# Automatisez avec LE REPOS

ONTAP Select

NetApp
April 01, 2025

# Sommaire

# Automatisez avec LE REPOS

# Concepts

## Base de services Web REST

Representational State Transfer (REST) est un style qui permet de créer des applications Web distribuées. Lorsqu'il est appliqué à la conception d'une API de services Web, il établit un ensemble de technologies et de meilleures pratiques pour l'exposition des ressources basées sur serveur et la gestion de leurs États. Il utilise des protocoles et des normes courants pour offrir une base flexible pour le déploiement et la gestion des clusters ONTAP Select.

### Contraintes classiques et architectures

REST a été formellement articulé par Roy Fielding dans son doctorat "thèse" À UC Irvine en 2000. Il définit un style architectural à travers un ensemble de contraintes, qui ont collectivement amélioré les applications basées sur le Web et les protocoles sous-jacents. Ces contraintes créent une application de services web RESTful basée sur une architecture client/serveur utilisant un protocole de communication sans état.

### Ressources et représentation d'état

Les ressources sont les composants de base d'un système basé sur le 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 les ressources. Une ressource peut être un fichier, une transaction commerciale, un processus ou une entité administrative. L'une des premières tâches de conception d'une application basée sur des services Web REST consiste à identifier les ressources.
- Définition des États de ressource et des opérations d'état associées
  Les ressources se trouvent toujours dans un des États finis. Les États, ainsi que les opérations associées utilisées pour affecter les changements d'état, doivent être clairement définis.

Les messages sont échangés entre le client et le serveur pour accéder aux ressources et les modifier selon le modèle CRUD générique (Créer, lire, mettre à jour et Supprimer).

### Terminaux URI

Chaque ressource REST doit être définie et mise à disposition à l'aide d'un schéma d'adressage bien défini. Les noeuds finaux où les ressources sont situées et identifiées utilisent un URI (Uniform Resource identifier). L'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 avec les services Web pour identifier et accéder aux ressources. Les ressources sont généralement exposées dans une structure hiérarchique similaire à un répertoire de fichiers.

### Messages HTTP

Le protocole HTTP (Hypertext Transfer Protocol) est le protocole utilisé par le client et le serveur de services Web pour échanger des messages de requête et de réponse sur les ressources. Dans le cadre de la conception d'une application de services Web, les verbes HTTP (COMME GET et POST) sont mappées aux ressources et aux actions de gestion d'état correspondantes.

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

#### Formatage JSON

Bien que les informations puissent être structurées et transférées de plusieurs façons entre un client et un serveur, l'option la plus populaire (et celle utilisée avec l'API REST de déploiement) est JavaScript Object notation (JSON). JSON est une norme de l'industrie qui représente les structures de données simples en texte brut et permet de transférer les informations d'état décrivant les ressources.

## Comment accéder à l'API de déploiement

En raison de la flexibilité inhérente des services web REST, l'API de déploiement ONTAP Select est accessible de plusieurs façons.

#### Déployez l'interface utilisateur native de l'utilitaire

La principale façon dont vous accédez à l'API consiste à utiliser l'interface utilisateur Web de ONTAP Select Deploy. Le navigateur fait des appels à l'API et reformate les données en fonction de la conception de l'interface utilisateur. Vous pouvez également accéder à l'API via l'interface de ligne de commande de l'utilitaire de déploiement.

#### Page de documentation en ligne sur le déploiement de ONTAP Select

La page de documentation en ligne de ONTAP Select Deploy fournit un point d'accès alternatif lorsque vous utilisez un navigateur. En plus de fournir un moyen d'exécuter directement des appels API individuels, la page comprend également une description détaillée de l'API, y compris les paramètres d'entrée et d'autres options pour chaque appel. Les appels API sont organisés en plusieurs zones fonctionnelles ou catégories différentes.

#### Programme personnalisé

Vous pouvez accéder à l'API de déploiement à l'aide de plusieurs langages et outils de programmation différents. Les options les plus populaires sont Python, Java et Curl. Un programme, un script ou un outil qui utilise l'API agit comme un client de services Web REST. L'utilisation d'un langage de programmation permet de mieux comprendre l'API et offre une possibilité d'automatiser les déploiements ONTAP Select.

## Déploiement des versions d'API

Un numéro de version est attribué à l'API REST incluse dans ONTAP Select Deploy. Le numéro de version de l'API est indépendant du numéro de version de déploiement. Il est important de connaître la version d'API incluse dans votre version de déploiement et d'identifier en quoi cela pourrait affecter votre utilisation de l'API.

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

**Déploiement 2.8 et versions ultérieures**

ONTAP Select Deploy 2.8, ainsi que que que toutes les versions ultérieures incluent la version 3 de l'API REST.

#### Déployer les versions 2.7.2 et antérieures

ONTAP Select Deploy 2.7.2. Toutes les versions antérieures comprennent 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 le déploiement de la version 2.8 ou ultérieure à partir d'une version antérieure incluant la version 2 de l'API, vous devez mettre à jour tout code existant qui accède directement à l'API ainsi que tous les scripts à l'aide de l'interface de ligne de commande.

## Caractéristiques opérationnelles de base

Alors QUE REST établit un ensemble commun de technologies et de meilleures pratiques, les détails de chaque API peuvent varier en fonction des choix de conception. Vous devez connaître les détails et les caractéristiques opérationnelles de l'API de déploiement ONTAP Select avant d'utiliser l'API.

#### Hôte de l'hyperviseur ou nœud ONTAP Select

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

#### Identifiants d'objets

Un identifiant unique est attribué à chaque instance de ressource ou objet lors de sa création. Ces identifiants sont globalement uniques dans une instance spécifique du déploiement ONTAP Select. Après l'émission d'un appel API qui crée une nouvelle instance d'objet, la valeur d'ID associée est renvoyée à l'appelant dans le `location` En-tête de la réponse HTTP. Vous pouvez extraire l'identificateur et l'utiliser sur les appels suivants lorsque vous faites référence à l'instance de ressource.

> Le contenu et la structure interne des identificateurs d'objet peuvent changer à tout moment. Vous ne devez utiliser les identificateurs sur les appels API applicables que si nécessaire lorsque vous faites référence aux objets associés.

#### Demander des identifiants

Un identifiant unique est attribué à chaque requête d'API réussie. L'identifiant est renvoyé dans le `request-id` En-tête de la réponse HTTP associée. Vous pouvez utiliser un identificateur de demande pour faire référence collectivement aux activités d'une seule transaction de réponse de requête API spécifique. Par exemple, vous pouvez récupérer tous les messages d'événement pour une transaction basée sur l'ID de demande

#### Appels synchrones et asynchrones

Il existe deux méthodes principales pour qu'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 demande et répond avec le code d'état 202. Cela indique que le serveur a accepté

la demande du client et a lancé une tâche en arrière-plan pour terminer la demande. Le succès ou l'échec final n'est pas immédiatement disponible et doit être déterminé par d'autres appels API.

#### Confirmez la fin d'une tâche en cours d'exécution

Généralement, toute opération qui peut prendre un certain temps est traitée de manière asynchrone à l'aide d'un
tâche en arrière-plan sur le serveur. Avec l'API REST de Deploy, chaque tâche en arrière-plan est ancrée dans un
Objet de travail qui suit la tâche et fournit des informations, telles que l'état actuel. Un objet travail,
Avec son identifiant unique, est renvoyé dans la réponse HTTP après la création d'une tâche en arrière-plan.

Vous pouvez interroger l'objet Job directement pour déterminer le succès ou l'échec de l'appel API associé. Pour plus d'informations, reportez-vous à la section *traitement asynchrone à l'aide de l'objet travail*.

Outre l'objet travail, vous pouvez déterminer la réussite ou l'échec d'un de plusieurs manières
demande, comprenant :

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

#### Sécurité

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

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

## Transaction d'API de demande et de réponse

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

#### Variables d'entrée contrôlant une requête API

Vous pouvez contrôler le traitement d'un appel API à l'aide de paramètres définis dans la requête HTTP.

#### En-têtes de demande

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

- type-contenu
  Si le corps de la demande inclut JSON, cet en-tête doit être défini sur application/json.
- accepter
  Si le corps de réponse inclut JSON, cet en-tête doit être défini sur application/json.
- autorisation
  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 demande

Le contenu du corps de la demande varie en fonction de l'appel spécifique. Le corps de requête HTTP comprend l'un des éléments suivants :

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

#### Filtrer les objets

Lors de l'émission d'un appel 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 associer :

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

En plus d'une correspondance exacte, d'autres opérateurs sont disponibles pour renvoyer un ensemble d'objets sur une plage de valeurs. ONTAP Select prend en charge les opérateurs de filtrage indiqués ci-dessous.

| Opérateur | Description |
|---|---|
| = | Égal à |
| < | Inférieur à |
| > | Supérieur à |
| &lt ;= | Inférieur ou égal à |
| >= | Supérieur ou égal à |
| | Ou |
| ! | Différent de |
| * | Un caractère générique gourmand |

Vous pouvez également renvoyer un ensemble d'objets en fonction de la définition ou non d'un champ spécifique à l'aide du mot clé null ou de sa négation (!null) dans le cadre de la requête.

#### Sélection des champs d'objet

Par défaut, l'émission d'un appel API à l'aide DE GET renvoie uniquement les attributs qui identifient de manière unique l'objet ou les objets. Cet ensemble minimal de champs sert de clé pour chaque objet et varie

en fonction du type d'objet. Vous pouvez sélectionner des propriétés d'objet supplémentaires à l'aide du paramètre de requête de champs de la manière suivante :

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

Vous devez toujours identifier les champs spécifiques que vous souhaitez. Vous ne devriez récupérer que l'ensemble de champs bon marché ou coûteux si nécessaire. Cette classification économique et onéreuse est déterminée par NetApp sur la base de l'analyse interne des performances. La classification d'un champ donné peut changer à tout moment.

#### Trier les objets dans le jeu de sortie

Les enregistrements d'une collection de ressources sont renvoyés dans l'ordre par défaut défini par l'objet. Vous pouvez modifier l'ordre à l'aide du paramètre de requête order_by avec le nom de champ et la direction de tri comme suit :
`order_by=<field name> asc|desc`

Par exemple, vous pouvez trier le champ de type par ordre décroissant, suivi d'un ID par ordre croissant :
`order_by=type desc, id asc`

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

#### Pagination

Lors de l'émission d'un appel API à l'aide DE GET pour accéder à une collection d'objets du même type, tous les objets correspondants sont renvoyés par défaut. Si nécessaire, vous pouvez limiter le nombre d'enregistrements renvoyés à l'aide du paramètre de requête max_records avec la demande. Par exemple :
`max_records=20`

Si nécessaire, vous pouvez combiner ce paramètre avec d'autres paramètres de requête pour affiner le jeu de résultats. Par exemple, ce qui suit renvoie jusqu'à 10 événements système générés après le temps spécifié :
`time⇒ 2019-04-04T15:41:29.140265Z&max_records=10`

Vous pouvez émettre plusieurs requêtes à la page via les événements (ou tout type d'objet). Chaque appel d'API suivant doit utiliser une nouvelle valeur de temps basée sur le dernier événement du dernier jeu de résultats.

### Interpréter une réponse API

Chaque requête d'API génère une réponse au client. Vous pouvez examiner la réponse pour déterminer s'il a réussi et qu'il récupère des données supplémentaires si nécessaire.

#### Code d'état HTTP

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

| Code | Signification | Description |
|---|---|---|
| 200 | OK | Indique que les appels qui ne créent pas d'objet ont réussi. |
| 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 en arrière-plan longue durée a été démarrée pour exécuter la demande, mais l'opération n'a pas encore été terminée. |
| 400 | Demande incorrecte | L'entrée de la demande n'est pas reconnue ou est inappropriée. |
| 403 | Interdit | L'accès est refusé en raison d'une erreur d'autorisation. |
| 404 | Introuvable | La ressource mentionnée dans la demande n'existe pas. |
| 405 | Méthode non autorisée | Le verbe HTTP de la demande n'est pas pris en charge pour la ressource. |
| 409 | Conflit | La tentative de création d'un objet a échoué car celui-ci existe déjà. |
| 500 | Erreur interne | Une erreur interne générale s'est produite sur le serveur. |
| 501 | Non mis en œuvre | L'URI est connu mais ne peut pas exécuter la demande. |

#### En-têtes de réponse

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

- id-demande
  Un identifiant de demande unique est attribué à chaque demande d'API réussie.
- emplacement
  Lorsqu'un objet est créé, l'en-tête d'emplacement inclut l'URL complète vers le nouvel objet, y compris l'identifiant unique de l'objet.

#### Corps de réponse

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

- Objet unique
  Un objet peut être renvoyé avec un ensemble de champs en fonction de la requête. Par exemple, vous pouvez utiliser OBTENIR pour extraire les propriétés sélectionnées d'un cluster à l'aide de l'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` indique le nombre d'enregistrements et d'enregistrements contenant un tableau des instances d'objet. Par exemple, vous pouvez extraire tous les nœuds définis dans un cluster spécifique.
- Objet travail
  Si un appel API est traité de manière asynchrone, un objet travail est renvoyé, qui ancres la tâche d'arrière-plan. Par exemple, la demande POST utilisée pour déployer un cluster est traitée de manière asynchrone et renvoie un objet Job.

- Objet erreur
  Si une erreur se produit, un objet erreur est toujours renvoyé. Par exemple, vous recevrez une erreur lors de la tentative de création d'un cluster dont le nom existe déjà.
- Vide
  Dans certains cas, aucune donnée n'est renvoyée et le corps de réponse est vide. Par exemple, le corps de réponse est vide après avoir utilisé SUPPRIMER pour supprimer un hôte existant.

## Traitement asynchrone à l'aide de l'objet de travail

Certains appels API de déploiement, en particulier ceux qui créent ou modifient une ressource, peuvent prendre plus de temps que d'autres appels. Le déploiement de ONTAP Select traite ces requêtes à long terme de manière asynchrone.

### Demandes asynchrones décrites à l'aide de l'objet travail

Après avoir effectué un appel API qui s'exécute de manière asynchrone, le code de réponse HTTP 202 indique que la demande a été validée et acceptée avec succès, mais pas encore terminée. La requête est traitée comme une tâche d'arrière-plan qui continue à s'exécuter après la réponse HTTP initiale au client. La réponse inclut l'objet Job qui fixe la requête, y compris son identifiant unique.

Reportez-vous à la page de documentation en ligne de ONTAP Select Deploy pour déterminer quels appels API fonctionnent de manière asynchrone.

### Interroger l'objet travail associé à une demande API

L'objet travail renvoyé dans la réponse HTTP contient plusieurs propriétés. Vous pouvez interroger la propriété d'état pour déterminer si la demande a bien été effectuée. Un objet travail peut être dans l'un des États suivants :

- En file d'attente
- Exécution
- Réussite
- Panne

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

- Demande d'interrogation standard
  L'état actuel du travail est immédiatement renvoyé
- Demande d'interrogation longue
  L'état du travail est renvoyé uniquement lorsque l'une des situations suivantes se produit :
  - L'état a changé plus récemment que la valeur date-heure fournie sur la demande d'interrogation
  - La valeur de temporisation a expiré (1 à 120 secondes)

Interrogation standard et interrogation longue utilisent le même appel API pour interroger un objet travail. Cependant, une demande d'interrogation 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 de déploiement.

#### Procédure générale d'émission d'une demande asynchrone

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

1. Lancez l'appel d'API asynchrone.
2. Recevoir une réponse HTTP 202 indiquant que la demande a été acceptée avec succès.
3. Extraire l'identifiant de l'objet travail du corps de réponse.
4. Dans une boucle, effectuez les opérations suivantes dans chaque cycle :
   a. Obtenir l'état actuel du travail avec une demande de sondage long
   b. Si le travail est dans un état autre que terminal (en file d'attente, en cours d'exécution), exécutez de nouveau la boucle.
5. Arrêter lorsque le travail atteint un état terminal (réussite, échec).

# Accès à l'aide d'un navigateur

## Avant d'accéder à l'API avec un navigateur

Vous devez connaître plusieurs éléments avant d'utiliser la page de documentation en ligne de déploiement.

#### Plan de déploiement

Si vous prévoyez d'émettre des appels d'API dans le cadre de l'exécution de tâches de déploiement ou d'administration spécifiques, vous devez envisager de créer un plan de déploiement. Ces plans peuvent être formels ou informels, et contiennent généralement vos buts et les appels API à utiliser. Consultez le document processus de workflow à l'aide de l'API REST de déploiement pour plus d'informations.

#### Exemples JSON et définitions de paramètres

Chaque appel d'API est décrit sur la page de documentation à l'aide d'un format cohérent. Le contenu inclut des notes d'implémentation, des paramètres de requête et des codes d'état HTTP. En outre, vous pouvez afficher les détails du fichier JSON utilisé avec les demandes et les réponses de l'API, comme suit :

- Exemple de valeur
  Si vous cliquez sur *exemple de valeur* sur un appel d'API, une structure JSON type pour l'appel s'affiche. Vous pouvez modifier l'exemple selon vos besoins et l'utiliser comme saisie pour votre demande.
- Modèle
  Si vous cliquez sur *Model*, 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 d'API que vous effectuez à l'aide de la page de documentation de déploiement sont des opérations en direct. Veillez à ne pas créer, mettre à jour ou supprimer une configuration ou d'autres données par erreur.

## Accédez à la page de documentation de déploiement

Vous devez accéder à la page de documentation en ligne ONTAP Select Deploy pour afficher la documentation de l'API et lancer manuellement un appel d'API.

**Avant de commencer**

Vous devez disposer des éléments suivants :

- L'adresse IP ou le nom de domaine du ONTAP Select déploient la machine virtuelle
- Nom d'utilisateur et mot de passe pour l'administrateur

**Étapes**

1. Saisissez l'URL dans votre navigateur et appuyez sur **entrée** :

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

2. Connectez-vous à l'aide du nom d'utilisateur et du mot de passe administrateur.

**Résultat**

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

## Comprendre et exécuter un appel d'API

Les détails de tous les appels API sont documentés et affichés dans un format commun sur la page Web de documentation en ligne de ONTAP Select Deploy. En comprenant un seul appel API, vous pouvez accéder aux détails de tous les appels API et les interpréter.

**Avant de commencer**

Vous devez vous connecter à la page Web de documentation en ligne de ONTAP Select Deploy. Vous devez disposer de l'identifiant unique attribué à votre cluster ONTAP Select au moment de sa création.

**Description de la tâche**

Vous pouvez récupérer les informations de configuration décrivant un cluster ONTAP Select à l'aide de son identifiant unique. Dans cet exemple, tous les champs classés comme peu coûteux sont retournés. Toutefois, conformément aux bonnes pratiques, vous devez uniquement demander les champs spécifiques nécessaires.

**Étapes**

1. Sur la page principale, faites défiler vers le bas et cliquez sur **Cluster**.
2. Cliquez sur **GET /cluster/{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 API

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

### Comprendre les appels d'API utilisés dans les workflows

La page de documentation en ligne de ONTAP Select comprend les détails de chaque appel d'API REST. Au lieu de répéter ces détails ici, chaque appel d'API utilisé dans les exemples de flux de travail comprend uniquement les informations dont vous avez besoin pour localiser l'appel sur la page de documentation. Après avoir localisé un appel API spécifique, vous pouvez vérifier les détails complets de l'appel, y compris les paramètres d'entrée, les formats de sortie, les codes d'état HTTP et le type de traitement de la demande.

Les informations suivantes sont incluses pour chaque appel d'API au sein d'un flux de travail afin de localiser l'appel sur la page de documentation :

- Catégorie
  Les appels API sont organisés sur la page de documentation en zones ou catégories liées aux fonctions. Pour localiser un appel API spécifique, faites défiler la page jusqu'en bas et cliquez sur la catégorie API applicable.
- 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 d'accès est ajoutée à l'URL principale pour former l'URL complète identifiant la ressource.

### Créez une URL pour accéder directement à l'API REST

En plus de la page de documentation ONTAP Select, vous pouvez également accéder à l'API REST de déploiement directement via un langage de programmation comme Python. Dans ce cas, l'URL principale est légèrement différente de l'URL utilisée lors de l'accès à la page de documentation en ligne. Lorsque vous accédez directement à l'API, vous devez ajouter /api au domaine et à la chaîne de port. Par exemple : `http://deploy.mycompany.com/api`

## Workflow 1 : créez un cluster d'évaluation à un seul nœud sur ESXi

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

Le workflow de création de cluster diffère dans les cas suivants :

- L'hôte ESXi n'est pas géré par vCenter (hôte autonome)
- Plusieurs nœuds ou hôtes sont utilisés dans le 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

- Depuis la version ONTAP Select 9.10.1, il n'est plus possible de déployer un nouveau cluster sur l'hyperviseur KVM.
- Depuis ONTAP Select 9.11.1, toutes les fonctionnalités de gestion ne sont plus disponibles pour les clusters et hôtes KVM existants, à l'exception des fonctions de mise hors ligne et de suppression.

### 1. Enregistrer les informations d'identification du serveur vCenter

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

| Catégorie | Verbe HTTP | Chemin |
|---|---|---|
| Déployez | POST | /sécurité/informations d'identification |

**Gondolage**

```
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

**Sortie**

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

### 2. Enregistrez un hôte d'hyperviseur

Vous devez ajouter un hôte d'hyperviseur où la machine virtuelle contenant le nœud ONTAP Select sera exécutée.

| Catégorie | Verbe HTTP | Chemin |
|---|---|---|
| Cluster | POST | /hôtes |

**Gondolage**

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

**Entrée JSON (step02)**

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

**Type de traitement**

Asynchrone

**Sortie**

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

### 3. Créez un cluster

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

| Catégorie | Verbe HTTP | Chemin |
|---|---|---|
| Cluster | POST | /clusters |

**Gondolage**

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

```
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 (étape03)**

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

**Type de traitement**

Synchrone

**Sortie**

- 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} |

**Gondolage**

Vous devez fournir l'ID de 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

**Sortie**

Aucune

#### 5. Récupérez le nom du nœud

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

| Catégorie | Verbe HTTP | Chemin |
|---|---|---|
| Cluster | OBTENEZ | /clusters/{cluster_id}/nœuds |

**Gondolage**

Vous devez fournir l'ID de cluster.

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

**Type de traitement**

Synchrone

**Sortie**

- Le tableau enregistre chaque élément décrivant un seul nœud avec l'ID et le nom uniques

### 6. Configurer les nœuds

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

| Catégorie | Verbe HTTP | Chemin |
|---|---|---|
| Cluster | CHEMIN | /clusters/{cluster_id}/nodes/{node_id} |

**Gondolage**

Vous devez fournir l'ID de cluster et l'ID de 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 sera exécuté.

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

**Type de traitement**

Synchrone

**Sortie**

Aucune

### 7. Récupérez les réseaux de nœuds

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

| Catégorie | Verbe HTTP | Chemin |
| --- | --- | --- |
| Cluster | OBTENEZ | /clusters/{cluster_id}/nodes/{node_id}/networks |

**Gondolage**

Vous devez fournir l'ID de cluster et l'ID de 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

**Sortie**

- Tableau de deux enregistrements décrivant chacun un seul réseau pour le nœud, y compris l'ID et le but uniques

### 8. Configurez 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 un seul nœud.

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

| Catégorie | Verbe HTTP | Chemin |
| --- | --- | --- |
| Cluster | CORRECTIF | /clusters/{cluster_id}/noeuds/{node_id}/réseaux/{network_id} |

**Gondolage**

Vous devez fournir l'ID de cluster, l'ID de nœud et l'ID 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 indiquer le nom du réseau.

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

**Type de traitement**

Synchrone

**Sortie**

Aucune

### 9. Configurez le pool de stockage de nœuds

La dernière étape de la configuration d'un nœud consiste à relier un pool de stockage. Vous pouvez déterminer les pools de stockage disponibles via le client Web vSphere, ou éventuellement via l'API REST de déploiement.

| Catégorie | Verbe HTTP | Chemin |
|---|---|---|
| Cluster | CORRECTIF | /clusters/{cluster_id}/noeuds/{node_id}/réseaux/{network_id} |

**Gondolage**

Vous devez fournir l'ID de cluster, l'ID de nœud et l'ID 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 (par étape 09)**

La capacité du pool est de 2 To.

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

**Type de traitement**

Synchrone

**Sortie**

Aucune

### 10. Déployer le cluster

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

| Catégorie | Verbe HTTP | Chemin |
|---|---|---|
| Cluster | POST | /clusters/{cluster_id}/deploy |

**Gondolage**

Vous devez fournir l'ID de 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 (step10)**

Vous devez fournir le mot de passe pour le compte d'administrateur ONTAP.

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

**Type de traitement**

Asynchrone

**Sortie**

- Objet travail

# Accès avec Python

## Avant d'accéder à l'API à l'aide de Python

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

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

- La dernière version applicable de Python2 doit être installée.
  Les codes échantillons ont été testés à l'aide de Python2. Ils devraient également être portables à Python3, mais n'ont pas été testés pour la compatibilité.
- Les requêtes et les bibliothèques urllib3 doivent être installées.
  Vous pouvez utiliser pip ou un autre outil de gestion Python, selon les besoins de votre environnement.
- Le poste de travail client sur lequel s'exécutent les scripts doit disposer d'un accès réseau à la machine virtuelle ONTAP Select Deploy.

En outre, vous devez disposer des informations suivantes :

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

## Comprendre les scripts Python

Les exemples de scripts Python vous permettent d'effectuer plusieurs tâches différentes. Vous devez comprendre les scripts avant de les utiliser dans une instance de déploiement actif.

### Caractéristiques de conception communes

Les scripts ont été conçus avec les caractéristiques communes suivantes :

- Exécuter à partir de l'interface de ligne de commande sur un ordinateur client
  Vous pouvez exécuter les scripts Python à partir de n'importe quelle machine cliente correctement configurée. Voir *avant de commencer* pour plus d'informations.
- Acceptez les paramètres d'entrée de l'interface de ligne
  Chaque script est contrôlé au niveau de l'interface de ligne de commande via des paramètres d'entrée.
- Lire le fichier d'entrée
  Chaque script lit un fichier en 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. Lorsque vous ajoutez une licence de nœud, vous devez fournir un fichier de licence valide.
- Utilisez 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.

### Création d'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 modifier le script dans votre environnement de déploiement comme suit :

- Depuis la version ONTAP Select 9.10.1, il n'est plus possible de déployer un nouveau cluster sur l'hyperviseur KVM.
- Depuis ONTAP Select 9.11.1, toutes les fonctionnalités de gestion ne sont plus disponibles pour les clusters et hôtes KVM existants, à l'exception des fonctions de mise hors ligne et de suppression.

- Hyperviseur
  Vous pouvez le déployer sur ESXI ou KVM (selon la version de déploiement). Lors du déploiement de VMware ESXi, l'hyperviseur peut être géré par vCenter ou être un hôte autonome.
- Taille du cluster
  Vous pouvez déployer un cluster à un ou plusieurs nœuds.
- 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 du compte utilisateur admin
- 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 du compte utilisateur admin
- Nom du fichier de licence
- Nom d'utilisateur ONTAP avec privilèges pour ajouter la licence
- Mot de passe de l'utilisateur ONTAP

#### Supprime 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 du compte utilisateur admin
- Nom du fichier de configuration JSON

# Exemples de code Python

## Pour créer un cluster

Vous pouvez utiliser le script suivant pour créer un cluster basé sur les 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 permettre la création d'un cluster par script

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

#### Cluster à un seul nœud 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 un seul nœud sur ESXi à l'aide de 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 un seul nœud sur KVM**

- Depuis la version ONTAP Select 9.10.1, il n'est plus possible de déployer un nouveau cluster sur l'hyperviseur KVM.
- Depuis ONTAP Select 9.11.1, toutes les fonctionnalités de gestion ne sont plus disponibles pour les clusters et hôtes KVM existants, à l'exception des fonctions de mise hors ligne et de suppression.

```
{
  "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

Vous pouvez utiliser le script suivant pour ajouter une licence pour un noeud 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

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 de support commun

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

Vous pouvez utiliser le script suivant pour redimensionner les nœuds d'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