Microsoft Azure

DevOps pour le Cloud… et réciproquement…

Version 1.0
Publication : 13 mai 2014
Auteurs :
Hervé LECLERC (CTO - Alter Way)
Stéphane GOUDEAU (Cloud Architect - Microsoft)

Alter Way

Microsoft | Openness

Microsoft

Avertissement

Ce document s'adresse aux architectes, aux développeurs, aux chefs de projet, ainsi qu'aux responsables d'infrastructure qui souhaitent avoir une vision plus complète sur les objectifs et les moyens liées à la mise en oeuvre d'une démarche DevOps dans Azure.

Introduction

« DevOps ». Au tout début, il s'agit d'une idée.

Une idée portée par quelques pionniers, parmi lesquels John Allspaw, Paul Hammond et Patrick Debois. L'idée d'un monde dans lequel chaque composante de l'organisation d'une entreprise collaborerait efficacement pour l'atteinte des mêmes objectifs. Cette idée est à l'origine d'un mouvement dont l'influence dans l'IT ne cesse de s'amplifier.

A cette accélération correspond celle de l'adoption du Cloud. Et force est de constater que le succès de la mise en œuvre d'un démarche DevOps et la réussite d'une évolution vers le Cloud sont intimement liés.

Cette idée, de multiples solutions logicielles Open Source ou propriétaires, nativement intégrées ou non sur la plateforme Microsoft Azure, permettent aujourd'hui de l'ancrer dans la réalité.

Cette idée, avec ce document, nous nous proposons de vous la faire partager.

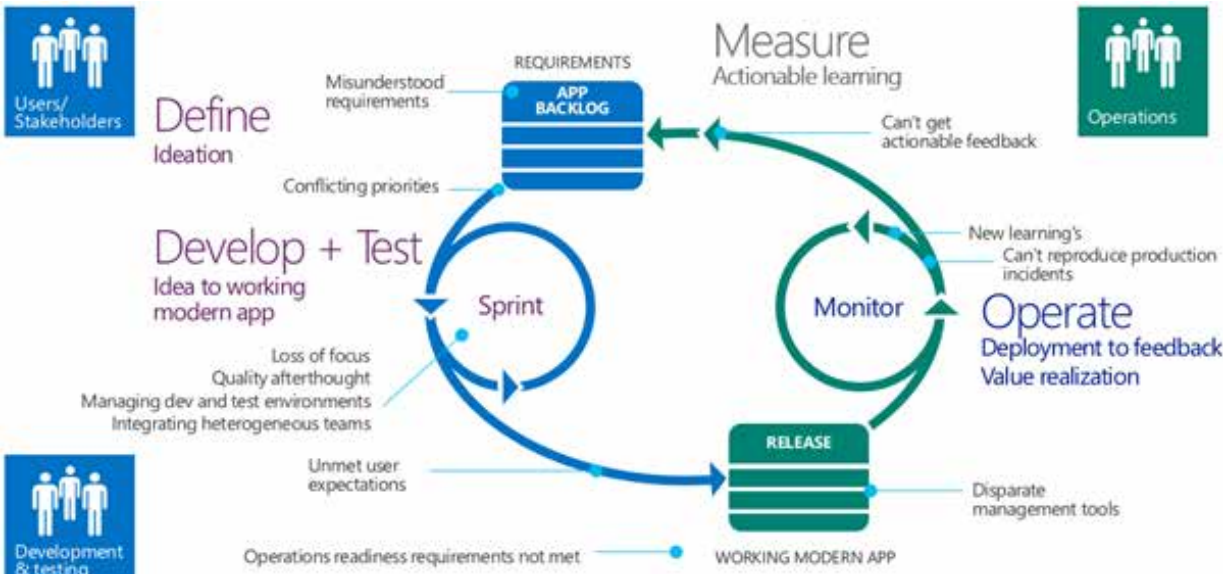
Sommaire

La démarche « DevOps »

POURQUOI ADOPTER UNE DÉMARCHE « DEVOPS » ?

« DevOps » est un terme inventé par Patrick Desbois en 2009. Comme il le dit lui-même : « There never was a grand plan about thinking about DevOps as a word, I was busy working on a project which had agile development and I was so envying these people about their methodology and I like to do something similar in my world system administration, so I called it initially agile system administration, but it's pretty long. I just picked the word DevOpsDays as Dev and Ops working together because agile system administration was also too narrow on focusing on sysadmins only. ».

La démarche « DevOps » est donc issue d'un constat : celui des conséquences négatives issues de la séparation des développeurs et des responsables opérationnels d'une organisation.



Parmi les obstacles fréquemment rencontrés, citons :

- La rigidité des processus liés aux transitions entre les différentes étapes du cycle de vie logiciel.
- L'absence de coordination dans la gestion des releases :
  - o Processus ad-hoc, souvent non outillés, conduisant à un manque de visibilité sur le statut de la release
  - o Insuffisance de la documentation et des artefacts de configuration et déploiement
- L'inefficacité des outils de collaboration :
  - o Manque d'outils offrant des boucles de rétroaction continue avec les utilisateurs
  - o Manque d'intégration entre les outils utilisés pour gérer les charges de travail développement et opérations)
- La durée trop élevée des cycles de déploiement (insuffisance de l'automatisation de certaines opérations qui conduit à des échecs en production)
- Les différences entre environnements de production et environnements de développement complexifiant le diagnostic et la résolution des incidents de production
- L'absence de traçabilité entre le code source et le déploiement en production correspondant
- La nécessité de disposer de données et des configurations de production pour reproduire les conditions d'erreurs sur les plateformes de tests.

L'ensemble de ces problématiques s'inscrivent dans un « Wall of Confusion » (formule d'Andrew Clay Shafer) entre développeurs et responsables opérationnels au sein d'une organisation.

Souvent, les développeurs se concentrent sur la réponse aux exigences fonctionnelles par la production d'un livrable, mais pas sur la maintenance de la solution en fonctionnement opérationnel. De leur côté, pour limiter les risques de problèmes inattendus, les administrateurs système tentent d'édicter des règles pour les déploiements qui s'avèrent être des contraintes architecturales pour le développement d'applications.

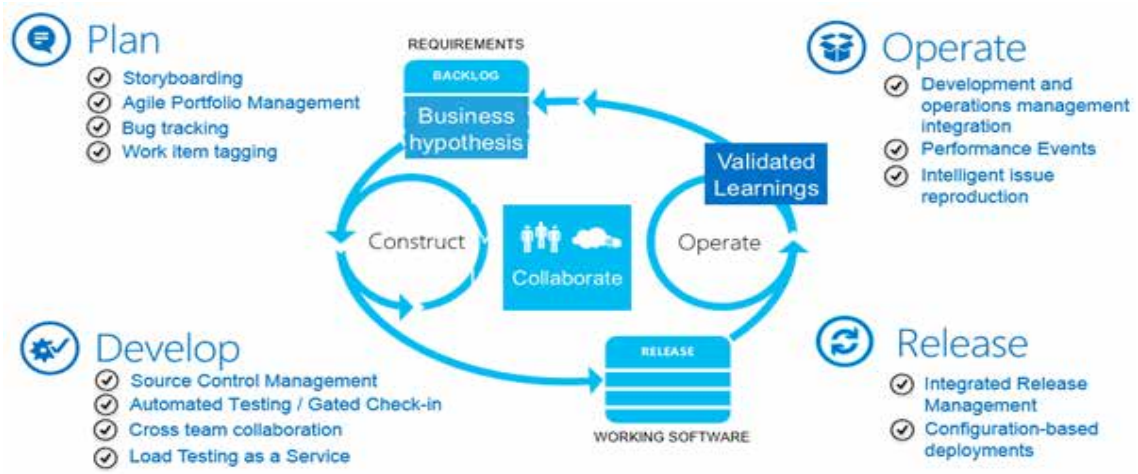
Chaque groupe optimise ce qu'il considère être son périmètre, ce qui crée des conflits. Le développeur est à l'origine de changements, que le responsable des opérations souhaite minimiser pour réduire le risque de dysfonctionnement.

LE PROCESSUS DE « CONTINUOUS DELIVERY »

Ce processus se fonde sur l'approche historique de l'ALM (Application LifeCycle Management) en regroupant de multiples pratiques destinées à apporter de la valeur en continu. La rationalisation induite par sa mise en œuvre permet de déployer le plus fréquemment possible de nouvelles versions d'une application ou d'un service.

L'utilisation combinée des solutions Microsoft Azure et de Visual Studio Online, grandement facilitée par leur intégration native, est un parfait exemple d'outillage permettant de gérer ce type de processus.

Le schéma suivant représente les différentes phases et outils d'un processus de « Continuous Delivery ».



Ce processus englobe :

• L'intégration continue qui vise à réduire les efforts d'intégration en assurant automatiquement la génération, l'assemblage et les tests des composants de l'application

• Le déploiement continu qui consiste à automatiser le déploiement du logiciel construit sur la plateforme cible (test, intégration, pré-production voire production) à chaque nouvelle génération de livrable

• Le suivi en continu du comportement de l'application sur le plan technique mais également du point de vue de son usage afin d'améliorer l'efficacité de la solution.

La démarche « DevOps »

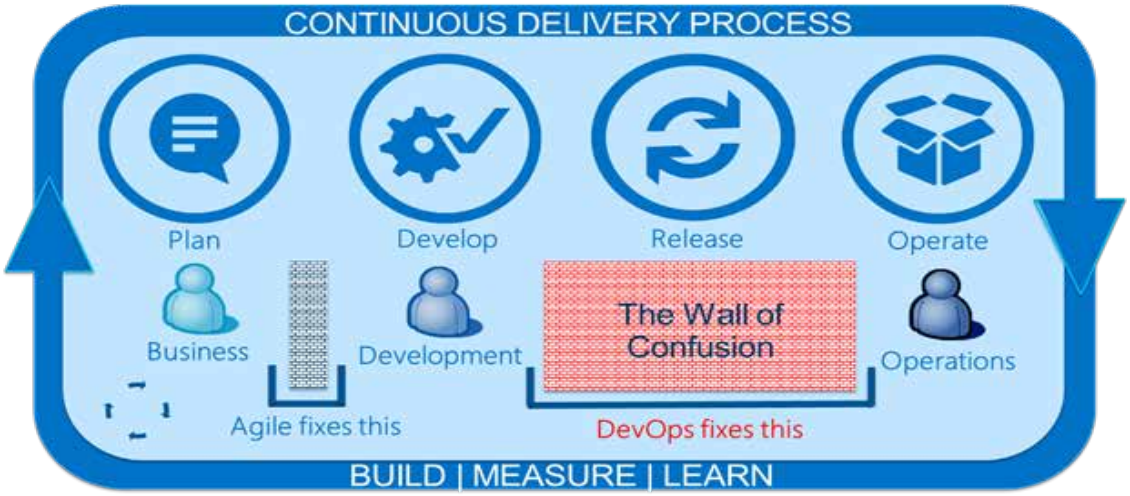
EN QUOI CONSISTE LA DÉMARCHE « DEVOPS »

La démarche « DevOps » est une philosophie, une façon de penser. Elle vise à établir une collaboration plus étroite et plus efficace entre les équipes de développement et d'infrastructure en permettant une accélération des déploiements tout en réduisant les frictions opérationnelles.

Elle représente un élément clé de la mise en place d'un processus de « Continuous Delivery » et se traduit par la mise en application de différents concepts d'ordre culturels et technologiques.

« DEVOPS » ET « CONTINUOUS DELIVERY »

La démarche « DevOps » se situe au cœur du processus de « Continuous Delivery ».



LA CULTURE « DEVOPS »

Une des caractéristiques de la démarche « DevOps » est de vouloir changer la dynamique dans laquelle les équipes de développement et opérations interagissent les unes avec les autres. Cette évolution est encouragée en prônant des valeurs fondamentales comme le respect mutuel, la confiance réciproque, la systématisation du partage de l'information et la responsabilité.

L'objectif est de permettre aux membres de chaque organisation d'avoir une meilleure compréhension du point de vue d'autres acteurs avec lesquels ils sont en relation et de modifier leur comportement en conséquence. L'ignorance délibérée des besoins de l'autre n'est plus autorisée car les résultats, comme la livraison d'un service livré en production sont, en fin de compte, partagés. Cela conduit à l'élaboration de mesures fondées sur la performance commune.

La culture « DevOps » favorise le développement des compétences de l'ensemble des acteurs du système dans une recherche perpétuelle d'amélioration (« Kaizen »). La pratique systématique de cette approche est la condition sine qua non de cette progression. L'organisation devient ainsi plus prompte à s'adapter et recherche le changement plutôt que de le fuir.

En outre, elle propose une vision positive de l'échec. En effet, les organisations doivent apprendre des succès, mais également de leurs échecs. Elles doivent donc accepter de prendre des risques. L'adoption ce type de démarche offre la possibilité d'anticiper et définir de nouveaux besoins opérationnels résultant de cette prise de risque.

Un des exemples de mise en place de ce type d'évolution culturelle est la démarche d'introduction volontaire de défauts dans le système (« Fault injection testing ») afin de constater au plus tôt la capacité du système à se remettre en service après un dysfonctionnement. Cela permet également de définir et partager les plans d'escalade et processus internes associés. Avec l'accélération de l'évolution vers le Cloud, ce type de test présente un intérêt renforcé car l'infrastructure matérielle est de moins en moins sous contrôle des directions informatique et la probabilité d'inter-dépendances de services liés à des SLA différents est de plus en plus forte.

Pour garantir la résilience du système, cette approche doit être couplée avec une conception d'une architecture dite « FailSafe ». Toute solution d'échelle significative étant sujette au dysfonctionnement, il est préférable d'intégrer des techniques de tolérance aux pannes dès la conception des services pour réduire l'impact lorsque des défaillances se produisent.

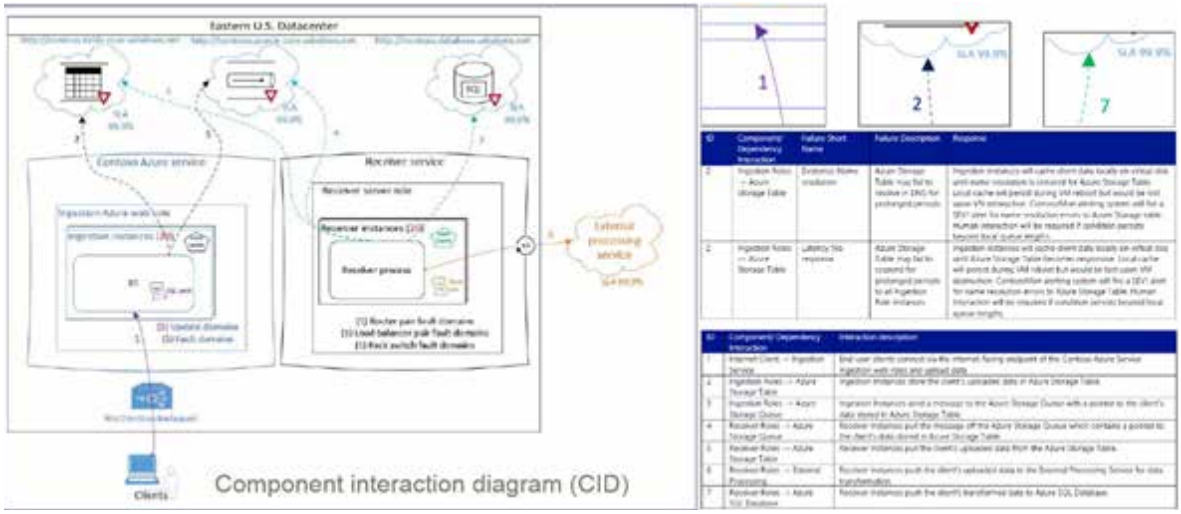
De multiples activités de conception contribuent à optimiser la solution et à prioriser les mesures correctives, notamment celles liées à la gestion de son évolutivité, de sa disponibilité et de sa résilience :

• Le découpage de l'application en composants fonctionnels dont la criticité métier diffère, ce qui permet de faire des choix technologique en fonction de critères de priorité et coûts.

• La définition d'un modèle de cycle de vie d'application décrivant le comportement attendu en production (charge variable aux différentes heures de la journée, ...).

• La définition d'un plan et d'un modèle de disponibilité permettant d'identifier le niveau de disponibilité attendue pour une composante de l'application et de hiérarchiser les étapes d'une remédiation sur des modes de défaillance pré-identifiés. Ces éléments sont susceptibles d'influencer bon nombre des décisions prises au cours du développement de la solution.

• Identifier les points de défaillance et des modes de défaillance. Pour créer une architecture résiliente, il est important de déterminer et documenter ce qui peut provoquer une panne. Identifier les points sujets à panne au sein d'une solution et les modes de défaillance qui en résultent permet de prendre des décisions sur les stratégies de résilience et de disponibilité dès la phase de développement. Cette démarche, baptisée « Resilience Modeling and Analysis (RMA) » se fonde sur une analyse et une modélisation de la résilience des services composant l'application.



La démarche « DevOps »

LA CULTURE « DEVOPS » - suite

L'approche RMA est issue du standard de l'industrie « Failure Mode and Effects Analysis (FMEA). »
Elle propose de créer et gérer un modèle qui détaille les possibilités de pannes de l'application et l'ensemble des actions qui peuvent en atténuer l'impact. RMA peut donc être utilisé pour modéliser les différents points de l'architecture présentant un risque de défaillance, de documenter l'impact correspondant et de définir des contre-mesures afin de s'assurer que la panne ne puisse pas se produire ou pour limiter son incidence.

« DEVOPS » : PERSPECTIVE TECHNOLOGIQUE

Sur le plan technologique la démarche DevOps se fonde sur de multiples principes.

Les machines peuvent être automatiquement provisionnées grâce à la virtualisation, au Cloud et par la prise en charge par de multiples outils d'orchestration. L'environnement de déploiement d'une application (sa configuration et sa plateforme cible) est un élément fondamental dont va dépendre la réussite du projet. A ce titre, il se doit d'être géré comme n'importe quel autre livrable du projet (archivage et contrôle des mises à jour, gestion des versions).

Afin d'éviter que le code d'une application soit étroitement couplé à un profil d'infrastructure spécifique, des « blueprints » sont associés aux applications précisant les « règles », descriptives ou prescriptives, ainsi que les paramètres d'entrée qui permettent aux services Cloud de connaître les actions à effectuer au nom de l'application.

Cette démarche est facilitée par l'évolution technologique. L'infrastructure est maintenant dynamiquement modifiable via des interfaces de programmation. Les descriptions de configuration peuvent donc être fournies aux services Cloud via des API bien définies. Elles sont directement corrélées aux profils de services d'infrastructure que propose le Cloud afin de configurer un ensemble d'instances et les différentes ressources connexes. En conséquence, les équipes de gestion opérationnelle sont amenés à écrire du code (scripts, workflows) pour provisionner et configurer les environnements cibles. A ce titre, il leur est recommandé de tirer parti des pratiques issues du monde du développement (tels que l'utilisation de référentiels partagés avec contrôle de version, l'automatisation des tests,...)

L'instrumentation d'une solution est essentielle pour pouvoir observer son comportement sur le plan fonctionnel (pour obtenir des informations permettant de répondre aux besoins fluctuants des utilisateurs) et sur le plan technique (afin de pouvoir corriger les dysfonctionnements au plus tôt, identifier les limites en termes de performance).

Les outils « DevOps »

Parallèlement à la mise en place de ces bonnes pratiques, ont émergées de multiples outils axés sur une démarche DevOps. Ces outils se fondent sur la manipulation d'objets communs, partagés entre les équipes de développement et de production.

CATÉGORISATION DES OBJETS COMMUNS

Les objets communs et leurs mécanismes de partage associés sont les suivants :

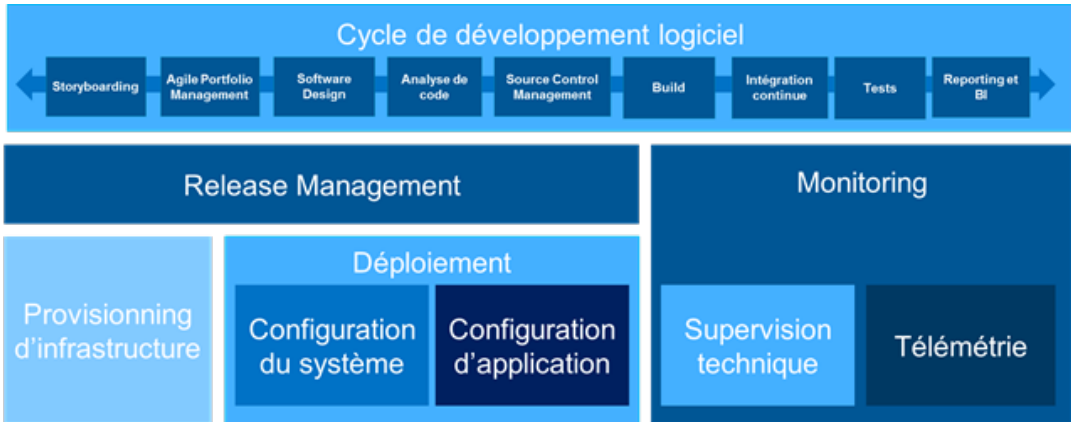
OBJETS COMMUNS	MÉCANISMES DE PARTAGE
Binaires	Référentiel de binaires et packages associés
Déploiement	Mécanismes d'automatisation de déploiement et de de configuration d'application multi-tiers
Environnement	Création d'environnements évolutifs (scaling up/down/out), de préférence à partir d'un modèle unique pour de multiples plateformes
Objets de test	Utilisation des mêmes objets de test entre les différentes équipes pour valider la bonne adéquation avec le fonctionnel attendu et le bon comportement en terme de performances, fiabilité et haute disponibilité
Données	Adaptation des données de production pour les rendre consommables comme données de test

Idéalement, les objets sont tous issus de la même source. Ainsi, il ne peut y avoir de de divergence de vue vis-à-vis de ces objets entre les développeurs et les équipes de gestion opérationnelle.

CATÉGORISATION DES OUTILS DEVOPS

La mise en œuvre d'une démarche « DevOps » ne repose pas uniquement sur un seul outil. Le principe est donc de s'appuyer sur chaque type d'outil pour ce qu'il fait de mieux. Et c'est de leur niveau d'intégration que dépendra la capacité à répondre aux problématiques précédemment énoncées.

Ces outils apportent une réponse à de multiples problématiques que l'on peut regrouper par thème.



Les outils « DevOps »

GESTION DES LIVRABLES (« RELEASE MANAGEMENT »)

La gestion des livrables suppose la mise en place d'outils permettant d'échanger les données appropriées dans un format cohérent et exploitable. Elle doit également offrir la capacité de recréer le même ensemble d'objets en cas d'erreurs de production. Idéalement, il ne devrait y avoir aucun transfert explicite entre les différentes équipes.

Les éléments correspondant aux livrables devraient donc être globalement partagés dans un référentiel unique commun contenant les différents livrables (binaires, dépendances, fichiers de configuration, fichiers de déploiement, fichiers de test et résultats associés).

Ce référentiel devrait offrir un contrôle de version, ainsi qu'un tableau de bord permettant de suivre l'état d'avancement du déploiement des différentes versions des applicatifs déployés) comme le propose Microsoft Release Management.

Il faut également un outil permettant de construire et gérer un pipeline de gestion des livrables dans lequel développeurs et responsables système vont pouvoir définir les actions et critères de déploiement, de vérification et de validation avec contrôle manuel ou automatique. Ce pipeline devrait pouvoir utiliser des objets partagés et communs. Idéalement il devrait être totalement automatisé.

PROVISIONING D'INFRASTRUCTURE

Le rôle de ce type d'outil est de permettre la mise en service un environnement d'exécution pour une instance de système d'exploitation spécifique choisie par l'utilisateur à partir d'une liste d'images pré-définies.

Le fonctionnement varie alors considérablement suivant que ce « provisioning » soit réalisé sur des serveurs virtualisés (avec ou sans Cloud) ou des instances de système d'exploitation natif. Il est en effet possible de produire automatiquement un environnement entier à partir d'un environnement « bare-metal » avec des outils comme System Center Virtual Machine Manager (SCVMM) ou de cibler un hyperviseur disposant d'un référentiel d'images virtuelles. Dans un cas comme dans l'autre, l'outil de « provisioning » doit permettre de réserver et configurer les ressources CPU, le stockage, tout en prenant en compte d'autres services, telles que l'authentification.

Le « provisioning » est réalisé sur la base de « modèles d'infrastructure » qui vont pouvoir être partagés et répliqués sur les différentes plateformes (développement, intégration, recette, pré-production, production) dans des configurations qui pourront varier en termes de dimensionnement. L'intérêt d'automatiser le provisioning d'infrastructure est non seulement d'accélérer la mise à disposition des environnements d'exécution mais aussi de pouvoir répondre à d'éventuels changements si nécessaire.

Le choix de tel ou tel autre outil pourra se faire sur de nombreux critères :

- Système d'exploitation sur lequel est lancé le « provisioning » (exemple : les commandes PowerShell s'exécutent dans Windows, même si elles permettent de provisionner des machines virtuelles en se basant sur des distributions Linux).
- Type d'usage (Vagrant cible plutôt les tests ou le développement)
- Gestion de configuration : la plupart des outils ne se limitent pas à un simple « provisioning »

Parmi les logiciels de provisioning disponibles aujourd'hui, citons : Ansible, Chef, Puppet, Salt, Vagrant, Sky-tap, Cfengine, Cmdlet PowerShell Azure, Confi, Microsoft Deployment Toolkit (MDT), System Center Virtual Machine Manager (SCVMM).

Une fois qu'une instance de système d'exploitation minimale est en cours d'exécution et dispose d'une connectivité réseau, il est alors temps de faire appel à l'outil de configuration du système.

GESTION DE CONFIGURATION

La gestion de configuration suppose la création d'un modèle de déploiement unique partagé entre les développeurs et les équipes de gestion opérationnelle. Il est nécessaire de s'assurer que les environnements de développement, de test et de production sont tous configurés de la même manière. Cela garantit qu'il n'y a pas de surprises liées aux subtiles différences entre les environnements. Il faut donc offrir la possibilité d'encoder ces configurations dans un langage permettant de les créer et les faire évoluer.

Parmi les autres caractéristiques des outils de gestion de configuration citons :

• La validation de la bonne exécution d'une configuration

• La capacité à pouvoir appliquer plusieurs fois la même configuration sans provoquer d'erreur ni en modifier le résultat (idempotence)

• La possibilité d'effectuer un rollback sur une configuration précédente

Il est souvent plus simple et plus rapide de relancer les opérations de « provisioning » et configuration que de tenter de réparer un environnement présentant un défaut.

Les objectifs de maintien de configurations système et d'application sont souvent en désaccord. Les configurations système doivent généralement être maintenues aussi cohérentes que possible, tandis que les configurations d'application peuvent changer constamment selon l'évolution des besoins commerciaux et des cycles de déploiement. Ces besoins opposés sont une des raisons pour lesquelles Le déploiement reconnaît une séparation conceptuelle entre la gestion de la configuration des systèmes et des applications.

Cette distinction se retrouve dans les outils de gestion de configuration :

• Configuration système : PowerShell DSC, Puppet, Chef, Salt, Ansible, CFEngine, Juju,...

• Configuration d'application : Capistrano, Fabric, Func, ControlTier

Bien que les outils open source présents dans cette liste aient été à l'origine conçus pour automatiser la configuration de systèmes d'exploitation Linux, leur arrivée dans le monde de l'entreprise a permis leur extension vers les environnements Windows.

Les outils « DevOps »

MONITORING

Avec la réduction de la durée d'un cycle de production logicielle induite par la mise en œuvre de processus de « Continous Delivery », il devient encore plus important de recueillir des informations techniques en phase de tests, mais aussi en production.

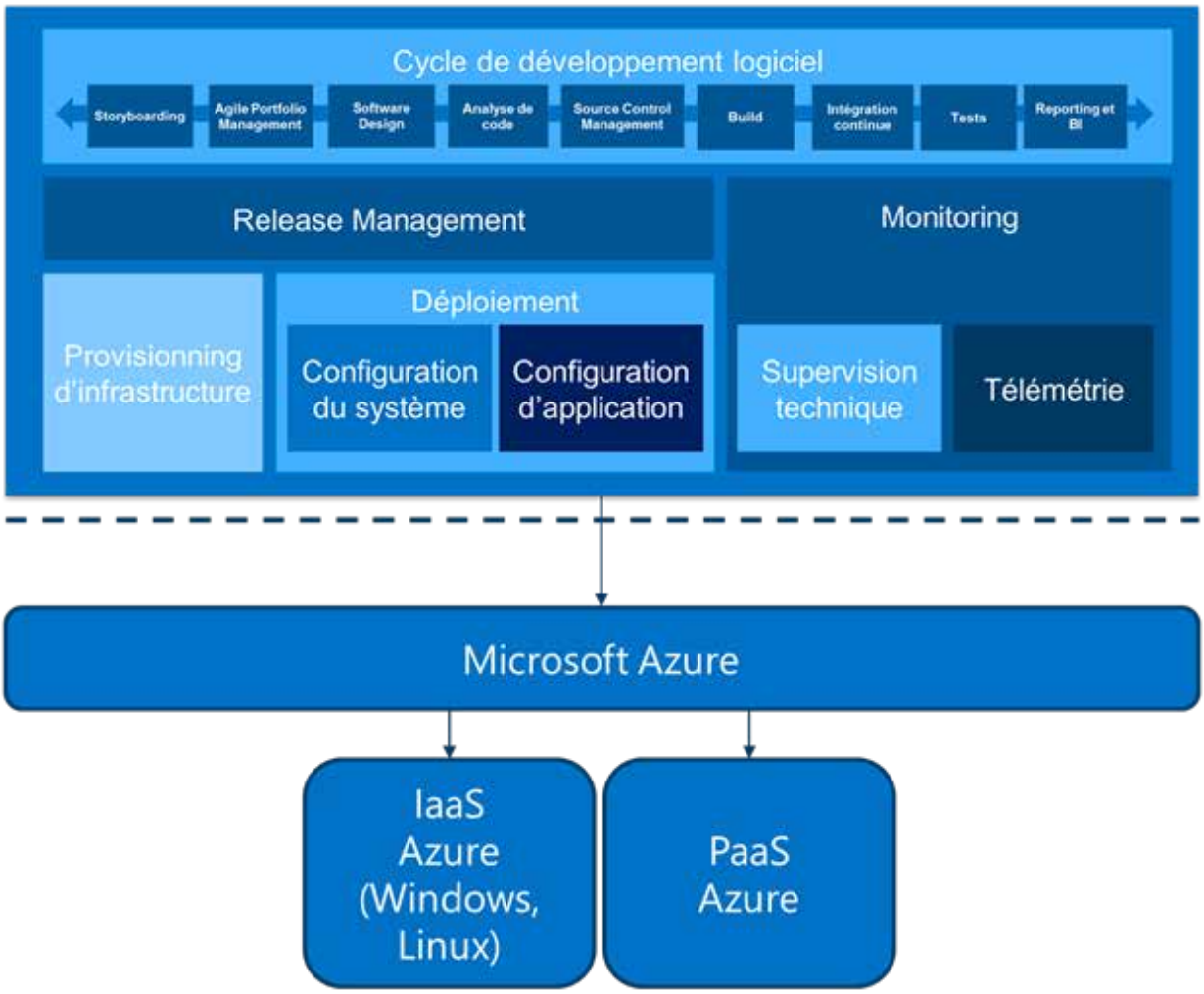
Parmi les produits disponibles sur ce créneau citons des solutions Open Source comme Zabbix et Nagios ou propriétaires comme System Center Operations Manager (SCOM) et Visual Studio Online Application Insights.

« DevOps » pour le Cloud

AZURE : « DEVOPS » POUR LE CLOUD

Microsoft propose diverses technologies « DevOps » (« Service Templates » de System Center Virtual Machine Manager, « VM Role Definition » du Windows Azure Pack, PowerShell DSC, Azure Resource Manager) permettant de répondre aux exigences précédemment décrites (configuration déclarative, idempotence) pour automatiquement déployer et configurer des applications dans les environnements Cloud privés et publics Microsoft.

Dans ce document, nous nous limiterons volontairement au contexte Azure.



La mise en place d'un environnement DevOps dans Azure repose sur la mise en œuvre d'un modèle de service uniforme permettant de cibler l'ensemble des ressources de la plateforme Azure, qu'il s'agisse de ressources PaaS ou IaaS.

« DevOps » pour le Cloud

PROVISIONING DANS AZURE AVEC DE MULTIPLES LANGAGES

Le provisioning de ressources Windows et Linux peut être directement assuré par la mise en œuvre des différents SDK Azure, des Cmdlets Azure PowerShell, de l'API REST de management, ou du langage de scripting Azure Cross-Platform Command-Line Interface.

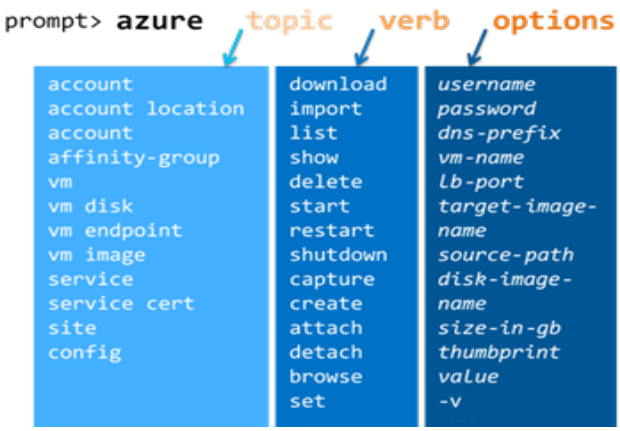
POWERSHELL ET AZURE

PowerShell est le langage de scripting et d'automatisation proposé par défaut sur la plateforme Windows. Un module Azure est disponible pour enrichir ce langage de multiples Cmdlets permettant d'automatiser le provisioning des différentes ressources Azures.

AZURE CROSS-PLATFORM COMMAND-LINE INTERFACE

Azure Cross-Platform Command-Line Interface (xplat-cli) est le langage de scripting en ligne de commande pour les environnements Windows, Mac et Linux. Il est développé en javascript et nécessite l'installation de node.js.

La figure suivante présente différentes commandes xplat-cli.



CONFIGURATION DANS AZURE AVEC WINDOWS MANAGEMENT FRAMEWORK

La configuration système des environnements Windows peut être réalisée grâce au Windows Management Framework (version 5 aujourd'hui disponible en preview) qui regroupe différentes technologies parmi lesquelles les DSC et OneGet.

CONFIGURATION DANS AZURE AVEC WINDOWS MANAGEMENT

La configuration système des environnements Windows peut être réalisée grâce au Windows Management Framework (version 5 aujourd'hui disponible en preview) qui regroupe différentes technologies parmi lesquelles les DSC et OneGet.

WINDOWS POWERSHELL DESIRED STATE CONFIGURATION

La version 4.0 de PowerShell proposait déjà une évolution majeure avec « Desired State Configuration » (DSC). La version 5.0 du Windows Management Framework vient enrichir les DSC. Le principe de DSC est de définir l'état d'une instance de serveur de Windows dans le code, en utilisant des blocs de configuration pour spécifier les composantes Windows à installer et les compléments associés (entrées de clés de Registre, création de fichiers, routines d'installation...).

Configuration de l'environnement (Développement -> Test -> Production)	$SystemDrive = "C:" $DemoFolder = "$SystemDrive\Demo" $global:WebServerCount = 3

Configuration déclarative	WindowsFeature IIS {
	Name = "Web-Server"
	Ensure = "Present"
	}

Automatisation idempotente	foreach -parallel ($featureName in $Name)
	{
	$feature = Get-WindowsFeature -Name $featureName
	if(($Ensure -eq "Present") -and (!$feature.Installed))
	{
	Install-WindowsFeature -Name $featureName
	}
	....
	}

Un script de DSC est idempotent. Grâce à DSC, il est maintenant possible de gérer dans un référentiel de code source les multiples versions d'une configuration des informations permettant de déployer, configurer et gérer des services logiciels et l'environnement dans lequel ils sont exécutés. Pour ce faire, DSC fournit un ensemble de ressources pour spécifier de façon déclarative la configuration de l'environnement logiciel.

WINDOWS POWERSHELL ONEGET

Windows PowerShell OneGet est une technologie dont l'objectif est de simplifier considérablement la découverte et l'installation de logiciel sur les machines Windows.

Avec OneGet, il est possible de :

• Gérer une liste de dépôts de logiciels dans lequel les packages logiciels peuvent être recherchés, acquis et installés

• Installer et désinstaller silencieusement des packages d'un ou plusieurs référentiels avec une seule commande PowerShell

« DevOps » pour le Cloud

POWERSHELL ET AZURE

OneGet fonctionne avec le référentiel communautaire « Chocolatey » qui recense plus de 1 700 logiciels uniques. La prise en charge de dépôts supplémentaires sera proposée dans des versions ultérieures.

Pour l'utiliser, il suffit d'importer le module OneGet dans PowerShell.

```
Import-Module -Name OneGet
```

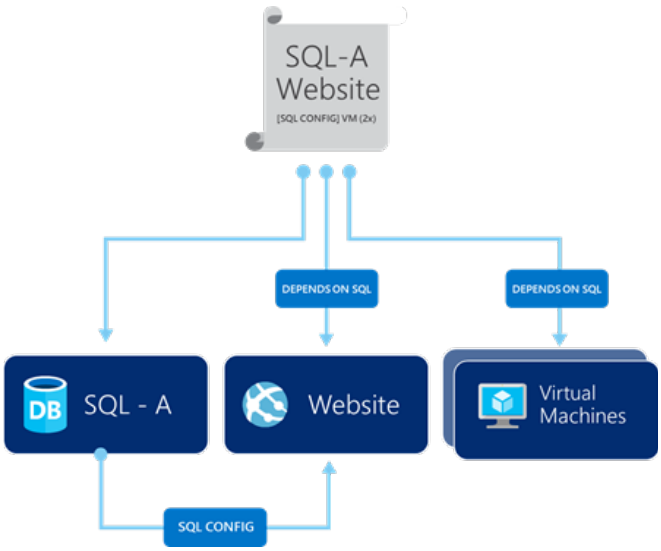
On dispose alors des commandes OneGet suivantes :

```
Get-Command -Module OneGet

CommandType     Name                     ModuleName
-----------     ----                     ----------
Cmdlet          Add-PackageSource        OneGet
Cmdlet          Find-Package             OneGet
Cmdlet          Get-Package              OneGet
Cmdlet          Get-PackageSource        OneGet
Cmdlet          Install-Package          OneGet
Cmdlet          Remove-PackageSource     OneGet
Cmdlet          Uninstall-Package        OneGet
```

PROVISIONING ET CONFIGURATION AVEC « AZURE RESOURCE MANAGER »

A l'origine, la gestion d'une ressource Azure (base de données, machine virtuelle, compte de stockage, site web) était « unitaire ». Cette situation rendait complexe la gestion d'une application composée de plusieurs ressources.



En effet, il était difficile de :

- Déployer ou de mettre à jour un groupe de ressources de façon récurrente
- Gérer les permissions sur un groupe de ressources
- Disposer d'une représentation offrant des informations de supervision technique et de facturation pour un ensemble de ressources

Il est maintenant possible de créer et gérer des groupes de ressources grâce au « Azure Resource Manager » qui permet de déclarer une entité de gestion dans laquelle sont intégrés des regroupements de multiples ressources de même type ou non. L'appartenance à un groupe de ressources est exclusive. Les ressources peuvent être multi-régions. Dans ce contexte, un groupe de ressources est un conteneur logique destiné à faciliter la gestion du cycle de vie d'un regroupement de multiples ressources, comme dans le cas d'une application construite autour d'un site Web, d'une base SQL Database et d'une machine virtuelle.

La solution proposée pour le déploiement et la configuration d'un groupe de ressources est déclarative afin de faciliter la configuration des ressources, de leurs dépendances, de leurs interconnexions, de la gestion d'identité entre ces ressources, et de leur facturation. Elle se fonde sur l'utilisation d'un modèle baptisé « Azure Template » qui va pouvoir garantir l'idempotence, simplifier l'orchestration, la gestion du cycle déploiement, le retour sur une version antérieure. Ces templates sont implémentés en json et peuvent donc être gérés dans un contrôleur de code source.

Ils peuvent être directement édités depuis une extension d'édition Json disponible dans l'Update 2 de Visual Studio 2013. Le fichier correspondant est composé de multiples sections, l'une décrivant les paramètres (« parameters ») du fichier template, l'autre les ressources Azure associées au modèle.

Lorsqu'il s'agit de ressources Azure de type « Machine virtuelle », la solution proposée avec « Azure Resource Manager » est extensible grâce à l'utilisation du « VM agent » qui permet d'installer et gérer des extensions pour interagir avec la VM, à l'emploi de Windows Management Framework V5 (DSC, OneGet) au sein de la VM, ou à la mise en œuvre de solutions comme Puppet, Chef, Ansible, Salt, par exemple, dans le cas de Machines virtuelles Linux.
Fig. 1

Il devient alors possible de déployer d'un simple click une application composée de multiples ressources depuis le nouveau portail Azure (disponible en preview à l'adresse : https://portal.azure.com)
Fig. 2

Une expérience encore plus étendue de gestion et de déploiement d'applications multi-ressources peut être obtenue depuis un script PowerShell.
Fig. 3

Fig. 1

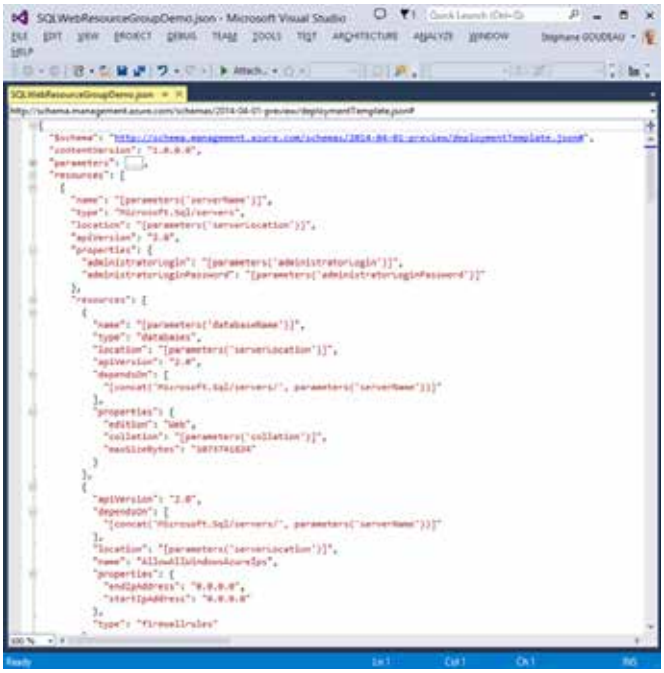


Fig. 2

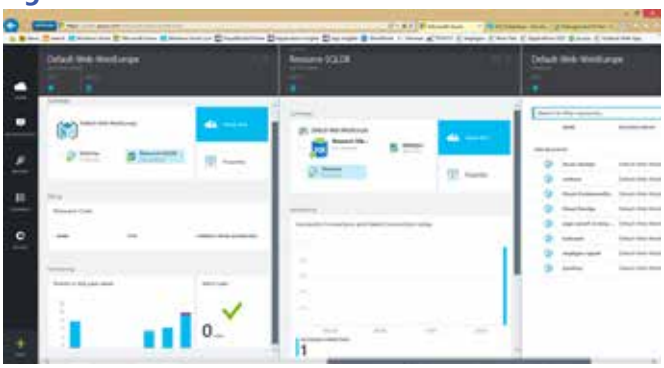
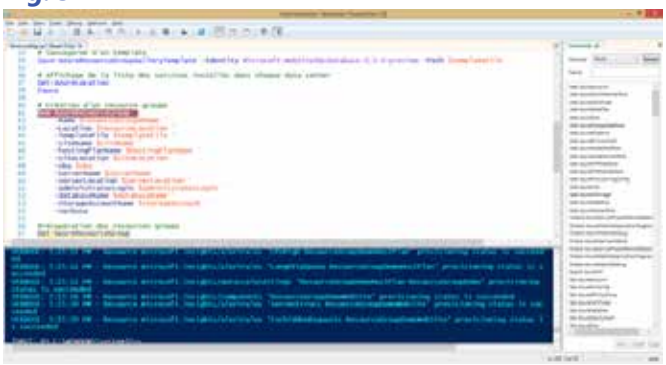


Fig. 3



« DevOps » pour le Cloud

```
Param(
    #Paramètres du Azure Ressource Group
    $administratorLogin = "stephgou",
    $databaseName = "ResourceGroupDemoDatabase",
    $hostingPlanName = "ResourceGroupDemoHostPlan",
    $resourceGroupeName = "ResourceGroupDemo",
    $resourceLocation = "West Europe",
    $siteLocation = "West Europe",
    $siteName = "ResourceGroupDemoWebSite",
    $serverName = "stephgou",
    $serverLocation = "West Europe",
    $sku = "Free",
    $storageAccount = "stephgouvmstorage",
    $templateFile = "C:\DEMOS\66 - DEVOPS\DevOps
Demo\AzureResourceMgr\SQLWebResourceGroupDemo.json"
    )

cls

Add-AzureAccount

Switch-AzureMode -Name AzureResourceManager

Get-Command -Module AzureResourceManager | Get-Help | Format-Table Name, Synopsis

Get-AzureAccount

# Liste des templates json dans la galerie
Get-AzureResourceGroupGalleryTemplate

# Affichage des infos sur un template en particulier
Get-AzureResourceGroupGalleryTemplate -Identity Microsoft.WebSiteSQLDatabase.0.2.0-
preview

# Sauvegarde d'un template
Save-AzureResourceGroupGalleryTemplate -Identity Microsoft.WebSiteSQLDatabase.0.2.0-
preview -Path $templateFile

# Affichage de la liste des services installés dans chaque data center
Get-AzureLocation
Pause

# Création d'un resource groupe
New-AzureResourceGroup
       -Name $resourceGroupeName `
       -Location $resourceLocation `
       -TemplateFile $templateFile `
       -siteName $siteName `
       -hostingPlanName $hostingPlanName `
       -siteLocation $siteLocation `
       -sku $sku `
       -serverName $serverName `
       -serverLocation $serverLocation `
       -administratorLogin $administratorLogin `
       -databaseName $databaseName `
       -StorageAccountName $storageAccount `
    -
    -Verbose

#récupération des resources groups
Get-AzureResourceGroup

#récupération des ressources dans un groupe
Get-AzureResource -ResourceGroupName $resourceGroupeName

#ajout d'une ressource à un groupe
New-AzureResource
```

STRATÉGIE DEVOPS MICROSOFT POUR LES SYSTÈMES NON WINDOWS

Pour les systèmes non Windows, Microsoft fonde sa stratégie DevOps sur l'utilisation de technologies Open Source reconnues par le marché, comme Puppet (Puppet Labs) et Chef (Opscode) dont les capacités fonctionnelles sont relativement similaires.

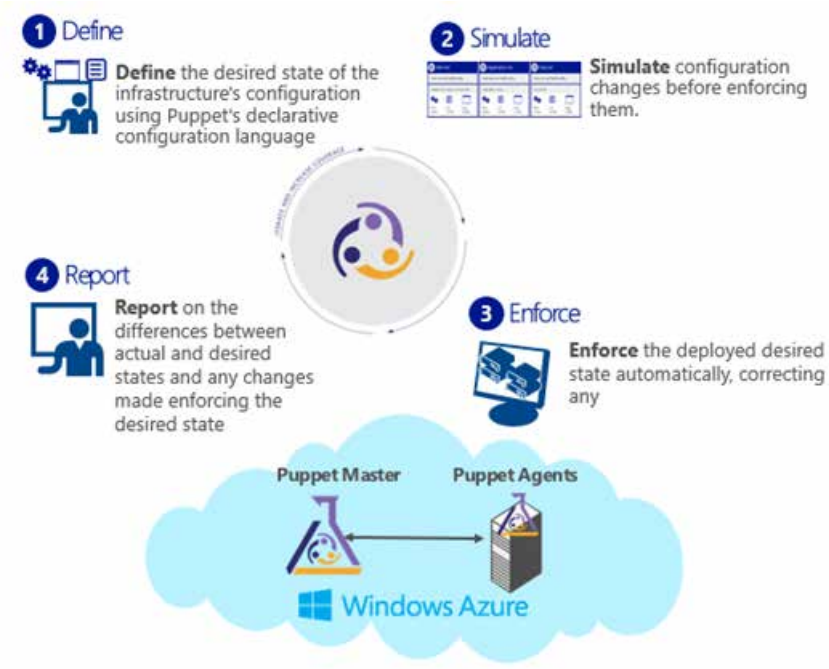
PUPPET

Puppet est un outil Open Source (édité par la société Puppet Labs) pour le provisioning et la gestion de configuration. Il permet de définir l'état souhaité d'un ensemble de composantes d'une infrastructure avec un langage déclaratif implémenté comme un DSL décrivant des relations de dépendances entre ressources. Cette définition d'état est baptisée « module ». Puppet configure tous les composantes du « module » et les maintient en l'état via l'application d'un modèle contrôlé par un agent « node ».

Puppet opère généralement selon un modèle client/serveur dans lequel le client (l'agent installé sur l'infrastructure déployée) contacte le serveur « Puppetmaster » périodiquement pour récupérer les dernières informations de configuration. Le client valide alors si le système est conforme et joue le scénario de configuration (le « manifeste » décrivant les ressources en modifiant l'environnement, lorsque cela est nécessaire. Lorsqu'elle ces opérations sont terminées, le client signale les modifications qu'il a appliquées au serveur.

Le principe de Puppet est le suivant. Puppet compile l'ensemble des fichiers de configuration, les regroupe dans un catalogue téléchargé en local par le client Puppet afin qu'il puisse instancier les éléments correspondants sur la cible. Concrètement, cela se traduit par le déploiement d'un jeu d'instructions exécutable permettant de modifier le système.

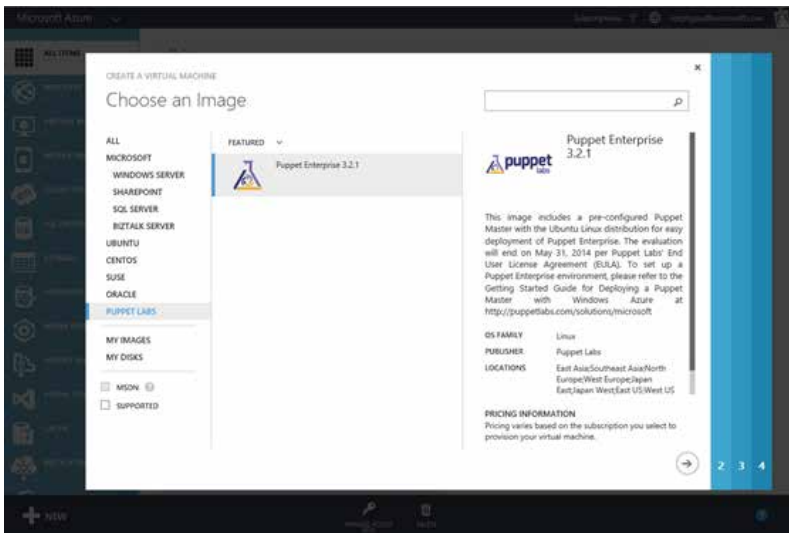
Puppet permet ainsi de modéliser la configuration d'un ensemble de systèmes, physiques ou virtuels, sur des cloud public ou privés, sur des systèmes d'exploitation Linux ou Windows et d'exécuter ce modèle.



« DevOps » pour le Cloud

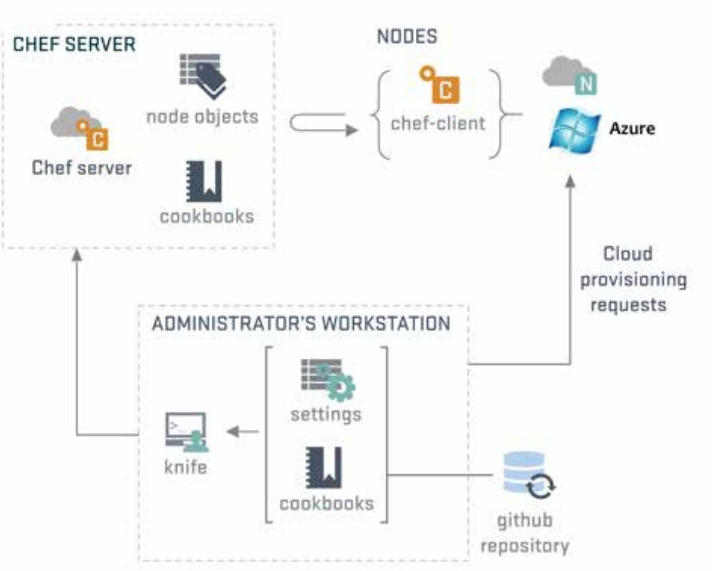
PUPPET - suite

Il est possible de déployer un serveur Puppet directement à partir de la galerie d'images du portail Azure.



CHEF

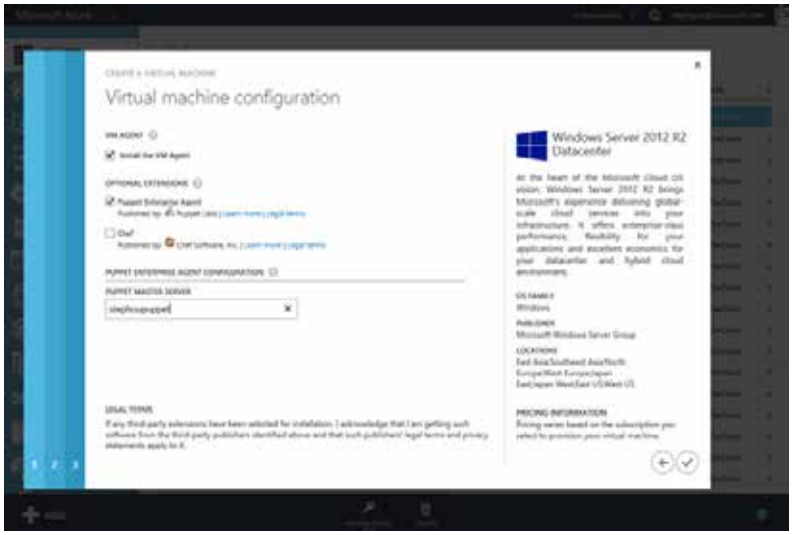
Comme Puppet, Chef est un outil Open Source (édité par la société Opscode) pour le provisioning et la gestion de configuration. Chef permet de modéliser la configuration d'un ensemble de systèmes, physiques ou virtuels, Linux ou Windows et d'exécuter ce modèle. Avec Chef, il est donc possible de mettre en place un processus répétable associé à un état de configuration défini dans une « policy » implémentée dans le langage Ruby. La configuration complète d'un environnement, baptisée « recipe » est donc le résultat de la combinaison d'une série de « policies ». Le « Cookbook » est le package contenant « recipes », modèles, fichiers et ressources personnalisées.



Chef propose plusieurs « cookbooks » ciblant des solutions Microsoft comme Windows, SQL Server, ou Internet Information Server. Chef supporte aussi Windows Azure, en permettant la construction automatisée et la configuration de machines virtuelles dans Azure.

Chef fournit également un utilitaire en ligne de commande baptisé « knife » qui fournit une interface entre le poste de travail Chef et le serveur. Il existe une implémentation de knife pour Azure. La version 1.2.2 permet non seulement de provisionner les machines mais aussi de configurer certains aspects spécifiques de la plateforme Azure comme la configuration des réseaux virtuels privés ou la définition de groupes d'affinité pour garantir une latence minimale entre les serveurs du même groupe, au sein d'un même datacenter.

Chef opère dans le mode client/serveur de la façon suivante. Le client Chef est utilisé pour créer la définition de configuration qui devra être déployée et maintenue sur les serveurs « Nodes » rassemblées au sein d'un même environnement (développement, test, pré-production, production). Le serveur Chef joue le rôle de « Hub » central, qui garantit que l'information pertinente pour l'ensemble des nœuds est disponible et son application dans le cadre des « policies » qui ont été définies.



AUTRES OUTILS DEVOPS OPEN SOURCE

LE ROLE DE MS OPEN TECH

Pour promouvoir davantage les investissements sur l'interopérabilité, les standards ouverts et l'Open Source, Microsoft a lancé une filiale baptisée Microsoft Open Technologies, Inc. (MS Open Tech) au début de 2012.

A ce titre, les équipes de MS Open Tech produisent du code Open Source et font la promotion du développement et de l'adoption de standards ouverts.

Une partie de ces activités concerne les outils de DevOps pour Microsoft Azure.

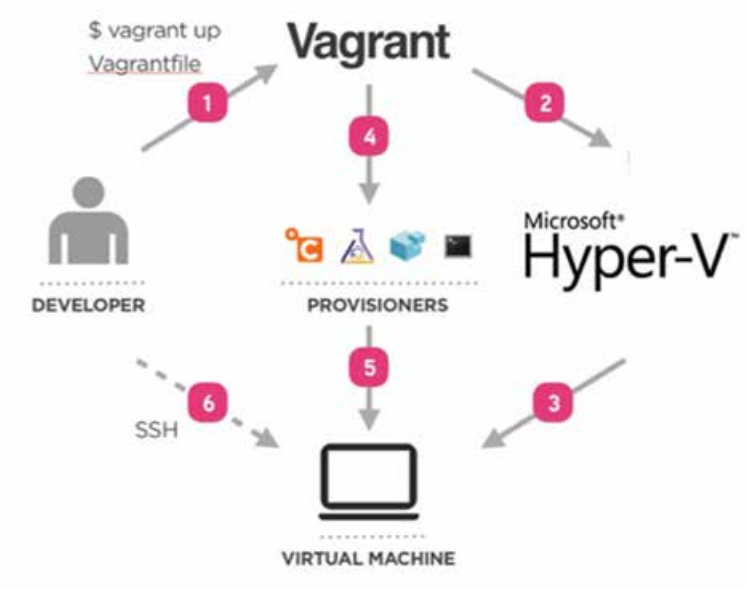
« DevOps » pour le Cloud

VAGRANT

Vagrant permet de configurer des environnements de développement ou de test reproductibles et portables. Vagrant isole les dépendances et leur configuration dans un environnement unique « jetable », cohérent, sans pour autant sacrifier les outils avec lesquels on a l'habitude de travailler (éditeurs, navigateurs, débogueurs, etc..).

Le fichier Vagrant permet de monter très rapidement un environnement de développement entièrement configuré. Il peut alors être partagé au sein d'une équipe de développeur afin que chacun se crée son propre environnement de développement à partir de la même configuration (que ce soit sur Linux, Mac OS X ou Windows). Ainsi le code s'exécute sur le même environnement, avec les mêmes contraintes et la même configuration.

Les ingénieurs systèmes peuvent également utiliser ce type d'environnement pour valider leur script d'automatisation de configuration.



MS Open Tech a récemment publié un provider Hyper-V (la version incluse dans Windows 8.1) pour Vagrant 1.5 et les versions suivantes. Les machines peuvent donc être provisionnées directement sur Hyper-V et pourront prochainement l'être sur Azure. La configuration automatique du système est alors assurée par l'utilisation de scripts shell Unix ou Powershell, ou d'outils comme Chef, ou Puppet.

SALT

Salt est un système de gestion de configuration, capable de maintenir des nœuds distants dans un état défini (par exemple, veiller à ce que des applications soient installées ou à ce que tel ou tel service soit en cours d'exécution). C'est aussi un système distribué d'exécution à distance qui peut exécuter des commandes et interroger des nœuds distants, soit individuellement ou selon divers critères de sélection.
Pour ce faire, Salt offre un bus de communication dynamique pour l'orchestration des étapes de constructions d'une architecture, l'exécution à distance, la gestion de configuration et autres tâches.

Salt, peut être utilisé pour gérer les ressources d'Azure. La gestion des ressources est réalisée grâce au module Salt Azure, qui requiert le SDK Python pour Azure. Avec ce module, les développeurs peuvent déployer, inspecter et supprimer les machines virtuelles et gérer l'automatisation de l'infrastructure Azure et la configuration des machines virtuelles.

ANSIBLE

Ansible se veut être une solution très simple de gestion automatisée des configurations et déploiements système.

Contrairement à Chef et Puppet, Ansible est un système complètement décentralisé. Là où Puppet et Chef utilisent un serveur central qui maintient les configurations des serveurs administrés, Ansible pilote directement les systèmes distants via le protocole SSH.

Avec Puppet ou Chef les clients (machines) vont régulièrement interroger le système central pour savoir si des tâches sont à réaliser. Un agent installé sur la machine cible est nécessaire pour maintenir la relation entre le système central et le serveur administré.

Ansible est un système sans agent : La plateforme sur laquelle est installée Ansible devient le centre de commande des déploiements. Toute machine Ansible peut devenir centre de commande.
Dans un mode de fonctionnement standard, Ansible demande au système distant d'exécuter les commandes qu'il lui transmet.

Le cœur d'Ansible est écrit en Python mais est extensible avec des modules « custom » écrits dans n'importe quels langages disponibles sur la plateforme de commandement (Python, Bash, Ruby, ...).

Ansible peut être invoqué de différentes manières:

• En ligne de commande : ansible

o ansible all –m ping : Ansible va lancer en parallèle un « ping » (test de connectivité en fait) sur toute les machines présentes dans le fichier d'inventaire

o ansible webservers –a reboot : Ansible va demander aux serveurs du groupe webservers d'exécuter la commande reboot

• En exécution de playbook (recettes) : ansible-playbook

o ansible-playbook lamp-stack.yml : Ansible va exécuter le playbook lamp-stack.yml

« DevOps » pour le Cloud

ANSIBLE - suite

Les actions peuvent être transmises en mode push (par défaut) et en mode pull.

Ansible s'appuie sur des inventaires de machines pour déployer/contrôler massivement (en parallèle ou non) des configurations de machines.

Un exemple de fichier d'inventaire machines :

```
[webservers]
webserver1.example.com
webserver2.example.com
[dbservers]
dbserver1.example.com
dbserver2.example.com
```

Les playbooks permettent de décrire les états attendus des systèmes distants. Ils sont écrits en YAML. Ils sont composés de tâches qui doivent être exécutées.

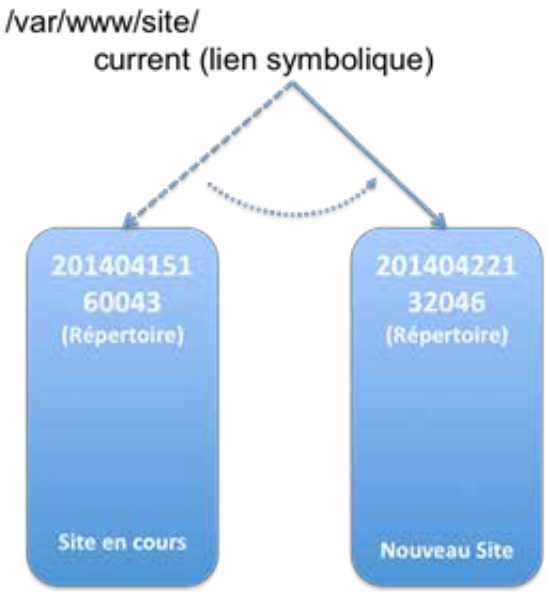
CAPISTRANO

Capistrano est un framework agnostique de déploiement.

Il est basé sur les mêmes principes qu'Ansible. C'est donc un système décentralisé qui permet le pilotage en parallèle de serveurs distants. Il utilise le protocole SSH pour dialoguer avec les systèmes distants. Il est capable d'utiliser les gestionnaires de code source tels que Accurev, BZR, CVS, Darcs, Perforce, SVN, Mercurial, Git.

Il est capable de gérer différents environnements : dev, pré-production, production... Il propose des fonctionnalités très intéressantes de retour arrière en cas d'échec. Son framework est extensible exclusivement en Ruby.

Capistrano utilise une stratégie simple pour gérer les déploiements. Il utilise un pivot représentant le site courant qui est en fait un lien symbolique sur une version du code source du site. Les retours arrière sont donc gérés très facilement par une simple modification du lien symbolique. Il est possible d'indiquer à Capistrano le nombre de versions à garder en ligne. La purge des anciennes versions est gérée de manière automatique.



Capistrano est aussi basé sur l'exécution de tâches. Le fichier utilisé par Capistrano est nommé « recipe » (recette) et doit s'appeler Capfile.
Ce fichier est composé de variables et de tâches permettant de réaliser le déploiement.
En voici un exemple correspondant à la version 2 de Capistrano.

```
set :application, "webapp"
set :stages, %w(production staging)
set :default_stage, "staging"

set :scm, :git
set :repository,  "git@github.com:example/webapp.git"
set :deploy_to, "/home/deploy_user/webapp"
set :deploy_via, :remote_cache

set :user, "deploy_user"
set :use_sudo, false

ssh_options[:forward_agent] = true
ssh_options[:port] = 3456

set :keep_releases, 10

namespace :deploy do

 desc "Surcharge de la tache restart"
 task :restart, :roles => :app, :except => { :no_release => true } do
  run "#{try_sudo} touch #{File.join(current_path,'tmp','restart.txt')}"
 end

 desc "Liens symboliques supplementaires"
 task :post_symlink, :roles => :app, :except => { :no_release => true } do
  ["config/database.yml", "config/config.yml"].each do |path|
   run "ln -fs #{shared_path}/#{path} #{release_path}/#{path}"
  end
 end

end

after  "deploy",              "deploy:post_symlink"
after  "deploy:restart",        "deploy:cleanup"
```

La nouvelle version de Capistrano (version 3) est disponible, elle permet d'écrire les recettes de déploiement de manière encore plus propre notamment grâce à la boite à outils SSHKit. Cette nouvelle version n'est pas encore complète et pour l'instant, ne propose d'interagir qu'avec le gestionnaire de code source Git.

DevOps et Azure : Retour d'expérience Alter Way

INTRODUCTION

Les chapitres précédents nous ont montrés l'étendue des possibles en termes d'outillage DevOps Open Source. Nous avons choisi pour illustrer ce livre blanc une automatisation complète du pilotage d'une infrastructure Azure basée sur l'utilisation avancée d'Ansible.

Nous commencerons par détailler les composants d'Ansible utilisés avant d'entrer dans le détail du pilotage d'Azure via cet outil.

PRÉSENTATION DÉTAILLÉE DES ÉLÉMENTS D'ANSIBLE MIS EN ŒUVRE

PLAYBOOKS

Les playbooks écrits en YAML permettent de décrire les états attendus des systèmes distants, par exemple :

```
---
- hosts: webservers
  sudo: yes
  remote_user: deployadm
  tasks:
  - name: Installation des paquets système nécessaires si besoin
    yum: name={{ item }} state=installed
    with_items:
       - apache2
       - php5
  - name: Copie des fichiers de configuration
    template: src="php.ini.j2" dest="/etc/php5/php.ini"
    notify:
    - restart apache
  handlers:
     - name: restart apache
       service: name="apache" state=started enabled=yes
```

Dans cet exemple Ansible pilote le groupe de machines webservers, en utilisant l'utilisateur deployadm, en exécutant toutes les commandes dans un environnement sudo, en installant les paquets système apache2 et php5 grâce au module yum d'Ansible, puis copie un fichier sur les systèmes distants mis en forme à partir d'un gabarit générique (template). Si l'état du système passe à « changé » alors le service apache sera redémarré.

Les actions à déclencher sont appelées grâce aux modules Ansible.

MODULES

Les modules permettent, de manière agnostique, d'interagir avec les systèmes distants (services, commandes système, fichiers, etc)
Les modules Ansible sont l'équivalent des ressources Puppet, en plus puissant.

Les modules Ansible sont catégorisés en différents groupes :

- cloud
- commands
- database
- files
- internal
- inventory
- messaging
- monitoring
- net_infrastructure
- network
- notification
- packaging
- source_control
- system
- utilities
- web_infrastructure

Ansible permet de construire des playbooks très élaborés en maniant des concepts avancés.
Ansible met à disposition les éléments structurels suivants :

- Les conditions
- Les boucles
- Les variables
- Les rôles
- Les inclusions

LES CONDITIONS

Ansible sait gérer des exécutions de tâches ou des inclusions conditionnées par des tests.
Exemple :

```
vars_files:
      - "vars/external_vars.yml"
      - [ "vars/{{ facter_operatingsystem }}.yml", "vars/defaults.yml" ]
```

Ici, on charge dynamiquement un fichier de variables ayant pour nom le nom du système d'exploitation qui a été collecté sur la machine distante. Si ce fichier n'est pas trouvé on prendra le fichier var/defaults.yml. C'est par cette méthode que l'on pourra par exemple définir une variable packager qui aura pour valeur yum si la variable facter_operatingsystem est CentOS ou RedHat et la valeur apt si facter_operatingsystem vaut Debian.

Notre playbook de déploiement sera donc complètement agnostique par rapport aux systèmes d'exploitation des machines cibles.

```
- include: hadoop_master.yml
  when: ha_enabled

- include: hadoop_master_no_ha.yml
  when: not ha_enabled
```

Dans cet exemple on inclura le fichier haddoop_master.yml si la variable ha_enabled est True, dans le cas contraire, on inclura le fichier haddoop_master_no_ha.yml.

```
- name: Copy the se linux fix file
  copy: src=cgrulesengd.pp dest=/opt/cgrulesengd.pp
  register: se_run

- name: allow se linux policy
  shell: chdir=/opt semodule -i cgrulesengd.pp
  when: se_run.changed
```

Dans cet exemple, la tâche "allow se linux policy" ne sera exécutée que si la variable se_run.changed est True. Cette variable est valorisée par le résultat de l'état de la copie d'un fichier. Si le fichier cgrulesengd.pp n'existe pas ou est différent alors l'état du système sera à « changé ».

DevOps et Azure : Retour d'expérience Alter Way

LES BOUCLES

Les boucles sont dans Ansible des instructions très puissantes. Elles permettent d'exécuter la même tâche plusieurs fois avec des arguments différents.

```
- name: Installation des paquets système
  action: "{{ packager }} pkg={{ item }} state=latest"
  with_items:
  - {{ apache }}
  - mysql-server
  - php5
```

Dans cet exemple, le système distant doit installer ou se mettre à jour avec la dernière version des paquets Apache, MySql-server et PHP5 en une seule instruction. Le paquet apache est une variable, elle sera apache2 sur Debian et httpd sur CentOS. On aura mis en paramètre le nom de ce paquet pour rester agnostique par rapport au système d'exploitation des machines cibles.

```
- name: copy templates
  template: src={{ item.src }} dest={{ item.dest }}
  with_items:
        - src: templates/testsource1
          dest: /example/dest1/test.conf
        - src: templates/testsource2
          dest: /example/dest2/test.conf
```

Les boucles peuvent utiliser plusieurs arguments.

```
- copy: src={{ item }} dest=/etc/fooapp/ owner=root mode=600
      with_fileglob: /playbooks/files/fooapp/*
```

Les boucles peuvent prendre des listes de fichiers en argument

```
- name: Create file for demo
  action: debug msg="{{ item }}"
  with_lines:
       - ls -1 /tmp
```

LES VARIABLES

Les variables dans Ansible peuvent être utilisées au niveau des lignes de commande, dans les tâches, ainsi que dans les fichiers de « template ».

Au niveau de la ligne de commande :
- ansible-playbook site.yml --extra-vars="user=example domain=alterway.fr". les variables user et domain pourront être utilisées dans les playbooks ou templates sous la forme {{ user }} et {{ domain }}
Au niveau des fichiers YML on référencera une variable simple de cette façon
- variable : valeur
Les variables peuvent être simples ou complexes, statique ou dynamiques.

Quelques exemples :

```
vars:
   simple: “hello”
   a_list:
        - a
        - b
   complex:
      ghostbusters: [ 'egon', 'ray', 'peter', 'winston' ]
      mice: [ 'pinky', 'brain', 'larry' ]
```

LES RÔLES

Les rôles ont été créés dans Ansible pour favoriser la réutilisabilité des playbooks et permettre aux teams DevOps d’avoir une approche en « layers ».

Ainsi un déploiement d’une architecture complexe sera un playbook constitué de plusieurs rôles. Bien sûr, l’exécution des rôles peut être conditionnée en fonction de paramètres.

```
---
- hosts: webs
  vars_files:
   - vars/default.yml
  roles:
    - { role: nginx,  tags: ["nginx-server", "reverse-proxy"] }
    - { role: apache, tags: ["webserver"] }
    - { role: supervisord, tags: ["supervisord"] }
    - { role: php-fpm, tags: ["php-fpm"] }
    - { role: redis-client, tags: ["redis"] }
```

Ce playbook va déployer sur le groupe de machine webs des frontaux web constitués de différents middle-wares configurés dans chacun des rôles.

Il est ainsi très facile de faire évoluer ou changer les stacks logicielles.
Un rôle est un ensemble de tâches, variables, fichiers, templates, et handlers. Il suffit d’organiser les répertoires de la façon suivante pour utiliser des rôles dans les playbooks.

```
roles/
  nginx/
    tasks/
    templates/
    vars/
    handlers/
  apache/
    tasks/
    templates/
    vars/
    handlers/
...
```

DevOps et Azure : Retour d'expérience Alter Way

PILOTAGE AZURE PAR ANSIBLE

Ce livre blanc illustre le caractère agnostique des outils DevOps mis à disposition par la communauté. Nous avons pour l'occasion mis en œuvre un playbook permettant de déployer une architecture complète sur Azure.

Ce playbook permet de créer sur Azure :
- Un groupe d'affinité
- Un compte de stockage et un container rattaché au groupe d'affinité
- Un service cloud rattaché au groupe d'affinité
- Et des VMs utilisant tous les éléments précédemment créés

Aucun module Ansible n'étant disponible à date pour Azure, nous avons créé les modules Ansible nécessaires pour le pilotage Azure. Ceux-ci sont maintenant disponibles sur github (https://github.com/herveleclerc/ansible-azure) simplement à des fins d'illustration.

Voici le playbook très simple permettant de provisionner une VM sur un environnement Azure sur un réseau virtuel Azure (VNET) existant.

```
- hosts: localhost
  vars_files:
      - ../ANSIBLE/vars/azure.yml
  tasks:
  - name: "Create an Affinity Group"
    azure_affinity_group: >
       subscription_id="{{ subscription_id }}"
       certificate_path="{{ certificate_path }}"
       state=present
       name="{{ affinity_group_name }}"
       label="{{ affinity_group_label }}"
       location="{{ affinity_group_location }}"
       desc="{{ affinity_group_desc }}"
    register: affinity_group

  - name: "Create a Storage Account"
    azure_storage_account: >
       subscription_id="{{ subscription_id }}"
       certificate_path="{{ certificate_path }}"
       state=present
       name="{{ storage_account_name }}"
       affinity_group={{ affinity_group.name }}
       container_name="{{ storage_account_container_name }}"
       label="{{ storage_account_label }}"
       desc="{{ storage_account_desc }}"
    register: storage_account

  - name: "Create an Hosted Service"
    azure_hosted_service: >
       subscription_id="{{ subscription_id }}"
       certificate_path="{{ certificate_path }}"
       state=present
       name="{{ hosted_service_name }}"
       affinity_group={{ affinity_group.name }}
       label="{{ hosted_service_label }}"
       desc="{{ hosted_service_desc }}"
    register: hosted_service

  - name: "Create a VM"
    azure_vm: >
       subscription_id'{{ subscription_id }}'
       certificate_path={{ certificate_path }}'
       state=present
       vm_name='{{ vm_name }}'
       image_name='{{ image_name }}'
       sshcert='{{ sshcert }}'
       fingerprint='{{ fingerprint }}'
       username='{{ username }}'
       password='{{ password }}'
       affinity_group='{{ affinity_group.name }}'
       service_name='{{ storage_account.name }}'
       hosted_service_name='{{ hosted_service.name }}'
       virtual_network_name='{{ virtual_network_name }}'
       subnet_name='{{ subnet_name }}'
    register: vm
```

Dans ce playbook les informations retournées par une tâche sont stockées dans des variables par le biais de la commande « register ». Cette variable peut être utilisée dans une autre tâche du playbook en utilisant son nom et l'attribut à utiliser, ex : affinity_group.name

Lorsque l'on lance un playbook voici la sortie standard que l'on obtient.

Exemple lors de la création d'une VM Azure via la commande : ansible-playbook tutorial.yml –v
(Exécution en mode verbeux grâce à l'argument –v)

```
PLAY [localhost] ***************************************************************

TASK: [Create an Affinity Group] **********************************************
ok: [localhost] => {"changed": false, "msg": "Affinity group awdevopsaffg1 already
exist", "name": "awdevopsaffg1"}

TASK: [Create a Storage Account] **********************************************
changed: [localhost] => {"changed": true, "name": "awdevopsstor1", "result": true}

TASK: [Create an Hosted Service] **********************************************
changed: [localhost] => {"changed": true, "name": "apacheaw1", "result": true}

TASK: [Create a VM] ***********************************************************
changed: [localhost] => {"changed": true, "result": true, "vm_name": "apachevm1"}

PLAY RECAP ********************************************************************
localhost                  : ok=4    changed=3    unreachable=0    failed=0
```

Ansible propose également la possibilité d'avoir des plugins de « callback » qui permettent de mettre en place des notifications, logging, etc au niveau de toutes les déclenchements ou résultats d'exécution d'Ansible.

Dans cet exemple la création du groupe d'affinité awdevopsaffg1 n'échoue pas mais notifie simplement que celui-ci existe et qu'il va être utilisé. Cet exemple illustre les aspects de ré-entrance du système.

Dans ce playbook nous faisons appel à 4 modules custom développés pour Azure :
azure_affinity_group
azure_storage_account
azure_hosted_service
azure_vm

Voici, pour exemple, des sections de code commentées d'un des modules (azure_affinity_ group)

```
try:
    from azure import *
    from azure.servicemanagement import *
except ImportError:
    print "failed=True msg='azure required for this module'"
    sys.exit(1)
```

Le SDK d'Azure doit être installé pour pouvoir interagir avec la plateforme.

```
def wait_for_completion(promise,sms):
    if not promise: return
    while True:
        time.sleep(5)
        operation_result = sms.get_operation_status(promise.request_id)
        print('   ' + operation_result.status)
        if (operation_result.status == "Succeeded"):
            return
```

L'API Azure étant asynchrone nous attendons la fin de la tâche avant de passer à une tâche suivante.

```
def main():
   module = AnsibleModule(
        argument_spec = dict(
            state = dict(choices=['list', 'present', 'absent', 'detail'],
default='list'),
            subscription_id = dict(type='str'),
            certificate_path = dict(type='str'),
            name = dict(type='str',default=''),
            location = dict(type='str'),
            label = dict(type='str'),
            desc = dict(type='str'),
        ),
        required_together = (
            ['subscription_id', 'certificate_path'],
        ),
    )
```

DevOps et Azure : Retour d'expérience Alter Way

PILOTAGE AZURE PAR ANSIBLE

On déclare sous forme de « dict » les paramètres attendus par les modules et leurs interdépendances.

```
try:
      sms = ServiceManagementService(subscription_id, certificate_path)
   except:
      module.fail_json(msg='unable to acces to azure check
subscription_id/certificate_path')
```

On se connecte à Azure en utilisant un identifiant de souscription et la clé privée associée à un compte d'administration Azure de cette souscription.

```
locations = sms.list_locations()
affinity_groups=sms.list_affinity_groups()
```

L 'objet ServiceManagementService possède un grand nombre de méthodes permettant de faire des actions CRUD sur Azure. Ici nous récupérons la liste des locations et des groupes d'affinité. Ces listes nous permettront de vérifier les paramètres passés au module.

```
if state == 'present':
       for affinity_group in affinity_groups:
           if name == affinity_group.name:
               module.exit_json(changed=False, msg="Affinity group %s already exist"
% name, name=name)
       res=sms.create_affinity_group(name, label, location, desc)
       wait_for_completion(res,sms)
```

Si le paramètre d'état est « list », nous demandons au module de nous renvoyer une liste JSON des groupes d'affinité disponibles.

```
if state == 'list':
       ag=[]
       for affinity_group in affinity_groups:
          ag.append(affinity_group.name)
       module.exit_json(changed=True, affinity_groups=ag)
```

Nous demandons les informations détaillées d'un groupe d'affinité si le paramètre d'état est « detail ».Ces données sont retournées au format JSON dans la variable « affinity_groups »

```
if state == 'detail':
       for affinity_group in affinity_groups:
           if name == affinity_group.name:
               properties=sms.get_affinity_group_properties(name)
               properties_json={
                   'description' : properties.description,
                   'label': properties.label,
                   'name': properties.name,
                   'capabilities': properties.capabilities
               }

               module.exit_json(changed=True, affinity_groups=properties_json)
       module.exit_json(changed=False, msg="Affinity group %s does not exist" % name)
```

Si le paramètre d'état est « present », si le groupe d'affinité est présent dans la liste des groupes d'affinité, on retourne alors un message d'existence du groupe d'affinité. Dans le cas contraire, le groupe d'affinité est créé. A la fin de la création on retourne son nom dans la variable « name ».

```
if state == 'absent':
       for affinity_group in affinity_groups:
           if name == affinity_group.name:
               res=sms.delete_affinity_group(name)
               wait_for_completion(res,sms)
               module.exit_json(changed=True, result='Deleted', name=name)

       module.exit_json(changed=False, msg="Affinity group %s does not exist" % name)
```

Si le paramètre d'état est « absent», si le groupe d'affinité existe on essaye de le supprimer. Cette suppression n'est possible que si aucun composant Azure n'est rattaché à ce groupe d'affinité.

```
from ansible.module_utils.basic import *

main()
```

On rend la main à Ansible pour l'exécution du module

SYNTHÈSE

Ansible est un outil DevOps simple, extensible, agnostique, souple et orienté architecture en « layers ». Il peut être même utilisé pour le pilotage d'architectures Windows par le biais de WinRM (Windows Remote Management, implémentation du protocole WS-Management) et utilisé sous forme de module comme illustré dans la vidéo suivante : https://www.youtube.com/watch?v=fGCh3ZfaaLU.

Azure proposant des services IAAS hybride, cet outil est particulièrement adapté à la démarche DevOps que nous préconisons d'adopter également sur cette plateforme.

Conclusion

Avec l'avènement du Cloud, il n'est plus possible de se limiter à un raisonnement unitaire pour chaque VM sur laquelle s'exécute telle ou telle composante de l'application. L'unité de déploiement a évolué et s'étend au périmètre de l'application elle-même. Cette évolution n'est pas sans incidence sur la gestion opérationnelle. Elle implique la redéfinition des modèles de conception et l'automatisation des étapes du cycle de vie de l'application ce qui inclut le provisioning des ressources, la gestion de la configuration, le déploiement d'applications, la gestion des mises à jour logicielles, la supervision….

Dans ce nouveau contexte, développeurs et responsables de l'exploitation doivent nécessairement travailler ensemble pour garantir le succès du déploiement de l'application. Pour ce faire, c'est l'intégralité des éléments constitutifs du cycle de vie d'une application qui doit être manipulable par logiciel, en se basant sur des modèles déclaratifs. C'est là l'un des axes technologiques retenus par la plupart des outils s'inscrivant dans une démarche « DevOps ». Cette approche devient donc un ingrédient indispensable du succès de la mise en place d'un service dans le Cloud tandis que le Cloud constitue un facteur d'accélération de la diffusion des bonnes pratiques « DevOps ».

La dimension « DevOps » est au cœur de la stratégie Microsoft pour la plateforme Azure. Scott Guthrie, vice-président exécutif du groupe « Cloud and Enterprise » s'en est fait l'écho à l'occasion de l'évènement « Build », en avril 2014 : « Imagine a world where infrastructure and platform services blend together in one seamless experience, so developers and IT professionals no longer have to work in disparate environments in the cloud. Microsoft has been rapidly innovating to solve this problem, and we have taken a big step toward that vision today. »

L'évolution du portail Azure avec l'intégration du « Azure Resource Manager », les technologies de provisioning Azure (Cmdlet PowerShell, xplat-cli), de configuration Windows (PowerShell DSC et OneNet) et l'ouverture de la plateforme Microsoft Azure vers les solutions Open Source (Puppet, Chef) concrétisent une première étape de cette vision.

Références techniques

AZURE : « DEVOPS » POUR LE CLOUD

MS OpenTech et DevOps :
http://msopentech.com/blog/project-categories/devops

« CONTINUOUS DELIVERY »

Ouvrage de Jez Humble et David Farley :
« Continuous Delivery : Reliable Software Releases through Build, Test, and Deployment Automation »

« FAILSAFE »

Livre blanc de Marc Mercuri, Ulrich Homann et Andrew Townhill :
« Failsafe: Guidance for Resilient Cloud Architectures »
http://msdn.microsoft.com/en-us/library/windowsazure/jj853352.asPX

POWERSHELL

Azure PowerShell :
http://msopentech.com/blog/project-categories/devops

AZURE : « DEVOPS » POUR LE CLOUD

MS OpenTech et DevOps :
http://msopentech.com/blog/project-categories/devops

Modules PowerShell :
http://msdn.microsoft.com/en-us/library/windows/desktop/dd878324(v=vs.85).aspx

Tutoriaux pour l'écriture de Cmdlets :
http://msdn.microsoft.com/en-us/library/windows/desktop/dd878321(v=vs.85).aspx

Création d'un Workflow avec activités Windows PowerShell
http://msdn.microsoft.com/en-us/library/windows/desktop/hh852760(v=vs.85).aspx

DSC

Windows PowerShell Desired State Configuration :
http://technet.microsoft.com/en-us/library/dn249912.aspx

Démarrer avec Windows PowerShell Desired State Configuration
http://technet.microsoft.com/en-us/library/dn249918.aspx

Déployer une configuration DSC sur une VM Azure
http://www.powershellmagazine.com/2014/04/02/pushing-desired-state-configuration-to-an-azure-vm

PUPPET

Documentation Puppet Labs :
https://puppetlabs.com/puppet/what-is-puppet
http://docs.puppetlabs.com/guides/puppet_internals.html
http://docs.puppetlabs.com/guides/introduction.html
https://puppetlabs.com/puppet/enterprise-vs-open-source

MS OpenTech :
http://msopentech.com/opentech-projects/puppet
http://msopentech.com/blog/2013/12/11/windows-azure-provisioning-via-puppet

CHEF

Documents Opscode :
http://docs.opscode.com/chef_overview.html
http://www.opscode.com/chef/#which-chef
http://community.opscode.com/cookbooks/iis
http://community.opscode.com/cookbooks/windows
http://community.opscode.com/cookbooks/sql_server

MS OpenTech :
https://github.com/opscode/knife-azure
http://msopentech.com/opentech-projects/chef
http://blogs.technet.com/b/openness/archive/2012/12/13/opscode-chef-windows-azure.aspx
http://msopentech.com/blog/2014/03/07/using-chefs-knives-manage-windows-azure-resources

Microsoft Azure