Giorno 0/1

NetApp Automation

NetApp
November 18, 2025

Sommario

Giorno 0/1

Panoramica della soluzione ONTAP Day 0/1

È possibile utilizzare la soluzione di automazione ONTAP day 0/1 per distribuire e configurare un cluster ONTAP tramite Ansible. La soluzione è disponibile presso "Hub di automazione NetApp Console".

Opzioni flessibili di implementazione ONTAP

A seconda dei tuoi requisiti, puoi utilizzare l'hardware on-premise o simulare ONTAP per implementare e configurare un cluster ONTAP utilizzando Ansible.

Hardware on-premise

Puoi implementare questa soluzione utilizzando hardware on-premise che esegue ONTAP, come un FAS o un sistema AFF. Devi utilizzare una macchina virtuale Linux per implementare e configurare il cluster ONTAP utilizzando Ansible.

Simula ONTAP

Per implementare questa soluzione utilizzando un simulatore ONTAP, è necessario scaricare la versione più recente di simulate ONTAP dal sito di supporto NetApp. Simulate ONTAP è un simulatore virtuale per il software ONTAP. Simulate ONTAP viene eseguito in un hypervisor VMware su un sistema Windows, Linux o Mac. Per gli host Windows e Linux, è necessario utilizzare l'hypervisor VMware Workstation per eseguire questa soluzione. Se si dispone di un sistema operativo Mac, utilizzare l'hypervisor VMware Fusion.

Design a più strati

Il framework Ansible semplifica lo sviluppo e il riutilizzo dei task logici e dell'esecuzione dell'automazione. Il framework distingue tra le attività decisionali (livello logico) e le fasi di esecuzione (livello di esecuzione) nell'automazione. La comprensione del funzionamento di questi livelli consente di personalizzare la configurazione.

Un "playbook" Ansible esegue una serie di task dall'inizio alla fine. La `site.yml` guida contiene la `logic.yml` guida e la `execution.yml` guida.

Quando viene eseguita una richiesta, il `site.yml` playbook viene chiamato per primo il `logic.yml` playbook, quindi chiama il `execution.yml` playbook per eseguire la richiesta di servizio.

Non è necessario utilizzare il livello logico del framework. Il livello logico fornisce opzioni per espandere la capacità del framework oltre i valori hard-coded per l'esecuzione. Ciò consente di personalizzare le funzionalità del framework, se necessario.

Livello logico

Il livello logico è costituito dai seguenti elementi:

- `logic.yml`Il manuale
- File di operazioni logiche all'interno della `logic-tasks` directory

Il livello logico offre la possibilità di prendere decisioni complesse senza la necessità di una significativa integrazione personalizzata (ad esempio, la connessione a ServiceNOW). Il livello logico è configurabile e fornisce l'ingresso ai microservizi.

È inoltre prevista la capacità di bypassare il livello logico. Se si desidera ignorare il livello logico, non definire la `logic_operation` variabile. La invocazione diretta del `logic.yml` playbook offre la possibilità di effettuare qualche livello di debug senza esecuzione. È possibile utilizzare un'istruzione "debug" per verificare che il valore di `raw_service_request` sia corretto.

Considerazioni importanti:

- Il `logic.yml` playbook ricerca la `logic_operation` variabile. Se la variabile è definita nella richiesta, carica un file di attività dalla `logic-tasks` directory. Il file di attività deve essere un file .yml. Se non esiste un file di attività corrispondente e la `logic_operation` variabile è definita, il livello logico non riesce.
- Il valore predefinito della `logic_operation` variabile è `no-op`. Se la variabile non è definita in modo esplicito, per impostazione predefinita è `no-op`, che non esegue alcuna operazione.
- Se la `raw_service_request` variabile è già definita, l'esecuzione procede al livello di esecuzione. Se la variabile non è definita, il livello logico non riesce.

Livello di esecuzione

Il livello di esecuzione è costituito dai seguenti elementi:

- `execution.yml`Il manuale

Il livello di esecuzione effettua le chiamate API per configurare un cluster ONTAP. Il `execution.yml` playbook richiede che la `raw_service_request` variabile sia definita al momento dell'esecuzione.

Supporto per la personalizzazione

È possibile personalizzare questa soluzione in vari modi a seconda delle proprie esigenze.

Le opzioni di personalizzazione includono:

- Modifica dei playbook Ansible
- Aggiunta di ruoli

Personalizza i file Ansible

La tabella seguente descrive i file Ansible personalizzabili contenuti in questa soluzione.

Posizione	Descrizione
`playbooks/inventory/hosts`	Contiene un singolo file con un elenco di host e gruppi.
`playbooks/group_vars/all/*`	Ansible fornisce un modo pratico per applicare le variabili a più host contemporaneamente. È possibile modificare uno o tutti i file contenuti in questa cartella, inclusi `cfg.yml` `clusters.yml` , `defaults.yml`, `services.yml`, `standards.yml` e `vault.yml`.
`playbooks/logic-tasks`	Supporta le attività decisionali all'interno di Ansible e mantiene la separazione di logica ed esecuzione. È possibile aggiungere file a questa cartella che corrispondono al servizio pertinente.
`playbooks/vars/*`	Valori dinamici utilizzati nei playbook e nei ruoli Ansible per consentire la personalizzazione, la flessibilità e la riutilizzabilità delle configurazioni. Se necessario, è possibile modificare uno o tutti i file contenuti in questa cartella.

Personalizzare i ruoli

Puoi anche personalizzare la soluzione aggiungendo o cambiando ruoli Ansible, anche chiamati microservizi. Per ulteriori informazioni, vedere "Personalizza".

Preparare l'uso della soluzione ONTAP Day 0/1

Prima di implementare la soluzione di automazione, devi preparare l'ambiente ONTAP e installare e configurare Ansible.

Considerazioni iniziali di pianificazione

È necessario analizzare i requisiti e le considerazioni seguenti prima di utilizzare questa soluzione per implementare un cluster ONTAP.

Requisiti di base

Per utilizzare questa soluzione è necessario soddisfare i seguenti requisiti di base:

- Devi avere accesso al software ONTAP on-premise o tramite un simulatore ONTAP.
- È necessario sapere come utilizzare il software ONTAP.
- Devi sapere come utilizzare gli strumenti software di automazione Ansible.

Considerazioni sulla pianificazione

Prima di implementare questa soluzione di automazione, è necessario decidere:

- Posizione in cui eseguire il nodo di controllo Ansible.
- Sistema ONTAP (hardware on-premise) o simulatore ONTAP.
- Se è necessario personalizzare o meno.

Preparare il sistema ONTAP

Utilizzando un sistema ONTAP on-premise o simulando ONTAP, devi preparare l'ambiente prima di poter implementare la soluzione di automazione.

Facoltativamente, installare e configurare simulate ONTAP

Per implementare questa soluzione attraverso un simulatore di ONTAP, è necessario scaricare ed eseguire simulate ONTAP.

Prima di iniziare

- È necessario scaricare e installare l'hypervisor VMware che si intende utilizzare per eseguire simulate ONTAP.
  - Se si dispone di un sistema operativo Windows o Linux, utilizzare VMware Workstation.
  - Se si dispone di un sistema operativo Mac, utilizzare VMware Fusion.

> Se si utilizza un sistema operativo Mac OS, è necessario disporre di un processore Intel.

Fasi

Per installare due simulatori ONTAP nell'ambiente locale, attenersi alla procedura seguente:

1. Scaricare simulate ONTAP dal "Sito di supporto NetApp".

   Anche se si installano due simulatori ONTAP, è sufficiente scaricare una sola copia del software.

2. Se non è già in esecuzione, avviare l'applicazione VMware.
3. Individuare il file del simulatore scaricato e fare clic con il pulsante destro del mouse per aprirlo con l'applicazione VMware.
4. Impostare il nome della prima istanza di ONTAP.
5. Attendere l'avvio del simulatore e seguire le istruzioni per creare un cluster a nodo singolo.

   Ripetere la procedura per la seconda istanza di ONTAP.

6. In alternativa, è possibile aggiungere un complemento di dischi completo.

   Da ciascun cluster, eseguire i seguenti comandi:

```
security unlock -username <user_01>
security login password -username <user_01>
set -priv advanced
systemshell local
disk assign -all -node <Cluster-01>-01
```

Stato del sistema ONTAP

Devi verificare lo stato iniziale del sistema ONTAP, sia on-premise che in esecuzione attraverso un simulatore ONTAP.

Verificare che siano soddisfatti i seguenti requisiti di sistema ONTAP:

- ONTAP è installato e in esecuzione senza cluster ancora definiti.
- ONTAP viene avviato e visualizza l'indirizzo IP per accedere al cluster.
- La rete è raggiungibile.
- Si dispone delle credenziali di amministratore.
- Viene visualizzato il banner del messaggio del giorno (MOTD) con l'indirizzo di gestione.

Installare il software di automazione richiesto

Questa sezione fornisce informazioni su come installare Ansible e preparare la soluzione di automazione per l'implementazione.

Installa Ansible

Ansible può essere installato su sistemi Linux o Windows.

Il metodo di comunicazione predefinito utilizzato da Ansible per comunicare con un cluster ONTAP è SSH.

Fare riferimento a "Introduzione a NetApp e Ansible: Installare Ansible" per installare Ansible.

Ansible deve essere installato sul nodo di controllo del sistema.

Scaricare e preparare la soluzione di automazione

Per scaricare e preparare la soluzione di automazione per la distribuzione, è possibile attenersi alla seguente procedura.

1. Scarica il "ONTAP - giorno 0/1 controlli dello stato" soluzione di automazione tramite l'interfaccia utente web della console. La soluzione è confezionata come `ONTAP_DAY0_DAY1.zip`.
2. Estrarre la cartella zip e copiare i file nella posizione desiderata sul nodo di controllo all'interno dell'ambiente Ansible.

Configurazione iniziale del framework Ansible

Eseguire la configurazione iniziale del framework Ansible:

1. Passare a `playbooks/inventory/group_vars/all`.
2. Decrittografare il `vault.yml` file:

   ```
   ansible-vault decrypt playbooks/inventory/group_vars/all/vault.yml
   ```

   Quando viene richiesta la password del vault, immettere la seguente password temporanea:

   ```
   NetApp123!
   ```

   "NetApp123!" è una password temporanea per decrittografare il `vault.yml` file e la password del vault corrispondente. Dopo il primo utilizzo, è **necessario** crittografare il file utilizzando la propria password.

3. Modificare i seguenti file Ansible:

   - `clusters.yml` - Modificare i valori in questo file per adattarli all'ambiente.
   - `vault.yml` - Dopo aver decrittografato il file, modificare i valori del cluster ONTAP, del nome utente e della password in base all'ambiente in uso.
   - `cfg.yml` - Impostare il percorso del file per `log2file` e impostare `show_request` in `cfg` a `True` per visualizzare `raw_service_request` .

     La `raw_service_request` variabile viene visualizzata nei file di registro e durante l'esecuzione.

   Ogni file elencato contiene commenti con istruzioni su come modificarlo in base alle proprie esigenze.

4. Crittografare nuovamente il `vault.yml` file:

   ```
   ansible-vault encrypt playbooks/inventory/group_vars/all/vault.yml
   ```

   Viene richiesto di scegliere una nuova password per il vault al momento della crittografia.

5. Navigare `playbooks/inventory/hosts` e impostare un interprete Python valido.

6. Implementare il `framework_test` servizio:

   Il seguente comando esegue il `na_ontap_info` modulo con un `gather_subset` valore di `cluster_identity_info` . In questo modo, la configurazione di base risulta corretta e si verifica la possibilità di comunicare con il cluster.

   ```
   ansible-playbook -i inventory/hosts site.yml -e
   cluster_name=<CLUSTER_NAME>
   -e logic_operation=framework-test
   ```

   Eseguire il comando per ciascun cluster.

   Se l'operazione ha esito positivo, si dovrebbe visualizzare un output simile al seguente esempio:

   ```
   PLAY RECAP
   ********************************************************************
   *********
   localhost : ok=12 changed=1 unreachable=0 failed=0 skipped=6
   The key is 'rescued=0' and 'failed=0'..
   ```

Implementare il cluster ONTAP utilizzando la soluzione

Dopo aver completato la preparazione e il planning, sei pronto a utilizzare la soluzione ONTAP Day 0/1 per configurare rapidamente un cluster ONTAP utilizzando Ansible.

In qualsiasi momento durante le fasi di questa sezione, è possibile scegliere di testare una richiesta invece di eseguirla. Per testare una richiesta, modificare il `site.yml` playbook sulla riga di comando in `logic.yml`.

> La `docs/tutorial-requests.txt` posizione contiene la versione finale di tutte le richieste di servizio utilizzate durante questa procedura. In caso di difficoltà nell'esecuzione di una richiesta di servizio, è possibile copiare la richiesta pertinente dal `tutorial-requests.txt` file nella `playbooks/inventory/group_vars/all/tutorial-requests.yml` posizione e modificare i valori codificati come richiesto (indirizzo IP, nomi aggregati e così via). A questo punto, è possibile eseguire correttamente la richiesta.

Prima di iniziare

- È necessario che Ansible sia installato.
- È necessario aver scaricato la soluzione ONTAP Day 0/1 ed estratto la cartella nella posizione desiderata sul nodo di controllo Ansible.
- Lo stato del sistema ONTAP deve soddisfare i requisiti e l'utente deve disporre delle credenziali necessarie.
- È necessario aver completato tutte le attività richieste indicate nella "Preparatevi"sezione .

Negli esempi di questa soluzione vengono utilizzati "Cluster_01" e "Cluster_02" come nomi per i due cluster. È necessario sostituire questi valori con i nomi dei cluster nel proprio ambiente.

Fase 1: Configurazione iniziale del cluster

A questo punto, è necessario eseguire alcune operazioni iniziali di configurazione del cluster.

Fasi

1. Individuare la `playbooks/inventory/group_vars/all/tutorial-requests.yml` posizione e rivedere la `cluster_initial` richiesta nel file. Apportare le modifiche necessarie al proprio ambiente.
2. Creare un file nella `logic-tasks` cartella per la richiesta di servizio. Ad esempio, creare un file denominato `cluster_initial.yml`.

   Copiare le seguenti righe nel nuovo file:

```
- name: Validate required inputs
  ansible.builtin.assert:
    that:
    - service is defined

- name: Include data files
  ansible.builtin.include_vars:
    file:   "{{ data_file_name }}.yml"
  loop:
  - common-site-stds
  - user-inputs
  - cluster-platform-stds
  - vserver-common-stds
  loop_control:
    loop_var:    data_file_name

- name: Initial cluster configuration
  set_fact:
    raw_service_request:
```

3. Definire la `raw_service_request` variabile.

   È possibile utilizzare una delle seguenti opzioni per definire la `raw_service_request` variabile nel `cluster_initial.yml` file creato nella `logic-tasks` cartella:

   - **Opzione 1**: Definire manualmente la `raw_service_request` variabile.

     Aprire il `tutorial-requests.yml` file utilizzando un editor e copiare il contenuto dalla riga 11 alla riga 165. Incollare il contenuto sotto la `raw service request` variabile nel nuovo `cluster_initial.yml` file, come illustrato negli esempi seguenti:


3 # This file contains the final version of the various service
4 # requests used throughout the tutorial in TUTORIAL.md.
5 #---
6 #---
7 # cluster_initial:
8 #
9 #---
11 service: cluster_initial
create
13 std_name: none
14 req_details:
16 ontap_aggr:
17 - hostname: "{{ cluster_name }}"
18 disk_count: 24
19 name: n01_aggr1
20 nodes: "{{ cluster_name }}-01"

Mostra esempio

File di esempio `cluster_initial.yml`:

```
- name: Validate required inputs
  ansible.builtin.assert:
    that:
    - service is defined

- name: Include data files
  ansible.builtin.include_vars:
    file:   "{{ data_file_name }}.yml"
  loop:
  - common-site-stds
  - user-inputs
  - cluster-platform-stds
  - vserver-common-stds
  loop_control:
    loop_var:    data_file_name

- name: Initial cluster configuration
  set_fact:
    raw_service_request:
     service:          cluster_initial
     operation:         create
     std_name:           none
     req_details:

      ontap_aggr:
      - hostname:                  "{{ cluster_name }}"
        disk_count:                24
        name:                      n01_aggr1
        nodes:                     "{{ cluster_name }}-01"
        raid_type:                 raid4

      - hostname:                  "{{ peer_cluster_name }}"
        disk_count:                24
        name:                      n01_aggr1
        nodes:                     "{{ peer_cluster_name }}-01"
        raid_type:                 raid4

      ontap_license:
      - hostname:                  "{{ cluster_name }}"
        license_codes:
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
```

```
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA

- hostname:                   "{{ peer_cluster_name }}"
  license_codes:
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
    - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
```

```
      - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXAAAAAAAAAAAAAA

  ontap_motd:
  - hostname:                     "{{ cluster_name }}"
    vserver:                      "{{ cluster_name }}"
    message:                      "New MOTD"

  - hostname:                     "{{ peer_cluster_name }}"
    vserver:                      "{{ peer_cluster_name }}"
    message:                      "New MOTD"

  ontap_interface:
  - hostname:                     "{{ cluster_name }}"
    vserver:                      "{{ cluster_name }}"
    interface_name:               ic01
    role:                         intercluster
    address:                      10.0.0.101
    netmask:                      255.255.255.0
    home_node:                    "{{ cluster_name }}-01"
    home_port:                    e0c
    ipspace:                      Default
    use_rest:                     never

  - hostname:                     "{{ cluster_name }}"
    vserver:                      "{{ cluster_name }}"
    interface_name:               ic02
    role:                         intercluster
    address:                      10.0.0.101
    netmask:                      255.255.255.0
    home_node:                    "{{ cluster_name }}-01"
    home_port:                    e0c
```

```
    ipspace:                    Default
    use_rest:                   never

  - hostname:                   "{{ peer_cluster_name }}"
    vserver:                    "{{ peer_cluster_name }}"
    interface_name:             ic01
    role:                       intercluster
    address:                    10.0.0.101
    netmask:                    255.255.255.0
    home_node:                  "{{ peer_cluster_name }}-01"
    home_port:                  e0c
    ipspace:                    Default
    use_rest:                   never

  - hostname:                   "{{ peer_cluster_name }}"
    vserver:                    "{{ peer_cluster_name }}"
    interface_name:             ic02
    role:                       intercluster
    address:                    10.0.0.101
    netmask:                    255.255.255.0
    home_node:                  "{{ peer_cluster_name }}-01"
    home_port:                  e0c
    ipspace:                    Default
    use_rest:                   never

  ontap_cluster_peer:
  - hostname:                   "{{ cluster_name }}"
    dest_cluster_name:          "{{ peer_cluster_name }}"
    dest_intercluster_lifs:     "{{ peer_lifs }}"
    source_cluster_name:        "{{ cluster_name }}"
    source_intercluster_lifs:   "{{ cluster_lifs }}"
    peer_options:
      hostname:                 "{{ peer_cluster_name }}"
```

- **Opzione 2**: Utilizzare un modello Jinja per definire la richiesta:

  È anche possibile utilizzare il seguente formato di modello Jinja per ottenere il `raw_service_request` valore.

```
raw_service_request: "{{ cluster_initial }}"
```

4. Eseguire la configurazione iniziale del cluster per il primo cluster:

```
ansible-playbook -i inventory/hosts site.yml -e
cluster_name=<Cluster_01>
```

Prima di procedere, verificare che non vi siano errori.

5. Ripetere il comando per il secondo cluster:

```
ansible-playbook -i inventory/hosts site.yml -e
cluster_name=<Cluster_02>
```

Verificare che non siano presenti errori per il secondo cluster.

Quando scorri verso l'alto verso l'inizio dell'output Ansible dovresti vedere la richiesta inviata al framework, come mostrato nel seguente esempio:

Mostra esempio

```
TASK [Show the raw_service_request]
********************************************************************
****************************************
ok: [localhost] => {
    "raw_service_request": {
        "operation": "create",
        "req_details": {
            "ontap_aggr": [
                {
                    "disk_count": 24,
                    "hostname": "Cluster_01",
                    "name": "n01_aggr1",
                    "nodes": "Cluster_01-01",
                    "raid_type": "raid4"
                }
            ],
            "ontap_license": [
                {
                    "hostname": "Cluster_01",
                    "license_codes": [
                        "XXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
```

```
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXAAAAAAAAAAAAA"
                    ]
                }
            ],
            "ontap_motd": [
                {
                    "hostname": "Cluster_01",
                    "message": "New MOTD",
                    "vserver": "Cluster_01"
                }
            ]
        },
        "service": "cluster_initial",
        "std_name": "none"
    }
}
```

6. Accedere a ciascuna istanza di ONTAP e verificare che la richiesta sia stata eseguita correttamente.

Fase 2: Configurare intercluster LIF

Ora puoi configurare i LIF intercluster LIF aggiungendo le definizioni LIF alla `cluster_initial` richiesta e definendo il `ontap_interface` microservizio.

La definizione del servizio e la richiesta lavorano insieme per determinare l'azione:

- Se si fornisce una richiesta di servizio per un microservizio che non è presente nelle definizioni di servizio, la richiesta non viene eseguita.
- Se si fornisce una richiesta di servizio con uno o più microservizi definiti nelle definizioni di servizio, ma omessi dalla richiesta, la richiesta non viene eseguita.

Il `execution.yml` playbook valuta la definizione del servizio analizzando l'elenco dei microservizi nell'ordine elencato:

- Se nella richiesta è presente una voce con una chiave dizionario corrispondente alla `args` voce contenuta nelle definizioni di microservizi, la richiesta viene eseguita.
- Se nella richiesta di servizio non è presente alcuna voce corrispondente, la richiesta viene ignorata senza errori.

Fasi

1. Passare al `cluster_initial.yml` file creato in precedenza e modificare la richiesta aggiungendo le seguenti righe alle definizioni della richiesta:

```
ontap_interface:
- hostname:                    "{{ cluster_name }}"
  vserver:                     "{{ cluster_name }}"
  interface_name:              ic01
  role:                        intercluster
  address:                     <ip_address>
  netmask:                     <netmask_address>
  home_node:                   "{{ cluster_name }}-01"
  home_port:                   e0c
  ipspace:                     Default
  use_rest:                    never

- hostname:                    "{{ cluster_name }}"
  vserver:                     "{{ cluster_name }}"
  interface_name:              ic02
  role:                        intercluster
  address:                     <ip_address>
  netmask:                     <netmask_address>
  home_node:                   "{{ cluster_name }}-01"
  home_port:                   e0c
  ipspace:                     Default
  use_rest:                    never

- hostname:                    "{{ peer_cluster_name }}"
  vserver:                     "{{ peer_cluster_name }}"
  interface_name:              ic01
  role:                        intercluster
  address:                     <ip_address>
  netmask:                     <netmask_address>
  home_node:                   "{{ peer_cluster_name }}-01"
  home_port:                   e0c
  ipspace:                     Default
  use_rest:                    never

- hostname:                    "{{ peer_cluster_name }}"
  vserver:                     "{{ peer_cluster_name }}"
  interface_name:              ic02
  role:                        intercluster
  address:                     <ip_address>
  netmask:                     <netmask_address>
  home_node:                   "{{ peer_cluster_name }}-01"
  home_port:                   e0c
  ipspace:                     Default
  use_rest:                    never
```

2. Eseguire il comando:

```
ansible-playbook -i inventory/hosts  site.yml -e
cluster_name=<Cluster_01> -e peer_cluster_name=<Cluster_02>
```

3. Effettua l'accesso a ciascuna istanza per verificare se le LIF sono state aggiunte al cluster:

Mostra esempio

```
Cluster_01::> net int show
  (network interface show)
           Logical    Status     Network            Current
Current Is
Vserver     Interface  Admin/Oper Address/Mask       Node
Port    Home
----------- ---------- ---------- ------------------ -------------
------- ----
Cluster_01
            Cluster_01-01_mgmt up/up 10.0.0.101/24   Cluster_01-01
e0c     true
            Cluster_01-01_mgmt_auto up/up 10.101.101.101/24
Cluster_01-01 e0c true
            cluster_mgmt up/up    10.0.0.110/24      Cluster_01-01
e0c     true
5 entries were displayed.
```

Il risultato mostra che le LIF sono state **non** aggiunte. Questo perché il `ontap_interface` microservizio deve ancora essere definito nel `services.yml` file.

4. Verificare che le LIF siano state aggiunte alla `raw_service_request` variabile.

Mostra esempio

Il seguente esempio mostra che le LIF sono state aggiunte alla richiesta:


```
            "ontap_interface": [
                 {
                     "address": "10.0.0.101",
                     "home_node": "Cluster_01-01",
                     "home_port": "e0c",
                     "hostname": "Cluster_01",
                     "interface_name": "ic01",
                     "ipspace": "Default",
                     "netmask": "255.255.255.0",
                     "role": "intercluster",
                     "use_rest": "never",
                     "vserver": "Cluster_01"
                 },
                 {
                     "address": "10.0.0.101",
                     "home_node": "Cluster_01-01",
                     "home_port": "e0c",
                     "hostname": "Cluster_01",
                     "interface_name": "ic02",
                     "ipspace": "Default",
                     "netmask": "255.255.255.0",
                     "role": "intercluster",
                     "use_rest": "never",
                     "vserver": "Cluster_01"
                 },
                 {
                     "address": "10.0.0.101",
                     "home_node": "Cluster_02-01",
                     "home_port": "e0c",
                     "hostname": "Cluster_02",
                     "interface_name": "ic01",
                     "ipspace": "Default",
                     "netmask": "255.255.255.0",
                     "role": "intercluster",
                     "use_rest": "never",
                     "vserver": "Cluster_02"
                 },
                 {
                     "address": "10.0.0.126",
                     "home_node": "Cluster_02-01",
                     "home_port": "e0c",
                     "hostname": "Cluster_02",
```

```
                    "interface_name": "ic02",
                    "ipspace": "Default",
                    "netmask": "255.255.255.0",
                    "role": "intercluster",
                    "use_rest": "never",
                    "vserver": "Cluster_02"
                }
            ],
```

5. Definire il `ontap_interface` microservizio in `cluster_initial` nel `services.yml` file.

   Copiare le seguenti righe nel file per definire il microservizio:

```
        - name: ontap_interface
          args: ontap_interface
          role: na/ontap_interface
```

6. Ora che il `ontap_interface` microservizio è stato definito nella richiesta e nel `services.yml` file, eseguire nuovamente la richiesta:

```
ansible-playbook -i inventory/hosts  site.yml -e
cluster_name=<Cluster_01> -e peer_cluster_name=<Cluster_02>
```

7. Accedere a ciascuna istanza di ONTAP e verificare che le LIF siano state aggiunte.

Fase 3: In alternativa, configurare più cluster

Se necessario, puoi configurare più cluster nella stessa richiesta. Quando si definisce la richiesta, è necessario fornire i nomi delle variabili per ciascun cluster.

Fasi

1. Aggiungere una voce per il secondo cluster nel `cluster_initial.yml` file per configurare entrambi i cluster nella stessa richiesta.

   Nell'esempio seguente viene visualizzato il `ontap_aggr` campo dopo l'aggiunta della seconda voce.

```
ontap_aggr:
 - hostname:                    "{{ cluster_name }}"
   disk_count:                  24
   name:                        n01_aggr1
   nodes:                       "{{ cluster_name }}-01"
   raid_type:                   raid4

 - hostname:                    "{{ peer_cluster_name }}"
   disk_count:                  24
   name:                        n01_aggr1
   nodes:                       "{{ peer_cluster_name }}-01"
   raid_type:                   raid4
```

2. Applicare le modifiche per tutti gli altri elementi in `cluster_initial`.
3. Aggiungere il peering dei cluster alla richiesta copiando le seguenti righe nel file:

```
ontap_cluster_peer:
- hostname:                     "{{ cluster_name }}"
  dest_cluster_name:            "{{ cluster_peer }}"
  dest_intercluster_lifs:       "{{ peer_lifs }}"
  source_cluster_name:          "{{ cluster_name }}"
  source_intercluster_lifs:     "{{ cluster_lifs }}"
  peer_options:
    hostname:                   "{{ cluster_peer }}"
```

4. Eseguire la richiesta Ansible:

```
ansible-playbook -i inventory/hosts -e cluster_name=<Cluster_01>
site.yml -e peer_cluster_name=<Cluster_02> -e
cluster_lifs=<cluster_lif_1_IP_address,cluster_lif_2_IP_address>
-e peer_lifs=<peer_lif_1_IP_address,peer_lif_2_IP_address>
```

Fase 4: Configurazione SVM iniziale

In questa fase della procedura è necessario configurare le SVM nel cluster.

Fasi

1. Aggiornare la `svm_initial` richiesta nel `tutorial-requests.yml` file per configurare un peer relationship SVM e SVM.

   È necessario configurare quanto segue:

   - SVM

- La relazione peer della SVM
- L'interfaccia SVM per ciascuna SVM

2. Aggiornare le definizioni delle variabili nelle definizioni delle `svm_initial` richieste. È necessario modificare le seguenti definizioni di variabile:
   - `cluster_name`
   - `vserver_name`
   - `peer_cluster_name`
   - `peer_vserver`

   Per aggiornare le definizioni, rimuovere il simbolo * '{}'* dopo `req_details` per la `svm_initial` definizione e aggiungere la definizione corretta.

3. Creare un file nella `logic-tasks` cartella per la richiesta di servizio. Ad esempio, creare un file denominato `svm_initial.yml`.

   Copiare le seguenti righe nel file:

```
- name: Validate required inputs
  ansible.builtin.assert:
    that:
    - service is defined

- name: Include data files
  ansible.builtin.include_vars:
    file:   "{{ data_file_name }}.yml"
  loop:
  - common-site-stds
  - user-inputs
  - cluster-platform-stds
  - vserver-common-stds
  loop_control:
    loop_var:    data_file_name

- name: Initial SVM configuration
  set_fact:
    raw_service_request:
```

4. Definire la `raw_service_request` variabile.

   È possibile utilizzare una delle seguenti opzioni per definire la `raw_service_request` variabile `svm_initial` nella `logic-tasks` cartella:

   - **Opzione 1**: Definire manualmente la `raw_service_request` variabile.

     Aprire il `tutorial-requests.yml` file utilizzando un editor e copiare il contenuto dalla riga 179 alla

riga 222. Incollare il contenuto sotto la `raw service request` variabile nel nuovo `svm_initial.yml` file, come illustrato negli esempi seguenti:

Mostra esempio

File di esempio `svm_initial.yml`:

```
- name: Validate required inputs
  ansible.builtin.assert:
    that:
    - service is defined

- name: Include data files
  ansible.builtin.include_vars:
    file:   "{{ data_file_name }}.yml"
  loop:
  - common-site-stds
  - user-inputs
  - cluster-platform-stds
  - vserver-common-stds
  loop_control:
    loop_var:    data_file_name

- name: Initial SVM configuration
  set_fact:
    raw_service_request:
     service:          svm_initial
     operation:        create
     std_name:         none
     req_details:

      ontap_vserver:
      - hostname:                   "{{ cluster_name }}"
        name:                       "{{ vserver_name }}"
        root_volume_aggregate:      n01_aggr1

      - hostname:                   "{{ peer_cluster_name }}"
       name:                       "{{ peer_vserver }}"
       root_volume_aggregate:      n01_aggr1

      ontap_vserver_peer:
      - hostname:                   "{{ cluster_name }}"
        vserver:                    "{{ vserver_name }}"
        peer_vserver:               "{{ peer_vserver }}"
        applications:               snapmirror
        peer_options:
          hostname:                 "{{ peer_cluster_name }}"

      ontap_interface:
```

```
      - hostname:                    "{{ cluster_name }}"
        vserver:                     "{{ vserver_name }}"
        interface_name:              data01
        role:                        data
        address:                     10.0.0.200
        netmask:                     255.255.255.0
        home_node:                   "{{ cluster_name }}-01"
        home_port:                   e0c
        ipspace:                     Default
        use_rest:                    never

      - hostname:                    "{{ peer_cluster_name }}"
        vserver:                     "{{ peer_vserver }}"
        interface_name:              data01
        role:                        data
        address:                     10.0.0.201
        netmask:                     255.255.255.0
        home_node:                   "{{ peer_cluster_name }}-01"
        home_port:                   e0c
        ipspace:                     Default
        use_rest:                    never
```

- **Opzione 2**: Utilizzare un modello Jinja per definire la richiesta:

  È anche possibile utilizzare il seguente formato di modello Jinja per ottenere il `raw_service_request` valore.

```
raw_service_request: "{{ svm_initial }}"
```

5. Eseguire la richiesta:

```
ansible-playbook -i inventory/hosts -e cluster_name=<Cluster_01> -e
peer_cluster_name=<Cluster_02> -e peer_vserver=<SVM_02>  -e
vserver_name=<SVM_01> site.yml
```

6. Accedere a ciascuna istanza di ONTAP e convalidare la configurazione.
7. Aggiungere le interfacce della SVM.

   Definire il `ontap_interface` servizio in `svm_initial` nel `services.yml` file ed eseguire nuovamente la richiesta:

```
ansible-playbook -i inventory/hosts -e cluster_name=<Cluster_01> -e
peer_cluster_name=<Cluster_02> -e peer_vserver=<SVM_02>  -e
vserver_name=<SVM_01> site.yml
```

8. Effettuare l'accesso a ciascuna istanza di ONTAP e verificare che le interfacce della SVM siano state configurate.

Fase 5: Se si desidera, definire una richiesta di servizio in modo dinamico

Nei passi precedenti, la `raw_service_request` variabile è codificata. Ciò è utile per l'apprendimento, lo sviluppo e il test. È inoltre possibile generare dinamicamente una richiesta di servizio.

La sezione seguente fornisce un'opzione per produrre dinamicamente il necessario `raw_service_request` se non si desidera integrarlo con sistemi di livello superiore.

- Se la `logic_operation` variabile non è definita nel comando, il `logic.yml` file non importa alcun file dalla `logic-tasks` cartella. Ciò significa che i `raw_service_request` devono essere definiti all'esterno di Ansible e forniti al framework al momento dell'esecuzione.
- Il nome del file di un'operazione nella `logic-tasks` cartella deve corrispondere al valore della `logic_operation` variabile senza estensione .yml.
- I file di attività nella `logic-tasks` cartella definiscono dinamicamente un `raw_service_request`. l'unico requisito è che un valido `raw_service_request` sia definito come l'ultima attività nel file pertinente.

Definizione dinamica di una richiesta di servizio

Esistono diversi modi per applicare un'attività logica per definire dinamicamente una richiesta di servizio. Di seguito sono elencate alcune di queste opzioni:

- Utilizzo di un file attività Ansible dalla `logic-tasks` cartella
- Richiamo di un ruolo personalizzato che restituisce dati adatti alla conversione in un ruolo `raw_service_request` variabile.
- Richiamo di un altro strumento all'esterno dell'ambiente Ansible per i dati richiesti. Ad esempio, una chiamata API REST a Active IQ Unified Manager.

I seguenti comandi di esempio definiscono dinamicamente una richiesta di servizio per ogni cluster utilizzando il `tutorial-requests.yml` file:

```
ansible-playbook -i inventory/hosts -e cluster2provision=Cluster_01
-e logic_operation=tutorial-requests site.yml
```

```
ansible-playbook -i inventory/hosts -e cluster2provision=Cluster_02
-e logic_operation=tutorial-requests site.yml
```

Fase 6: Distribuire la soluzione ONTAP Day 0/1

In questa fase, dovresti aver già completato quanto segue:

- Revisionato e modificato tutti i file in in `playbooks/inventory/group_vars/all` base alle proprie esigenze. Ogni file contiene commenti dettagliati che consentono di apportare le modifiche.
- Aggiunti tutti i file di attività richiesti alla `logic-tasks` directory.
- Aggiunti tutti i file di dati necessari alla `playbook/vars` directory.

Utilizzare i seguenti comandi per implementare la soluzione ONTAP Day 0/1 e verificare lo stato di salute della distribuzione:

> In questa fase, il file dovrebbe essere già stato decrittografato e modificato `vault.yml` e deve essere crittografato con la nuova password.

- Eseguire il servizio ONTAP Day 0:

```
ansible-playbook -i playbooks/inventory/hosts playbooks/site.yml -e
logic_operation=cluster_day_0 -e service=cluster_day_0 -vvvv --ask-vault
-pass <your_vault_password>
```

- Eseguire il servizio ONTAP Day 1:

```
ansible-playbook -i playbooks/inventory/hosts playbooks/site.yml -e
logic_operation=cluster_day_1 -e service=cluster_day_0 -vvvv --ask-vault
-pass <your_vault_password>
```

- Applicare le impostazioni a livello di cluster:

```
ansible-playbook -i playbooks/inventory/hosts playbooks/site.yml -e
logic_operation=cluster_wide_settings -e service=cluster_wide_settings
-vvvv --ask-vault-pass <your_vault_password>
```

- Eseguire i controlli dello stato di salute:

```
ansible-playbook -i playbooks/inventory/hosts playbooks/site.yml -e
logic_operation=health_checks -e service=health_checks -e
enable_health_reports=true -vvvv --ask-vault-pass <your_vault_password>
```

Personalizzare la soluzione ONTAP Day 0/1

Per personalizzare la soluzione ONTAP Day 0/1 in base ai tuoi requisiti, puoi aggiungere o modificare i ruoli Ansible.

I ruoli rappresentano i microservizi all'interno del framework Ansible. Ogni microservizio esegue un'operazione. Ad esempio, ONTAP Day 0 è un servizio che contiene più microservizi.

Aggiungi ruoli Ansible

Puoi aggiungere ruoli Ansible per personalizzare la soluzione per il tuo ambiente. I ruoli richiesti sono definiti dalle definizioni dei servizi all'interno del framework Ansible.

Un ruolo deve soddisfare i seguenti requisiti per essere utilizzato come microservizio:

- Accettare un elenco di argomenti nella `args` variabile.
- Utilizza la struttura Ansible "Block, rescue, Always" con determinati requisiti per ogni blocco.
- Utilizza un singolo modulo Ansible e definisci un singolo task all'interno del blocco.
- Implementare tutti i parametri del modulo disponibili in base ai requisiti descritti in questa sezione.

Struttura di microservizio richiesta

Ogni ruolo deve supportare le seguenti variabili:

- `mode`: Se la modalità è impostata sul `test` ruolo tenta di importare il `test.yml` che mostra cosa fa il ruolo senza eseguirlo.

  Non è sempre possibile implementare questo processo a causa di alcune interdipendenze.

- `status`: Lo stato generale dell'esecuzione del playbook. Se il valore non è impostato `success` sul ruolo non viene eseguito.
- `args` : Elenco di dizionari specifici per ruolo con chiavi che corrispondono ai nomi dei parametri del ruolo.
- `global_log_messages`: Raccoglie i messaggi di registro durante l'esecuzione del playbook. Ogni volta che viene eseguito il ruolo viene generata una voce.
- `log_name`: Il nome utilizzato per fare riferimento al ruolo all'interno delle `global_log_messages` voci.
- `task_descr`: Una breve descrizione delle funzioni del ruolo.
- `service_start_time`: La data e l'ora utilizzate per tenere traccia dell'ora di esecuzione di ciascun ruolo.
- `playbook_status`: Lo stato del playbook Ansible.
- `role_result`: La variabile che contiene l'output del ruolo ed è inclusa in ogni messaggio all'interno delle `global_log_messages` voci.

Esempio di struttura dei ruoli

Nell'esempio seguente viene fornita la struttura di base di un ruolo che implementa un microservizio. È necessario modificare le variabili in questo esempio per la propria configurazione.

Mostra esempio

Struttura dei ruoli di base:

```
- name:  Set some role attributes
  set_fact:
    log_name:     "<LOG_NAME>"
    task_descr:   "<TASK_DESCRIPTION>"

-  name: "{{ log_name }}"
   block:
      -  set_fact:
            service_start_time: "{{ lookup('pipe', 'date
+%Y%m%d%H%M%S') }}"

      -  name: "Provision the new user"
         <MODULE_NAME>:

#-----------------------------------------------------------------
            # COMMON ATTRIBUTES

#-----------------------------------------------------------------
            hostname:            "{{
clusters[loop_arg['hostname']]['mgmt_ip'] }}"
            username:            "{{
clusters[loop_arg['hostname']]['username'] }}"
            password:            "{{
clusters[loop_arg['hostname']]['password'] }}"

            cert_filepath:       "{{ loop_arg['cert_filepath']
| default(omit) }}"
            feature_flags:       "{{ loop_arg['feature_flags']
| default(omit) }}"
            http_port:           "{{ loop_arg['http_port']
| default(omit) }}"
            https:               "{{ loop_arg['https']
| default('true') }}"
            ontapi:              "{{ loop_arg['ontapi']
| default(omit) }}"
            key_filepath:        "{{ loop_arg['key_filepath']
| default(omit) }}"
            use_rest:            "{{ loop_arg['use_rest']
| default(omit) }}"
            validate_certs:      "{{ loop_arg['validate_certs']
| default('false') }}"
```

```
            <MODULE_SPECIFIC_PARAMETERS>

#-------------------------------------------------------------
            # REQUIRED ATTRIBUTES

#-------------------------------------------------------------
            required_parameter:     "{{ loop_arg['required_parameter']
}}"

#-------------------------------------------------------------
            # ATTRIBUTES w/ DEFAULTS

#-------------------------------------------------------------
            defaulted_parameter:    "{{ loop_arg['defaulted_parameter']
| default('default_value') }}"

#-------------------------------------------------------------
            # OPTIONAL ATTRIBUTES

#-------------------------------------------------------------
            optional_parameter:     "{{ loop_arg['optional_parameter']
| default(omit) }}"
         loop:    "{{ args }}"
         loop_control:
            loop_var:   loop_arg
         register:   role_result

   rescue:
      -  name: Set role status to FAIL
         set_fact:
            playbook_status:   "failed"

   always:
      -  name: add log msg
         vars:
            role_log:
               role: "{{ log_name }}"
               timestamp:
                  start_time: "{{service_start_time}}"
                  end_time: "{{ lookup('pipe', 'date +%Y-%m-
%d@%H:%M:%S') }}"
               service_status: "{{ playbook_status }}"
               result: "{{role_result}}"
         set_fact:
            global_log_msgs:   "{{ global_log_msgs + [ role_log ] }}"
```

Variabili utilizzate nel ruolo di esempio:

- `<NAME>`: Un valore sostituibile che deve essere fornito per ogni microservizio.
- `<LOG_NAME>`: Il nome breve del ruolo utilizzato per la registrazione. Ad esempio, `ONTAP_VOLUME`.
- `<TASK_DESCRIPTION>`: Una breve descrizione delle funzioni del microservizio.
- `<MODULE_NAME>`: Il nome del modulo Ansible per l'attività.

> Il playbook di livello superiore `execute.yml` specifica la `netapp.ontap` raccolta. Se il modulo fa parte dell' `netapp.ontap`insieme, non è necessario specificare completamente il nome del modulo.

- `<MODULE_SPECIFIC_PARAMETERS>`: Parametri del modulo Ansible specifici del modulo utilizzato per implementare il microservizio. Nell'elenco seguente vengono descritti i tipi di parametri e le relative modalità di raggruppamento.
  - Parametri richiesti: Tutti i parametri richiesti sono specificati senza alcun valore predefinito.
  - Parametri che hanno un valore predefinito specifico per il microservizio (non uguale a un valore predefinito specificato nella documentazione del modulo).
  - Tutti i parametri rimanenti utilizzano `default(omit)` come valore predefinito.

Utilizzo di dizionari multilivello come parametri del modulo

Alcuni moduli Ansible forniti da NetApp utilizzano dizionari multi-livello per i parametri dei moduli (ad esempio gruppi di policy QoS fissi e adattivi).

L'uso `default(omit)` da solo non funziona quando si utilizzano questi dizionari, specialmente quando ne esistono più di uno e si escludono a vicenda.

Se è necessario utilizzare dizionari multilivello come parametri del modulo, è necessario suddividere la funzionalità in più microservizi (ruoli) in modo che ciascuno di essi possa fornire almeno un valore del dizionario di secondo livello per il dizionario pertinente.

Gli esempi seguenti mostrano gruppi di criteri QoS fissi e adattivi suddivisi in due microservizi.

Il primo microservizio contiene valori di gruppo di criteri QoS fissi:

```
fixed_qos_options:
  capacity_shared:           "{{
loop_arg['fixed_qos_options']['capacity_shared']         | default(omit)
}}"
  max_throughput_iops:       "{{
loop_arg['fixed_qos_options']['max_throughput_iops']     | default(omit)
}}"
  min_throughput_iops:       "{{
loop_arg['fixed_qos_options']['min_throughput_iops']     | default(omit)
}}"
  max_throughput_mbps:       "{{
loop_arg['fixed_qos_options']['max_throughput_mbps']     | default(omit)
}}"
  min_throughput_mbps:       "{{
loop_arg['fixed_qos_options']['min_throughput_mbps']     | default(omit)
}}"
```

Il secondo microservizio contiene i valori dei gruppi di criteri QoS adattivi:

```
adaptive_qos_options:
  absolute_min_iops:         "{{
loop_arg['adaptive_qos_options']['absolute_min_iops'] | default(omit) }}"
  expected_iops:             "{{
loop_arg['adaptive_qos_options']['expected_iops']     | default(omit) }}"
  peak_iops:                 "{{
loop_arg['adaptive_qos_options']['peak_iops']         | default(omit) }}"
```

Informazioni sul copyright

Informazioni sul marchio commerciale