ONTAP

NetApp Automation

NetApp
November 18, 2025

Inhalt

ONTAP

Tag 0/1

Überblick über die ONTAP-Tag-0/1-Lösung

Mit der ONTAP Day 0/1 Automatisierungslösung können Sie einen ONTAP Cluster mithilfe von Ansible bereitstellen und konfigurieren. Die Lösung ist erhältlich bei "NetApp Console Automatisierungszentrale"Die

Flexible Implementierungsoptionen für ONTAP

Je nach Ihren Anforderungen können Sie lokale Hardware verwenden oder ONTAP simulieren, um einen ONTAP Cluster mithilfe von Ansible zu implementieren und zu konfigurieren.

On-Premises-Hardware

Sie können diese Lösung mit On-Premises-Hardware mit ONTAP wie einem FAS oder einem AFF System implementieren. Sie müssen eine Linux VM verwenden, um den ONTAP-Cluster mit Ansible zu implementieren und zu konfigurieren.

ONTAP simulieren

Um diese Lösung mit einem ONTAP-Simulator implementieren zu können, müssen Sie die aktuellste Version von Simulate ONTAP von der NetApp Support-Website herunterladen. Simulieren ONTAP ist ein virtueller Simulator für ONTAP Software. Simulieren Sie ONTAP in einem VMware Hypervisor auf einem Windows-, Linux- oder Mac-System. Für Windows- und Linux-Hosts müssen Sie den VMware Workstation-Hypervisor verwenden, um diese Lösung auszuführen. Wenn Sie über ein Mac-Betriebssystem verfügen, verwenden Sie den VMware Fusion-Hypervisor.

Mehrlagiges Design

Das Ansible-Framework vereinfacht die Entwicklung und Wiederverwendung von Automatisierungsausführung und logischen Aufgaben. Das Framework unterscheidet zwischen den Entscheidungsaufgaben (Logikschicht) und den Ausführungsschritten (Ausführungsebene) in der Automatisierung. Wenn Sie verstehen, wie diese Ebenen funktionieren, können Sie die Konfiguration anpassen.

In einem Ansible-„Playbook“ werden verschiedene Aufgaben vom Anfang bis zum Ende ausgeführt. Das `site.yml` Playbook enthält das `logic.yml` Playbook und das `execution.yml` Playbook.

Wenn eine Anfrage ausgeführt wird, ruft das `site.yml` Playbook zuerst in das `logic.yml` Playbook auf und ruft dann das Playbook zur Ausführung der Service-Anfrage auf `execution.yml`.

Sie müssen die logische Schicht des Frameworks nicht verwenden. Die Logikebene bietet Optionen zur Erweiterung der Funktionalität des Frameworks über die hartcodierten Werte für die Ausführung hinaus. Auf diese Weise können Sie die Framework-Funktionen bei Bedarf anpassen.

Logische Ebene

Die Logikschicht besteht aus folgenden Komponenten:

- Das Playbook `logic.yml`
- Logische Aufgabendateien im `logic-tasks` Verzeichnis

Die Logikebene bietet die Möglichkeit für komplexe Entscheidungen, ohne dass eine umfassende benutzerdefinierte Integration erforderlich ist (z. B. eine Verbindung zu ServiceNow). Die Logikebene ist konfigurierbar und liefert die Eingabe zu Microservices.

Die Möglichkeit, die Logikschicht zu umgehen, wird ebenfalls bereitgestellt. Wenn Sie die logische Ebene umgehen möchten, definieren Sie die Variable nicht `logic_operation`. Der direkte Aufruf des `logic.yml` Playbooks ermöglicht es, ein gewisses Maß an Debugging ohne Ausführung durchzuführen. Sie können eine „Debug“-Anweisung verwenden, um zu überprüfen, ob der Wert des `raw_service_request` korrekt ist.

Wichtige Überlegungen:

- Das `logic.yml` Playbook sucht nach der `logic_operation` Variablen. Wenn die Variable in der Anfrage definiert ist, wird eine Aufgabendatei aus dem Verzeichnis geladen `logic-tasks`. Die Task-Datei muss eine .yml-Datei sein. Wenn keine passende Task-Datei vorhanden ist und die `logic_operation` Variable definiert ist, schlägt die Logikebene fehl.
- Der Standardwert der `logic_operation` Variable ist `no-op`. Wenn die Variable nicht explizit definiert ist, wird standardmäßig auf, gesetzt `no-op`, das keine Operationen ausführt.
- Wenn die `raw_service_request` Variable bereits definiert ist, wird die Ausführung zur Ausführungsebene fortgesetzt. Wenn die Variable nicht definiert ist, schlägt die logische Ebene fehl.

Ausführungsebene

Die Ausführungsebene besteht aus folgenden Komponenten:

- Das Playbook `execution.yml`

Die Ausführungsebene führt die API-Aufrufe zum Konfigurieren eines ONTAP-Clusters durch. Das `execution.yml` Playbook setzt voraus, dass die `raw_service_request` Variable bei der Ausführung definiert ist.

Unterstützung für Anpassungen

Sie können diese Lösung auf verschiedene Weise an Ihre Anforderungen anpassen.

Die Anpassungsoptionen umfassen:

- Ändern von Ansible Playbooks
- Hinzufügen von Rollen

Ansible-Dateien anpassen

In der folgenden Tabelle werden die in dieser Lösung enthaltenen anpassbaren Ansible-Dateien beschrieben.

Standort	Beschreibung
`playbooks/inventory/hosts`	Enthält eine einzelne Datei mit einer Liste von Hosts und Gruppen.
`playbooks/group_vars/all/*`	Ansible bietet eine praktische Möglichkeit, Variablen auf mehrere Hosts gleichzeitig anzuwenden. Sie können alle oder alle Dateien in diesem Ordner ändern, einschließlich `cfg.yml`, , , `clusters.yml` `defaults.yml` , `services.yml` `standards.yml` und `vault.yml`.

Standort	Beschreibung
`playbooks/logic-tasks`	Unterstützung von Entscheidungsaufgaben innerhalb von Ansible und Beibehaltung der Trennung von Logik und Ausführung Sie können diesem Ordner Dateien hinzufügen, die dem entsprechenden Dienst entsprechen.
`playbooks/vars/*`	Dynamische Werte, die in Ansible Playbooks und Rollen verwendet werden, um Anpassungen, Flexibilität und Wiederverwendbarkeit von Konfigurationen zu ermöglichen. Bei Bedarf können Sie alle oder alle Dateien in diesem Ordner ändern.

Anpassen von Rollen

Sie können die Lösung auch anpassen, indem Sie Ansible-Rollen, auch Microservices genannt, hinzufügen oder ändern. Weitere Informationen finden Sie unter "Anpassen".

Bereiten Sie sich auf die Verwendung der Lösung für den ONTAP Tag 0/1 vor

Vor der Implementierung der Automatisierungslösung müssen Sie die ONTAP-Umgebung vorbereiten und Ansible installieren und konfigurieren.

Erste Überlegungen zur Planung

Lesen Sie sich die folgenden Anforderungen und Überlegungen durch, bevor Sie diese Lösung zum Bereitstellen eines ONTAP-Clusters verwenden.

Grundvoraussetzungen

Sie müssen die folgenden grundlegenden Anforderungen erfüllen, um diese Lösung verwenden zu können:

- Sie müssen auf die ONTAP-Software zugreifen können – entweder vor Ort oder über einen ONTAP-Simulator.
- Sie müssen wissen, wie Sie die ONTAP Software nutzen.
- Sie müssen wissen, wie Sie die Automatisierungssoftware-Tools von Ansible verwenden können.

Überlegungen zur Planung

Vor der Implementierung dieser Automatisierungslösung müssen Sie folgende Entscheidungen treffen:

- Der Speicherort, an dem der Ansible-Steuerungsknoten ausgeführt werden soll.
- Dem ONTAP System, entweder vor Ort Hardware oder einem ONTAP Simulator.
- Ob Sie eine Anpassung benötigen.

Bereiten Sie das ONTAP-System vor

Unabhängig davon, ob Sie ein lokales ONTAP System nutzen oder ONTAP simulieren, müssen Sie die Umgebung vorbereiten, bevor die Automatisierungslösung implementiert werden kann.

Optional können Sie Simulate ONTAP installieren und konfigurieren

Wenn Sie diese Lösung über einen ONTAP Simulator bereitstellen möchten, müssen Sie Simulate ONTAP herunterladen und ausführen.

Bevor Sie beginnen

- Sie müssen den VMware Hypervisor herunterladen und installieren, den Sie verwenden werden, um Simulate ONTAP auszuführen.
  - Wenn Sie über ein Windows- oder Linux-Betriebssystem verfügen, verwenden Sie VMware Workstation.
  - Wenn Sie ein Mac-Betriebssystem verwenden, verwenden Sie VMware Fusion.

> Wenn Sie ein Mac-Betriebssystem verwenden, benötigen Sie einen Intel-Prozessor.

Schritte

Gehen Sie wie folgt vor, um zwei ONTAP Simulatoren in Ihrer lokalen Umgebung zu installieren:

1. Laden Sie Simulate ONTAP aus dem "NetApp Support Website".

> Obwohl Sie zwei ONTAP Simulatoren installieren, müssen Sie nur eine Kopie der Software herunterladen.

2. Wenn die Anwendung noch nicht ausgeführt wird, starten Sie die VMware-Anwendung.
3. Suchen Sie die heruntergeladene Simulatordatei, und klicken Sie mit der rechten Maustaste, um sie mit der VMware-Anwendung zu öffnen.
4. Legen Sie den Namen der ersten ONTAP-Instanz fest.
5. Warten Sie, bis der Simulator hochgefahren ist, und befolgen Sie die Anweisungen zum Erstellen eines einzelnen Node-Clusters.

   Wiederholen Sie die Schritte für die zweite ONTAP-Instanz.

6. Fügen Sie optional eine vollständige Datenträgerergänzung hinzu.

   Führen Sie in jedem Cluster die folgenden Befehle aus:

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

Der Status des ONTAP Systems

Sie müssen den Anfangsstatus des ONTAP Systems überprüfen, unabhängig davon, ob es sich vor Ort befindet oder über einen ONTAP Simulator ausgeführt wird.

Stellen Sie sicher, dass die folgenden ONTAP-Systemanforderungen erfüllt sind:

- ONTAP ist installiert und läuft ohne Cluster definiert.
- ONTAP wird gebootet und zeigt die IP-Adresse für den Zugriff auf das Cluster an.
- Das Netzwerk ist erreichbar.
- Sie haben Admin-Anmeldedaten.

- Das MOTD-Banner (Message of the Day) wird mit der Managementadresse angezeigt.

Installieren Sie die erforderliche Automatisierungssoftware

Dieser Abschnitt enthält Informationen über die Installation von Ansible und die Vorbereitung der Automatisierungslösung für die Implementierung.

Installation Von Ansible

Ansible kann auf Linux oder Windows Systemen installiert werden.

Die standardmäßige Kommunikationsmethode, die Ansible für die Kommunikation mit einem ONTAP-Cluster verwendet, ist SSH.

Informationen zur Installation von Ansible finden Sie unter"Erste Schritte mit NetApp und Ansible – Installation von Ansible".

Ansible muss auf dem Steuerungsknoten des Systems installiert sein.

Laden Sie die Automatisierungslösung herunter und bereiten Sie sie vor

Sie können die Automatisierungslösung mit den folgenden Schritten herunterladen und für die Implementierung vorbereiten.

1. Laden Sie die "ONTAP - Tag 0/1 Health Checks" Automatisierungslösung über die Web-Benutzeroberfläche der Konsole. Die Lösung ist verpackt als `ONTAP_DAY0_DAY1.zip`Die
2. Extrahieren Sie den ZIP-Ordner und kopieren Sie die Dateien an den gewünschten Speicherort auf dem Steuerknoten in Ihrer Ansible-Umgebung.

Anfängliche Ansible-Framework-Konfiguration

Führen Sie die Erstkonfiguration des Ansible-Frameworks durch:

1. Navigieren Sie zu `playbooks/inventory/group_vars/all`.
2. Entschlüsseln der `vault.yml` Datei:

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

   Wenn Sie zur Eingabe des Vault-Passworts aufgefordert werden, geben Sie das folgende temporäre Passwort ein:

   ```
   NetApp123!
   ```

   „NetApp123!“ ist ein temporäres Kennwort zum Entschlüsseln der `vault.yml` Datei und des entsprechenden Vault-Passworts. Nach der ersten Verwendung müssen Sie die Datei mit Ihrem eigenen Passwort verschlüsseln.

3. Ändern Sie die folgenden Ansible-Dateien:
   - `clusters.yml` - Ändern Sie die Werte in dieser Datei, um Ihre Umgebung anzupassen.
   - `vault.yml` - Nach der Entschlüsselung der Datei, ändern Sie die ONTAP-Cluster, Benutzername und Passwort-Werte, um Ihre Umgebung anzupassen.

- `cfg.yml` - Setzen Sie den Dateipfad für `log2file` und `show_request` unter `cfg` auf `True`, um die anzuzeigen `raw_service_request`.

  Die `raw_service_request` Variable wird in den Protokolldateien und während der Ausführung angezeigt.

> Jede aufgeführte Datei enthält Kommentare mit Anweisungen, wie sie entsprechend Ihren Anforderungen geändert werden kann.

4. Verschlüsseln Sie die Datei erneut `vault.yml`:

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

> Sie werden bei der Verschlüsselung aufgefordert, ein neues Passwort für den Tresor auszuwählen.

5. Navigieren Sie zu `playbooks/inventory/hosts` einem gültigen Python Interpreter und legen Sie ihn fest.
6. Bereitstellung des `framework_test` Service:

   Mit dem folgenden Befehl wird das Modul mit dem `gather_subset` Wert `cluster_identity_info` ausgeführt `na_ontap_info`. Dadurch wird überprüft, ob die Grundkonfiguration korrekt ist und ob Sie mit dem Cluster kommunizieren können.

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

   Führen Sie den Befehl für jedes Cluster aus.

   Wenn der Erfolg erfolgreich ist, sollte die Ausgabe wie im folgenden Beispiel angezeigt werden:

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

Implementieren Sie den ONTAP Cluster mit der Lösung

Nach Abschluss der Vorbereitung und Planung sind Sie bereit, mit der ONTAP Day 0/1 Lösung schnell einen ONTAP Cluster mithilfe von Ansible zu konfigurieren.

Sie können jederzeit während der Schritte in diesem Abschnitt auswählen, ob Sie eine Anforderung testen möchten, anstatt sie tatsächlich auszuführen. Um eine Anforderung zu testen, ändern Sie das `site.yml` Playbook in der Befehlszeile in `logic.yml`.

Der `docs/tutorial-requests.txt` Speicherort enthält die endgültige Version aller Service-Requests, die während dieses Verfahrens verwendet werden. Wenn Sie Schwierigkeiten bei der Ausführung einer Service-Anfrage haben, können Sie die entsprechende Anforderung aus der Datei an den `playbooks/inventory/group_vars/all/tutorial-requests.yml` Speicherort kopieren `tutorial-requests.txt` und die hartcodierten Werte nach Bedarf ändern (IP-Adresse, Aggregatnamen usw.). Die Anforderung sollte dann erfolgreich ausgeführt werden können.

Bevor Sie beginnen

- Ansible muss installiert sein.
- Sie müssen die ONTAP Day 0/1-Lösung heruntergeladen und den Ordner an den gewünschten Speicherort auf dem Ansible-Steuerungsknoten extrahiert haben.
- Der ONTAP-Systemstatus muss die Anforderungen erfüllen und Sie müssen über die erforderlichen Anmeldedaten verfügen.
- Sie müssen alle erforderlichen Aufgaben abgeschlossen haben, die im Abschnitt beschrieben "Vorbereiten"sind.

Die Beispiele in dieser Lösung verwenden „Cluster_01“ und „Cluster_02“ als Namen für die beiden Cluster. Sie müssen diese Werte durch die Namen der Cluster in Ihrer Umgebung ersetzen.

Schritt: Erstkonfiguration des Clusters

In dieser Phase müssen Sie zunächst einige Schritte zur Cluster-Konfiguration durchführen.

Schritte

1. Navigieren Sie zum `playbooks/inventory/group_vars/all/tutorial-requests.yml` Speicherort und prüfen Sie die `cluster_initial` Anforderung in der Datei. Nehmen Sie alle erforderlichen Änderungen an Ihrer Umgebung vor.
2. Erstellen Sie eine Datei im `logic-tasks` Ordner für die Service-Anfrage. Erstellen Sie beispielsweise eine Datei mit dem Namen `cluster_initial.yml`.

   Kopieren Sie die folgenden Zeilen in die neue Datei:

```
- 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. Definieren Sie die `raw_service_request` Variable.

   Sie können eine der folgenden Optionen verwenden, um die Variable in der Datei zu `cluster_initial.yml` definieren `raw_service_request`, die Sie im Ordner erstellt `logic-tasks` haben:

   - **Option 1**: Variable manuell definieren `raw_service_request`.

     Öffnen Sie die `tutorial-requests.yml` Datei mit einem Editor und kopieren Sie den Inhalt von Zeile 11 in Zeile 165. Fügen Sie den Inhalt unter der Variablen in der neuen `cluster_initial.yml` Datei ein `raw service request`, wie in den folgenden Beispielen gezeigt:

Beispiel anzeigen

Beispieldatei `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 }}"
```

- **Option 2**: Verwenden Sie eine Jinja-Vorlage, um die Anforderung zu definieren:

  Sie können auch das folgende Jinja-Vorlagenformat verwenden, um den Wert zu erhalten `raw_service_request`.

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

4. Führen Sie die Erstkonfiguration des Clusters für das erste Cluster durch:

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

Vergewissern Sie sich, dass keine Fehler vorliegen, bevor Sie fortfahren.

5. Wiederholen Sie den Befehl für das zweite Cluster:

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

Vergewissern Sie sich, dass beim zweiten Cluster keine Fehler auftreten.

Wenn Sie zu Beginn der Ansible-Ausgabe nach oben scrollen, sollten Sie die an das Framework gesendete Anforderung sehen, wie im folgenden Beispiel gezeigt:

Beispiel anzeigen


```
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",
```

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

6. Melden Sie sich bei jeder ONTAP-Instanz an und überprüfen Sie, ob die Anforderung erfolgreich war.

Schritt 2: Konfigurieren der Intercluster LIFs

Sie können jetzt die Intercluster LIFs konfigurieren, indem Sie der Anforderung die LIF-Definitionen hinzufügen `cluster_initial` und den Microservice definieren `ontap_interface`.

Die Servicedefinition und die Anforderung arbeiten zusammen, um die Aktion zu bestimmen:

- Wenn Sie eine Service-Anfrage für einen Microservice bereitstellen, der nicht in den Servicedefinitionen enthalten ist, wird die Anforderung nicht ausgeführt.
- Wenn Sie eine Service-Anfrage mit einer oder mehreren in den Servicedefinitionen definierten Microservices bereitstellen, aber aus der Anfrage ausgelassen werden, wird die Anforderung nicht ausgeführt.

Das `execution.yml` Playbook wertet die Dienstdefinition aus, indem die Liste der Microservices in der aufgeführten Reihenfolge durchsucht wird:

- Wenn in der Anfrage ein Eintrag mit einem Wörterbuchschlüssel vorhanden ist, der dem Eintrag in den Microservice-Definitionen entspricht `args`, wird die Anforderung ausgeführt.
- Wenn in der Service-Anfrage kein übereinstimmender Eintrag vorhanden ist, wird die Anforderung

fehlerfrei übersprungen.

Schritte

1. Navigieren Sie zu der `cluster_initial.yml` zuvor erstellten Datei, und ändern Sie die Anforderung, indem Sie den Anforderungsdefinitionen die folgenden Zeilen hinzufügen:

```
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. Führen Sie den Befehl aus:

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

3. Melden Sie sich bei jeder Instanz an, um zu überprüfen, ob die LIFs dem Cluster hinzugefügt wurden:

Beispiel anzeigen

```
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.
```

Die Ausgabe zeigt an, dass die LIFs **nicht** hinzugefügt wurden. Der Grund dafür ist, dass der `ontap_interface` Microservice noch in der Datei definiert werden `services.yml` muss.

4. Vergewissern Sie sich, dass die LIFs der Variable hinzugefügt wurden `raw_service_request`.

Beispiel anzeigen

Im folgenden Beispiel werden die LIFs zur Anforderung hinzugefügt:


```
            "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. Definieren Sie den `ontap_interface` Microservice unter `cluster_initial` in der `services.yml` Datei.

   Kopieren Sie die folgenden Zeilen in die Datei, um den Microservice zu definieren:

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

6. Nachdem nun der `ontap_interface` Microservice in der Anfrage und der Datei definiert wurde `services.yml`, führen Sie die Anforderung erneut aus:

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

7. Loggen Sie sich bei jeder ONTAP Instanz ein und überprüfen Sie, ob die LIFs hinzugefügt wurden.

Schritt 3: Optional mehrere Cluster konfigurieren

Bei Bedarf können Sie mehrere Cluster in derselben Anforderung konfigurieren. Sie müssen beim Definieren der Anforderung für jedes Cluster Variablennamen angeben.

Schritte

1. Fügen Sie einen Eintrag für das zweite Cluster in der Datei hinzu `cluster_initial.yml`, um beide Cluster in derselben Anforderung zu konfigurieren.

   Im folgenden Beispiel wird das Feld angezeigt `ontap_aggr`, nachdem der zweite Eintrag hinzugefügt wurde.

```
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. Übernehmen Sie die Änderungen für alle anderen Elemente unter `cluster_initial`.
3. Fügen Sie Cluster-Peering zur Anforderung hinzu, indem Sie die folgenden Zeilen in die Datei kopieren:

```
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. Ansible-Anforderung ausführen:

```
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>
```

Schritt 4: Anfängliche SVM-Konfiguration

An dieser Stelle des Verfahrens konfigurieren Sie die SVMs im Cluster.

Schritte

1. Anforderung `tutorial-requests.yml` zur Konfiguration einer SVM- und SVM-Peer-Beziehung aktualisieren `svm_initial`

   Sie müssen Folgendes konfigurieren:

   - Das SVM

- Die SVM-Peer-Beziehung
- Die SVM-Schnittstelle für jede SVM

2. Aktualisieren Sie die Variablendefinitionen in den `svm_initial` Anforderungsdefinitionen. Sie müssen die folgenden Variablendefinitionen ändern:
   - `cluster_name`
   - `vserver_name`
   - `peer_cluster_name`
   - `peer_vserver`

   Um die Definitionen zu aktualisieren, entfernen Sie das **'{}'** nach `req_details` für die `svm_initial` Definition und fügen Sie die korrekte Definition hinzu.

3. Erstellen Sie eine Datei im `logic-tasks` Ordner für die Service-Anfrage. Erstellen Sie beispielsweise eine Datei mit dem Namen `svm_initial.yml`.

   Kopieren Sie die folgenden Zeilen in die Datei:

```
- 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. Definieren Sie die `raw_service_request` Variable.

   Sie können eine der folgenden Optionen verwenden, um die Variable für `svm_initial` im Ordner zu `logic-tasks` definieren `raw_service_request`:

   - **Option 1**: Variable manuell definieren `raw_service_request`.

     Öffnen Sie die `tutorial-requests.yml` Datei mit einem Editor und kopieren Sie den Inhalt von

Zeile 179 in Zeile 222. Fügen Sie den Inhalt unter der Variablen in der neuen `svm_initial.yml` Datei ein `raw service request`, wie in den folgenden Beispielen gezeigt:

Beispiel anzeigen

Beispieldatei `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
```

- **Option 2**: Verwenden Sie eine Jinja-Vorlage, um die Anforderung zu definieren:

  Sie können auch das folgende Jinja-Vorlagenformat verwenden, um den Wert zu erhalten `raw_service_request`.

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

5. Anforderung ausführen:

```
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. Melden Sie sich bei jeder ONTAP Instanz an und validieren Sie die Konfiguration.
7. Fügen Sie die SVM-Schnittstellen hinzu.

   Definieren Sie den `ontap_interface` Dienst unter `svm_initial` in der `services.yml` Datei und führen Sie die Anforderung erneut aus:

```
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. Loggen Sie sich bei jeder ONTAP Instanz ein und überprüfen Sie, ob die SVM-Schnittstellen konfiguriert sind.

Schritt 5: Optional können Sie eine Service-Anfrage dynamisch definieren

In den vorherigen Schritten ist die `raw_service_request` Variable hartcodiert. Dies ist nützlich für Lernen, Entwicklung und Tests. Sie können auch eine Serviceanfrage dynamisch generieren.

Der folgende Abschnitt bietet eine Option zum dynamischen Erstellen des erforderlichen `raw_service_request`, wenn Sie es nicht in höhere Systeme integrieren möchten.

- Wenn die `logic_operation` Variable im Befehl nicht definiert ist, importiert die `logic.yml` Datei keine Datei aus dem `logic-tasks` Ordner. Das bedeutet, dass die `raw_service_request` außerhalb von Ansible definiert und bei der Ausführung dem Framework zur Verfügung gestellt werden muss.
- Ein Aufgabendateiname im `logic-tasks` Ordner muss mit dem Wert der Variablen ohne die Erweiterung .yml übereinstimmen `logic_operation`.
- Die Aufgabendateien im `logic-tasks` Ordner definieren dynamisch ein `raw_service_request`. die einzige Voraussetzung ist, dass ein gültiges `raw_service_request` als letzte Aufgabe in der entsprechenden Datei definiert wird.

Dynamische Definition von Service-Anfragen

Es gibt mehrere Möglichkeiten, eine logische Aufgabe anzuwenden, um eine Service-Anfrage dynamisch zu definieren. Einige dieser Optionen sind unten aufgeführt:

- Verwenden einer Ansible-Aufgabendatei aus dem `logic-tasks` Ordner
- Aufrufen einer benutzerdefinierten Rolle, die Daten zurückgibt, die für die Konvertierung in eine Variable geeignet `raw_service_request` sind.
- Aufruf eines weiteren Tools außerhalb der Ansible-Umgebung, um die erforderlichen Daten bereitzustellen Beispielsweise ein REST-API-Aufruf an Active IQ Unified Manager.

Mit den folgenden Beispielbefehlen können Sie mithilfe der Datei für jedes Cluster eine Service-Anfrage dynamisch definieren `tutorial-requests.yml`:

```
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
```

Schritt 6: Implementierung der ONTAP-Lösung Tag 0/1

In dieser Phase sollten Sie bereits Folgendes abgeschlossen haben:

- Alle Dateien in wurden entsprechend Ihren Anforderungen überprüft und geändert `playbooks/inventory/group_vars/all`. Jede Datei enthält detaillierte Kommentare, mit denen Sie die Änderungen vornehmen können.
- Erforderliche Aufgabendateien wurden dem Verzeichnis hinzugefügt `logic-tasks`.
- Alle erforderlichen Datendateien wurden dem Verzeichnis hinzugefügt `playbook/vars`.

Verwenden Sie die folgenden Befehle, um die ONTAP Day 0/1-Lösung bereitzustellen und den Zustand Ihrer Bereitstellung zu überprüfen:

Zu diesem Zeitpunkt sollten Sie die Datei bereits entschlüsselt und geändert haben `vault.yml` und sie muss mit Ihrem neuen Passwort verschlüsselt werden.

- Führen Sie den ONTAP-Tag-0-Service aus:

```
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>
```

- Führen Sie den ONTAP Day 1-Service aus:

```
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>
```

- Clusterweite Einstellungen anwenden:

```
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>
```

- Führen Sie Zustandsprüfungen durch:

```
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>
```

Passen Sie die ONTAP Day 0/1-Lösung an

Zur Anpassung der ONTAP-Day-0/1-Lösung an Ihre Anforderungen können Sie Ansible-

Rollen hinzufügen oder ändern.

Rollen stellen die Microservices im Ansible-Framework dar. Jeder Microservice führt einen Vorgang durch. Beispielsweise ist ONTAP Tag 0 ein Service, der mehrere Microservices umfasst.

Ansible-Rollen hinzufügen

Sie können Ansible-Rollen hinzufügen, um die Lösung an Ihre Umgebung anzupassen. Erforderliche Rollen werden durch Servicedefinitionen im Ansible Framework definiert.

Eine Rolle muss die folgenden Anforderungen erfüllen, um als Microservice verwendet werden zu können:

- Akzeptieren Sie eine Liste der Argumente in der `args` Variablen.
- Nutzen Sie die Ansible-Struktur „Block, Rescue, Always“ mit bestimmten Anforderungen für jeden Block.
- Verwenden Sie ein einzelnes Ansible-Modul und definieren Sie eine einzelne Aufgabe innerhalb des Blocks.
- Implementieren Sie alle verfügbaren Modulparameter gemäß den in diesem Abschnitt beschriebenen Anforderungen.

Erforderliche Microservice-Struktur

Jede Rolle muss die folgenden Variablen unterstützen:

- `mode`: Wenn der Modus auf die Rolle eingestellt ist `test`, versucht der zu importieren, der `test.yml` zeigt, was die Rolle tut, ohne sie tatsächlich auszuführen.

  Dies ist aufgrund bestimmter Abhängigkeiten nicht immer möglich.

- `status`: Der Gesamtstatus der Ausführung des Playbooks. Wenn der Wert nicht auf die Rolle gesetzt `success` ist, wird nicht ausgeführt.
- `args` : Eine Liste rollenspezifischer Wörterbücher mit Schlüsseln, die den Rollenparameternamen entsprechen.
- `global_log_messages`: Sammelt Protokollmeldungen während der Ausführung des Playbooks. Bei jeder Ausführung der Rolle wird ein Eintrag generiert.
- `log_name`: Der Name, der auf die Rolle innerhalb der Einträge verweist `global_log_messages`.
- `task_descr`: Eine kurze Beschreibung dessen, was die Rolle tut.
- `service_start_time`: Der Zeitstempel, der verwendet wird, um die Zeit zu verfolgen, zu der jede Rolle ausgeführt wird.
- `playbook_status`: Der Status des Ansible-Playbooks.
- `role_result`: Die Variable, die die Rollenausgabe enthält und in jeder Nachricht innerhalb der Einträge enthalten `global_log_messages` ist.

Beispiel für eine Rollenstruktur

Das folgende Beispiel zeigt die grundlegende Struktur einer Rolle, die einen Microservice implementiert. Sie müssen die Variablen in diesem Beispiel für Ihre Konfiguration ändern.

Beispiel anzeigen

Grundlegende Rollenstruktur:

```
- 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 ] }}"
```

Variablen, die in der Beispielrolle verwendet werden:

- `<NAME>`: Ein austauschbarer Wert, der für jeden Microservice bereitgestellt werden muss.
- `<LOG_NAME>`: Der Kurzname der Rolle, die für Protokollierungszwecke verwendet wird. `ONTAP_VOLUME`Beispiel: .
- `<TASK_DESCRIPTION>`: Eine kurze Beschreibung dessen, was der Microservice tut.
- `<MODULE_NAME>`: Der Ansible-Modulname für die Aufgabe.

> Im Playbook der obersten Ebene `execute.yml` wird die Sammlung angegeben `netapp.ontap`. Wenn das Modul Teil der Sammlung ist `netapp.ontap`, muss der Modulname nicht vollständig angegeben werden.

- `<MODULE_SPECIFIC_PARAMETERS>`: Ansible-Modulparameter, die spezifisch für das Modul sind, das zur Implementierung des Microservices verwendet wird. In der folgenden Liste werden die Parametertypen und deren Gruppierung beschrieben.
  - Erforderliche Parameter: Alle erforderlichen Parameter werden ohne Standardwert angegeben.
  - Parameter, die einen für den Microservice spezifischen Standardwert haben (nicht der gleiche Wert wie ein in der Moduldokumentation spezifizierter Standardwert).
  - Alle verbleibenden Parameter werden als Standardwert verwendet `default(omit)`.

Verwendung von mehrstufigen Wörterbüchern als Modulparameter

Einige von NetApp bereitgestellte Ansible-Module verwenden mehrstufige Wörterbücher für Modulparameter (z. B. feste und adaptive QoS-Richtliniengruppen).

Allein zu verwenden `default(omit)` funktioniert nicht, wenn diese Wörterbücher verwendet werden, besonders wenn es mehrere gibt und sie sich gegenseitig ausschließen.

Wenn Sie Multi-Level-Wörterbücher als Modulparameter verwenden müssen, sollten Sie die Funktionalität in mehrere Microservices (Rollen) aufteilen, so dass jeder garantiert mindestens einen Second-Level-Wörterbuchwert für das jeweilige Wörterbuch liefern kann.

Die folgenden Beispiele zeigen feste und anpassungsfähige QoS-Richtliniengruppen, die sich auf zwei Microservices verteilen.

Der erste Microservice enthält feste QoS-Richtliniengruppenwerte:

```
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)
}}"
```

Der zweite Microservice enthält die Werte der adaptiven QoS-Richtliniengruppe:

```
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) }}"
```