus CyberSecurity, 1 bd Jean moulin 78990 Elancourt, France*

Abstract

This paper aims to share with the community the road to an "as close as possible" CI/CD integration with air gaps on the way to the production. We will share some technical feedback, as well as the impact on the team (skillset) and the return on investment. During this CI/CD journey, we have strongly pushed all security aspect into the process from OS and component hardening to ACL computation for the whole infrastructure accordingly to the component deployed.

Keywords

Automation, security by design, Trade off, training, system engineering, IaC, Infrastructure as code, CaC, Configuration as Code, automatisation, sécurité par conception, compromis, formation, ingénierie système

1. Introduction

Remplacer, augmenter, accélérer … voici quelques effets envisageables avec les mécanismes d'automatisation. Ces dix dernières années, nous avons pu expérimenter différentes options dans l'optique d'améliorer la posture du défenseur. Elle passe par les mécanismes d'automatisation d'un SOC moderne avec la détection, l'enrichissement, l'analyse, notification supportée tant par des systèmes experts que des moteurs d'Intelligence Artificielle, mais aussi par différentes formes d'automatisations permettant de renforcer le niveau de sécurité d'un Système d'Information. Nous vous proposons de partager quelques-unes de ses évolutions, souvent efficaces, parfois jonchées de quelques désillusions sur le retour sur investissement financier direct. Ces initiatives déplaçant souvent la charge de travail soit à autre moment du cycle de vie du projet, soit sur d'autres compétences ayant alors un impact sur la constitution de nos équipes types par rapport aux années 2000. Nous partagerons également dans ce papier les impacts sur les équipes des différentes options abordées.

Une des particularités de notre expérience est de tenter d'intégrer ces évolutions d'intégration continue pour des systèmes dit déconnectés et donc aussi comment tirer profit au mieux des concepts d'automatisation et simplification de mise en œuvre (déploiement / administration / maintenance) et d'intégration continue avec son déploiement (CI/CD) sans avoir la capacité de pousser directement en production les modifications.

Pour donner un peu de contexte, mais sans pour autant rentrer dans le détail car ce n'est pas l'objectif de ce papier, les réflexions se sont basées sur des déploiements de plusieurs systèmes SOC similaires ou « cousins » dans leurs technologies utilisées. Ces similarités de déploiement ouvraient la porte à une tentative de standardisation pour augmenter la productivité. Nous ne parlons pas ici de notre SOC MSSP où nous avons tous nos personnels à son chevet mais de systèmes SOC que nous livrons à des tiers (clients ou autre partie de nos organisations), qui sont déployés sur des sites géographiques distincts, pour lesquels nous n'avons pas de possibilité de faire de maintenance à distance et qui sont gérés au quotidien par des tiers.

C&ESAR'21: Computer Electronics Security Application Rendezvous, November 16-17, 2021, Rennes, France
EMAIL: nicolas.loriot at airbus.com (N. Loriot); fabien.lebosse at airbus.com (F. Lebossé);

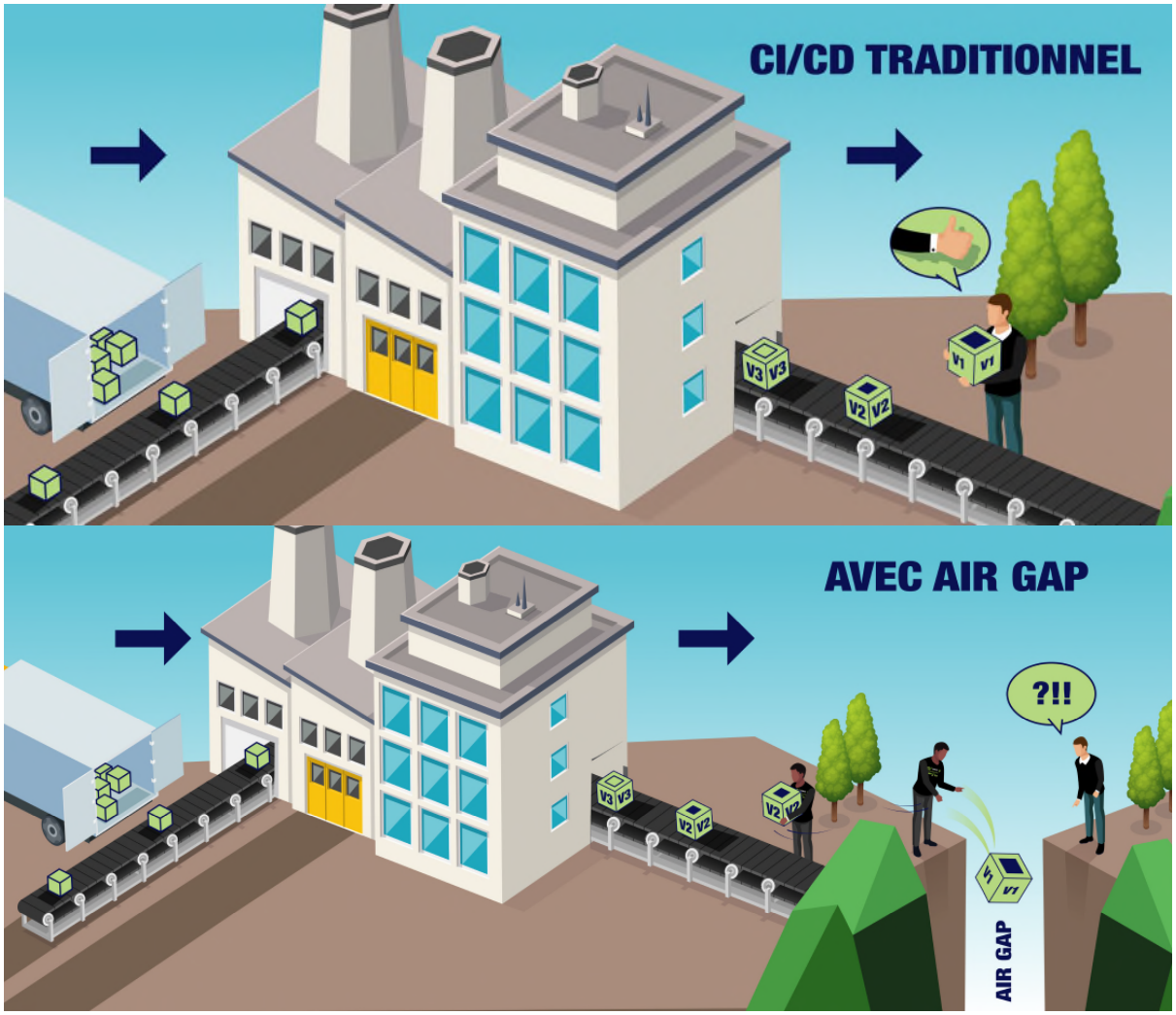

Figure 1 :CI/CD … avec Air Gap … il y a comme un os

D'une certaine manière, nous nous sommes appliqués à pousser au plus loin le concept de DevSecOps au profit d'un système déployé via « air gap » sans jamais pouvoir l'atteindre de part cette séparation. Nous aborderons dans cet article la manière nous avons franchi cet « air gap » et ses limites.

2. Premiers pas pour introduire la sécurité via l'automatisation

Pour commencer sur la posture visant à remplacer du personnel, la réponse simple est qu'elle a souvent généré plus de frustration que de succès. Après plus de 10 ans, nous n'avons pas d'exemple d'ampleur permettant de dire que l'automatisation nous a permis d'économiser notablement l'humain sur les activités. En fait, les rares fois où nous nous sommes approchés d'un mécanisme d'automatisation permettant de remplacer l'humain, c'était dans les processus d'installation et de mise à jour d'un système. L'installation automatisée unitaire d'une machine, machine virtuelle ou container est monnaie courante de nos jours. Nous y avons introduit depuis maintenant une dizaine d'années les processus de sécurité afin de s'assurer de leur déploiement. Cela passe par la mise en place automatique des guides de durcissement, l'ajustement des contrôles d'intégrités, l'application des règles de filtrages en fonction des composants déployés sur la machine et le transfert des journaux vers un puits de logs ou un SOC.

2.1. L'Installation automatisée, une opportunité pour assurer un niveau minimal de sécurité local

Nous avons commencé dans cette voie il y a une décennie avec les technologies d'automatisation intégrant des outils du type cfengine, puppet, ansible... et un simple outil de gestion de configuration pour suivre les révisions de code de type git, teamforge, … Ce type d'outils d'automatisation nécessite souvent un composant central qui a accès à la majorité de votre parc. Ce composant est donc sensible à plus d'un titre : il a des privilèges élevés sur les équipements pour les paramétrer et, comme évoqué plus tôt, il a accès à la majorité de votre parc. Une vulnérabilité sur ce dernier met donc votre système en bien mauvaise posture. Pour les nostalgiques des scores CVSS v2, la note "Target distribution" passe directement à « High » avec ce type d'outils car souvent les ¾ de votre parc sont alors impactés.

Afin de limiter ce défaut inhérent à ces technologies, nos premiers déploiements se sont faits en dissociant l'activité serveur et l'activité client locale à la machine. Le composant serveur n'était pas utilisé et nous avions une rupture protocolaire avec un autre outil pour pousser les « promesses » de paramétrage au client local. Un premier service se chargeait alors de pousser les nouvelles configurations à la demande d'un opérateur et l'application des nouvelles promesses se faisait soit par une action manuelle de l'opérateur, soit par une tâche planifiée qui vérifiait régulièrement la présence d'une nouvelle promesse pour l'appliquer. La carte d'identité de la machine (un fichier contenant les informations de configuration spécifiques et nécessaire à sa configuration) était alors paramétrée en local, en fait soit modifiée en SSH ou poussée par SFTP.

Ce type de technologies a l'avantage d'assurer en parallèle la vérification de conformité et le retour à l'état initial si un écart est constaté. L'implémentation des guides de durcissements des systèmes d'exploitation et des applications dans les promesses permet alors d'être notifié en cas de changement de configuration indésirable (message de log d'erreur signifiant un écart de configuration) et d'appliquer à nouveau la configuration cible souhaitée. Ainsi, même un administrateur local qui modifierait des droits sur certains fichiers par mégarde serait rattrapé au prochain passage de l'outil d'automatisation.

2.2. Impact sur nos équipes de conception et d'intégration

La mise en place de cette activité a eu trois effets sur nos équipes qui historiquement étaient composées d'intégrateurs systèmes, réseaux ou sécurité et administrateur.

Le premier, et le plus évident, est l'acquisition de compétences de type développement et/ou d'un nouveau langage pour les personnels en charge de déployer et paramétrer les systèmes. Anciennement, ce type de personnel connaissait des langages de scripting (shell, perl, python principalement) permettant d'effectuer quelques actions d'administration et d'automatiser au plus ces dernières, mais l'utilisait principalement de manière unitaire.

Figure 2 : Exemple de discussion de soucis d'avant

Ainsi, un composant ou un déploiement qui posait soucis pouvait se régler de manière assez unitaire. Le manuel du composant (commande man sous Unix) et la journalisation / code d'erreur de l'application permettaient d'identifier et traiter relativement simplement la panne.

Les processus de déploiement automatisés, eux, entraînent l'imbrication de nombreux composants avec des dépendances entre ces derniers. Une des conséquences a été de basculer les divers outils de scripts et paramétrage pré existant dans différents langages (shell, perl, python, …) dans un langage commun de l'outil de déploiement nécessitant une acceptation du changement par les personnels de l'équipe plus ou moins évidente au début de l'aventure. Une autre conséquence a été le diagnostic d'une panne qui s'est avéré un peu plus complexe. L'imbrication des composants bien plus forte entraîne alors une résolution qui pourrait presque s'apparenter à la recherche du patient zéro lors d'une intrusion informatique et du chemin d'attaque utilisé. La correction ne porte plus exclusivement sur le composant en erreur, elle peut provenir d'héritages appelés par d'autres fonctions. En effet, les langages d'automatisation permettent de faire appel à des héritages ou inclusions (include) dans le code. En plus il arrive qu'un de vos collègues, en corrigeant un autre défaut ou en livrant une nouvelle fonctionnalité, introduise une régression impactant exclusivement le déploiement de votre composant.

Figure 3 : Exemple de discussion de soucis de maintenant

Le deuxième est une perte relative d'autonomie sur les systèmes en question pour ajuster des soucis de déploiement sur le terrain. En effet, avant, il suffisait de connaître le composant posant soucis et de modifier sa configuration. Ce nouveau mode de déploiement impose, soit d'être désactivé

pour repasser “à l’ancien” et donc prive le système des prochaines mises à jour, soit de faire les modifications au niveau de la promesse souvent hébergée en central. Si votre système est distribué géographiquement et que vous faites face à un souci de déploiement qui vous prive de la liaison avec le site central, alors cette modification est rendue impossible. Ce type de technologie augmente la dépendance à l’accès au site central hébergeant les configurations. L’autre souci est que la modification des promesses directement sur le terrain doit d’être effectuée par une personne ayant une idée des impacts croisés. Le système étant automatisé, la modification va être poussée à toutes les machines impactées et comme dirait une citation pertinente dans ce cas : “on a tous une plateforme de validation, certains ont la chance de l’avoir distincte de la production”. Pour éviter ce souci, la posture retenue était principalement la désactivation du mécanisme d’automatisation sur la machine à modifier ; la modification “à la main” de la configuration impactée et le développement d’une solution pérenne en usine avant de redéployer cette dernière sur le parc après validation en pré production. Je précise ici, que nous avons majoritairement des déploiements déconnectés de l’usine comme évoqué en introduction. Nous avons cependant constaté une contrepartie positive à cette relative perte d’autonomie terrain, la situation entre le terrain et la pré production étant normée, l’assurance de diagnostiquer sur une plateforme similaire au client simplifie les phases de diagnostic. Il est facile pour l’administrateur terrain de s’assurer que la configuration déployée sur tous les composants est bien à jour et partager le numéro de version utilisé à l’équipe support.

Le troisième est le gain en validation. En effet, avant, il fallait vérifier sur toutes les machines que le paramétrage, souvent manuel, était correctement effectué. C’est d’ailleurs une des activités qui a basculé en premier sur l’automatisation afin de scripter ces vérifications. Grâce à cette méthode de déploiement, il suffit de valider la “recette de cuisine”, la technique d’application étant toujours la même, la validation sur une machine permet de s’assurer de la conformité de toutes les autres machines du même type … pour ce qui est couvert par l’automatisation.

Nos premières automatisations étaient assez ciblées. Nous étions en mode hybride (manuel et automatisation) et seuls les composants devant être installés et paramétrés plus d’un certain nombre de fois (souvent >5 en première approche) étaient inclus dans ce processus. Pour fixer ce seuil « S », nous estimions le cout de l’automatisation par rapport à la mise en œuvre manuelle avec une logique de type. $S = (\text{Cout installation manuelle} / \mathit{Cout\ de\ l'automatisation})$. En effet, pourquoi passer 2 jours (16h) à chercher à automatiser une tâche pour faire 3 installations qui prenaient 2h chacune à la main. Dans ce cas, le S vaut 8. Nous pouvions refaire encore 5 fois le travail à la main avant de rentabiliser.

Avec le temps, nous avons rapidement diminué le seuil pour lequel nous nous attaquions à l’automatisation pour finalement arriver quasiment au chiffre de “1”. En effet, nous nous sommes malheureusement rendu compte que les rescapés de cette automatisation étaient souvent les vilains petits canards posant problème en production. Les processus manuels de paramétrage n’étaient pas toujours pleinement respectés par les personnels en charge du déploiement. L’autre révélateur a également été la réinstallation de plateforme de pré production, parfois quelques années plus tard, où le sachant n’étaient soit plus présents, soit plus disponibles. Les parties du périmètre non automatisées rendaient alors ces réinstallations plus coûteuses, parfois à cause de leur complexité, parfois également du fait d’une documentation imprécise ou qui ne prenait pas en compte une évolution introduite entre deux versions qui changeait le mode opératoire. Tout ça pour dire, qu’il y a le « 1 » que l’on croit unique dans un monde idéal et le « 1 » terrain qui lui est plusieurs. L’automatisation au final de ces derniers permet d’assurer au mieux la reproductivité du processus.

Vous aurez toutefois noté que nous avons écrit plus haut « quasiment au chiffre de 1 ». En effet, il reste certains cas où soit le seuil S est soit vraiment trop élevé pour que le jeu en vaille la chandelle, soit que l’automatisation d’un composant génère des effets de bord comme évoqué plus haut de par l’imbrication, l’héritage et les tentatives de mutualisation des composants. Parfois, vouloir tout gérer en un point génère une complexité plus grande que le problème que l’on veut résoudre. Comme communément dit, si un problème est trop complexe, c’est qu’il n’a pas été assez découpé. Cela nous a valu quelques modules qui se sont trop complexifiés et que l’on a parfois séparés en deux. La quête de la standardisation / unification ayant elle aussi ses limites.

2.3. Impact sur la chaîne de production et de déploiement

Cependant ce processus de déploiement favorisant la sécurité avec un mode de déploiement disjoint du logiciel d'automatisation, posait un souci sur le suivi des déploiements depuis un poste central.

Jusqu'ici, nous avons discuté du résultat, mais peu de sa mise en œuvre … or cette partie est prépondérante dans le processus. Afin de rendre viable ce projet, nous avons donc utilisé un gestionnaire de configuration (type teamforge, github, …) et un processus de déploiement continue sur notre plateforme de développement et intégration.

La totalité des configurations était regroupée dans un unique dépôt de gestion de configuration. Le CI/CD dans sa première version était d'une « approche artisanale » loin des outils actuels. Les tests unitaires et/ou syntaxiques se faisant en local à chaque commit ou sur demande de l'utilisateur (et donc à son bon vouloir). Le processus de « push » des modifications sur la plateforme d'intégration comportait le suivi des versions. Ce « push » poussait alors la nouvelle configuration générée par la forge sur un espace de stockage. Ce stockage était partagé avec tous les postes cibles et la mise en place de tâches périodiques assurait le déploiement automatisé des configurations quelques minutes plus tard. La bonne application des configurations pouvant être vérifiée au moyen de l'outil (affichage de la version des configurations).

La mise en place s'avérait donc assez simple dans son concept et composé de :

- un traditionnel outil de gestion de configuration
- un script de packaging des configurations
- une livraison via un « push » sur la plateforme d'intégration du code, scripts et binaires nécessaires
- une action planifiée qui allait chercher régulièrement la présence de mises à jour pour l'exécuter en local le cas échéant sur chacun des équipements
- un outil de vérification de la bonne application des modifications.

Les modifications notables dans la chaîne de production étaient le lieu où mettre à niveau ses configurations qui changeait, la transcription dans un autre langage des configurations et le déploiement automatisé sur toutes les machines.

2.4. Retour d'expérience

Gain en qualité, reproductibilité du processus, validation d'un équipement dans un contexte permettant de s'assurer que tous les autres dans le même contexte seront similaires, sécurité implémenté dans le processus de déploiement, rapidité de déploiement des correctifs sur le parc. Voici quelques-uns des gains générés par cette méthode dans notre environnement.

Cependant, la pièce a également une deuxième face. Il n'y a pas de gain réel financier sur la partie conception / intégration / premier déploiement. Le gain obtenu en déploiement est compensé par la mise en place et l'ingénierie plus coûteuse du mécanisme d'automatisation. Ce coût est d'ailleurs, sans surprise, d'autant plus élevé que vous devez gérer des configurations de déploiement distinctes (version cousine évoquée dans l'introduction) par rapport à un déploiement type toujours identique. Cette nouvelle méthode a conduit à une relative perte d'autonomie locale sur les machines automatiquement déployées. Par exemple, si lors d'un déploiement ou d'une mise à jour, la configuration réseau poussée sur une machine est erronée, alors la machine va perdre la connectivité et ne sera plus joignable. Si un administreur est dépêché en local pour investiguer, identifie la cause et modifie la configuration à la main alors la machine va récupérer le réseau et de nouveau télécharger la configuration erronée entrainant de nouveau la perte de connectivité. Pour corriger définitivement, il faut soit modifier la matrice de configuration en central (mais en générant ici un fork par rapport à la livraison) soit demander une nouvelle livraison (via une nouvelle « build »). Pour terminer sur le sujet technique, la méthode mise en place donnait un faible retour sur le statut du déploiement sur le parc (ou du moins avec latence) car pour des raisons de sécurité le service d'automatisation ne disposait pas du composant « serveur » afin de limiter la surface d'attaque possible. Seul le composant client était déployé en local. Le statut de déploiement était vérifié régulièrement (tâche planifiée) en local et

poussé en central par un mécanisme « custom » mais à une fréquence trop faible pour la patience humaine. Nous constations régulièrement des connections SSH d'administrateurs impatients pour aller vérifier en local avant même d'avoir le retour d'information du mécanisme « custom ». Ce point était source de frustration.

Le métier d'intégration se déplaçant vers le codage (IaC et CaC), certains employés se sont retrouvés moins à l'aise lors de ce changement. Sur ce sujet impact sur le personnel, pas de si grandes nouveautés à première vue. Il faut toujours maîtriser l'intégration système comme avant. Le plus grand changement était le passage à un langage lié à notre outil d'automatisation qui était différent des outils de Scripting habituels et prendre une nouvelle habitude à savoir ne plus modifier les configurations directement sur les machines mais via la chaine de production. Cette nouvelle habitude fut plus ou moins longue à mettre en place suivant les acteurs. Nous sommes passés par une étape où certains continuaient de faire des modifications (souvent) de dernière minute pour être prêt pour une démonstration interne ou une mise à disposition pour validation … modifications qui ne survivaient pas plus que quelques minutes avec, à la clé, quelques sueurs froides du responsable du composant au moment de présenter ses résultats (qui ne marchaient plus).

D'un point de vue exploitation ultérieure des travaux, nous nous sommes aperçu que le packaging des configurations dans un dépôt/archive unique a rendu très complexe la réutilisation du travail d'intégration en externe du projet d'origine. L'intrication des modules/configurations ne permettant que difficilement de séparer la partie spécifique projet (c'est à dire potentiellement confidentielle) et la partie « commune » et donc potentiellement exportable publiquement. Ce dernier point étant identifié comme important pour les prochaines évolutions afin de partager au plus grand nombre les travaux et ainsi améliorer le retour sur investissement avec d'autres projets dans notre structure.

3. En route vers l'infrastructure codée

3.1. La chaîne de production, maillon essentiel de l'IaC et CaC

L'objectif de notre nouvelle chaîne de production a été la diminution des actions humaines et par conséquent des erreurs associées, mais aussi d'améliorer la robustesse de notre processus d'intégration et de livraison.

Le gestionnaire de configuration devient l'outil central de notre processus d'intégration, validation et livraison. Toutes les données liées à un projet passent au moins une fois par celui-ci au cours de l'intégration/livraison d'une version. Ce nombre de passages peut se trouver augmenté dans le cas d'un développement interne car ces « projets dans le projet » suivent également ce même cycle de vie avant d'être mis à disposition de la « build finale ». Finalement c'est la même logique que pour les développements logiciels. Le code principal fait appel à des librairies qui sont soit des composants externes, soit des développements d'une autre personne ou équipe interne. La compilation de tous ces composants permet alors de faire le produit final livré au client.

Les tests sur les données en entrée ont été améliorés et déportés sur des VM/container (chaque test bloquant la suite de la procédure)

- Ajout de tests de code, tests sécurité, qualité de code, linter
- Gestion des tests unitaires
- Versionning
- Déploiement sur les plateformes intégration (CI/CD)
- Gestion des tests système sur la nouvelle version déployée
- Affichage des résultats des tests système dans un tableau de bord

L'évolution de notre processus de livraison est un bon exemple de notre volonté d'amélioration de notre intégration continue et automatisation associée. Les actions humaines ont été réduites au minimum (lancement d'un pipeline CI/CD, vérification du checksum de la livraison). Les données en entrées sont vérifiées (mise à jour ou installateur de COTS, packages OS, packages internes, documentations…) puis regroupées pour construire « build » l'image de la livraison.
Sans modification des données en entrée, le « build » d'une nouvelle livraison fournira une image strictement identique dans son contenu permettant ainsi, soit de reproduire une version antérieure au

besoin, soit de reproduire depuis une autre chaine de « build » identique mais déportée exactement la même version. La confiance dans les sorties des pipelines CI/CD a été renforcée, la reproductibilité d'une erreur étant bien sûr un avantage pour sa résolution.

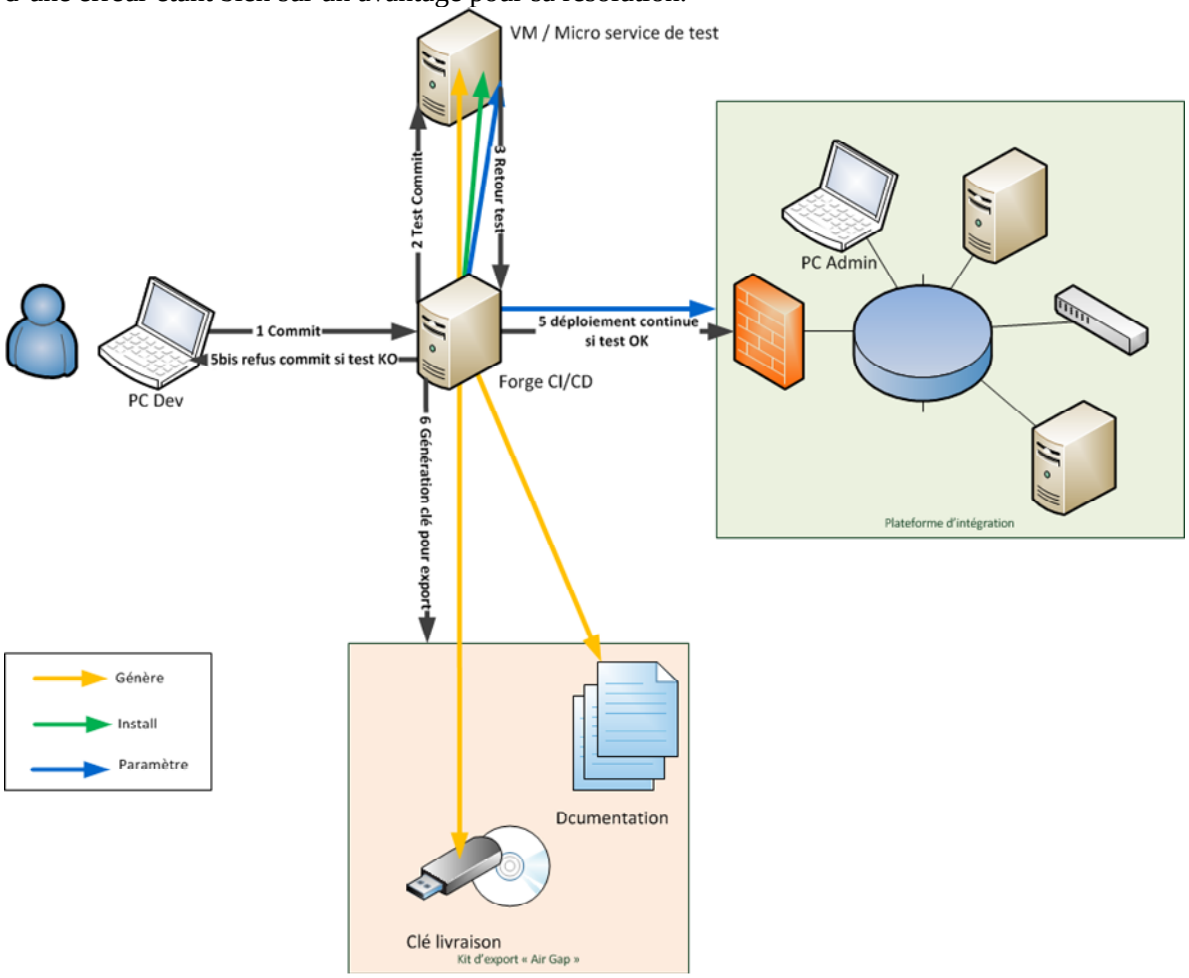

Figure 4 : Chaîne CI/CD (vue simplifiée) et génération de son relais via « Air gap »

Un second objectif de notre nouvelle chaîne est de réutiliser au maximum le travail d'intégration entre nos différents projets. Nos travaux sont divisés en projets unitaires (dans l'outil de gestion de configuration) correspondant souvent à un produit et/ou service unique. L'intégration d'un service est maintenant divisée en deux :

- Une partie commune aux différents projets, partageable
- Une partie spécifique contenant des infos confidentielles ou du moins spécifiques au projet

Prenons l'exemple de la configuration du service SSH sur deux projets distincts. La partie commune sera le module permettant de déployer un service SSH fonctionnel avec une configuration déjà sécurisée. Dans la partie spécifique projet, le projet « A » utilise ce module avec le paramètre « allow_groups » autorisant les utilisateurs « admin » à se connecter. Le projet « B » utilise ce même module avec le paramètre « ciphers » autorisant un algorithme moins robuste mais nécessaire à la connexion d'un système légèrement obsolète.

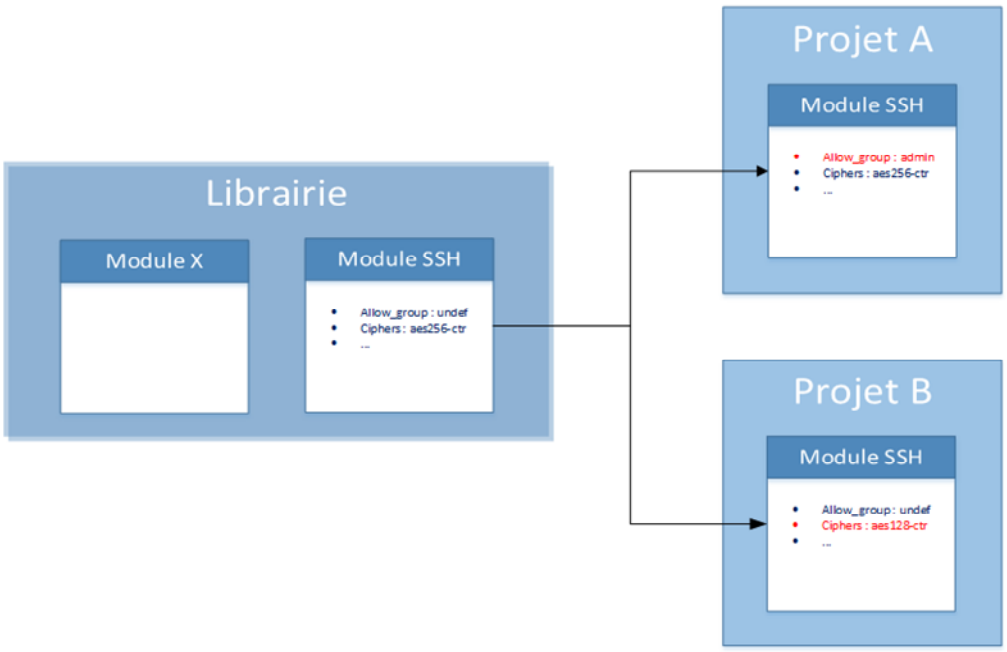

Figure 5 : Exemple d'instanciation d'un module

Un 3ème projet souhaitant utiliser ce module SSH pourra

- Utiliser le module sans modification s'il correspond au besoin
- Apporter une modification au module sous la forme d'un paramètre optionnel (pas d'impact sur les autres projets)
- Modifier le comportement par défaut du module (impact sur la validation des projets « A » et « B »)

Un retour d'expérience étant que garder une nouvelle version d'un module iso fonctionnelle (et donc utiliser des paramètres optionnels pour ajouter des fonctions) est une bonne pratique qui est encouragée dans notre chaîne d'intégration / production afin d'éviter de créer des régressions sur les autres projets « cousins ». Quand un changement plus profond est nécessaire, une revue avec les différents projets est alors réalisée afin de voir comment et quand introduire un éventuel changement majeur. Il est possible de désynchroniser le changement par projet en réalisant des « fork » dans la forge logicielle et en tirant une branche de maintenance, mais nous nous efforçons de limiter ces cas au plus possible afin d'éviter la multiplication des « forks » qui finissent par augmenter les coûts, notamment au niveau de la validation.
L'objectif ici est d'accroitre au mieux le retour sur investissement de l'activité d'ingénierie de cette activité d'automatisation qui, contrairement à des environnements cloud, traite moins de volume de par les quantités livrées et le mode de déploiement en « air gap ».

3.2. Aperçu du concept mis en œuvre

3.2.1. Méthode pour industrialiser l'installation au-delà de l' « air gap »

Comme évoqué en introduction, le souci est de se rapprocher au plus de l'esprit CI/CD sans pouvoir l'atteindre avec le fameux « air gap » sur le trajet. La première partie du trajet étant constituée de l'usine de production décrite au paragraphe précédent avec la génération du média de livraison.

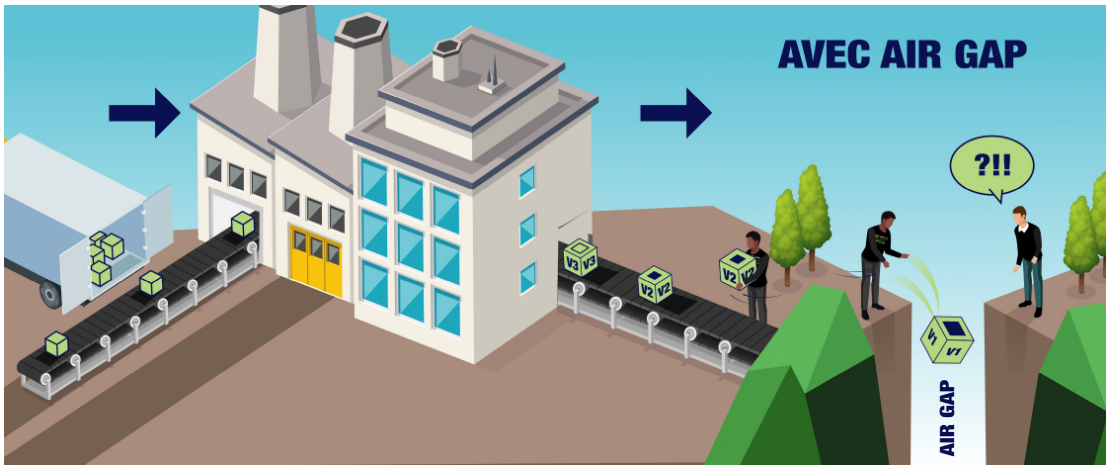

Figure 6 (rappel) CI/CD … avec Air Gap … il y a comme un os

L'idée était alors de trouver un moyen de rebondir pour que la chaine puisse continuer à s'exécuter au mieux une fois de l'autre côté …

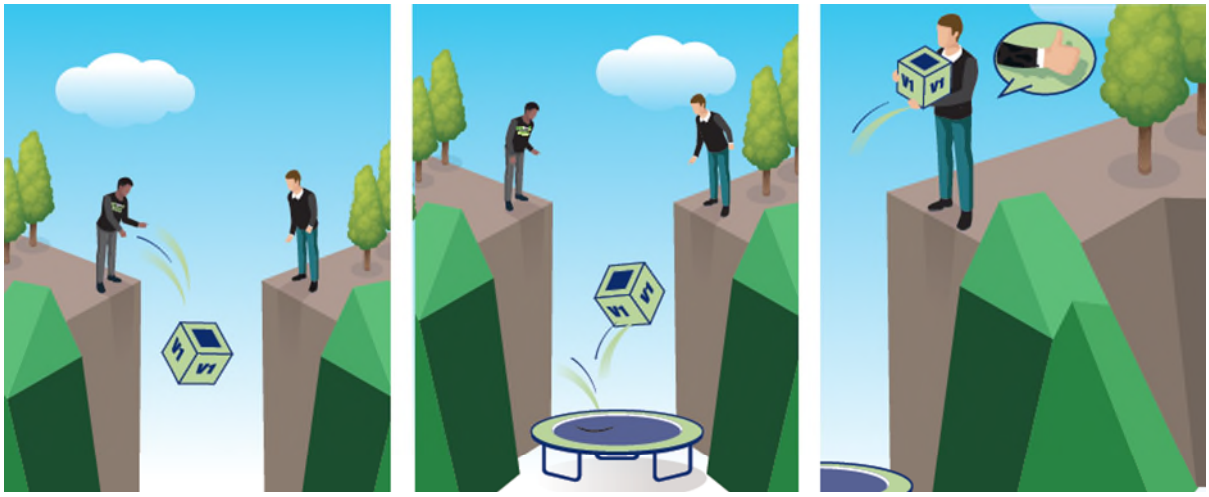

Figure 7 : Chaine CI/CD comment rebondir au-delà de l'« Air gap »

Ce rebond est dans notre implémentation représenté techniquement par une machine dédiée qui prend en entrée l'architecture cible du déploiement. La configuration de l'architecture type de déploiement (outil Seed) comprend notamment les informations telles que

- Nom de domaine du système
- Plan d'adressage retenu
- Nombre de postes de travail
- Nombre de sites distants
- Equipements sur les sites distants
- Types d'interconnexions avec les sites distants
- …

Les cartes d'identité présentes en local sur les hôtes dans nos premiers déploiements sont désormais remplacées par une carte d'identité contenant les informations du système dans son intégralité. Ce fichier de configuration porte le même nom « seed » que l'outil qui sert à le générer. Cet outil seed est un développement interne qui a vocation à présenter une IHM à la personne en charge du déploiement afin qu'elle puisse configurer la topologie de son déploiement. Il comporte également des choix de types de déploiements système afin de gagner du temps (un utilisateur avancé peut générer à la main ce fichier) et de générer des configurations systèmes pré-testées en usine. Par exemple, de choisir si tout le système est déployé en local ou si certains composants sont sur des sites distants qui peuvent être distincts, nécessitant alors une liaison VPN pour y accéder.

Le fichier « seed » alors importé dans la machine initiale « garden » qui se charge alors d'automatiser la génération du système en fonction des contraintes déclarées (oui, nos équipes ont eu le petit côté poétique de la graine plantée dans le jardin qui génère un bel arbre et des fruits …). Cette machine génère donc à partir de l'architecture (seed) et de l'image de la livraison de notre chaîne CI/CD les éléments pour réaliser le déploiement en mode « air gap » chez le client final, à savoir :

- Média d'installation automatique (Kickstart, Preseed ou ESXi suivant le projet) des machines de bases incluant les partitionnements et paramètres de durcissement nécessaires au niveau de l'image
- Script d'installation des switchs
- Calcul des configurations des pare-feux réseaux
- Calcul des configurations VPN
- …

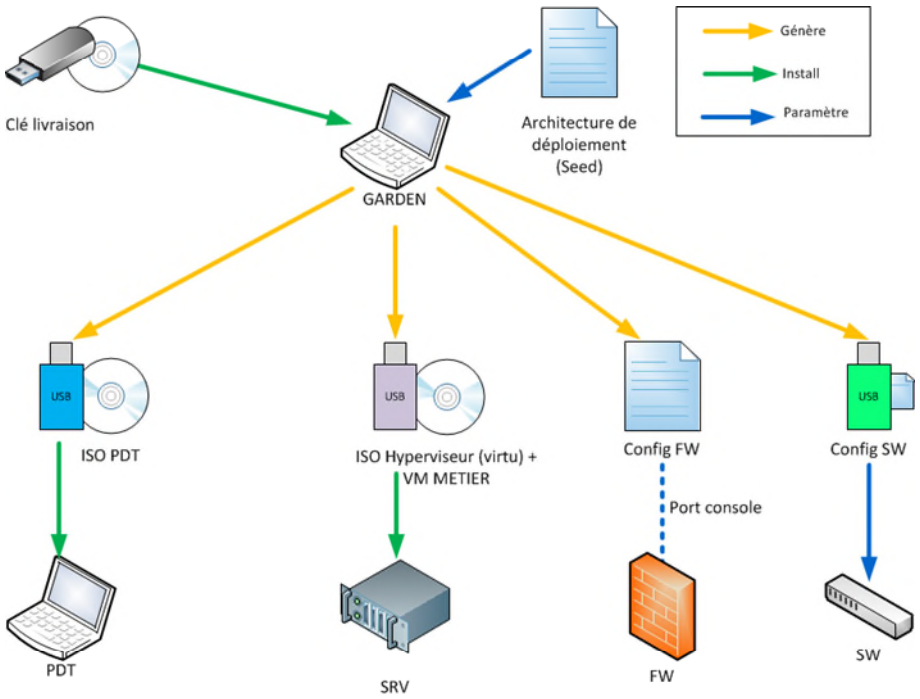

Figure 8 : « Garden » : Usine de production locale pour déploiement initial

Ensuite, la post-installation des machines déployées est réalisée en appliquant les spécifications par type de machines, ces dernières se chargeant de spécialiser les machines pour leurs tâches métiers. Par exemple, le kickstart va déployer un linux de base. Les post-installations vont le transformer en serveur d'administration en y installant les logiciels manquants et paramétrant ces derniers ainsi que les règles de pare-feu local en fonction de son adresse IP et des applicatifs déployés pour n'ouvrir que les ports nécessaires.
Une finalisation, ou plutôt une personnalisation éventuelle est possible par une surcharge de certaines variables depuis la console d'administration du système pour ajuster le fonctionnement des machines. Ce dernier point a été rajouté pour permettre une souplesse locale lors du déploiement entre plusieurs modes de fonctionnement pour un composant.

3.2.2. Méthode pour industrialiser la mise à jour au-delà de l' « air gap »

Dans une optique de simplification des processus, chaque version générée par notre usine de « build » permet à la fois une installation d'un nouveau système et la mise à jour d'un système existant.

La mise à jour d'un système a été simplifiée grâce à la présence du service de télédistribution. Les promesses étant fournies sous forme de paquets, il suffit alors de mettre à jour le miroir du système

avec le contenu de la clé de livraison pour permettre ensuite de diffuser les mises à jour que celles-ci portent sur les logiciels ou sur leurs paramétrages.

3.3. Le calcul des politiques de sécurité à l'échelle du système déployé

Sur une nouvelle génération de programmes, nous nous sommes donc attelés à essayer de diminuer les soucis vus précédemment. Nous avons d'ailleurs encore augmenté le taux d'automatisation des systèmes. En effet, nous avons aussi introduit l'automatisation de la politique de sécurité des composants réseaux dont pare-feu et vpn alors qu'avant nous nous limitions aux ACL des pare-feux locaux. La sécurité des composants serveurs ayant progressée avec les dernières générations d'outil, nous les avons intégrés afin d'avoir le retour direct de l'état des machines cibles. Afin de permettre une surcharge particulière, nous avons aussi intégré des composants permettant un ajustement centralisé de certains paramètres dans des variantes la plupart du temps prédéfinies en usine mais aussi une personnalisation possible au niveau du déploiement.

Le coût et l'importance de l'ingénierie a donc encore augmenté, les imbrications entre les composants étant maintenant passées d'une machine à tout le système. La partie émergée de l'iceberg c'est un déploiement encore plus facilité et une politique de sécurité adaptée en fonction du déploiement des composants sur les différents sites, tout étant calculé au choix de déploiement. La partie cachée de l'iceberg, c'est un système bien plus complexe d'ingénieries, une imbrication d'appel à multiples niveaux des classes dans les promesses. Nous sommes ici pleinement dans une infrastructure (IaC) et une configuration (CaC) qui est "codées". La compétence nécessaire à l'écriture de ses promesses se rapproche encore plus ici de l'ingénierie logicielle et demande encore plus dans cette dernière génération des compétences que certains administrateurs / intégrateurs historiques n'ont pas. L'accompagnement de ces derniers par des formations est donc nécessaire.

Maintenant, quand le système fonctionne bien, le déploiement se fait de manière très fluide, et il est possible d'envoyer directement le matériel sur leurs sites d'emplois sans pré-installation. L'installation se résumant à un démarrage sur une clé USB avec l'image de base, le nom/IP de la machine et la finalisation se fait par le processus d'automatisation centralisé. Le déploiement de la sécurité (durcissement, ACL, …) tant au niveau d'une machine que du parc (ouverture des règles de pare-feu et VPN via l'automatisation) se voit donc amélioré. L'absence de paramétrage manuel non validé assure un niveau de qualité et de sécurité plus élevé. Le revers de la médaille est lorsqu'il y a un accroc, le diagnostic est plus compliqué et la correction est encore moins facile sur le terrain. L'impact système d'une modification mal calibrée étant plus important.

3.4. Mise sous (auto) supervision automatisée

Tout comme les politiques de sécurités évoquées au paragraphe précédent étaient calculées automatiquement par rapport au déploiement souhaité, la mise sous auto-supervision a été automatisée. Chaque machine déployée est créée avec le paramétrage du transfert des journaux vers un serveur Syslog qui a été automatiquement déployé dans l'architecture de base. Les applications déployées sont également automatiquement paramétrées pour renvoyer leurs journaux. Nous collectons donc automatiquement les journaux OS et applicatifs du parc déployé. Cette phase est nécessaire mais pas suffisante.

Afin que les journaux collectés soient pris en compte et comme nous savons exactement ce qui est déployé, les journaux sont automatiquement redirigés dans des archives par type de machines et/ou sites d'origine. Les index des outils de gestion des logs sont automatiquement créés toujours en cohérence avec les machines et applications déployées.

Un « Câblage » automatique via le déploiement de parseurs idoines vers les jeux de règles de corrélation et le déploiement automatique de ces règles et des tableaux de bord associés aux composants déployés est également réalisé en poussant les paramétrages via les API disponibles dans le SIEM utilisé.

A la fin de cette séquence d'installation, nous avons donc une chaîne complète : journaux applicatifs et OS ➔ collecteur de logs ➔ index du SIEM ➔ Parseurs ➔ règles de détection et tableau de bord pour l'auto-supervision qui est effective de manière automatique. Le système que l'on

vient de déployer est alors automatiquement mis sous supervision. Comme pour les règles de sécurité, si un composant est ajouté ou retiré lors d'une mise à jour de l'architecture, alors le déploiement de la supervision sera lui aussi adapté par le déploiement des nouvelles configurations nécessaires ou suppression de configuration.

3.5. Prise en compte de plan de secours

L'automatisation permet de prévoir certains plans de continuité d'activité dès la mise en œuvre, notamment par la prise en compte de la gestion de la haute disponibilité (HA) des composants. Les configurations pourront être déployées en HA ou sans suivant l'architecture sélectionnée.

L'automatisation utilisée offre aussi la possibilité de modifier rapidement le comportement d'une machine en lui changeant sa carte d'identité (utilisée pour appliquer sa configuration cible au déploiement pour rappel) permettant d'ajouter temporairement un nouveau service à une machine afin de reprendre l'activité. Ce scénario est utile quand une panne matérielle d'une machine non redondée arrive. La redistribution de ses services est donc facilitée par ces mécanismes même s'il n'existe pas de mécanisme système de redondance.

3.6. Système logiciel faiseur de système … fait par des humains et donc faillible comme les autres

Ce système d'automatisation du déploiement et du maintien du SI est partiellement caché (enfoui) dans son système et a sa propre architecture et complexité masquées aux administrateurs systèmes classiques. Sa complexité est quasiment invisible pour les exploitants quand tout fonctionne parfaitement (n'est-ce pas son rôle après tout). Il peut cependant arriver que des bugs se glissent dans cette chaîne, et dans ce cas, la complexité de diagnostic est augmentée pour l'utilisateur final. On touche ici les limites de l'implémentation du concept DevOps dans notre métier. Cet « air gap » entre le « build » et l'exploitation est en contradiction avec la volonté DevOps de rapprocher les équipes opérationnelles des équipes d'intégration/développement. Un bug qui serait corrigé en quelques heures sur un SI ayant appliqué le DevOps à 100% peut être corrigé en quelques jours si le support est difficile, voire en quelques mois si une nouvelle version est nécessaire.
Cette distance entre « Dev » et « Ops » peut aussi générer beaucoup de frustration des deux côtés. La vie d'un SI est faite de modifications mineures mais pouvant faciliter grandement le travail des opérationnels.

Les erreurs dans le code peuvent générer des erreurs dans le déploiement … les « bonnes ACL » peuvent aussi rapidement devenir les mauvaises car trop permissives si le code a mal calculé les règles de filtrage lié au déploiement. Le point positif est que la reproduction en usine est plus aisée si le partage du fichier (seed) de déploiement est possible car il est possible de reproduire ainsi les conditions de l'erreur. La validation de ces générations de configuration (en usine) revêt une importance encore plus grande. La mise en place d'automatisation de certains tests pour passer l'échelle est obligatoire afin de pouvoir tester la combinatoire possible de cas de déploiement.

Ce système étant le point central de pilotage du SI avec accès relativement privilégié aux machines (après tout, il dispose des privilèges pour les paramétrer), ce composant devient une cible de choix pour un éventuel attaquant. Les contrôles d'accès pour utiliser cet outil sont à soigner en respectant les bonnes pratiques courantes d'administration telles que l'accès par les seuls profils administrateur depuis des zones réservées à ces derniers ou l'usage d'une solution à multiples facteurs. Les activités de maintien en condition de sécurité (MCS) sont primordiales sur ce type de composant avec une bonne réactivité afin de couvrir au plus tôt une faille sur le logiciel.

3.7. Retour d'expérience

Par rapport à notre première génération évoquée au chapitre 2, nous avons pu observer les points suivants :

- fiabilité du déploiement et niveau de qualité / reproductivité augmenté,
- gain encore plus prononcé en temps sur les processus de déploiement (temps de déploiement divisé par 2 environ),
- gain niveau de sécurité (notamment gestion des règles de filtrage) cohérente au niveau système grâce à l'automatisation en évitant les erreurs de paramétrage (oublie d'ajout ou oublie de suppression de règles inutiles suite à un dé commissionnement d'un service ou équipement),
- bascule des compétences des équipes encore plus vers l'ingénierie logicielle, certains profils réticents à cette nouvelle manière de concevoir et déployer s'intègrent plus difficilement dans ce type de projet. Il convient d'accompagner les personnels et de les former à l'usage de ces outils. Des notions de base, mais néanmoins solides, de développement et d'identification de complexité de code pour les éviter, seront recherchées,
- augmentation de la complexité de conception,
- augmentation de la complexité de diagnostic en cas de panne sur le terrain, mais reproduction en usine facilitée et cas de panne moins nombreux,
- expérience plus coûteuse qu'espérée initialement, le retour sur investissement se fait sur la mutualisation de certains développements et le partage des modules communs avec d'autres projets.

Par rapport au dernier point, le vrai gain se fait donc dans le cadre d'une démarche d'entreprise d'utiliser ce type de déploiement en tant que standard. La mutualisation se fait alors sur les modules dits « communs » et sur l'usine de production logicielle CI/CD. L'implémentation dans le cadre d'un seul projet fera gagner sur la qualité et vitesse d'exécution en déploiement, mais moins probablement en coût suivant les nombres de déploiements visés. Pour rappel, ici, nous ne sommes pas dans des déploiements de clouds publics mais dans le déploiement de nombre raisonnable de systèmes dont le nombre reste encore sur deux chiffres.

4. Pistes d'évolutions

Sans être exhaustif, voici quelques activités dans notre radar d'évolution et d'amélioration continue :

- Augmentation du périmètre de tests automatiques au niveau système, il n'y en a jamais assez quand un système peut proposer une combinatoire qui est difficilement testable humainement ou dans un temps raisonnable. L'objectif n'est pas que de réduire (éventuellement) les coûts, mais aussi assurer une validation la plus rapide possible afin de pouvoir délivrer des correctifs au client final dans des délais raisonnables malgré la complexité du système.
- Industrialisation des plans de replis pour la résilience incluant la sécurité (ACL & Co) pour éviter les plans de replis avec un niveau de sécurité moindre et surtout les actions manuelles nécessaires dans ces moments stressants.
- Rajouter une section sur la prise en compte par défaut des modes de replis ou conservatoire dans la conception des systèmes. Aujourd'hui la résilience n'est incluse dans l'automatisation quasiment que pour la gestion des pannes (défaillance avec de la haute disponibilité) mais la gestion des postures de replis partielle, isolation … pour des raisons de sécurité en réponse à une attaque n'est pas encore dans les mœurs… mais ce sujet pourrait faire l'objet d'un développement dans une autre publication.

5. Conclusions

Nous avons initié il y a plus d'une décennie notre transformation vers l'automatisation et une démarche de type IaC / CaC (Infrastructure codée ; Configuration Codée) permettant de garantir un niveau de qualité reproductible des déploiements.

Côté rentabilité, il y a un gain de temps notable sur les phases de déploiement, mais la charge fortement augmentée sur la partie mise au point initiale. Le gain financier total n'est possible à court terme que pour de gros déploiements. Cependant, le gain se fera sur le long terme avec l'opération de MCO simplifiée. Il faut prévoir dans ses modèles de coût et de plan de charge ce transfert d'activité au démarrage du projet.

Afin d'améliorer cette rentabilité, la réutilisation des parties de codes sur d'autres projets doit être recherchée, pour synergie, mais est aussi passage obligé pour rentabiliser l'investissement initial de l'usine logicielle car, à défaut d'être dans des infrastructures hautement scalables comme le cloud, il faut générer le volume par d'autres artifices (les projets cousins). Cette initiative a donc besoin d'être dans une démarche générale d'entreprise pour assurer la rentabilité.

La sécurité, si intégrée dans la conception initiale, est un des grands gagnants de cette nouvelle norme. L'intégration dans la conception initiale permettra notamment un durcissement automatique du système en fonction des différents types de déploiements possibles de votre système calculés et appliqués au déploiement et durant toutes les évolutions du système au lieu du traditionnel durcissement unitaire de plusieurs machines autonomes. L'ajout d'un service dans le SI entraîne automatique l'ouverture des flux associés, le changement de durcissement du serveur si nécessaire et l'installation du dit service avec les règles de durcissement de l'application. Le retrait de ce service, entraine automatiquement la fermeture et adaptation du durcissement pour revenir à l'état maximal possible sans erreur humaine (souvent un oubli ou manque de temps) possible dans le processus. Un autre grand gagnant, non évoqué jusqu'ici, est le processus d'homologation du système d'information ainsi déployé. Par conception, la reproductibilité du déploiement permet d'augmenter la confiance (élément clé d'une homologation) dans le déploiement. Les audits techniques intrusifs peuvent principalement se dérouler sur une plateforme de pré-production ou sur un seul des systèmes de production (les autres étant des jumeaux) limitant ainsi au maximum les impacts sur la production à quelques relevés tout en étant certains de regarder la même chose si la version est identique.

Côté client, le système se déploie plus vite et de manière plus fiable, mais les diagnostics sont également complexifiés en cas de problème ce qui peut générer une légère perte d'autonomie qu'il faut intégrer. Cette perte d'autonomie est également présente dans la capacité à customiser localement un déploiement. Encore plus qu'avant, il faut rechercher une proximité avec son client par des points réguliers afin de leur laisser une capacité d'adaptation. Cela passe par la mise en œuvre de mécanismes de variables / options qui doivent être définies conjointement.

Côté personnel industriel, cette démarche a entrainé une évolution du métier historique. Nous sommes passés d'une équipe majoritairement composée d'administrateurs système à une équipe d'administrateurs système cumulant les rôles de développeurs (en tant qu'outils) et disposant de compétences d'ingénierie système. Cette mutation demande d'accompagner les équipes au travers de formations ou de recrutements ad hoc afin que l'initiative soit un succès. A noter, en gestion de compétences, les personnels prenant l'habitude que tout fonctionne bien et donc travaillent moins souvent sur le sujet (finalement, les pannes aidaient à un entraînement régulier), les expertises sur certains sujets se sont concentrées sur moins de personnels … ce qui peut poser soucis lors des phases de diagnostic plus complexes ou en cas de départ d'un personnel. Il faut donc beaucoup plus anticiper les changements de personnels si une ressource devient unique pour assurer la résilience.