Automatiser avec REST

ONTAP Select

NetApp
January 29, 2026

Sommaire

Automatiser avec REST

Concepts

Base de services Web REST pour le déploiement et la gestion des clusters ONTAP Select

Le transfert d'état représentatif (REST) est un style de création d'applications web distribuées. Appliqué à la conception d'une API de services web, il établit un ensemble de technologies et de bonnes pratiques pour exposer les ressources serveur et gérer leurs états. Il utilise les protocoles et normes les plus courants pour fournir une base flexible au déploiement et à la gestion des clusters ONTAP Select .

Architecture et contraintes classiques

REST a été formellement articulé par Roy Fielding dans son doctorat "thèse" Créé à l'Université de Californie à Irvine en 2000, il définit un style architectural grâce à un ensemble de contraintes qui, collectivement, améliorent les applications web et les protocoles sous-jacents. Ces contraintes établissent une application de services web RESTful basée sur une architecture client/serveur utilisant un protocole de communication sans état.

Ressources et représentation de l'État

Les ressources sont les composants de base d'un système web. Lors de la création d'une application de services web REST, les premières tâches de conception incluent :

- Identification des ressources système ou serveur. Chaque système utilise et gère des ressources. Une ressource peut être un fichier, une transaction métier, un processus ou une entité administrative. L'une des premières tâches de la conception d'une application basée sur les services Web REST est d'identifier les ressources.
- Définition des états des ressources et des opérations associées. Les ressources sont toujours dans un état parmi un nombre fini. Ces états, ainsi que les opérations associées utilisées pour modifier les états, doivent être clairement définis.

Des messages sont échangés entre le client et le serveur pour accéder et modifier l'état des ressources selon le modèle générique CRUD (Créer, Lire, Mettre à jour et Supprimer).

Points de terminaison URI

Chaque ressource REST doit être définie et rendue disponible selon un schéma d'adressage précis. Les points de terminaison où les ressources sont localisées et identifiées utilisent un identifiant de ressource uniforme (URI). Cet URI fournit un cadre général pour créer un nom unique pour chaque ressource du réseau. L'URL (Uniform Resource Locator) est un type d'URI utilisé par les services web pour identifier et accéder aux ressources. Les ressources sont généralement présentées dans une structure hiérarchique similaire à un répertoire de fichiers.

messages HTTP

Le protocole HTTP (Hypertext Transfer Protocol) est utilisé par le client et le serveur de services Web pour échanger des messages de requête et de réponse concernant les ressources. Lors de la conception d'une application de services Web, les verbes HTTP (tels que GET et POST) sont associés aux ressources et aux

actions de gestion d'état correspondantes.

HTTP est un protocole sans état. Par conséquent, pour associer un ensemble de requêtes et de réponses connexes dans une même transaction, des informations supplémentaires doivent être incluses dans les en-têtes HTTP accompagnant les flux de données de requêtes/réponses.

Formatage JSON

Bien que les informations puissent être structurées et transférées entre un client et un serveur de plusieurs manières, l'option la plus courante (et celle utilisée avec l'API REST Deploy) est JavaScript Object Notation (JSON). JSON est une norme industrielle pour la représentation de structures de données simples en texte brut et permet de transférer des informations d'état décrivant les ressources.

Comment accéder à l'API ONTAP Select Deploy

En raison de la flexibilité inhérente aux services Web REST, l'API ONTAP Select Deploy est accessible de plusieurs manières différentes.

Déployer l'interface utilisateur native de l'utilitaire

L'accès à l'API se fait principalement via l'interface utilisateur Web ONTAP Select Deploy. Le navigateur appelle l'API et reformate les données conformément à la conception de l'interface utilisateur. L'accès à l'API se fait également via l'interface de ligne de commande de l'utilitaire Deploy.

Page de documentation en ligne ONTAP Select Deploy

La page de documentation en ligne ONTAP Select Deploy offre un point d'accès alternatif pour l'utilisation d'un navigateur. Outre la possibilité d'exécuter directement des appels d'API individuels, cette page inclut une description détaillée de l'API, incluant les paramètres d'entrée et d'autres options pour chaque appel. Les appels d'API sont organisés en plusieurs domaines ou catégories fonctionnels.

Programme personnalisé

Vous pouvez accéder à l'API Deploy à l'aide de différents langages et outils de programmation. Parmi les plus courants, on trouve Python, Java et cURL. Un programme, un script ou un outil utilisant l'API agit comme un client de services Web REST. L'utilisation d'un langage de programmation vous permet de mieux comprendre l'API et d'automatiser les déploiements ONTAP Select .

Gestion des versions de l'API ONTAP Select Deploy

L'API REST incluse dans ONTAP Select Deploy possède un numéro de version. Ce numéro est indépendant de celui de Deploy. Veuillez connaître la version de l'API incluse dans votre version de Deploy et son impact potentiel sur son utilisation.

La version actuelle de l'utilitaire d'administration Deploy inclut la version 3 de l'API REST. Les versions précédentes de l'utilitaire Deploy incluent les versions d'API suivantes :

Déployer 2.8 et versions ultérieures
: ONTAP Select Deploy 2.8 et toutes les versions ultérieures incluent la version 3 de l'API REST.

Déployer 2.7.2 et versions antérieures
: ONTAP Select Deploy 2.7.2 et toutes les versions antérieures incluent la version 2 de l'API REST.

Les versions 2 et 3 de l'API REST ne sont pas compatibles. Si vous effectuez une mise à niveau vers Deploy 2.8 ou une version ultérieure à partir d'une version antérieure incluant la version 2 de l'API, vous devez mettre à jour tout code existant accédant directement à l'API, ainsi que tous les scripts utilisant l'interface de ligne de commande.

Caractéristiques opérationnelles de base de l'API ONTAP Select Deploy

Bien que REST établisse un ensemble commun de technologies et de bonnes pratiques, les détails de chaque API peuvent varier en fonction des choix de conception. Il est important de connaître les détails et les caractéristiques opérationnelles de l'API ONTAP Select Deploy avant de l'utiliser.

Hôte hyperviseur par rapport au nœud ONTAP Select

Un hôte hyperviseur est la plateforme matérielle principale qui héberge une machine virtuelle ONTAP Select . Lorsqu'une machine virtuelle ONTAP Select est déployée et active sur un hôte hyperviseur, elle est considérée comme un nœud ONTAP Select. Avec la version 3 de l'API REST Deploy, les objets hôte et nœud sont distincts. Cela permet une relation un-à-plusieurs, où un ou plusieurs nœuds ONTAP Select peuvent s'exécuter sur le même hôte hyperviseur.

Identifiants d'objet

Chaque instance ou objet de ressource se voit attribuer un identifiant unique lors de sa création. Ces identifiants sont uniques au sein d'une instance spécifique d' ONTAP Select Deploy. Après un appel d'API créant une instance d'objet, la valeur d'identifiant associée est renvoyée à l'appelant dans le `location` En-tête de la réponse HTTP. Vous pouvez extraire l'identifiant et l'utiliser lors des appels ultérieurs pour faire référence à l'instance de ressource.

Le contenu et la structure interne des identifiants d'objet peuvent changer à tout moment. Vous ne devez utiliser les identifiants que dans les appels d'API applicables, lorsque cela est nécessaire, pour faire référence aux objets associés.

Identifiants de demande

Chaque requête API réussie se voit attribuer un identifiant unique. Cet identifiant est renvoyé dans le `request-id` En-tête de la réponse HTTP associée. Vous pouvez utiliser un identifiant de requête pour désigner collectivement les activités d'une transaction API spécifique. Par exemple, vous pouvez récupérer tous les messages d'événement d'une transaction en fonction de l'identifiant de requête.

Appels synchrones et asynchrones

Il existe deux manières principales par lesquelles un serveur exécute une requête HTTP reçue d'un client :

- Synchrone Le serveur exécute la demande immédiatement et répond avec un code d'état 200, 201 ou 204.
- Asynchrone : le serveur accepte la requête et répond avec le code d'état 202. Cela indique que le serveur a accepté la requête du client et a lancé une tâche en arrière-plan pour la traiter. La réussite ou l'échec final n'est pas immédiatement disponible et doit être déterminé par des appels d'API supplémentaires.

Confirmer l'achèvement d'une tâche de longue durée

En général, toute opération longue est traitée de manière asynchrone via une tâche d'arrière-plan sur le serveur. Avec l'API REST Deploy, chaque tâche d'arrière-plan est ancrée par un objet Job qui la suit et fournit des informations, telles que son état actuel. Un objet Job, avec son identifiant unique, est renvoyé dans la réponse HTTP après la création d'une tâche d'arrière-plan.

Vous pouvez interroger directement l'objet Job pour déterminer la réussite ou l'échec de l'appel d'API associé. Pour plus d'informations, consultez la section « Traitement asynchrone avec l'objet Job ».

Outre l'utilisation de l'objet Job, il existe d'autres moyens de déterminer le succès ou l'échec d'une demande, notamment :

- Messages d'événement : vous pouvez récupérer tous les messages d'événement associés à un appel d'API spécifique à l'aide de l'ID de requête renvoyé avec la réponse d'origine. Ces messages contiennent généralement une indication de réussite ou d'échec et peuvent également être utiles lors du débogage d'une condition d'erreur.
- État ou statut des ressources Plusieurs ressources conservent une valeur d'état ou de statut que vous pouvez interroger pour déterminer indirectement la réussite ou l'échec d'une demande.

Sécurité

L'API de déploiement utilise les technologies de sécurité suivantes :

- Sécurité de la couche transport : tout le trafic envoyé sur le réseau entre le serveur et le client Deploy est chiffré via TLS. L'utilisation du protocole HTTP sur un canal non chiffré n'est pas prise en charge. La version 1.2 de TLS est prise en charge.
- Authentification HTTP : l'authentification de base est utilisée pour chaque transaction API. Un en-tête HTTP, contenant le nom d'utilisateur et le mot de passe dans une chaîne base64, est ajouté à chaque requête.

Transaction API de demande et de réponse pour ONTAP Select

Chaque appel à l'API Deploy est exécuté sous forme de requête HTTP adressée à la machine virtuelle Deploy, qui génère une réponse associée au client. Cette paire requête/réponse est considérée comme une transaction API. Avant d'utiliser l'API Deploy, vous devez vous familiariser avec les variables d'entrée disponibles pour contrôler une requête et le contenu de la réponse.

Variables d'entrée contrôlant une requête API

Vous pouvez contrôler la manière dont un appel API est traité via des paramètres définis dans la requête HTTP.

En-têtes de requête

Vous devez inclure plusieurs en-têtes dans la requête HTTP, notamment :

- type de contenu Si le corps de la demande inclut JSON, cet en-tête doit être défini sur application/json.
- accepter Si le corps de la réponse doit inclure du JSON, cet en-tête doit être défini sur application/json.
- L'authentification de base doit être définie avec le nom d'utilisateur et le mot de passe codés dans une chaîne base64.

Corps de la requête

Le contenu du corps de la requête varie selon l'appel. Le corps de la requête HTTP se compose de l'un des éléments suivants :

- Objet JSON avec variables d'entrée (comme le nom d'un nouveau cluster)
- Vide

Filtrer les objets

Lors d'un appel d'API utilisant GET, vous pouvez limiter ou filtrer les objets renvoyés en fonction de n'importe quel attribut. Par exemple, vous pouvez spécifier une valeur exacte à laquelle correspondre :

```
<field>=<query value>
```

Outre une correspondance exacte, d'autres opérateurs permettent de renvoyer un ensemble d'objets sur une plage de valeurs. ONTAP Select prend en charge les opérateurs de filtrage présentés ci-dessous.

Opérateur	Description
=	Égal à
<	Moins que
>	Plus grand que
⇐	Inférieur ou égal à
>=	Supérieur ou égal à
|	Ou
!	Pas égal à
*	Caractère générique gourmand

Vous pouvez également renvoyer un ensemble d'objets selon qu'un champ spécifique est défini ou non en utilisant le mot-clé null ou sa négation (!null) dans le cadre de la requête.

Sélection des champs d'objet

Par défaut, un appel d'API via GET ne renvoie que les attributs identifiant de manière unique le ou les objets. Cet ensemble minimal de champs sert de clé pour chaque objet et varie selon son type. Vous pouvez sélectionner des propriétés d'objet supplémentaires à l'aide du paramètre de requête fields, comme suit :

- Champs peu coûteux Spécifier `fields=*` pour récupérer les champs d'objet qui sont conservés dans la mémoire du serveur local ou qui nécessitent peu de traitement pour y accéder.
- Champs coûteux Spécifier `fields=**` pour récupérer tous les champs de l'objet, y compris ceux nécessitant un traitement serveur supplémentaire pour y accéder.
- Sélection de champ personnalisé Utiliser `fields=FIELDNAME` pour spécifier le champ exact souhaité. Lorsque vous demandez plusieurs champs, les valeurs doivent être séparées par des virgules sans espaces.

Il est recommandé de toujours identifier les champs spécifiques souhaités. Ne récupérez les champs les moins chers ou les plus coûteux que lorsque cela est nécessaire. La classification des champs les moins chers et les plus coûteux est déterminée par NetApp sur la base d'une analyse interne des performances. La classification d'un champ donné peut changer à tout moment.

Trier les objets dans l'ensemble de sortie

Les enregistrements d'une collection de ressources sont renvoyés dans l'ordre par défaut défini par l'objet. Vous pouvez modifier cet ordre à l'aide du paramètre de requête order_by, en indiquant le nom du champ et le sens de tri comme suit :

```
order_by=<field name> asc|desc
```

Par exemple, vous pouvez trier le champ type par ordre décroissant suivi de l'ID par ordre croissant :

```
order_by=type desc, id asc
```

Lorsque vous incluez plusieurs paramètres, vous devez séparer les champs par une virgule.

Pagination

Lors d'un appel API via GET pour accéder à une collection d'objets de même type, tous les objets correspondants sont renvoyés par défaut. Si nécessaire, vous pouvez limiter le nombre d'enregistrements renvoyés en utilisant le paramètre de requête max_records. Par exemple :

```
max_records=20
```

Si nécessaire, vous pouvez combiner ce paramètre avec d'autres paramètres de requête pour affiner les résultats. Par exemple, la requête suivante renvoie jusqu'à 10 événements système générés après l'heure spécifiée :

```
time⇒ 2019-04-04T15:41:29.140265Z&max_records=10
```

Vous pouvez émettre plusieurs requêtes pour parcourir les événements (ou tout type d'objet). Chaque appel d'API ultérieur doit utiliser une nouvelle valeur temporelle basée sur le dernier événement du dernier ensemble de résultats.

Interpréter une réponse API

Chaque requête API génère une réponse au client. Vous pouvez examiner cette réponse pour déterminer si elle a abouti et récupérer des données supplémentaires si nécessaire.

Code d'état HTTP

Les codes d'état HTTP utilisés par l'API REST Deploy sont décrits ci-dessous.

Code	Signification	Description
200	OK	Indique le succès des appels qui ne créent pas de nouvel objet.
201	Créé	Un objet est créé avec succès ; l'en-tête de réponse d'emplacement inclut l'identifiant unique de l'objet.
202	Accepté	Une tâche d'arrière-plan de longue durée a été démarrée pour exécuter la demande, mais l'opération n'est pas encore terminée.
400	Mauvaise demande	La demande d'entrée n'est pas reconnue ou est inappropriée.

Code	Signification	Description
403	Interdit	L'accès est refusé en raison d'une erreur d'autorisation.
404	Non trouvé	La ressource référencée dans la demande n'existe pas.
405	Méthode non autorisée	Le verbe HTTP dans la requête n'est pas pris en charge pour la ressource.
409	Conflit	Une tentative de création d'un objet a échoué car l'objet existe déjà.
500	Erreur interne	Une erreur interne générale s'est produite sur le serveur.
501	Non implémenté	L'URI est connu mais n'est pas capable d'exécuter la requête.

En-têtes de réponse

Plusieurs en-têtes sont inclus dans la réponse HTTP générée par le serveur de déploiement, notamment :

- request-id Chaque demande d'API réussie se voit attribuer un identifiant de demande unique.
- emplacement Lorsqu'un objet est créé, l'en-tête d'emplacement inclut l'URL complète du nouvel objet, y compris l'identifiant d'objet unique.

Corps de la réponse

Le contenu de la réponse associée à une requête API varie selon l'objet, le type de traitement et la réussite ou l'échec de la requête. Le corps de la réponse est affiché au format JSON.

- Objet unique : un objet unique peut être renvoyé avec un ensemble de champs en fonction de la requête. Par exemple, vous pouvez utiliser GET pour récupérer les propriétés sélectionnées d'un cluster à l'aide de son identifiant unique.
- Objets multiples : plusieurs objets d'une collection de ressources peuvent être renvoyés. Dans tous les cas, un format cohérent est utilisé, avec `num_records` indiquant le nombre d'enregistrements et les enregistrements contenant un tableau d'instances d'objet. Par exemple, vous pouvez récupérer tous les nœuds définis dans un cluster spécifique.
- Objet Job : si un appel d'API est traité de manière asynchrone, un objet Job est renvoyé, qui ancre la tâche en arrière-plan. Par exemple, la requête POST utilisée pour déployer un cluster est traitée de manière asynchrone et renvoie un objet Job.
- Objet d'erreur : si une erreur se produit, un objet d'erreur est toujours renvoyé. Par exemple, vous recevrez une erreur lorsque vous tenterez de créer un cluster avec un nom qui existe déjà.
- Vide : Dans certains cas, aucune donnée n'est renvoyée et le corps de la réponse est vide. Par exemple, le corps de la réponse est vide après l'utilisation de DELETE pour supprimer un hôte existant.

Traitement asynchrone à l'aide de l'objet Job pour ONTAP Select

Certains appels d'API Deploy, notamment ceux qui créent ou modifient une ressource, peuvent prendre plus de temps que d'autres. ONTAP Select Deploy traite ces requêtes de longue durée de manière asynchrone.

Requêtes asynchrones décrites à l'aide de l'objet Job

Après un appel d'API exécuté de manière asynchrone, le code de réponse HTTP 202 indique que la requête a été validée et acceptée, mais pas encore terminée. La requête est traitée en tâche d'arrière-plan et continue

de s'exécuter après la réponse HTTP initiale au client. La réponse inclut l'objet Job ancrant la requête, ainsi que son identifiant unique.

Vous devez vous référer à la page de documentation en ligne ONTAP Select Deploy pour déterminer quels appels d'API fonctionnent de manière asynchrone.

Interroger l'objet Job associé à une requête API

L'objet Job renvoyé dans la réponse HTTP contient plusieurs propriétés. Vous pouvez interroger la propriété « state » pour déterminer si la requête a abouti. Un objet Job peut être dans l'un des états suivants :

- En file d'attente
- Exécution
- Succès
- Échec

Il existe deux techniques que vous pouvez utiliser lors de l'interrogation d'un objet Job pour détecter un état terminal pour la tâche, soit un succès, soit un échec :

- Demande d'interrogation standard L'état actuel du travail est renvoyé immédiatement
- Demande d'interrogation longue L'état du travail est renvoyé uniquement lorsque l'un des événements suivants se produit :
  - L'état a changé plus récemment que la valeur de date et d'heure fournie dans la demande de sondage
  - La valeur du délai d'attente a expiré (1 à 120 secondes)

Les interrogations standard et longue utilisent le même appel d'API pour interroger un objet Job. Cependant, une requête longue inclut deux paramètres de requête : `poll_timeout` et `last_modified` .

Vous devez toujours utiliser une interrogation longue pour réduire la charge de travail sur la machine virtuelle Deploy.

Procédure générale pour émettre une requête asynchrone

Vous pouvez utiliser la procédure de haut niveau suivante pour terminer un appel d'API asynchrone :

1. Émettez l'appel API asynchrone.
2. Recevez une réponse HTTP 202 indiquant l'acceptation réussie de la demande.
3. Extraire l'identifiant de l'objet Job du corps de la réponse.
4. Dans une boucle, effectuez les opérations suivantes à chaque cycle :
   a. Obtenez l'état actuel du travail avec une requête longue durée
   b. Si le travail est dans un état non terminal (en file d'attente, en cours d'exécution), exécutez à nouveau la boucle.
5. Arrêtez-vous lorsque le travail atteint un état terminal (succès, échec).

Accès avec un navigateur

Avant d'accéder à l'API ONTAP Select Deploy avec un navigateur

Il y a plusieurs choses que vous devez savoir avant d'utiliser la page de documentation en ligne Deploy.

Plan de déploiement

Si vous prévoyez d'émettre des appels d'API dans le cadre de tâches de déploiement ou d'administration spécifiques, pensez à créer un plan de déploiement. Ce plan peut être formel ou informel et contient généralement vos objectifs et les appels d'API à utiliser. Pour plus d'informations, consultez la section Processus de workflow utilisant l'API REST Déployer.

Exemples JSON et définitions de paramètres

Chaque appel d'API est décrit dans la page de documentation selon un format cohérent. Le contenu comprend des notes d'implémentation, des paramètres de requête et des codes d'état HTTP. De plus, vous pouvez afficher des détails sur le JSON utilisé avec les requêtes et réponses d'API comme suit :

- Exemple de valeur : si vous cliquez sur « Exemple de valeur » lors d'un appel d'API, une structure JSON typique s'affiche. Vous pouvez modifier l'exemple selon vos besoins et l'utiliser comme entrée pour votre requête.
- Modèle Si vous cliquez sur *Modèle*, une liste complète des paramètres JSON s'affiche, avec une description pour chaque paramètre.

Attention lors de l'émission d'appels API

Toutes les opérations API effectuées via la page de documentation « Déployer » sont des opérations en direct. Veillez à ne pas créer, mettre à jour ou supprimer par erreur des configurations ou d'autres données.

Accéder à la page de documentation ONTAP Select Deploy

Vous devez accéder à la page de documentation en ligne ONTAP Select Deploy pour afficher la documentation de l'API, ainsi que pour émettre manuellement un appel d'API.

Avant de commencer

Vous devez avoir les éléments suivants :

- Adresse IP ou nom de domaine de la machine virtuelle ONTAP Select Deploy
- Nom d'utilisateur et mot de passe pour l'administrateur

Étapes

1. Tapez l'URL dans votre navigateur et appuyez sur **Entrée** :

   ```
   https://<ip_address>/api/ui
   ```

2. Sign in en utilisant le nom d'utilisateur et le mot de passe administrateur.

Résultat

La page Web de documentation de déploiement s'affiche avec les appels organisés par catégorie au bas de la page.

Comprendre et exécuter un appel d'API ONTAP Select Deploy

Les détails de tous les appels d'API sont documentés et affichés dans un format commun sur la page de documentation en ligne ONTAP Select Deploy. Comprendre un appel d'API permet d'accéder aux détails de tous les appels d'API et de les interpréter.

Avant de commencer

Vous devez être connecté à la page de documentation en ligne ONTAP Select Deploy. L'identifiant unique de votre cluster ONTAP Select doit avoir été attribué lors de sa création.

À propos de cette tâche

Vous pouvez récupérer les informations de configuration décrivant un cluster ONTAP Select grâce à son identifiant unique. Dans cet exemple, tous les champs classés comme peu coûteux sont renvoyés. Cependant, il est recommandé de ne demander que les champs spécifiques nécessaires.

Étapes

1. Sur la page principale, faites défiler vers le bas et cliquez sur **Cluster**.
2. Cliquez sur **GET /clusters/{cluster_id}** pour afficher les détails de l'appel d'API utilisé pour renvoyer des informations sur un cluster ONTAP Select .

Processus de flux de travail

Avant d'utiliser les workflows de l'API ONTAP Select Deploy

Vous devez vous préparer à examiner et à utiliser les processus de flux de travail.

Comprendre les appels API utilisés dans les workflows

La page de documentation en ligne ONTAP Select détaille chaque appel d'API REST. Plutôt que de répéter ces détails ici, chaque appel d'API utilisé dans les exemples de workflows inclut uniquement les informations nécessaires pour le localiser sur la page de documentation. Après avoir localisé un appel d'API spécifique, vous pouvez consulter ses détails complets, notamment les paramètres d'entrée, les formats de sortie, les codes d'état HTTP et le type de traitement de la requête.

Les informations suivantes sont incluses pour chaque appel d'API dans un workflow pour aider à localiser l'appel sur la page de documentation :

- Catégorie : les appels d'API sont organisés sur la page de documentation en domaines ou catégories fonctionnelles. Pour localiser un appel d'API spécifique, faites défiler la page jusqu'en bas et cliquez sur la catégorie d'API correspondante.
- Verbe HTTP : Le verbe HTTP identifie l'action effectuée sur une ressource. Chaque appel d'API est exécuté via un seul verbe HTTP.
- Chemin : le chemin détermine la ressource spécifique à laquelle l'action s'applique dans le cadre d'un appel. La chaîne de chemin est ajoutée à l'URL principale pour former l'URL complète identifiant la ressource.

Construire une URL pour accéder directement à l'API REST

Outre la page de documentation ONTAP Select , vous pouvez également accéder directement à l'API REST Deploy via un langage de programmation tel que Python. Dans ce cas, l'URL principale est légèrement différente de celle utilisée pour accéder à la page de documentation en ligne. Pour accéder directement à

l'API, vous devez ajouter /api à la chaîne de domaine et de port. Par exemple : `http://deploy.mycompany.com/api`

Workflow 1 : Créer un cluster d'évaluation à nœud unique ONTAP Select sur ESXi

Vous pouvez déployer un cluster ONTAP Select à nœud unique sur un hôte VMware ESXi géré par vCenter. Le cluster est créé avec une licence d'évaluation.

Le flux de travail de création de cluster diffère dans les situations suivantes :

- L'hôte ESXi n'est pas géré par vCenter (hôte autonome)
- Plusieurs nœuds ou hôtes sont utilisés au sein du cluster
- Le cluster est déployé dans un environnement de production avec une licence achetée
- L'hyperviseur KVM est utilisé à la place de VMware ESXi

1. Enregistrer les informations d'identification du serveur vCenter

Lors d'un déploiement sur un hôte ESXi géré par un serveur vCenter, vous devez ajouter des informations d'identification avant d'enregistrer l'hôte. L'utilitaire d'administration Deploy peut ensuite utiliser ces informations d'identification pour s'authentifier auprès de vCenter.

Catégorie	verbe HTTP	Chemin
Déployer	POSTE	/sécurité/informations d'identification

Boucle

```
curl -iX POST -H 'Content-Type: application/json' -u admin:<password> -k
-d @step01 'https://10.21.191.150/api/security/credentials'
```

Entrée JSON (étape 01)

```
{
  "hostname": "vcenter.company-demo.com",
  "type": "vcenter",
  "username": "misteradmin@vsphere.local",
  "password": "mypassword"
}
```

Type de traitement

Asynchrone

Sortir

- ID d'identification dans l'en-tête de réponse de localisation
- Objet de travail

2. Enregistrer un hôte hyperviseur

Vous devez ajouter un hôte hyperviseur sur lequel la machine virtuelle contenant le nœud ONTAP Select s'exécutera.

Catégorie	verbe HTTP	Chemin
Cluster	POSTE	/hôtes

Boucle

```
curl -iX POST -H 'Content-Type: application/json' -u admin:<password> -k
-d @step02 'https://10.21.191.150/api/hosts'
```

Entrée JSON (étape 02)

```
{
  "hosts": [
    {
      "hypervisor_type": "ESX",
      "management_server": "vcenter.company-demo.com",
      "name": "esx1.company-demo.com"
    }
  ]
}
```

Type de traitement

Asynchrone

Sortir

- ID d'hôte dans l'en-tête de réponse d'emplacement
- Objet de travail

3. Créer un cluster

Lorsque vous créez un cluster ONTAP Select , la configuration de cluster de base est enregistrée et les noms de nœuds sont automatiquement générés par Deploy.

Catégorie	verbe HTTP	Chemin
Cluster	POSTE	/groupes

Boucle

Le paramètre de requête node_count doit être défini sur 1 pour un cluster à nœud unique.

```
curl -iX POST -H 'Content-Type: application/json' -u admin:<password> -k
-d @step03 'https://10.21.191.150/api/clusters? node_count=1'
```

Entrée JSON (étape 03)

```
{
  "name": "my_cluster"
}
```

Type de traitement

Synchrone

Sortir

- ID de cluster dans l'en-tête de réponse d'emplacement

4. Configurer le cluster

Vous devez fournir plusieurs attributs dans le cadre de la configuration du cluster.

Catégorie	verbe HTTP	Chemin
Cluster	CORRECTIF	/clusters/{cluster_id}

Boucle

Vous devez fournir l'ID du cluster.

```
curl -iX PATCH -H 'Content-Type: application/json' -u admin:<password> -k
-d @step04 'https://10.21.191.150/api/clusters/CLUSTERID'
```

Entrée JSON (étape 04)

```
{
  "dns_info": {
    "domains": ["lab1.company-demo.com"],
    "dns_ips": ["10.206.80.135", "10.206.80.136"]
    },
    "ontap_image_version": "9.5",
    "gateway": "10.206.80.1",
    "ip": "10.206.80.115",
    "netmask": "255.255.255.192",
    "ntp_servers": {"10.206.80.183"}
}
```

Type de traitement

Synchrone

Sortir

Aucune

5. Récupérer le nom du nœud

L'utilitaire d'administration Deploy génère automatiquement les identifiants et les noms des nœuds lors de la création d'un cluster. Avant de configurer un nœud, vous devez récupérer l'identifiant attribué.

Catégorie	verbe HTTP	Chemin
Cluster	OBTENIR	/clusters/{cluster_id}/nodes

Boucle

Vous devez fournir l'ID du cluster.

```
curl -iX GET -u admin:<password> -k
'https://10.21.191.150/api/clusters/CLUSTERID/nodes?fields=id,name'
```

Type de traitement

Synchrone

Sortir

- Les enregistrements de tableau décrivent chacun un nœud unique avec l'ID et le nom uniques

6. Configurer les nœuds

Vous devez fournir la configuration de base du nœud, qui est le premier des trois appels d'API utilisés pour configurer un nœud.

Catégorie	verbe HTTP	Chemin
Cluster	CHEMIN	/clusters/{cluster_id}/nodes/{node_id}

Boucle

Vous devez fournir l'ID du cluster et l'ID du nœud.

```
curl -iX PATCH -H 'Content-Type: application/json' -u admin:<password> -k
-d @step06 'https://10.21.191.150/api/clusters/CLUSTERID/nodes/NODEID'
```

Entrée JSON (étape 06)

Vous devez fournir l'ID d'hôte sur lequel le nœud ONTAP Select s'exécutera.

```
{
  "host": {
    "id": "HOSTID"
    },
  "instance_type": "small",
  "ip": "10.206.80.101",
  "passthrough_disks": false
}
```

Type de traitement

Synchrone

Sortir

Aucune

7. Récupérer les réseaux de nœuds

Vous devez identifier les réseaux de données et de gestion utilisés par le nœud du cluster à nœud unique. Le réseau interne n'est pas utilisé avec un cluster à nœud unique.

Catégorie	verbe HTTP	Chemin
Cluster	OBTENIR	/clusters/{cluster_id}/nodes/{node_id}/networks

Boucle

Vous devez fournir l'ID du cluster et l'ID du nœud.

```
curl -iX GET -u admin:<password> -k 'https://10.21.191.150/api/
clusters/CLUSTERID/nodes/NODEID/networks?fields=id,purpose'
```

Type de traitement

Synchrone

Sortir

- Tableau de deux enregistrements décrivant chacun un réseau unique pour le nœud, y compris l'ID unique et l'objectif

8. Configurer la mise en réseau des nœuds

Vous devez configurer les réseaux de données et de gestion. Le réseau interne n'est pas utilisé avec un cluster à nœud unique.

Émettez l'appel API suivant deux fois, une fois pour chaque réseau.

Catégorie	verbe HTTP	Chemin
Cluster	CORRECTIF	/clusters/{cluster_id}/nodes/{node_id}/networks/{network_id}

Boucle

Vous devez fournir l'ID du cluster, l'ID du nœud et l'ID du réseau.

```
curl -iX PATCH -H 'Content-Type: application/json' -u admin:<password> -k
-d @step08 'https://10.21.191.150/api/clusters/
CLUSTERID/nodes/NODEID/networks/NETWORKID'
```

Entrée JSON (étape 08)

Vous devez fournir le nom du réseau.

```
{
  "name": "sDOT_Network"
}
```

Type de traitement

Synchrone

Sortir

Aucune

9. Configurer le pool de stockage de nœuds

La dernière étape de la configuration d'un nœud consiste à y associer un pool de stockage. Vous pouvez déterminer les pools de stockage disponibles via le client web vSphere ou, en option, via l'API REST Deploy.

Catégorie	verbe HTTP	Chemin
Cluster	CORRECTIF	/clusters/{cluster_id}/nodes/{node_id}/networks/{network_id}

Boucle

Vous devez fournir l'ID du cluster, l'ID du nœud et l'ID du réseau.

```
curl -iX PATCH -H 'Content-Type: application/json' -u admin:<password> -k
-d @step09 'https://10.21.191.150/api/clusters/ CLUSTERID/nodes/NODEID'
```

Entrée JSON (étape 09)

La capacité du pool est de 2 To.

```
{
  "pool_array": [
    {
      "name": "sDOT-01",
      "capacity": 2147483648000
    }
  ]
}
```

Type de traitement

Synchrone

Sortir

Aucune

10. Déployer le cluster

Une fois le cluster et le nœud configurés, vous pouvez déployer le cluster.

Catégorie	verbe HTTP	Chemin
Cluster	POSTE	/clusters/{cluster_id}/deploy

Boucle

Vous devez fournir l'ID du cluster.

```
curl -iX POST -H 'Content-Type: application/json' -u admin:<password> -k
-d @step10 'https://10.21.191.150/api/clusters/CLUSTERID/deploy'
```

Entrée JSON (étape 10)

Vous devez fournir le mot de passe du compte administrateur ONTAP .

```
{
  "ontap_credentials": {
    "password": "mypassword"
  }
}
```

Type de traitement

Asynchrone

Sortir

- Objet de travail

Informations connexes

"Déployer une instance d'évaluation de 90 jours d'un cluster ONTAP Select"

Accès avec Python

Avant d'accéder à l'API ONTAP Select Deploy à l'aide de Python

Vous devez préparer l'environnement avant d'exécuter les exemples de scripts Python.

Avant d'exécuter les scripts Python, vous devez vous assurer que l'environnement est correctement configuré :

- La dernière version applicable de Python 2 doit être installée. Les exemples de code ont été testés avec Python 2. Ils devraient également être portables vers Python 3, mais leur compatibilité n'a pas été testée.
- Les bibliothèques Requests et urllib3 doivent être installées. Vous pouvez utiliser pip ou un autre outil de gestion Python adapté à votre environnement.
- Le poste de travail client sur lequel les scripts s'exécutent doit avoir accès au réseau à la machine virtuelle ONTAP Select Deploy.

De plus, vous devez disposer des informations suivantes :

- Adresse IP de la machine virtuelle Deploy
- Nom d'utilisateur et mot de passe d'un compte administrateur de déploiement

Comprendre les scripts Python pour ONTAP Select Deploy

Les exemples de scripts Python vous permettent d'effectuer plusieurs tâches différentes. Il est important de bien comprendre ces scripts avant de les utiliser dans une instance Deploy en direct.

Caractéristiques de conception communes

Les scripts ont été conçus avec les caractéristiques communes suivantes :

- Exécuter depuis l'interface de ligne de commande sur un ordinateur client. Vous pouvez exécuter les scripts Python depuis n'importe quel ordinateur client correctement configuré. Consultez *Avant de commencer* pour plus d'informations.
- Accepter les paramètres d'entrée CLI Chaque script est contrôlé au niveau de la CLI via des paramètres d'entrée.
- Lire le fichier d'entrée Chaque script lit un fichier d'entrée en fonction de son objectif. Lors de la création ou de la suppression d'un cluster, vous devez fournir un fichier de configuration JSON. Lors de l'ajout d'une licence de nœud, vous devez fournir un fichier de licence valide.
- Utiliser un module de support commun. Le module de support commun *deploy_requests.py* contient une seule classe. Il est importé et utilisé par chacun des scripts.

Créer un cluster

Vous pouvez créer un cluster ONTAP Select à l'aide du script cluster.py. En fonction des paramètres de l'interface de ligne de commande et du contenu du fichier d'entrée JSON, vous pouvez adapter le script à votre environnement de déploiement comme suit :

- Hyperviseur : vous pouvez déployer sur ESXI ou KVM (selon la version de Deploy). Lors d'un déploiement sur ESXi, l'hyperviseur peut être géré par vCenter ou être un hôte autonome.
- Taille du cluster Vous pouvez déployer un cluster à nœud unique ou à nœuds multiples.
- Licence d'évaluation ou de production Vous pouvez déployer un cluster avec une licence d'évaluation ou achetée pour la production.

Les paramètres d'entrée CLI pour le script incluent :

- Nom d'hôte ou adresse IP du serveur de déploiement
- Mot de passe pour le compte utilisateur administrateur
- Nom du fichier de configuration JSON
- Indicateur détaillé pour la sortie du message

Ajouter une licence de nœud

Si vous choisissez de déployer un cluster de production, vous devez ajouter une licence pour chaque nœud à l'aide du script *add_license.py*. Vous pouvez ajouter la licence avant ou après le déploiement du cluster.

Les paramètres d'entrée CLI pour le script incluent :

- Nom d'hôte ou adresse IP du serveur de déploiement
- Mot de passe pour le compte utilisateur administrateur
- Nom du fichier de licence
- Nom d'utilisateur ONTAP avec privilèges pour ajouter la licence
- Mot de passe pour l'utilisateur ONTAP

Supprimer un cluster

Vous pouvez supprimer un cluster ONTAP Select existant à l'aide du script *delete_cluster.py*.

Les paramètres d'entrée CLI pour le script incluent :

- Nom d'hôte ou adresse IP du serveur de déploiement
- Mot de passe pour le compte utilisateur administrateur
- Nom du fichier de configuration JSON

Exemples de code Python

Script pour créer un cluster ONTAP Select

Vous pouvez utiliser le script suivant pour créer un cluster basé sur des paramètres définis dans le script et un fichier d'entrée JSON.

```
#!/usr/bin/env python
##-----------------------------------------------------------------------
#
# File: cluster.py
#
# (C) Copyright 2019 NetApp, Inc.
#
# This sample code is provided AS IS, with no support or warranties of
# any kind, including but not limited for warranties of merchantability
# or fitness of any kind, expressed or implied. Permission to use,
# reproduce, modify and create derivatives of the sample code is granted
# solely for the purpose of researching, designing, developing and
# testing a software application product for use with NetApp products,
# provided that the above copyright notice appears in all copies and
# that the software application product is distributed pursuant to terms
# no less restrictive than those set forth herein.
#
##-----------------------------------------------------------------------

import traceback
import argparse
```

```
import json
import logging

from deploy_requests import DeployRequests


def add_vcenter_credentials(deploy, config):
    """ Add credentials for the vcenter if present in the config """
    log_debug_trace()

    vcenter = config.get('vcenter', None)
    if vcenter and not deploy.resource_exists('/security/credentials',
                                              'hostname', vcenter[
'hostname']):
        log_info("Registering vcenter {} credentials".format(vcenter[
'hostname']))
        data = {k: vcenter[k] for k in ['hostname', 'username', 'password
']}
        data['type'] = "vcenter"
        deploy.post('/security/credentials', data)


def add_standalone_host_credentials(deploy, config):
    """ Add credentials for standalone hosts if present in the config.
        Does nothing if the host credential already exists on the Deploy.
    """
    log_debug_trace()

    hosts = config.get('hosts', [])
    for host in hosts:
        # The presense of the 'password' will be used only for standalone
hosts.
        # If this host is managed by a vcenter, it should not have a host
'password' in the json.
        if 'password' in host and not deploy.resource_exists(
'/security/credentials',
                                                             'hostname',
host['name']):
            log_info("Registering host {} credentials".format(host['name
']))
            data = {'hostname': host['name'], 'type': 'host',
                    'username': host['username'], 'password': host[
'password']}
            deploy.post('/security/credentials', data)
```

```
def register_unkown_hosts(deploy, config):
    ''' Registers all hosts with the deploy server.
        The host details are read from the cluster config json file.

        This method will skip any hosts that are already registered.
        This method will exit the script if no hosts are found in the
config.
    '''
    log_debug_trace()

    data = {"hosts": []}
    if 'hosts' not in config or not config['hosts']:
        log_and_exit("The cluster config requires at least 1 entry in the
'hosts' list got {}".format(config))

    missing_host_cnt = 0
    for host in config['hosts']:
        if not deploy.resource_exists('/hosts', 'name', host['name']):
            missing_host_cnt += 1
            host_config = {"name": host['name'], "hypervisor_type": host[
'type']}
            if 'mgmt_server' in host:
                host_config["management_server"] = host['mgmt_server']
                log_info(
                   "Registering from vcenter {mgmt_server}".format(**
host))

            if 'password' in host and 'user' in host:
                host_config['credential'] = {
                    "password": host['password'], "username": host['user
']}

            log_info("Registering {type} host {name}".format(**host))
            data["hosts"].append(host_config)

    # only post /hosts if some missing hosts were found
    if missing_host_cnt:
        deploy.post('/hosts', data, wait_for_job=True)


def add_cluster_attributes(deploy, config):
    ''' POST a new cluster with all needed attribute values.
        Returns the cluster_id of the new config
    '''
    log_debug_trace()
```

```
    cluster_config = config['cluster']
    cluster_id = deploy.find_resource('/clusters', 'name', cluster_config
['name'])

    if not cluster_id:
        log_info("Creating cluster config named {name}".format(
**cluster_config))

        # Filter to only the valid attributes, ignores anything else in
the json
        data = {k: cluster_config[k] for k in [
            'name', 'ip', 'gateway', 'netmask', 'ontap_image_version',
'dns_info', 'ntp_servers']}

        num_nodes = len(config['nodes'])

        log_info("Cluster properties: {}".format(data))

        resp = deploy.post('/v3/clusters?node_count={}'.format(num_nodes),
data)
        cluster_id = resp.headers.get('Location').split('/')[-1]

    return cluster_id


def get_node_ids(deploy, cluster_id):
    ''' Get the the ids of the nodes in a cluster. Returns a list of
node_ids.'''
    log_debug_trace()

    response = deploy.get('/clusters/{}/nodes'.format(cluster_id))
    node_ids = [node['id'] for node in response.json().get('records')]
    return node_ids


def add_node_attributes(deploy, cluster_id, node_id, node):
    ''' Set all the needed properties on a node '''
    log_debug_trace()

    log_info("Adding node '{}' properties".format(node_id))

    data = {k: node[k] for k in ['ip', 'serial_number', 'instance_type',
                                 'is_storage_efficiency_enabled'] if k in
node}
    # Optional: Set a serial_number
    if 'license' in node:
        data['license'] = {'id': node['license']}
```

```
    # Assign the host
    host_id = deploy.find_resource('/hosts', 'name', node['host_name'])
    if not host_id:
        log_and_exit("Host names must match in the 'hosts' array, and the
nodes.host_name property")

    data['host'] = {'id': host_id}

    # Set the correct raid_type
    is_hw_raid = not node['storage'].get('disks')  # The presence of a
list of disks indicates sw_raid
    data['passthrough_disks'] = not is_hw_raid

    # Optionally set a custom node name
    if 'name' in node:
        data['name'] = node['name']

    log_info("Node properties: {}".format(data))
    deploy.patch('/clusters/{}/nodes/{}'.format(cluster_id, node_id),
data)


def add_node_networks(deploy, cluster_id, node_id, node):
    ''' Set the network information for a node '''
    log_debug_trace()

    log_info("Adding node '{}' network properties".format(node_id))

    num_nodes = deploy.get_num_records('/clusters/{}/nodes'.format
(cluster_id))

    for network in node['networks']:

        # single node clusters do not use the 'internal' network
        if num_nodes == 1 and network['purpose'] == 'internal':
            continue

        # Deduce the network id given the purpose for each entry
        network_id = deploy.find_resource('/clusters/{}/nodes/{}/networks
'.format(cluster_id, node_id),
                                          'purpose', network['purpose'])
        data = {"name": network['name']}
        if 'vlan' in network and network['vlan']:
            data['vlan_id'] = network['vlan']

        deploy.patch('/clusters/{}/nodes/{}/networks/{}'.format(
```

```
cluster_id, node_id, network_id), data)


def add_node_storage(deploy, cluster_id, node_id, node):
    ''' Set all the storage information on a node '''
    log_debug_trace()

    log_info("Adding node '{}' storage properties".format(node_id))
    log_info("Node storage: {}".format(node['storage']['pools']))

    data = {'pool_array': node['storage']['pools']}  # use all the json
properties
    deploy.post(
        '/clusters/{}/nodes/{}/storage/pools'.format(cluster_id, node_id),
data)

    if 'disks' in node['storage'] and node['storage']['disks']:
        data = {'disks': node['storage']['disks']}
        deploy.post(
            '/clusters/{}/nodes/{}/storage/disks'.format(cluster_id,
node_id), data)


def create_cluster_config(deploy, config):
    ''' Construct a cluster config in the deploy server using the input
json data '''
    log_debug_trace()

    cluster_id = add_cluster_attributes(deploy, config)

    node_ids = get_node_ids(deploy, cluster_id)
    node_configs = config['nodes']

    for node_id, node_config in zip(node_ids, node_configs):
        add_node_attributes(deploy, cluster_id, node_id, node_config)
        add_node_networks(deploy, cluster_id, node_id, node_config)
        add_node_storage(deploy, cluster_id, node_id, node_config)

    return cluster_id


def deploy_cluster(deploy, cluster_id, config):
    ''' Deploy the cluster config to create the ONTAP Select VMs. '''
    log_debug_trace()
    log_info("Deploying cluster: {}".format(cluster_id))

    data = {'ontap_credential': {'password': config['cluster'][
```

```
'ontap_admin_password']}}
    deploy.post('/clusters/{}/deploy?inhibit_rollback=true'.format
(cluster_id),
                data, wait_for_job=True)


def log_debug_trace():
    stack = traceback.extract_stack()
    parent_function = stack[-2][2]
    logging.getLogger('deploy').debug('Calling %s()' % parent_function)


def log_info(msg):
    logging.getLogger('deploy').info(msg)


def log_and_exit(msg):
    logging.getLogger('deploy').error(msg)
    exit(1)


def configure_logging(verbose):
    FORMAT = '%(asctime)-15s:%(levelname)s:%(name)s: %(message)s'
    if verbose:
        logging.basicConfig(level=logging.DEBUG, format=FORMAT)
    else:
        logging.basicConfig(level=logging.INFO, format=FORMAT)
        logging.getLogger('requests.packages.urllib3.connectionpool'
).setLevel(
            logging.WARNING)


def main(args):
    configure_logging(args.verbose)
    deploy = DeployRequests(args.deploy, args.password)

    with open(args.config_file) as json_data:
        config = json.load(json_data)

        add_vcenter_credentials(deploy, config)

        add_standalone_host_credentials(deploy, config)

        register_unkown_hosts(deploy, config)

        cluster_id = create_cluster_config(deploy, config)
```

```
        deploy_cluster(deploy, cluster_id, config)


def parseArgs():
    parser = argparse.ArgumentParser(description='Uses the ONTAP Select
Deploy API to construct and deploy a cluster.')
    parser.add_argument('-d', '--deploy', help='Hostname or IP address of
Deploy server')
    parser.add_argument('-p', '--password', help='Admin password of Deploy
server')
    parser.add_argument('-c', '--config_file', help='Filename of the
cluster config')
    parser.add_argument('-v', '--verbose', help='Display extra debugging
messages for seeing exact API calls and responses',
                        action='store_true', default=False)
    return parser.parse_args()

if __name__ == '__main__':
    args = parseArgs()
    main(args)
```

JSON pour le script permettant de créer un cluster ONTAP Select

Lors de la création ou de la suppression d'un cluster ONTAP Select à l'aide des exemples de code Python, vous devez fournir un fichier JSON en entrée du script. Vous pouvez copier et modifier l'exemple JSON approprié en fonction de vos plans de déploiement.

Cluster à nœud unique sur ESXi

```
{
  "hosts": [
    {
      "password": "mypassword1",
      "name": "host-1234",
      "type": "ESX",
      "username": "admin"
    }
  ],

  "cluster": {
    "dns_info": {
      "domains": ["lab1.company-demo.com", "lab2.company-demo.com",
        "lab3.company-demo.com", "lab4.company-demo.com"
        ],
```

```
    "dns_ips": ["10.206.80.135", "10.206.80.136"]
    },
    "ontap_image_version": "9.7",
    "gateway": "10.206.80.1",
    "ip": "10.206.80.115",
    "name": "mycluster",
    "ntp_servers": ["10.206.80.183", "10.206.80.142"],
    "ontap_admin_password": "mypassword2",
    "netmask": "255.255.254.0"
},

"nodes": [
  {
    "serial_number": "3200000nn",
    "ip": "10.206.80.114",
    "name": "node-1",
    "networks": [
      {
        "name": "ontap-external",
        "purpose": "mgmt",
        "vlan": 1234
      },
      {
        "name": "ontap-external",
        "purpose": "data",
        "vlan": null
      },
      {
        "name": "ontap-internal",
        "purpose": "internal",
        "vlan": null
      }
    ],
    "host_name": "host-1234",
    "is_storage_efficiency_enabled": false,
    "instance_type": "small",
    "storage": {
      "disk": [],
      "pools": [
        {
          "name": "storage-pool-1",
          "capacity": 4802666790125
        }
      ]
    }
```

```
    }
  ]
}
```

Cluster à nœud unique sur ESXi utilisant vCenter

```
{
  "hosts": [
    {
      "name":"host-1234",
      "type":"ESX",
      "mgmt_server":"vcenter-1234"
    }
  ],

  "cluster": {
    "dns_info": {"domains": ["lab1.company-demo.com", "lab2.company-
demo.com",
      "lab3.company-demo.com", "lab4.company-demo.com"
      ],
      "dns_ips": ["10.206.80.135","10.206.80.136"]
  },

  "ontap_image_version":"9.7",
  "gateway":"10.206.80.1",
  "ip":"10.206.80.115",
  "name":"mycluster",
  "ntp_servers": ["10.206.80.183","10.206.80.142"],
  "ontap_admin_password":"mypassword2",
  "netmask":"255.255.254.0"
  },

  "vcenter": {
    "password":"mypassword2",
    "hostname":"vcenter-1234",
    "username":"selectadmin"
  },

  "nodes": [
    {
      "serial_number": "3200000nn",
      "ip":"10.206.80.114",
      "name":"node-1",
      "networks": [
        {
```

```
            "name":"ONTAP-Management",
            "purpose":"mgmt",
            "vlan":null
          },
          {
            "name": "ONTAP-External",
            "purpose":"data",
            "vlan":null
          },
          {
            "name": "ONTAP-Internal",
            "purpose":"internal",
            "vlan":null
          }
        ],

        "host_name": "host-1234",
        "is_storage_efficiency_enabled": false,
        "instance_type": "small",
        "storage": {
          "disk":[],
          "pools": [
            {
              "name": "storage-pool-1",
              "capacity":5685190380748
            }
          ]
        }
      }
    ]
}
```

Cluster à nœud unique sur KVM

```
{
  "hosts": [
    {
      "password": "mypassword1",
      "name":"host-1234",
      "type":"KVM",
      "username":"root"
    }
  ],

  "cluster": {
```

```
    "dns_info": {
      "domains": ["lab1.company-demo.com", "lab2.company-demo.com",
        "lab3.company-demo.com", "lab4.company-demo.com"
      ],

      "dns_ips": ["10.206.80.135", "10.206.80.136"]
    },

    "ontap_image_version": "9.7",
    "gateway":"10.206.80.1",
    "ip":"10.206.80.115",
    "name":"CBF4ED97",
    "ntp_servers": ["10.206.80.183", "10.206.80.142"],
    "ontap_admin_password": "mypassword2",
    "netmask":"255.255.254.0"
  },
  "nodes": [
    {
      "serial_number":"3200000nn",
      "ip":"10.206.80.115",
      "name": "node-1",
      "networks": [
        {
          "name": "ontap-external",
          "purpose": "mgmt",
          "vlan":1234
        },
        {
          "name": "ontap-external",
          "purpose": "data",
          "vlan": null
        },
        {
          "name": "ontap-internal",
          "purpose": "internal",
          "vlan": null
        }
      ],

      "host_name": "host-1234",
      "is_storage_efficiency_enabled": false,
      "instance_type": "small",
      "storage": {
        "disk": [],
        "pools": [
          {
```

```
            "name": "storage-pool-1",
            "capacity": 4802666790125
          }
        ]
      }
    }
  ]
}
```

Script pour ajouter une licence de nœud ONTAP Select

Vous pouvez utiliser le script suivant pour ajouter une licence pour un nœud ONTAP Select .

```
#!/usr/bin/env python
##-----------------------------------------------------------------------
#
# File: add_license.py
#
# (C) Copyright 2019 NetApp, Inc.
#
# This sample code is provided AS IS, with no support or warranties of
# any kind, including but not limited for warranties of merchantability
# or fitness of any kind, expressed or implied. Permission to use,
# reproduce, modify and create derivatives of the sample code is granted
# solely for the purpose of researching, designing, developing and
# testing a software application product for use with NetApp products,
# provided that the above copyright notice appears in all copies and
# that the software application product is distributed pursuant to terms
# no less restrictive than those set forth herein.
#
##-----------------------------------------------------------------------

import argparse
import logging
import json

from deploy_requests import DeployRequests


def post_new_license(deploy, license_filename):
    log_info('Posting a new license: {}'.format(license_filename))

    # Stream the file as multipart/form-data
    deploy.post('/licensing/licenses', data={},
```

```
                files={'license_file': open(license_filename, 'rb')})

    # Alternative if the NLF license data is converted to a string.
    # with open(license_filename, 'rb') as f:
    #    nlf_data = f.read()
    #    r = deploy.post('/licensing/licenses', data={},
    #                    files={'license_file': (license_filename,
nlf_data)})


def put_license(deploy, serial_number, data, files):
    log_info('Adding license for serial number: {}'.format(serial_number))

    deploy.put('/licensing/licenses/{}'.format(serial_number), data=data,
files=files)


def put_used_license(deploy, serial_number, license_filename,
ontap_username, ontap_password):
    ''' If the license is used by an 'online' cluster, a username/password
must be given. '''

    data = {'ontap_username': ontap_username, 'ontap_password':
ontap_password}
    files = {'license_file': open(license_filename, 'rb')}

    put_license(deploy, serial_number, data, files)


def put_free_license(deploy, serial_number, license_filename):
    data = {}
    files = {'license_file': open(license_filename, 'rb')}

    put_license(deploy, serial_number, data, files)


def get_serial_number_from_license(license_filename):
    ''' Read the NLF file to extract the serial number '''
    with open(license_filename) as f:
        data = json.load(f)

        statusResp = data.get('statusResp', {})
        serialNumber = statusResp.get('serialNumber')
        if not serialNumber:
            log_and_exit("The license file seems to be missing the
serialNumber")
```

```
        return serialNumber


def log_info(msg):
    logging.getLogger('deploy').info(msg)


def log_and_exit(msg):
    logging.getLogger('deploy').error(msg)
    exit(1)


def configure_logging():
    FORMAT = '%(asctime)-15s:%(levelname)s:%(name)s: %(message)s'
    logging.basicConfig(level=logging.INFO, format=FORMAT)
    logging.getLogger('requests.packages.urllib3.connectionpool').
setLevel(logging.WARNING)


def main(args):
    configure_logging()
    serial_number = get_serial_number_from_license(args.license)

    deploy = DeployRequests(args.deploy, args.password)

    # First check if there is already a license resource for this serial-
number
    if deploy.find_resource('/licensing/licenses', 'id', serial_number):

        # If the license already exists in the Deploy server, determine if
its used
        if deploy.find_resource('/clusters', 'nodes.serial_number',
serial_number):

            # In this case, requires ONTAP creds to push the license to
the node
            if args.ontap_username and args.ontap_password:
                put_used_license(deploy, serial_number, args.license,
                                 args.ontap_username, args.ontap_password)
            else:
                print("ERROR: The serial number for this license is in
use. Please provide ONTAP credentials.")
        else:
            # License exists, but its not used
            put_free_license(deploy, serial_number, args.license)
    else:
        # No license exists, so register a new one as an available license
```

```
for later use
        post_new_license(deploy, args.license)


def parseArgs():
    parser = argparse.ArgumentParser(description='Uses the ONTAP Select
Deploy API to add or update a new or used NLF license file.')
    parser.add_argument('-d', '--deploy', required=True, type=str, help=
'Hostname or IP address of ONTAP Select Deploy')
    parser.add_argument('-p', '--password', required=True, type=str, help
='Admin password of Deploy server')
    parser.add_argument('-l', '--license', required=True, type=str, help=
'Filename of the NLF license data')
    parser.add_argument('-u', '--ontap_username', type=str,
                        help='ONTAP Select username with privelege to add
the license. Only provide if the license is used by a Node.')
    parser.add_argument('-o', '--ontap_password', type=str,
                        help='ONTAP Select password for the
ontap_username. Required only if ontap_username is given.')
    return parser.parse_args()

if __name__ == '__main__':
    args = parseArgs()
    main(args)
```

Script pour supprimer un cluster ONTAP Select

Vous pouvez utiliser le script CLI suivant pour supprimer un cluster existant.

```
#!/usr/bin/env python
##-----------------------------------------------------------------------
#
# File: delete_cluster.py
#
# (C) Copyright 2019 NetApp, Inc.
#
# This sample code is provided AS IS, with no support or warranties of
# any kind, including but not limited for warranties of merchantability
# or fitness of any kind, expressed or implied. Permission to use,
# reproduce, modify and create derivatives of the sample code is granted
# solely for the purpose of researching, designing, developing and
# testing a software application product for use with NetApp products,
# provided that the above copyright notice appears in all copies and
# that the software application product is distributed pursuant to terms
# no less restrictive than those set forth herein.
```

```
#
##-------------------------------------------------------------------

import argparse
import json
import logging

from deploy_requests import DeployRequests

def find_cluster(deploy, cluster_name):
    return deploy.find_resource('/clusters', 'name', cluster_name)


def offline_cluster(deploy, cluster_id):
    # Test that the cluster is online, otherwise do nothing
    response = deploy.get('/clusters/{}?fields=state'.format(cluster_id))
    cluster_data = response.json()['record']
    if cluster_data['state'] == 'powered_on':
        log_info("Found the cluster to be online, modifying it to be
powered_off.")
        deploy.patch('/clusters/{}'.format(cluster_id), {'availability':
'powered_off'}, True)


def delete_cluster(deploy, cluster_id):
    log_info("Deleting the cluster({}).".format(cluster_id))
    deploy.delete('/clusters/{}'.format(cluster_id), True)
    pass


def log_info(msg):
    logging.getLogger('deploy').info(msg)


def configure_logging():
    FORMAT = '%(asctime)-15s:%(levelname)s:%(name)s: %(message)s'
    logging.basicConfig(level=logging.INFO, format=FORMAT)
    logging.getLogger('requests.packages.urllib3.connectionpool').
setLevel(logging.WARNING)


def main(args):
    configure_logging()
    deploy = DeployRequests(args.deploy, args.password)

    with open(args.config_file) as json_data:
        config = json.load(json_data)
```

```
        cluster_id = find_cluster(deploy, config['cluster']['name'])

        log_info("Found the cluster {} with id: {}.".format(config[
'cluster']['name'], cluster_id))

        offline_cluster(deploy, cluster_id)

        delete_cluster(deploy, cluster_id)


def parseArgs():
    parser = argparse.ArgumentParser(description='Uses the ONTAP Select
Deploy API to delete a cluster')
    parser.add_argument('-d', '--deploy', required=True, type=str, help=
'Hostname or IP address of Deploy server')
    parser.add_argument('-p', '--password', required=True, type=str, help
='Admin password of Deploy server')
    parser.add_argument('-c', '--config_file', required=True, type=str,
help='Filename of the cluster json config')
    return parser.parse_args()

if __name__ == '__main__':
    args = parseArgs()
    main(args)
```

Module Python de support commun pour ONTAP Select

Tous les scripts Python utilisent une classe Python commune dans un seul module.

```
#!/usr/bin/env python
##-----------------------------------------------------------------------
#
# File: deploy_requests.py
#
# (C) Copyright 2019 NetApp, Inc.
#
# This sample code is provided AS IS, with no support or warranties of
# any kind, including but not limited for warranties of merchantability
# or fitness of any kind, expressed or implied. Permission to use,
# reproduce, modify and create derivatives of the sample code is granted
# solely for the purpose of researching, designing, developing and
# testing a software application product for use with NetApp products,
# provided that the above copyright notice appears in all copies and
# that the software application product is distributed pursuant to terms
```

```
# no less restrictive than those set forth herein.
#
##------------------------------------------------------------------------

import json
import logging
import requests

requests.packages.urllib3.disable_warnings()

class DeployRequests(object):
    '''
    Wrapper class for requests that simplifies the ONTAP Select Deploy
    path creation and header manipulations for simpler code.
    '''

    def __init__(self, ip, admin_password):
        self.base_url = 'https://{}/api'.format(ip)
        self.auth = ('admin', admin_password)
        self.headers = {'Accept': 'application/json'}
        self.logger = logging.getLogger('deploy')

    def post(self, path, data, files=None, wait_for_job=False):
        if files:
            self.logger.debug('POST FILES:')
            response = requests.post(self.base_url + path,
                                     auth=self.auth, verify=False,
                                     files=files)
        else:
            self.logger.debug('POST DATA: %s', data)
            response = requests.post(self.base_url + path,
                                     auth=self.auth, verify=False,
                                     json=data,
                                     headers=self.headers)

        self.logger.debug('HEADERS: %s\nBODY: %s', self.filter_headers
(response), response.text)
        self.exit_on_errors(response)

        if wait_for_job and response.status_code == 202:
            self.wait_for_job(response.json())
        return response

    def patch(self, path, data, wait_for_job=False):
        self.logger.debug('PATCH DATA: %s', data)
        response = requests.patch(self.base_url + path,
```

```
                                    auth=self.auth, verify=False,
                                    json=data,
                                    headers=self.headers)
        self.logger.debug('HEADERS: %s\nBODY: %s', self.filter_headers
(response), response.text)
        self.exit_on_errors(response)

        if wait_for_job and response.status_code == 202:
            self.wait_for_job(response.json())
        return response

    def put(self, path, data, files=None, wait_for_job=False):
        if files:
            print('PUT FILES: {}'.format(data))
            response = requests.put(self.base_url + path,
                                    auth=self.auth, verify=False,
                                    data=data,
                                    files=files)
        else:
            self.logger.debug('PUT DATA:')
            response = requests.put(self.base_url + path,
                                    auth=self.auth, verify=False,
                                    json=data,
                                    headers=self.headers)

        self.logger.debug('HEADERS: %s\nBODY: %s', self.filter_headers
(response), response.text)
        self.exit_on_errors(response)

        if wait_for_job and response.status_code == 202:
            self.wait_for_job(response.json())
        return response

    def get(self, path):
        """ Get a resource object from the specified path """
        response = requests.get(self.base_url + path, auth=self.auth,
verify=False)
        self.logger.debug('HEADERS: %s\nBODY: %s', self.filter_headers
(response), response.text)
        self.exit_on_errors(response)
        return response

    def delete(self, path, wait_for_job=False):
        """ Delete's a resource from the specified path """
        response = requests.delete(self.base_url + path, auth=self.auth,
verify=False)
```

```
        self.logger.debug('HEADERS: %s\nBODY: %s', self.filter_headers
(response), response.text)
        self.exit_on_errors(response)

        if wait_for_job and response.status_code == 202:
            self.wait_for_job(response.json())
        return response

    def find_resource(self, path, name, value):
        ''' Returns the 'id' of the resource if it exists, otherwise None
'''
        resource = None
        response = self.get('{path}?{field}={value}'.format(
                            path=path, field=name, value=value))
        if response.status_code == 200 and response.json().get(
'num_records') >= 1:
            resource = response.json().get('records')[0].get('id')
        return resource

    def get_num_records(self, path, query=None):
        ''' Returns the number of records found in a container, or None on
error '''
        resource = None
        query_opt = '?{}'.format(query) if query else ''
        response = self.get('{path}{query}'.format(path=path, query
=query_opt))
        if response.status_code == 200 :
            return response.json().get('num_records')
        return None

    def resource_exists(self, path, name, value):
        return self.find_resource(path, name, value) is not None

    def wait_for_job(self, response, poll_timeout=120):
        last_modified = response['job']['last_modified']
        job_id = response['job']['id']

        self.logger.info('Event: ' + response['job']['message'])

        while True:
            response = self.get('/jobs/{}?fields=state,message&'
                                'poll_timeout={}&last_modified=>={}'
.format(
                                    job_id, poll_timeout, last_modified))

            job_body = response.json().get('record', {})
```

```
            # Show interesting message updates
            message = job_body.get('message', '')
            self.logger.info('Event: ' + message)

            # Refresh the last modified time for the poll loop
            last_modified = job_body.get('last_modified')

            # Look for the final states
            state = job_body.get('state', 'unknown')
            if state in ['success', 'failure']:
                if state == 'failure':
                    self.logger.error('FAILED background job.\nJOB: %s',
job_body)
                    exit(1)    # End the script if a failure occurs
                break

    def exit_on_errors(self, response):
        if response.status_code >= 400:
            self.logger.error('FAILED request to URL: %s\nHEADERS: %s
\nRESPONSE BODY: %s',
                              response.request.url,
                              self.filter_headers(response),
                              response.text)
        response.raise_for_status()    # Displays the response error, and
exits the script

    @staticmethod
    def filter_headers(response):
        ''' Returns a filtered set of the response headers '''
        return {key: response.headers[key] for key in ['Location',
'request-id'] if key in response.headers}
```

Script pour redimensionner les nœuds du cluster ONTAP Select

Vous pouvez utiliser le script suivant pour redimensionner les nœuds dans un cluster ONTAP Select .

```
#!/usr/bin/env python
##-----------------------------------------------------------------------
#
# File: resize_nodes.py
#
# (C) Copyright 2019 NetApp, Inc.
#
# This sample code is provided AS IS, with no support or warranties of
```

```
# any kind, including but not limited for warranties of merchantability
# or fitness of any kind, expressed or implied. Permission to use,
# reproduce, modify and create derivatives of the sample code is granted
# solely for the purpose of researching, designing, developing and
# testing a software application product for use with NetApp products,
# provided that the above copyright notice appears in all copies and
# that the software application product is distributed pursuant to terms
# no less restrictive than those set forth herein.
#
##------------------------------------------------------------------

import argparse
import logging
import sys

from deploy_requests import DeployRequests


def _parse_args():
    """ Parses the arguments provided on the command line when executing
this
        script and returns the resulting namespace. If all required
arguments
        are not provided, an error message indicating the mismatch is
printed and
        the script will exit.
    """

    parser = argparse.ArgumentParser(description=(
        'Uses the ONTAP Select Deploy API to resize the nodes in the
cluster.'
        ' For example, you might have a small (4 CPU, 16GB RAM per node) 2
node'
        ' cluster and wish to resize the cluster to medium (8 CPU, 64GB
RAM per'
        ' node). This script will take in the cluster details and then
perform'
        ' the operation and wait for it to complete.'
    ))
    parser.add_argument('--deploy', required=True, help=(
        'Hostname or IP of the ONTAP Select Deploy VM.'
    ))
    parser.add_argument('--deploy-password', required=True, help=(
        'The password for the ONTAP Select Deploy admin user.'
    ))
    parser.add_argument('--cluster', required=True, help=(
```

```
        'Hostname or IP of the cluster management interface.'
    ))
    parser.add_argument('--instance-type', required=True, help=(
        'The desired instance size of the nodes after the operation is
complete.'
    ))
    parser.add_argument('--ontap-password', required=True, help=(
        'The password for the ONTAP administrative user account.'
    ))
    parser.add_argument('--ontap-username', default='admin', help=(
        'The username for the ONTAP administrative user account. Default:
admin.'
    ))
    parser.add_argument('--nodes', nargs='+', metavar='NODE_NAME', help=(
        'A space separated list of node names for which the resize
operation'
        ' should be performed. The default is to apply the resize to all
nodes in'
        ' the cluster. If a list of nodes is provided, it must be provided
in HA'
        ' pairs. That is, in a 4 node cluster, nodes 1 and 2 (partners)
must be'
        ' resized in the same operation.'
    ))
    return parser.parse_args()


def _get_cluster(deploy, parsed_args):
    """ Locate the cluster using the arguments provided """

    cluster_id = deploy.find_resource('/clusters', 'ip', parsed_args
.cluster)
    if not cluster_id:
        return None
    return deploy.get('/clusters/%s?fields=nodes' % cluster_id).json()[
'record']


def _get_request_body(parsed_args, cluster):
    """ Build the request body """

    changes = {'admin_password': parsed_args.ontap_password}

    # if provided, use the list of nodes given, else use all the nodes in
the cluster
    nodes = [node for node in cluster['nodes']]
```

```
    if parsed_args.nodes:
        nodes = [node for node in nodes if node['name'] in parsed_args
.nodes]

    changes['nodes'] = [
        {'instance_type': parsed_args.instance_type, 'id': node['id']} for
node in nodes]

    return changes


def main():
    """ Set up the resize operation by gathering the necessary data and
then send
        the request to the ONTAP Select Deploy server.
    """

    logging.basicConfig(
        format='[%(asctime)s] [%(levelname)5s] %(message)s', level=
logging.INFO,)

    logging.getLogger('requests.packages.urllib3').setLevel(logging
.WARNING)

    parsed_args = _parse_args()
    deploy = DeployRequests(parsed_args.deploy, parsed_args
.deploy_password)

    cluster = _get_cluster(deploy, parsed_args)
    if not cluster:
        deploy.logger.error(
            'Unable to find a cluster with a management IP of %s' %
parsed_args.cluster)
        return 1

    changes = _get_request_body(parsed_args, cluster)
    deploy.patch('/clusters/%s' % cluster['id'], changes, wait_for_job
=True)

if __name__ == '__main__':
    sys.exit(main())
```

Informations sur le copyright

Informations sur les marques commerciales