



ONTAP

NetApp Automation

NetApp
November 18, 2025

목차

ONTAP	1
1일 차	1
ONTAP Day 0/1 솔루션 개요	1
ONTAP Day 0/1 솔루션을 사용할 준비를 하십시오	3
솔루션을 사용하여 ONTAP 클러스터를 구축합니다	6
ONTAP Day 0/1 솔루션을 사용자 지정합니다	27

ONTAP

1일 차

ONTAP Day 0/1 솔루션 개요

ONTAP 0/1일 자동화 솔루션을 사용하면 Ansible을 사용하여 ONTAP 클러스터를 배포하고 구성할 수 있습니다. 해당 솔루션은 다음에서 제공됩니다. "[NetApp Console 자동화 허브](#)".

유연한 ONTAP 구축 옵션

요구사항에 따라 온프레미스 하드웨어를 사용하거나 ONTAP를 시뮬레이션하여 Ansible을 사용하여 ONTAP 클러스터를 구축 및 구성할 수 있습니다.

온프레미스 하드웨어

FAS 또는 AFF 시스템과 같이 ONTAP을 실행하는 온프레미스 하드웨어를 사용하여 이 솔루션을 배포할 수 있습니다. Ansible을 사용하여 ONTAP 클러스터를 구축하고 구성하려면 Linux VM을 사용해야 합니다.

ONTAP 시뮬레이션

ONTAP 시뮬레이터를 사용하여 이 솔루션을 구축하려면 NetApp Support 사이트에서 Simulate ONTAP의 최신 버전을 다운로드해야 합니다. Simulate ONTAP는 ONTAP 소프트웨어용 가상 시뮬레이터입니다. ONTAP는 Windows, Linux 또는 Mac 시스템의 VMware 하이퍼바이저에서 실행됩니다. Windows 및 Linux 호스트의 경우 VMware Workstation 하이퍼바이저를 사용하여 이 솔루션을 실행해야 합니다. Mac OS를 사용하는 경우 VMware Fusion 하이퍼바이저를 사용하십시오.

레이어드 디자인

Ansible 프레임워크는 자동화 실행 및 논리 작업의 개발과 재사용을 간소화합니다. 프레임워크는 의사 결정 작업(논리 계층)과 자동화의 실행 단계(실행 계층)를 구별합니다. 이러한 레이어의 작동 방식을 이해하면 구성을 사용자 정의할 수 있습니다.

Ansible "플레이북"은 처음부터 끝까지 일련의 작업을 실행합니다. `site.yml` 플레이북에는 플레이북 및 `execution.yml` 플레이북이 포함되어 `logic.yml` 있습니다.

요청이 실행되면 `site.yml` Playbook은 먼저 Playbook을 호출한 `logic.yml` 다음 `execution.yml` Playbook을 호출하여 서비스 요청을 실행합니다.

프레임워크의 논리 계층을 사용할 필요는 없습니다. 논리 계층은 실행을 위해 하드 코드된 값 이상으로 프레임워크의 기능을 확장하는 옵션을 제공합니다. 따라서 필요한 경우 프레임워크 기능을 사용자 지정할 수 있습니다.

로직 레이어

논리 계층은 다음과 같이 구성됩니다.

- 플레이북 `logic.yml`
- 디렉토리 내의 논리 작업 파일 `logic-tasks`

논리 계층은 중대한 맞춤형 통합(예: ServiceNow에 연결) 없이도 복잡한 의사 결정을 위한 기능을 제공합니다. 논리 계층은 구성 가능하며 마이크로서비스에 입력을 제공합니다.

논리 계층을 우회하는 기능도 제공됩니다. 논리 계층을 무시하려면 변수를 정의하지 마십시오 `logic_operation`. 플레이북을 직접 호출하면 `logic.yml` 실행 없이 일부 수준의 디버깅을 수행할 수 있습니다. "debug" 문을 사용하여 의 값이 올바른지 확인할 수 `raw_service_request` 있습니다.

중요 고려 사항:

- `logic.yml` 플레이북에서 `logic_operation` 변수를 검색합니다. 변수가 요청에 정의되어 있으면 디렉터리에서 작업 파일을 `logic-tasks` 로드합니다. 작업 파일은 `.yml` 파일이어야 합니다. 일치하는 작업 파일이 없고 변수가 정의된 경우 `logic_operation` 논리 계층이 실패합니다.
- 변수의 `logic_operation` 기본값은 `no-op`입니다. 변수가 명시적으로 정의되지 않은 경우 기본값은 로 설정되며, 이 경우 `no-op` 아무 작업도 실행되지 않습니다.
- 변수가 이미 정의되어 있으면 `raw_service_request` 실행이 실행 계층으로 진행됩니다. 변수가 정의되지 않으면 논리 계층이 실패합니다.

실행 계층

실행 계층은 다음과 같이 구성됩니다.

- 플레이북 `execution.yml`

실행 계층은 ONTAP 클러스터를 구성하기 위해 API를 호출합니다. `execution.yml` Playbook을 사용하려면 실행 시 변수를 정의해야 `raw_service_request` 합니다.

사용자 지정 지원

요구 사항에 따라 다양한 방법으로 이 솔루션을 사용자 지정할 수 있습니다.

사용자 지정 옵션은 다음과 같습니다.

- Ansible 플레이북 수정
- 역할을 추가하는 중입니다

Ansible 파일 사용자 지정

다음 표에서는 이 솔루션에 포함된 사용자 지정 가능한 Ansible 파일을 설명합니다.

위치	설명
<code>playbooks/inventory/hosts</code>	호스트 및 그룹 목록이 포함된 단일 파일을 포함합니다.
<code>playbooks/group_vars/all/*</code>	Ansible을 사용하면 여러 호스트에 변수를 한 번에 간편하게 적용할 수 있습니다. 이 폴더에 있는 <code>clusters.yml</code> , <code>defaults.yml</code> , <code>services.yml</code> , <code>standards.yml</code> 및 <code>vault.yml</code> 를 포함한 모든 파일을 수정할 수 <code>cfg.yml</code> 있습니다.
<code>playbooks/logic-tasks</code>	Ansible 내에서 의사 결정 작업을 지원하고 논리와 실행의 분리를 유지합니다. 관련 서비스에 해당하는 파일을 이 폴더에 추가할 수 있습니다.
<code>playbooks/vars/*</code>	Ansible 플레이북 및 역할에 사용되는 동적인 가치를 제공하여 커스터마이징, 유연성 및 구성 재사용을 지원합니다. 필요한 경우 이 폴더의 모든 파일을 수정할 수 있습니다.

역할을 사용자 정의합니다

마이크로서비스라고도 하는 Ansible 역할을 추가 또는 변경하여 솔루션을 사용자 지정할 수도 있습니다. 자세한 내용은 [참조하십시오 "사용자 정의"](#).

ONTAP Day 0/1 솔루션을 사용할 준비를 하십시오

자동화 솔루션을 구축하기 전에 ONTAP 환경을 준비하고 Ansible을 설치 및 구성해야 합니다.

초기 계획 고려 사항

이 솔루션을 사용하여 ONTAP 클러스터를 구축하기 전에 다음 요구 사항 및 고려 사항을 검토해야 합니다.

기본 요구 사항

이 솔루션을 사용하려면 다음 기본 요구 사항을 충족해야 합니다.

- 온프레미스 또는 ONTAP 시뮬레이터를 통해 ONTAP 소프트웨어에 액세스할 수 있어야 합니다.
- ONTAP 소프트웨어 사용 방법을 알고 있어야 합니다.
- Ansible 자동화 소프트웨어 툴을 사용하는 방법을 알아야 합니다.

계획 고려 사항

이 자동화 솔루션을 배포하기 전에 다음을 결정해야 합니다.

- Ansible 제어 노드를 실행할 위치입니다.
- ONTAP 시스템의 온프레미스 하드웨어 또는 ONTAP 시뮬레이터입니다.
- 사용자 지정 필요 여부.

ONTAP 시스템을 준비합니다

온프레미스 ONTAP 시스템을 사용하든 ONTAP를 시뮬레이션하든 상관없이 자동화 솔루션을 구축하기 전에 환경을 준비해야 합니다.

필요에 따라 **Simulate ONTAP**를 설치 및 구성합니다

ONTAP 시뮬레이터를 통해 이 솔루션을 배포하려면 Simulate ONTAP를 다운로드하여 실행해야 합니다.

시작하기 전에

- Simulate ONTAP를 실행하는 데 사용할 VMware 하이퍼바이저를 다운로드하고 설치해야 합니다.
 - Windows 또는 Linux OS를 사용하는 경우 VMware Workstation을 사용하십시오.
 - Mac OS를 사용하는 경우 VMware Fusion를 사용합니다.



Mac OS를 사용하는 경우 Intel 프로세서가 있어야 합니다.

단계

로컬 환경에 두 개의 ONTAP 시뮬레이터를 설치하려면 다음 절차를 따르십시오.

1. 에서 Simulate ONTAP 를 ["NetApp Support 사이트"](#) 다운로드합니다.



두 개의 ONTAP 시뮬레이터를 설치하더라도 소프트웨어 복사본 하나만 다운로드하면 됩니다.

2. 아직 실행 중이 아닌 경우 VMware 애플리케이션을 시작합니다.
3. 다운로드한 시뮬레이터 파일을 찾아 마우스 오른쪽 버튼을 클릭하여 VMware 애플리케이션에서 엽니다.
4. 첫 번째 ONTAP 인스턴스의 이름을 설정합니다.
5. 시뮬레이터가 부팅될 때까지 기다린 다음 지침에 따라 단일 노드 클러스터를 생성합니다.

두 번째 ONTAP 인스턴스에 대해 이 단계를 반복합니다.

6. 필요에 따라 전체 디스크 보안을 추가합니다.

각 클러스터에서 다음 명령을 실행합니다.

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

ONTAP 시스템의 상태입니다

사내 시스템 또는 ONTAP 시뮬레이터를 통해 ONTAP 시스템의 초기 상태를 확인해야 합니다.

다음 ONTAP 시스템 요구 사항이 충족되는지 확인합니다.

- 아직 정의된 클러스터가 없는 상태에서 ONTAP가 설치되고 실행 중입니다.
- ONTAP가 부팅되고 클러스터에 액세스하기 위한 IP 주소가 표시됩니다.
- 네트워크에 연결할 수 있습니다.
- 관리자 자격 증명이 있습니다.
- MOTD(Message of the Day) 배너에 관리 주소가 표시됩니다.

필요한 자동화 소프트웨어를 설치합니다

이 섹션에서는 Ansible을 설치하고 구축을 위한 자동화 솔루션을 준비하는 방법에 대한 정보를 제공합니다.

Ansible 설치

Linux 또는 Windows 시스템에 Ansible을 설치할 수 있습니다.

Ansible이 ONTAP 클러스터와 통신하기 위해 사용하는 기본 통신 방법은 SSH입니다.

Ansible 설치에 대해서는 [을 "NetApp 및 Ansible 시작하기: Ansible을 설치하십시오"](#)참조하십시오.



시스템의 제어 노드에 Ansible을 설치해야 합니다.

자동화 솔루션을 다운로드하고 준비합니다

다음 단계에 따라 배포를 위한 자동화 솔루션을 다운로드하고 준비할 수 있습니다.

1. 다운로드 "**ONTAP - 1일차 및 1일차; 상태 점검**" 콘솔 웹 UI를 통한 자동화 솔루션. 솔루션은 다음과 같이 패키지가 됩니다. `ONTAP_DAY0_DAY1.zip`.
2. zip 폴더의 압축을 풀고 Ansible 환경 내의 제어 노드에서 원하는 위치로 파일을 복사합니다.

초기 **Ansible** 프레임워크 구성

Ansible 프레임워크의 초기 구성 수행:

1. 로 이동합니다 `playbooks/inventory/group_vars/all`.

2. 파일 암호 해독 `vault.yml`:

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

볼트 암호를 입력하라는 메시지가 나타나면 다음 임시 암호를 입력합니다.

```
NetApp123!
```



"NetApp123!"는 파일과 해당 볼트 암호를 해독하기 위한 임시 암호입니다. `vault.yml` 처음 사용한 후에는 * 자신의 암호를 사용하여 * 파일을 암호화해야 합니다.

3. 다음 Ansible 파일을 수정합니다.

- `clusters.yml` - 환경에 맞게 이 파일의 값을 수정합니다.
- `vault.yml` - 파일을 해독한 후 사용자 환경에 맞게 ONTAP 클러스터, 사용자 이름 및 암호 값을 수정합니다.
- `cfg.yml` - 에 대한 파일 경로를 `log2file` 설정하고 `cfg` 을(를) 표시하려면 을 `raw_service_request` (를 `True`) 로 설정합니다 `show_request`.

```
`raw_service_request`로그 파일과 실행 중에 변수가 표시됩니다.
```



나열된 각 파일에는 요구 사항에 따라 수정하는 방법에 대한 지침이 포함된 설명이 포함되어 있습니다.

4. 파일 다시 암호화 `vault.yml`:

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



암호화 시 볼트에 대한 새 암호를 선택하라는 메시지가 표시됩니다.

5. 유효한 Python 인터프리터를 찾아 `playbooks/inventory/hosts` 설정합니다.

6. 서비스 구축 `framework_test`:

다음 명령을 실행하면 `na_ontap_info` 값이 인 `cluster_identity_info` 모듈이 `gather_subset`

실행됩니다. 이렇게 하면 기본 구성이 올바른지 확인하고 클러스터와 통신할 수 있는지 확인합니다.

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

각 클러스터에 대해 명령을 실행합니다.

성공하면 다음 예와 유사한 출력이 표시됩니다.

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

솔루션을 사용하여 **ONTAP** 클러스터를 구축합니다

준비 및 계획을 완료하면 ONTAP Day 0/1 솔루션을 사용하여 Ansible을 사용하여 ONTAP 클러스터를 신속하게 구성할 수 있습니다.

이 섹션의 단계 중 언제든지 요청을 실제로 실행하는 대신 테스트할 수 있습니다. 요청을 테스트하려면 명령줄에서 플레이북을 로 logic.yml 변경합니다 site.yml.



docs/tutorial-requests.txt`위치에는 이 절차 전반에 걸쳐 사용된 모든 서비스 요청의 최종 버전이 포함되어 있습니다. 서비스 요청을 실행하는 데 문제가 있는 경우 파일에서 `playbooks/inventory/group_vars/all/tutorial-requests.yml` 위치로 관련 요청을 복사하고 필요에 따라 하드 코딩된 값(IP 주소, 집계 이름 등)을 수정할 수 tutorial-requests.txt 있습니다. 그러면 요청을 성공적으로 실행할 수 있습니다.

시작하기 전에

- Ansible을 설치해야 합니다.
- ONTAP DAY 0/1 솔루션을 다운로드하고 Ansible 제어 노드의 원하는 위치에 폴더를 추출해야 합니다.
- ONTAP 시스템 상태는 요구사항을 충족해야 하고 필요한 자격 증명이 있어야 합니다.
- 섹션에 요약된 모든 필수 작업을 완료해야 **"준비"**합니다.



이 솔루션의 예에서는 두 클러스터의 이름으로 "Cluster_01"과 "Cluster_02"를 사용합니다. 이러한 값은 사용자 환경에서 클러스터의 이름으로 대체해야 합니다.

1단계: 초기 클러스터 구성

이 단계에서는 몇 가지 초기 클러스터 구성 단계를 수행해야 합니다.

단계

1. 위치로 `playbooks/inventory/group_vars/all/tutorial-requests.yml` 이동하여 `cluster_initial` 파일의 요청을 검토합니다. 사용자 환경에 필요한 변경 작업을 수행합니다.
2. 서비스 요청에 대한 폴더에 파일을 만듭니다 `logic-tasks`. 예를 들어, 이라는 파일을 `cluster_initial.yml` 만듭니다.

다음 줄을 새 파일에 복사합니다.

```
- 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. ``raw_service_request`` 변수를 정의합니다.

다음 옵션 중 하나를 사용하여 폴더에 만든 파일의 `logic-tasks` 변수를 `cluster_initial.yml` 정의할 수 `raw_service_request` 있습니다.

- * 옵션 1 *: 변수를 수동으로 `raw_service_request` 정의합니다.

```
`tutorial-requests.yml` 편집기를 사용하여 파일을 열고 11줄의 콘텐츠를 165줄로
복사합니다. 다음 예제와 같이 새 파일의 변수 `cluster_initial.yml` 아래에 내용을
붙여 넣습니다 `raw service request`.
```

```
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 #-----
10 #-----
11 service: cluster_initial
12 operation: create
13 std_name: none
14 req_details:
15
16   ontap_aggr:
17     - hostname: "{{ cluster_name }}"
18       disk_count: 24
19       name: n01_aggr1
20       nodes: "{{ cluster_name }}-01"
```

`cluster_initial.yml` 예제 파일:

```
- 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:
```



```

home_node:                "{{ cluster_name }}-01"
home_port:                e0c
ipspace:                  Default
use_rest:                 never

- hostname:               "{{ peer_cluster_name }}"
  vservers:               "{{ 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 }}"
  vservers:               "{{ 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 }}"

```

◦ * 옵션 2 * : Jinja 템플릿을 사용하여 요청을 정의합니다.

다음 Jinja 템플릿 형식을 사용하여 값을 가져올 수도 raw_service_request 있습니다.

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

4. 첫 번째 클러스터에 대한 초기 클러스터 구성을 수행합니다.

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

계속하기 전에 오류가 없는지 확인하십시오.

5. 두 번째 클러스터에 대해 명령을 반복합니다.

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

두 번째 클러스터에 오류가 없는지 확인합니다.

Ansible 출력 시작 부분으로 스크롤하면 다음 예제에 표시된 것처럼 프레임워크로 전송된 요청을 볼 수 있습니다.

- 서비스 요청에 일치하는 항목이 없으면 오류 없이 요청을 건너뛵니다.

단계

1. `cluster_initial.yml` 이전에 만든 파일로 이동하고 요청 정의에 다음 행을 추가하여 요청을 수정합니다.

```

ontap_interface:
- hostname:                "{{ cluster_name }}"
  vservers:                "{{ 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 }}"
  vservers:                "{{ 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 }}"
  vservers:                "{{ 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 }}"
  vservers:                "{{ 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. 다음 명령을 실행합니다.

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

3. 각 인스턴스에 로그인하여 LIF가 클러스터에 추가되었는지 확인합니다.

예제 보기

```
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.
```

출력은 LIF가 추가되지 않았음을 보여줍니다 *. 이는 마이크로 서비스를 services.yml 파일에 정의해야 하기 ontap_interface 때문입니다.

4. LIF가 변수에 추가되었는지 확인 raw_service_request

다음 예제는 LIF가 요청에 추가되었음을 보여줍니다.

```
"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. `ontap_interface` `services.yml` 파일에서 마이크로서비스를 `cluster_initial` 정의합니다.

마이크로서비스를 정의하려면 파일에 다음 줄을 복사합니다.

```

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

```

6. 이제 마이크로 서비스가 요청과 `services.yml` 파일에 정의되었으므로 `ontap_interface` 요청을 다시 실행합니다.

```

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

```

7. 각 ONTAP 인스턴스에 로그인하여 LIF가 추가되었는지 확인합니다.

3단계: 필요에 따라 여러 클러스터를 구성합니다

필요한 경우 동일한 요청으로 여러 클러스터를 구성할 수 있습니다. 요청을 정의할 때 각 클러스터에 대해 가변 이름을 제공해야 합니다.

단계

1. 파일에 두 번째 클러스터에 대한 항목을 `cluster_initial.yml` 추가하여 동일한 요청에서 두 클러스터를 구성합니다.

다음 예제에서는 `ontap_aggr` 두 번째 항목이 추가된 후 필드를 표시합니다.

```

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. 에서 다른 모든 항목에 대한 변경 내용을 `cluster_initial` 적용합니다.

3. 다음 줄을 파일에 복사하여 요청에 클러스터 피어링을 추가합니다.

```

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 요청 실행:

```

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>

```

4단계: 초기 SVM 구성

절차의 이 단계에서는 클러스터에서 SVM을 구성합니다.

단계

1. `svm_initial`` 파일에서 요청을 `tutorial-requests.yml` 업데이트하여 SVM 및 SVM 피어 관계 구성

다음을 구성해야 합니다.

- SVM은

- SVM 피어 관계
- 각 SVM의 SVM 인터페이스

2. 요청 정의에서 변수 정의를 `svm_initial` 업데이트합니다. 다음 변수 정의를 수정해야 합니다.

- `cluster_name`
- `vserver_name`
- `peer_cluster_name`
- `peer_vserver`

정의를 업데이트하려면 정의에 대한 `svm_initial` 뒤에 *'{}'* 를 `req_details` 제거하고 올바른 정의를 추가하십시오.

3. 서비스 요청에 대한 폴더에 파일을 만듭니다 `logic-tasks`. 예를 들어, 이라는 파일을 `svm_initial.yml` 만듭니다.

다음 줄을 파일에 복사합니다.

```
- 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. `raw_service_request` 변수를 정의합니다.

다음 옵션 중 하나를 사용하여 `logic-tasks` 폴더에서 변수를 `svm_initial` 정의할 수 `raw_service_request` 있습니다.

- * 옵션 1 *: 변수를 수동으로 `raw_service_request` 정의합니다.

`tutorial-requests.yml` 편집기를 사용하여 파일을 열고 179줄의 내용을 222줄로 복사합니다. 다음 예제와 같이 새 파일의 변수 `svm_initial.yml` 아래에 내용을 붙여 넣습니다 `raw service request`.

```
177
178 svm_initial:
179   service:          svm_initial
180   operations:
181     std_name:       none
182     req_details:
183
184     ontap_vserver:
185       - hostname:   "{{ cluster_name }}"
186         name:        "{{ vserver_name }}"
187         root_volume_aggregate: n01_aggr1
188
189       - hostname:   "{{ peer_cluster_name }}"
190         name:        "{{ peer_vserver }}"
191         root_volume_aggregate: n01_aggr1
192
```

`svm_initial.yml` 예제 파일:

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

- name: Include data files
  ansible.builtin.include_vars:
    file:  "{{ data_file_name }}.yaml"
  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

```

◦ * 옵션 2 * : Jinja 템플릿을 사용하여 요청을 정의합니다.

다음 Jinja 템플릿 형식을 사용하여 값을 가져올 수도 raw_service_request 있습니다.

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

5. 다음 요청을 실행합니다.

```

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. 각 ONTAP 인스턴스에 로그인하고 구성을 검증합니다.

7. SVM 인터페이스를 추가합니다.

`ontap_interface` `services.yml` 파일에서 서비스를 `svm_initial` 정의하고 요청을 다시 실행합니다.

```
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. 각 ONTAP 인스턴스에 로그인하고 SVM 인터페이스가 구성되었는지 확인합니다.

5단계: 필요에 따라 서비스 요청을 동적으로 정의합니다

이전 단계에서 `raw_service_request` 변수는 하드 코딩됩니다. 이 기능은 학습, 개발 및 테스트에 유용합니다. 서비스 요청을 동적으로 생성할 수도 있습니다.

다음 섹션에서는 Required 를 상위 시스템과 통합하지 않으려는 경우 동적으로 생성할 수 있는 옵션을 `raw_service_request` 제공합니다.



- 명령에서 변수가 정의되지 않은 `logic.yml` 경우 `logic_operation` 파일은 폴더에서 파일을 가져오지 `logic-tasks` 않습니다. 이는 `raw_service_request` Ansible 외부에서 정의되고 실행 프레임워크에 제공되어야 함을 의미합니다.
- 폴더의 작업 파일 이름은 `logic-tasks.yml` 확장자가 없는 변수 값과 일치해야 `logic_operation` 합니다.
- 폴더의 작업 파일이 `logic-tasks` 동적으로 `raw_service_request` `정의합니다. 단, 유효한 를 `raw_service_request` 관련 파일의 마지막 작업으로 정의해야 합니다.

서비스 요청을 동적으로 정의하는 방법

서비스 요청을 동적으로 정의하기 위해 논리 작업을 적용하는 방법에는 여러 가지가 있습니다. 이러한 옵션 중 일부는 다음과 같습니다.

- 폴더의 Ansible 작업 파일 사용 `logic-tasks`
- `variable` 로 변환하기에 적합한 데이터를 반환하는 사용자 지정 역할 호출 `raw_service_request`
- Ansible 환경 외부에서 다른 툴을 호출하여 필요한 데이터를 제공합니다. 예를 들어, Active IQ Unified Manager에 대한 REST API 호출

다음 명령 예는 파일을 사용하여 각 클러스터에 대한 서비스 요청을 동적으로 `tutorial-requests.yml` 정의합니다.

```
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
```

6단계: ONTAP Day 0/1 솔루션을 배포합니다

이 단계에서는 다음을 이미 완료해야 합니다.

- 요구 사항에 따라 의 모든 파일을 검토하고 수정했습니다. `playbooks/inventory/group_vars/all` 각 파일에는 변경 작업에 도움이 되는 자세한 설명이 있습니다.
- 필요한 작업 파일을 `logic-tasks` 디렉토리에 추가했습니다.
- 필요한 데이터 파일을 `playbook/vars` 디렉토리에 추가했습니다.

다음 명령을 사용하여 ONTAP DAY 0/1 솔루션을 배포하고 배포 상태를 확인합니다.



이 단계에서는 이미 파일을 암호 해독하고 수정했으며 `vault.yml` 새 암호로 암호화해야 합니다.

- ONTAP Day 0 서비스 실행:

```
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>
```

- ONTAP Day 1 서비스 실행:

```
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>
```

- 클러스터 전체 설정 적용:

```
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>
```

- 상태 점검 실행:

```
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>
```

ONTAP Day 0/1 솔루션을 사용자 지정합니다

요구사항에 맞게 ONTAP Day 0/1 솔루션을 사용자 정의하려면 Ansible 역할을 추가하거나 변경할 수 있습니다.

역할은 Ansible 프레임워크 내의 마이크로서비스를 나타냅니다. 각 마이크로서비스는 하나의 작업을 수행합니다. 예를 들어 ONTAP Day 0은 여러 마이크로서비스를 포함하는 서비스입니다.

Ansible 역할을 추가합니다

Ansible 역할을 추가하여 환경에 맞게 솔루션을 사용자 지정할 수 있습니다. 필수 역할은 Ansible 프레임워크 내의 서비스 정의에 의해 정의됩니다.

마이크로서비스로 사용하려면 역할이 다음 요구 사항을 충족해야 합니다.

- 변수의 인수 목록을 수락합니다 `args`.
- 각 블록에 대한 특정 요구 사항이 있는 Ansible "블록, 구조, 항상" 구조를 사용합니다.
- 단일 Ansible 모듈을 사용하여 블록 내의 단일 작업을 정의합니다.
- 이 섹션에 설명된 요구 사항에 따라 사용 가능한 모든 모듈 매개 변수를 구현합니다.

필수 마이크로서비스 구조

각 역할은 다음 변수를 지원해야 합니다.

- `mode`: 모드가 역할로 설정된 경우 `test` 을(를) 가져오려고 시도하면 `test.yml` 실제로 해당 역할이 실행되지 않고 어떤 작업을 수행하는지를 보여줍니다.



특정 상호 의존성 때문에 이를 항상 구현할 수는 없습니다.

- `status`: Playbook 실행의 전반적인 상태. 값이 역할로 설정되지 않으면 `success` 실행되지 않습니다.
- `args`: 역할 매개 변수 이름과 일치하는 키가 있는 역할별 사전 목록입니다.
- `global_log_messages`: 플레이북 실행 중 로그 메시지를 수집합니다. 역할이 실행될 때마다 하나의 항목이 생성됩니다.
- `log_name`: 항목 내에서 역할을 참조하는 데 사용되는 이름입니다 `global_log_messages`.
- `task_descr`: 역할에 대한 간단한 설명입니다.
- `service_start_time`: 각 역할이 실행되는 시간을 추적하는 데 사용되는 타임 스탬프입니다.
- `playbook_status`: Ansible 플레이북의 상태.
- `role_result`: 역할 출력을 포함하고 항목 내의 각 메시지에 포함되는 변수입니다. `global_log_messages`

역할 구조의 예

다음 예제에서는 마이크로서비스를 구현하는 역할의 기본 구조를 제공합니다. 이 예제에서는 설정을 위해 변수를 변경해야 합니다.

기본 역할 구조:

```

- 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 ] }}"

```

예제 역할에 사용되는 변수:

- <NAME>: 각 마이크로서비스에 대해 제공되어야 하는 대체 가능한 값입니다.
- <LOG_NAME>: 로깅 목적으로 사용되는 역할의 짧은 형식 이름입니다. `ONTAP_VOLUME` 예를 들어,
- <TASK_DESCRIPTION>: 마이크로 서비스가 수행하는 작업에 대한 간략한 설명.
- <MODULE_NAME>: 작업에 대한 Ansible 모듈 이름.



최상위 `execute.yml` Playbook에서 `netapp.ontap` 컬렉션을 지정합니다. 모듈이 컬렉션의 일부인 경우 `netapp.ontap` 모듈 이름을 완전히 지정할 필요가 없습니다.

- <MODULE_SPECIFIC_PARAMETERS>: 마이크로 서비스를 구현하는 데 사용되는 모듈에 고유한 Ansible 모듈 매개 변수입니다. 다음 목록에서는 매개 변수 유형과 매개 변수를 그룹화하는 방법을 설명합니다.
 - 필수 매개 변수: 모든 필수 매개 변수는 기본값 없이 지정됩니다.
 - 마이크로서비스에 고유한 기본값을 갖는 매개 변수(모듈 설명서에 지정된 기본값과 다름)
 - 나머지 모든 매개 변수는 `default (omit)` 기본값으로 사용됩니다.

다중 수준 사전을 모듈 매개 변수로 사용

일부 NetApp에서 제공하는 Ansible 모듈은 모듈 매개 변수에 대해 다중 수준 사전을 사용합니다(예: 고정 및 적응형 QoS 정책 그룹).

이러한 사전이 둘 이상 있고 서로 배타적인 경우에는 `default (omit)` 단독으로 사용할 수 없습니다.

다중 수준 사전을 모듈 매개 변수로 사용해야 하는 경우 각 사전이 관련 사전에 대해 최소 1개의 2단계 사전 값을 제공하도록 여러 마이크로서비스(역할)로 기능을 분할해야 합니다.

다음 예에서는 고정 및 적응형 QoS 정책 그룹이 두 마이크로서비스로 분할되는 것을 보여 줍니다.

첫 번째 마이크로서비스에는 고정된 QoS 정책 그룹 값이 포함되어 있습니다.

```

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)
}}}"

```

두 번째 마이크로서비스에는 적응형 QoS 정책 그룹 값이 포함됩니다.

```

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) }}}"

```

저작권 정보

Copyright © 2025 NetApp, Inc. All Rights Reserved. 미국에서 인쇄된 본 문서의 어떠한 부분도 저작권 소유자의 사전 서면 승인 없이는 어떠한 형식이나 수단(복사, 녹음, 녹화 또는 전자 검색 시스템에 저장하는 것을 비롯한 그래픽, 전자적 또는 기계적 방법)으로도 복제될 수 없습니다.

NetApp이 저작권을 가진 자료에 있는 소프트웨어에는 아래의 라이선스와 고지사항이 적용됩니다.

본 소프트웨어는 NetApp에 의해 '있는 그대로' 제공되며 상품성 및 특정 목적에의 적합성에 대한 명시적 또는 묵시적 보증을 포함하여(이에 제한되지 않음) 어떠한 보증도 하지 않습니다. NetApp은 대체품 또는 대체 서비스의 조달, 사용 불능, 데이터 손실, 이익 손실, 영업 중단을 포함하여(이에 국한되지 않음), 이 소프트웨어의 사용으로 인해 발생하는 모든 직접 및 간접 손해, 우발적 손해, 특별 손해, 징벌적 손해, 결과적 손해의 발생에 대하여 그 발생 이유, 책임론, 계약 여부, 엄격한 책임, 불법 행위(과실 또는 그렇지 않은 경우)와 관계없이 어떠한 책임도 지지 않으며, 이와 같은 손실의 발생 가능성이 통지되었다 하더라도 마찬가지입니다.

NetApp은 본 문서에 설명된 제품을 언제든지 예고 없이 변경할 권리를 보유합니다. NetApp은 NetApp의 명시적인 서면 동의를 받은 경우를 제외하고 본 문서에 설명된 제품을 사용하여 발생하는 어떠한 문제에도 책임을 지지 않습니다. 본 제품의 사용 또는 구매의 경우 NetApp에서는 어떠한 특허권, 상표권 또는 기타 지적 재산권이 적용되는 라이선스도 제공하지 않습니다.

본 설명서에 설명된 제품은 하나 이상의 미국 특허, 해외 특허 또는 출원 중인 특허로 보호됩니다.

제한적 권리 표시: 정부에 의한 사용, 복제 또는 공개에는 DFARS 252.227-7013(2014년 2월) 및 FAR 52.227-19(2007년 12월)의 기술 데이터-비상업적 품목에 대한 권리(Rights in Technical Data -Noncommercial Items) 조항의 하위 조항 (b)(3)에 설명된 제한사항이 적용됩니다.

여기에 포함된 데이터는 상업용 제품 및/또는 상업용 서비스(FAR 2.101에 정의)에 해당하며 NetApp, Inc.의 독점 자산입니다. 본 계약에 따라 제공되는 모든 NetApp 기술 데이터 및 컴퓨터 소프트웨어는 본질적으로 상업용이며 개인 비용만으로 개발되었습니다. 미국 정부는 데이터가 제공된 미국 계약과 관련하여 해당 계약을 지원하는 데에만 데이터에 대한 전 세계적으로 비독점적이고 양도할 수 없으며 재사용이 불가능하며 취소 불가능한 라이선스를 제한적으로 가집니다. 여기에 제공된 경우를 제외하고 NetApp, Inc.의 사전 서면 승인 없이는 이 데이터를 사용, 공개, 재생산, 수정, 수행 또는 표시할 수 없습니다. 미국 국방부에 대한 정부 라이선스는 DFARS 조항 252.227-7015(b)(2014년 2월)에 명시된 권한으로 제한됩니다.

상표 정보

NETAPP, NETAPP 로고 및 <http://www.netapp.com/TM>에 나열된 마크는 NetApp, Inc.의 상표입니다. 기타 회사 및 제품 이름은 해당 소유자의 상표일 수 있습니다.