Day 1

NetApp Automation

NetApp
November 18, 2025

This PDF was generated from https://docs.netapp.com/zh-cn/netapp-automation/solutions/ontap-day01-overview.html on November 18, 2025. Always check docs.netapp.com for the latest.

目录

Day 1 . . . 1
ONTAP Day 1解决方案概述 . . . 1
灵活的ONTAP部署选项 . . . 1
分层设计 . . . 1
支持自定义 . . . 2
准备使用ONTAP Day 1解决方案 . . . 2
初始规划注意事项 . . . 3
准备ONTAP系统 . . . 3
安装所需的自动化软件 . . . 4
初始的Ands处理 框架配置 . . . 5
使用该解决方案部署ONTAP集群 . . . 6
开始之前 . . . 6
第1步：初始集群配置 . . . 6
第2步：配置集群间的生命周期 . . . 14
第3步：(可选)配置多个集群 . . . 19
第4步：初始SVM配置 . . . 20
第5步：(可选)动态定义服务请求 . . . 25
第6步：部署ONTAP Day 1解决方案 . . . 25
定制ONTAP Day 1解决方案 . . . 26
添加Ands处理 角色 . . . 26
示例角色结构 . . . 27
使用多层词典作为模块参数 . . . 30

Day 1

ONTAP Day 1解决方案概述

您可以使用ONTAP day 0/1 自动化解决方案，通过 Ansible 部署和配置ONTAP集群。解决方案可从以下途径获得： "NetApp Console自动化中心"。

灵活的ONTAP部署选项

根据您的要求、您可以使用内部硬件或模拟ONTAP来使用ONTAP部署和配置。

内部硬件

您可以使用运行ONTAP的内部硬件(例如FAS或AFF系统)部署此解决方案。您必须使用Linux VM使用ONTAP来部署和配置Storage Virtual Machine集群。

模拟ONTAP

要使用ONTAP模拟器部署此解决方案、您必须从NetApp支持站点下载最新版本的sim模拟ONTAP。模拟ONTAP是ONTAP软件的虚拟模拟器。模拟ONTAP在Windows、Linux或Mac系统上的VMware虚拟机管理程序中运行。对于Windows和Linux主机、必须使用VMware Workstation虚拟机管理程序来运行此解决方案。如果您使用的是Mac操作系统、请使用VMware Fusion虚拟机管理程序。

分层设计

通过该框架、可以简化自动化执行和逻辑任务的开发和重复使用。该框架区分了自动化中的决策任务(逻辑层)和执行步骤(执行层)。通过了解这些层的工作原理、您可以自定义配置。

一本安可赛"操作手册"从头到尾执行一系列任务。该 `site.yml`手册包含该手册 `logic.yml`和该 `execution.yml`手册。

运行请求时，该 `site.yml`操作手册首先调用该操作手册，然后调用该 `logic.yml`操作手册 `execution.yml`以执行服务请求。

您不需要使用框架的逻辑层。逻辑层提供了一些选项、可将框架的功能扩展到硬编码值之外、以供执行。这样、您可以根据需要自定义框架功能。

逻辑层

逻辑层由以下部分组成：

- 这是一本操作 `logic.yml`手册
- 目录中的逻辑任务文件 `logic-tasks`

逻辑层提供了复杂决策的功能、而无需进行大量的自定义集成(例如、连接到ServiceNow)。逻辑层是可配置的、可为微服务提供输入。

此外、还可以绕过逻辑层。如果要绕过逻辑层、请勿定义 `logic_operation`变量。通过直接调用该 `logic.yml`操作手册、可以在不执行的情况下执行某种级别的调试。您可以使用"debug "语句验证的值是否 `raw_service_request`正确。

重要注意事项：

- 该手册将 `logic.yml`搜索 `logic_operation`变量。如果在请求中定义了变量、则它会从目录中加载任务文件 `logic-tasks`。任务文件必须是.yml文件。如果没有匹配的任务文件且 `logic_operation`已定义变量、则逻辑层将失败。
- 变量的默认值 `logic_operation`为 `no-op`。如果未明确定义变量，则默认为 `no-op`，而不运行任何操作。
- 如果 `raw_service_request`已定义变量、则执行将继续到执行层。如果未定义此变量、则逻辑层将失败。

执行层

执行层由以下部分组成：

- 这是一本操作 `execution.yml`手册

执行层通过API调用来配置ONTAP集群。该 `execution.yml`手册要求 `raw_service_request`在执行时定义变量。

支持自定义

您可以根据自己的要求以各种方式自定义此解决方案。

自定义选项包括：

- 正在修改《安赛尔操作手册》
- 添加角色

自定义**Ans**｝文件

下表介绍了此解决方案中包含的可自定义的Ansable文件。

位置	说明
`playbooks/inventory/hosts`	包含一个包含主机和组列表的文件。
`playbooks/group_vars/all/*`	通过Ans可 轻松地一次性将变量应用于多个主机。您可以修改此文件夹中的任何或所有文件，包括 `cfg.yml`、、 `clusters.yml` `defaults.yml`、`services.yml`、 `standards.yml`和 `vault.yml`。
`playbooks/logic-tasks`	支持在Ans可 内执行决策任务、并保持逻辑与执行的分离。您可以向此文件夹添加与相关服务对应的文件。
`playbooks/vars/*`	在《Andsability操作手册》和角色中使用动态值、以实现配置的自定义、灵活性和可重用性。如有必要、您可以修改此文件夹中的任何或所有文件。

自定义角色

此外、您还可以通过添加或更改Ans担负 的角色(也称为微服务)来自定义解决方案。有关详细信息，请参见"自定义"。

准备使用ONTAP Day 1解决方案

在部署自动化解决方案之前、您必须准备ONTAP环境并安装和配置Ans得。

初始规划注意事项

在使用此解决方案部署ONTAP集群之前、您应查看以下要求和注意事项。

基本要求

要使用此解决方案、您必须满足以下基本要求：

- 您必须能够在内部或通过ONTAP模拟器访问ONTAP软件。
- 您必须了解如何使用ONTAP软件。
- 您必须了解如何使用Ans得 自动化软件工具。

规划注意事项

在部署此自动化解决方案之前、您必须确定：

- 要运行Ans得 可控制节点的位置。
- ONTAP系统、内部硬件或ONTAP模拟器。
- 是否需要自定义。

准备ONTAP系统

无论您是使用内部ONTAP系统还是模拟ONTAP、都必须先准备好环境、然后才能部署自动化解决方案。

(可选)安装和配置模拟ONTAP

如果要通过ONTAP模拟器部署此解决方案、则必须下载并运行模拟ONTAP。

开始之前

- 您必须下载并安装要用于运行模拟ONTAP的VMware虚拟机管理程序。
  - 如果您使用的是Windows或Linux操作系统、请使用VMware Workstation。
  - 如果您使用的是Mac操作系统、请使用VMware Fusion。

> 如果您使用的是Mac OS、则必须使用Intel处理器。

步骤

使用以下过程在本地环境中安装两个ONTAP管理器：

1. 从下载模拟ONTAP"NetApp 支持站点"。

   > 尽管您安装了两个ONTAP联机器、但只需下载一份软件副本即可。

2. 如果尚未运行、请启动VMware应用程序。
3. 找到已下载的模拟器文件、然后右键单击以使用VMware应用程序打开该文件。
4. 设置第一个ONTAP实例的名称。
5. 等待模拟器启动、然后按照说明创建单节点集群。

对第二个ONTAP实例重复上述步骤。

6. (可选)添加完整磁盘补充。

   从每个集群中、运行以下命令：

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

ONTAP系统的状态

您必须验证ONTAP系统的初始状态、无论是内部还是通过ONTAP模拟器运行。

验证是否满足以下ONTAP系统要求：

- ONTAP已安装并正在运行、但尚未定义集群。
- 此时、ONTAP将启动并显示用于访问集群的IP地址。
- 网络可访问。
- 您具有管理员凭据。
- 此时将显示每日消息(Message of the Day、MOTD)横幅以及管理地址。

安装所需的自动化软件

本节介绍有关如何安装Ands还是 准备部署自动化解决方案的信息。

安装**Ands**器

可以在Linux或Windows系统上安装Ans得。

Ands得以 与ONTAP集群进行通信的默认通信方法是SSH。

请参阅"NetApp 和 Ansible 入门：安装 Ansible"安装Ands得以 安装。

> 必须在系统的控制节点上安装有Ans得。

下载并准备自动化解决方案

您可以使用以下步骤下载并准备要部署的自动化解决方案。

1. 下载 "ONTAP - Day 1和amp；运行状况检查" 通过控制台 Web 用户界面实现自动化解决方案。该解决方案打包如下： `ONTAP_DAY0_DAY1.zip`。
2. 提取zip文件夹、并将文件复制到您的Ands得以 环境中控制节点上的所需位置。

初始的Ands处理 框架配置

执行Ans担负 框架的初始配置：

1. 导航到 `playbooks/inventory/group_vars/all`。
2. 解密 `vault.yml`文件：

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

   当系统提示您输入存储密码时、输入以下临时密码：

   ```
   NetApp123!
   ```

   > "NetApp 123！"是用于对文件和相应的存储密码进行解密的临时 `vault.yml`密码。首次使用后，*必须*使用您自己的密码对文件进行加密。

3. 修改以下的Ans的 文件：

   - `clusters.yml`修改此文件中的值以适合您的环境。
   - `vault.yml`-解密文件后、根据您的环境修改ONTAP集群、用户名和密码值。
   - `cfg.yml`-设置的文件路径 `log2file`，并在下 `cfg`设置 `show_request`为 `True`以显示 `raw_service_request`。

     此 `raw_service_request`变量将显示在日志文件中以及执行期间。

     > 列出的每个文件都包含注释、并说明如何根据您的要求对其进行修改。

4. 重新加密 `vault.yml`文件：

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

   > 系统会提示您在加密时为存储选择新密码。

5. 导航到 `playbooks/inventory/hosts`并设置有效的Python解释器。
6. 部署 `framework_test`服务：

   以下命令运行值为 `cluster_identity_info`的 `na_ontap_info`模块 `gather_subset`。这将验证基本配置是否正确、并验证您是否可以与集群进行通信。

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

   对每个集群运行命令。

   如果成功、您应看到类似于以下示例的输出：

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

使用该解决方案部署ONTAP集群

完成准备和规划后、您便可以使用ONTAP Day 1解决方案使用ONTAP来快速配置使用Ans得 集群了。

在执行本节中的步骤期间、您可以随时选择测试请求、而不是实际执行请求。要测试请求，请将命令行上的操作手册更 `site.yml`改为 `logic.yml`。

> 该 `docs/tutorial-requests.txt`位置包含此过程中使用的所有服务请求的最终版本。如果您在运行服务请求时遇到困难、可以将相关请求从文件复制 `tutorial-requests.txt`到该 `playbooks/inventory/group_vars/all/tutorial-requests.yml`位置、并根据需要修改硬编码值(IP地址、聚合名称等)。然后、您应该能够成功运行此请求。

开始之前

- 您必须安装了Ans得。
- 您必须已下载ONTAP Day 1解决方案并将该文件夹解压缩到了Ands得以 控制的节点上的所需位置。
- ONTAP系统状态必须满足要求、并且您必须具有必要的凭据。
- 您必须已完成本节中所述的所有必需任务"准备"。

> 本解决方案中的示例使用"Cluster_01"和"Cluster_02"作为两个集群的名称。您必须将这些值替换为环境中集群的名称。

第1步：初始集群配置

在此阶段、您必须执行一些初始集群配置步骤。

步骤

1. 导航到该 `playbooks/inventory/group_vars/all/tutorial-requests.yml`位置并查看 `cluster_initial`文件中的请求。对您的环境进行任何必要的更改。
2. 在文件夹中为服务请求创建一个文件 `logic-tasks`。例如，创建一个名为的文件 `cluster_initial.yml`。

   将以下行复制到新文件：

```
- 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. 定义 `raw_service_request`变量。

   您可以使用以下选项之一在文件夹中创建的文件 `logic-tasks`中定义 `raw_service_request`变量 `cluster_initial.yml`：

   - 选项**1**:手动定义 `raw_service_request`变量。

     使用编辑器打开 `tutorial-requests.yml`文件、并将内容从第11行复制到第165行。将内容粘贴到新文件中的变量 `cluster_initial.yml`下 `raw service request`、如以下示例所示：

显示示例

示例 `cluster_initial.yml`文件：

```
- 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
```

```
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
      - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA

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

```
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA
        - XXXXXXXXXXXXXXXAAAAAAAAAAAAAA

    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 }}"
```

- 选项**2**:使用Jinja模板定义请求:

  您也可以使用以下Jinja模板格式获取该 `raw_service_request`值。

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

4. 对第一个集群执行初始集群配置:

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

继续操作前、请确认没有错误。

5. 对第二个集群重复此命令：

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

确认第二个集群没有错误。

向上滚动到Andsent输出的开头时、您应看到发送到框架的请求、如以下示例所示：

显示示例

```
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": [
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
                        "XXXXXXXXXXXXXXXXAAAAAAAAAAAA",
```

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


6. 登录到每个ONTAP实例并验证请求是否成功。

第2步：配置集群间的生命周期

现在、您可以通过向请求添加LIF定义并定义 `ontap_interface`微服务来配置集群间LIF `cluster_initial`。

服务定义和请求共同确定操作：

- 如果您为服务定义中未包含的微服务提供服务请求、则不会执行此请求。
- 如果您为服务请求提供服务定义中定义的一个或多个微服务、但在请求中省略了该服务、则不会执行该请求。

该 `execution.yml`操作手册将按所列顺序扫描微服务列表、以评估服务定义：

- 如果请求中有一个条目，其词典密钥与微服务定义中包含的条目匹配 `args`，则执行该请求。
- 如果服务请求中没有匹配的条目、则会跳过此请求、并且不会出现错误。

步骤

1. 导航到 `cluster_initial.yml`先前创建的文件，并通过向请求定义添加以下行来修改请求：

```
    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. 运行命令：

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

3. 登录到每个实例以检查是否已将这些RIF添加到集群：

显示示例

```
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.
```

输出显示已*未*添加Lifs。这是因为 `ontap_interface`仍需要在文件中定义微服务 `services.yml`。

4. 验证是否已将这些生命周期添加到此变量中 `raw_service_request`。

显示示例

以下示例显示已将这些生命周期管理器添加到请求中：


```
            "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. 在文件中 `services.yml`的下定义 `ontap_interface`微服务 `cluster_initial。

   将以下行复制到文件中以定义微服务：

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

6. `ontap_interface`已在请求和文件中定义微服务 `services.yml、请再次运行请求：

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

7. 登录到每个ONTAP实例并验证是否已添加这些LUN。

第3步：(可选)配置多个集群

如果需要、您可以在同一请求中配置多个集群。定义请求时、必须为每个集群提供变量名称。

步骤

1. 在文件中为第二个集群添加一个条目 `cluster_initial.yml`、以便在同一请求中配置这两个集群。

   以下示例将在添加第二个条目后显示 `ontap_aggr`字段。

```
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. 将更改应用于下的所有其他项目 `cluster_initial`。
3. 通过将以下行复制到文件来向请求添加集群对等关系：

```
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. 运行Ands处理 请求：

```
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>
```

第4步：初始SVM配置

在此过程的此阶段、您需要配置集群中的SVM。

步骤

1. 更新 `svm_initial`文件中的请求 `tutorial-requests.yml`以配置SVM和SVM对等关系。

   您必须配置以下内容：

   - SVM

- SVM对等关系
- 每个SVM的SVM接口

2. 更新请求定义中的变量定义 `svm_initial`。您必须修改以下变量定义：
   - `cluster_name`
   - `vserver_name`
   - `peer_cluster_name`
   - `peer_vserver`

   要更新定义，请删除 `svm_initial`定义后的*‘{}'* `req_details`并添加正确的定义。

3. 在文件夹中为服务请求创建一个文件 `logic-tasks`。例如，创建一个名为的文件 `svm_initial.yml`。

   将以下行复制到文件：

```
- 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. 定义 `raw_service_request`变量。

   您可以使用以下选项之一为文件夹中 `logic-tasks`的定义 `raw_service_request`变量 `svm_initial`：

   - 选项**1**:手动定义 `raw_service_request`变量。

     使用编辑器打开 `tutorial-requests.yml`文件、并将内容从第`179`行复制到第`222`行。将内容粘贴到新文件中的变量 `svm_initial.yml`下 `raw service request`、如以下示例所示：

```
    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
```

显示示例

示例 `svm_initial.yml`文件：

```
- 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
```

- 选项**2**:使用Jinja模板定义请求:

  您也可以使用以下Jinja模板格式获取该 `raw_service_request`值。

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

5. 运行请求:

```
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. 登录到每个ONTAP实例并验证配置。
7. 添加SVM接口。

   在文件中 `services.yml`的下定义 `ontap_interface`服务 `svm_initial`、然后再次运行请求:

```
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. 登录到每个ONTAP实例并验证是否已配置SVM接口。

第5步：(可选)动态定义服务请求

在前面的步骤中、 `raw_service_request`变量是硬编码的。这对于学习、开发和测试非常有用。您还可以动态生成服务请求。

如果您不想将所需的与更高级别的系统集成、则下一节提供了一个动态生成所需的选项 `raw_service_request`。

> - 如果 `logic_operation`未在命令中定义变量、则该 `logic.yml`文件不会从文件夹导入任何文件 `logic-tasks`。这意味着`raw_service_request`必须在Ands还是 外部定义、并在执行时提供给框架。
> - 文件夹中的任务文件名 `logic-tasks`必须与不带`.yml`扩展名的变量值匹配 `logic_operation`。
> - 文件夹中的任务文件 `logic-tasks`会动态定义 `raw_service_request`。唯一的要求是将有效定义为相关文件中的最后一个任务。 `raw_service_request`

如何动态定义服务请求

可以通过多种方法应用逻辑任务来动态定义服务请求。下面列出了其中一些选项：

- 使用文件夹中的一个Ans} 任务文件 `logic-tasks`
- 调用返回适合转换为变量的数据的自定义角色 `raw_service_request`。
- 调用Ands得以 环境外部的另一个工具以提供所需数据。例如、对Active IQ Unified Manager的REST API调用。

以下示例命令使用文件为每个集群动态定义服务请求 `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
```

第6步：部署ONTAP Day 1解决方案

在此阶段、您应已完成以下操作：

- 已根据您的要求查看和修改中的所有文件 `playbooks/inventory/group_vars/all`。每个文件中都有详细的注释、以帮助您进行更改。
- 已将任何所需的任务文件添加到 `logic-tasks`目录。
- 已将任何所需的数据文件添加到 `playbook/vars`目录。

使用以下命令部署ONTAP Day 1解决方案并验证部署的运行状况：

在此阶段、您应已解密并修改 `vault.yml`文件、并且必须使用新密码对其进行加密。

- 运行ONTAP Day 0服务：

```
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>
```

- 运行ONTAP Day 1服务：

```
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>
```

- 应用集群范围设置：

```
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>
```

- 运行运行状况检查：

```
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>
```

定制ONTAP Day 1解决方案

要根据您的需求自定义ONTAP Day 1解决方案、您可以添加或更改Ands还是 角色。

角色表示在Andsent框架内的微服务。每个微服务执行一个操作。例如、ONTAP Day 0是包含多个微服务的服务。

添加Ands处理 角色

您可以添加Ans可 通过角色为您的环境自定义解决方案。所需角色由在Ands还是 在框架中定义的服务定义来定义。

角色必须满足以下要求才能用作微服务：

- 接受变量中的参数列表 `args`。
- 对于每个块、使用具有特定要求的"Andsing"结构。

- 使用单个Ans还是 模块并在块中定义单个任务。
- 根据本节详细介绍的要求实施每个可用的模块参数。

所需的微服务结构

每个角色都必须支持以下变量：

- `mode`：如果将模式设置为角色，则 `test`尝试导入显示该角色所执行的 `test.yml`操作而不实际执行该角色。

  由于某些相互依赖关系、并非总是可以实现这一点。

- `status`：执行手册的整体状态。如果未将此值设置为 `success`、则不会执行此角色。
- `args`：具有与角色参数名称匹配的关键字的角色专用词典列表。
- `global_log_messages`：在执行播放手册期间收集日志消息。每次执行角色时都会生成一个条目。
- `log_name`：用于引用条目中的角色的名称 `global_log_messages`。
- `task_descr`：有关该角色的作用的简要说明。
- `service_start_time`：用于跟踪每个角色执行时间的时间戳。
- `playbook_status`：Ans负责人 的状态。
- `role_result`：包含角色输出并包含在条目中的每条消息中的变量 `global_log_messages`。

示例角色结构

以下示例提供了实施微服务的角色的基本结构。您必须更改此示例中适用于您的配置的变量。

显示示例

基本角色结构：

```
- 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 ] }}"
```

示例角色中使用的变量：

- `<NAME>`：必须为每个微服务提供的可替换值。
- `<LOG_NAME>`：用于日志记录的角色的简称。例如， `ONTAP_VOLUME`。
- `<TASK_DESCRIPTION>`：微服务功能的简要说明。
- `<MODULE_NAME>`：任务的Ans担负 模块名称。

> 顶级 `execute.yml`操作手册用于指定 `netapp.ontap`集合。如果模块是集合的一部分 `netapp.ontap`，则无需完全指定模块名称。

- `<MODULE_SPECIFIC_PARAMETERS>`:特定于用于实施微服务的模块的Ans得 模块参数。以下列表介绍了参数的类型以及应如何对其进行分组。
  - 必需参数：所有必需参数均指定、无默认值。
  - 具有特定于微服务的默认值的参数(与模块文档指定的默认值不同)。
  - 所有剩余参数均 `default(omit)`用作默认值。

使用多层词典作为模块参数

某些NetApp提供的Ans典 模块使用多级别词典作为模块参数(例如、固定和自适应QoS策略组)。

如果使用这些词典、则单独使用 `default(omit)`不起作用、尤其是当有多个词典且它们互斥时。

如果您需要使用多级别词典作为模块参数、则应将此功能拆分为多个微服务(角色)、以确保每个微服务都能为相关词典至少提供一个二级词典值。

以下示例显示了固定和自适应QoS策略组、它们分别分布在两个微服务中。

第一个微服务包含固定的QoS策略组值：

```
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)
}}"
```

第二个微服务包含自适应QoS策略组值：

```
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) }}"
```

版权信息

版权所有 © 2025 NetApp, Inc.。保留所有权利。中国印刷。未经版权所有者事先书面许可，本文档中受版权保护的任何部分不得以任何形式或通过任何手段（图片、电子或机械方式，包括影印、录音、录像或存储在电子检索系统中）进行复制。

从受版权保护的 NetApp 资料派生的软件受以下许可和免责声明的约束：

本软件由 NetApp 按“原样”提供，不含任何明示或暗示担保，包括但不限于适销性以及针对特定用途的适用性的隐含担保，特此声明不承担任何责任。在任何情况下，对于因使用本软件而以任何方式造成的任何直接性、间接性、偶然性、特殊性、惩罚性或后果性损失（包括但不限于购买替代商品或服务；使用、数据或利润方面的损失；或者业务中断），无论原因如何以及基于何种责任理论，无论出于合同、严格责任或侵权行为（包括疏忽或其他行为），NetApp 均不承担责任，即使已被告知存在上述损失的可能性。

NetApp 保留在不另行通知的情况下随时对本文档所述的任何产品进行更改的权利。除非 NetApp 以书面形式明确同意，否则 NetApp 不承担因使用本文档所述产品而产生的任何责任或义务。使用或购买本产品不表示获得 NetApp 的任何专利权、商标权或任何其他知识产权许可。

本手册中描述的产品可能受一项或多项美国专利、外国专利或正在申请的专利的保护。

有限权利说明：政府使用、复制或公开本文档受 DFARS 252.227-7013（2014 年 2 月）和 FAR 52.227-19（2007 年 12 月）中“技术数据权利 — 非商用”条款第 (b)(3) 条规定的限制条件的约束。

本文档中所含数据与商业产品和/或商业服务（定义见 FAR 2.101）相关，属于 NetApp, Inc. 的专有信息。根据本协议提供的所有 NetApp 技术数据和计算机软件具有商业性质，并完全由私人出资开发。 美国政府对这些数据的使用权具有非排他性、全球性、受限且不可撤销的许可，该许可既不可转让，也不可再许可，但仅限在与交付数据所依据的美国政府合同有关且受合同支持的情况下使用。除本文档规定的情形外，未经 NetApp, Inc. 事先书面批准，不得使用、披露、复制、修改、操作或显示这些数据。美国政府对国防部的授权仅限于 DFARS 的第 252.227-7015(b)（2014 年 2 月）条款中明确的权利。

商标信息

NetApp、NetApp 标识和 http://www.netapp.com/TM 上所列的商标是 NetApp, Inc. 的商标。其他公司和产品名称可能是其各自所有者的商标。