Dia 0/1

NetApp Automation

NetApp
November 18, 2025

Índice

Dia 0/1

Visão geral da solução ONTAP Day 0/1

Você pode usar a solução de automação ONTAP day 0/1 para implantar e configurar um cluster ONTAP usando o Ansible. A solução está disponível em "Hub de automação do NetApp Console".

Opções flexíveis de implantação do ONTAP

Dependendo dos seus requisitos, use o hardware no local ou simule o ONTAP para implantar e configurar um cluster do ONTAP com o Ansible.

Hardware no local

É possível implantar essa solução usando hardware local executando ONTAP, como um sistema FAS ou AFF. Use uma máquina virtual do Linux para implantar e configurar o cluster do ONTAP usando o Ansible.

Simular ONTAP

Para implantar essa solução usando um simulador ONTAP, você deve baixar a versão mais recente do Simulate ONTAP no site de suporte da NetApp. Simule ONTAP é um simulador virtual para o software ONTAP. Simule a execução do ONTAP em um hypervisor VMware em um sistema Windows, Linux ou Mac. Para hosts Windows e Linux, você deve usar o hipervisor VMware Workstation para executar essa solução. Se você tiver um Mac os, use o hypervisor do VMware Fusion.

Design em camadas

A estrutura do Ansible simplifica o desenvolvimento e a reutilização de tarefas lógicas e de execução de automação. A estrutura faz uma distinção entre as tarefas de tomada de decisão (camada lógica) e as etapas de execução (camada de execução) na automação. Entender como essas camadas funcionam permite que você personalize a configuração.

Um "manual de estratégia" do Ansible executa uma série de tarefas do início ao fim. O `site.yml` manual de estratégia contém o `logic.yml` manual de estratégia e `execution.yml` o manual de estratégia.

Quando uma solicitação é executada, o `site.yml` manual de estratégia faz uma chamada primeiro para o `logic.yml` manual de estratégia e, em seguida, chama o `execution.yml` manual de estratégia para executar a solicitação de serviço.

Você não é obrigado a usar a camada lógica da estrutura. A camada lógica fornece opções para expandir a capacidade da estrutura além dos valores codificados para execução. Isso permite que você personalize os recursos da estrutura, se necessário.

Camada lógica

A camada lógica consiste no seguinte:

- O `logic.yml` manual de estratégia
- Arquivos de tarefa lógica dentro do `logic-tasks` diretório

A camada lógica fornece a capacidade para tomada de decisões complexas sem a necessidade de integração personalizada significativa (por exemplo, conetando-se ao ServiceNOW). A camada lógica é configurável e

fornece a entrada para microservices.

A capacidade de ignorar a camada lógica também é fornecida. Se você quiser ignorar a camada lógica, não defina a `logic_operation` variável. A invocação direta `logic.yml` do manual de estratégia fornece a capacidade de fazer algum nível de depuração sem execução. Você pode usar uma instrução "debug" para verificar se o valor do `raw_service_request` está correto.

Considerações importantes:

- O `logic.yml` manual de estratégia procura a `logic_operation` variável. Se a variável for definida na solicitação, ela carregará um arquivo de tarefa do `logic-tasks` diretório. O arquivo de tarefa deve ser um arquivo .yml. Se não houver nenhum arquivo de tarefa correspondente e a `logic_operation` variável for definida, a camada lógica falhará.
- O valor padrão `logic_operation` da variável é `no-op`. Se a variável não for definida explicitamente, ela será padrão para `no-op`, que não executa nenhuma operação.
- Se a `raw_service_request` variável já estiver definida, a execução prossegue para a camada de execução. Se a variável não estiver definida, a camada lógica falhará.

Camada de execução

A camada de execução consiste no seguinte:

- O `execution.yml` manual de estratégia

A camada de execução faz as chamadas de API para configurar um cluster ONTAP. O `execution.yml` manual de estratégia requer que a `raw_service_request` variável seja definida quando executada.

Suporte para personalização

Você pode personalizar esta solução de várias maneiras, dependendo de suas necessidades.

As opções de personalização incluem:

- Modificação de playbooks do Ansible
- Adicionar funções

Personalizar arquivos do Ansible

A tabela a seguir descreve os arquivos Ansible personalizáveis contidos nesta solução.

Localização	Descrição
`playbooks/inventory/hosts`	Contém um único arquivo com uma lista de hosts e grupos.
`playbooks/group_vars/all/*`	O Ansible fornece uma maneira conveniente de aplicar variáveis a vários hosts de uma só vez. Pode modificar qualquer ou todos os ficheiros desta pasta, incluindo `cfg.yml`, `clusters.yml`, , , `defaults.yml` `services.yml` `standards.yml` `vault.yml` e .
`playbooks/logic-tasks`	Dá suporte a tarefas de tomada de decisão no Ansible e mantém a separação entre lógica e execução. Pode adicionar ficheiros a esta pasta que correspondam ao serviço relevante.

Localização	Descrição
`playbooks/vars/*`	Valores dinâmicos usados nos playbooks e funções do Ansible para permitir a personalização, flexibilidade e reutilização de configurações. Se necessário, você pode modificar qualquer ou todos os arquivos nesta pasta.

Personalizar funções

Também é possível personalizar a solução adicionando ou alterando as funções do Ansible, também chamadas de microsserviços. Para obter mais detalhes, "Personalizar"consulte .

Prepare-se para usar a solução ONTAP Day 0/1

Antes de implantar a solução de automação, você precisa preparar o ambiente ONTAP e instalar e configurar o Ansible.

Considerações iniciais de Planejamento

Você deve analisar os requisitos e considerações a seguir antes de usar essa solução para implantar um cluster do ONTAP.

Requisitos básicos

Para usar essa solução, você precisa atender aos seguintes requisitos básicos:

- Você deve ter acesso ao software ONTAP, seja no local ou por meio de um simulador ONTAP.
- Você deve saber como usar o software ONTAP.
- Você precisa saber como usar as ferramentas de software de automação do Ansible.

Considerações de Planejamento

Antes de implantar essa solução de automação, você deve decidir:

- O local onde você executará o nó de controle do Ansible.
- O sistema ONTAP, seja hardware no local ou um simulador ONTAP.
- Se você vai ou não precisar de personalização.

Prepare o sistema ONTAP

Não importa se você está usando um sistema ONTAP no local ou simule o ONTAP, prepare o ambiente antes de implantar a solução de automação.

Opcionalmente, instale e configure o Simulate ONTAP

Se você quiser implantar essa solução através de um simulador ONTAP, você deve baixar e executar o Simulate ONTAP.

Antes de começar

- É necessário fazer o download e instalar o hypervisor VMware que você vai usar para executar o Simulate ONTAP.
  - Se você tiver um sistema operacional Windows ou Linux, use o VMware Workstation.
  - Se você tiver um Mac os, use o VMware Fusion.

Se você estiver usando um Mac os, você deve ter um processador Intel.

Passos

Use o seguinte procedimento para instalar dois simuladores ONTAP em seu ambiente local:

1. Faça o download do Simulate ONTAP no "Site de suporte da NetApp".

   Embora você instale dois simuladores ONTAP, você só precisa baixar uma cópia do software.

2. Se ele ainda não estiver em execução, inicie seu aplicativo VMware.
3. Localize o arquivo do simulador que foi baixado e clique com o botão direito do Mouse para abri-lo com o aplicativo VMware.
4. Defina o nome da primeira instância do ONTAP.
5. Aguarde a inicialização do simulador e siga as instruções para criar um cluster de nó único.

   Repita as etapas para a segunda instância do ONTAP.

6. Opcionalmente, adicione um complemento de disco completo.

   Em cada cluster, execute os seguintes comandos:

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

Estado do sistema ONTAP

Você deve verificar o estado inicial do sistema ONTAP, seja no local ou em execução através de um simulador ONTAP.

Verifique se os seguintes requisitos do sistema ONTAP são atendidos:

- O ONTAP está instalado e em execução sem cluster definido ainda.
- O ONTAP é inicializado e exibe o endereço IP para acessar o cluster.
- A rede é acessível.
- Você tem credenciais de administrador.
- O banner mensagem do dia (MOTD) é exibido com o endereço de gerenciamento.

Instale o software de automação necessário

Esta seção fornece informações sobre como instalar o Ansible e preparar a solução de automação para implantação.

Instalar o Ansible

O Ansible pode ser instalado em sistemas Linux ou Windows.

O método de comunicação padrão usado pelo Ansible para se comunicar com um cluster ONTAP é SSH.

Consulte a "Primeiros passos com o NetApp e o Ansible: Instale o Ansible"instalação do Ansible.

O Ansible precisa ser instalado no nó de controle do sistema.

Baixe e prepare a solução de automação

Você pode usar as etapas a seguir para baixar e preparar a solução de automação para implantação.

1. Baixe o "ONTAP - dia 0/1 verificações de estado" Solução de automação através da interface web do Console. A solução é empacotada como `ONTAP_DAY0_DAY1.zip`.
2. Extraia a pasta zip e copie os arquivos para o local desejado no nó de controle em seu ambiente Ansible.

Configuração inicial da estrutura do Ansible

Execute a configuração inicial da estrutura do Ansible:

1. Navegue até `playbooks/inventory/group_vars/all`.
2. Desencriptar o `vault.yml` ficheiro:

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

   Quando for solicitada a senha do cofre, digite a seguinte senha temporária:

   ```
   NetApp123!
   ```

   "NetApp123!" é uma senha temporária para descriptografar o `vault.yml` arquivo e a senha do cofre correspondente. Após o primeiro uso, você **deve** criptografar o arquivo usando sua própria senha.

3. Modifique os seguintes arquivos do Ansible:

   - `clusters.yml` - Modifique os valores neste arquivo para se adequar ao seu ambiente.
   - `vault.yml` - Depois de descriptografar o arquivo, modifique os valores de cluster, nome de usuário e senha do ONTAP para se adequar ao seu ambiente.
   - `cfg.yml` - Defina o caminho do arquivo `log2file` e defina `show_request` `cfg` como para `True` exibir o `raw_service_request`.

     A `raw_service_request` variável é exibida nos arquivos de log e durante a execução.

   Cada arquivo listado contém comentários com instruções sobre como modificá-lo de acordo com suas necessidades.

4. Recriptografe o `vault.yml` arquivo:

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

Você será solicitado a escolher uma nova senha para o cofre após a criptografia.

5. Navegue `playbooks/inventory/hosts` e defina um interpretador Python válido.
6. Implantar o `framework_test` serviço:

   O comando a seguir executa o `na_ontap_info` módulo com um `gather_subset` valor `cluster_identity_info` de . Isso valida que a configuração básica está correta e verifica se você pode se comunicar com o cluster.

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

   Execute o comando para cada cluster.

   Se for bem-sucedido, você verá uma saída semelhante ao seguinte exemplo:

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

Implante o cluster do ONTAP usando a solução

Após concluir a preparação e o Planejamento, você estará pronto para usar a solução ONTAP day 0/1 para configurar rapidamente um cluster do ONTAP usando o Ansible.

A qualquer momento durante as etapas desta seção, você pode optar por testar uma solicitação em vez de executá-la. Para testar uma solicitação, altere o `site.yml` manual na linha de comando para `logic.yml`.

A `docs/tutorial-requests.txt` localização contém a versão final de todos os pedidos de assistência utilizados durante este procedimento. Se tiver dificuldade em executar uma solicitação de serviço, você pode copiar a solicitação relevante do `tutorial-requests.txt` arquivo para `playbooks/inventory/group_vars/all/tutorial-requests.yml` o local e modificar os valores codificados conforme necessário (endereço IP, nomes agregados, etc.). Em seguida, você deve ser capaz de executar com sucesso a solicitação.

Antes de começar

- Tenha o Ansible instalado.
- Você precisa ter baixado a solução do dia 0/1 do ONTAP e extraído a pasta para o local desejado no nó de controle do Ansible.

- O estado do sistema ONTAP deve atender aos requisitos e você precisa ter as credenciais necessárias.
- Você deve ter concluído todas as tarefas necessárias descritas na "Prepare-se"seção.

Os exemplos dessa solução usam "Cluster_01" e "Cluster_02" como os nomes dos dois clusters. É necessário substituir esses valores pelos nomes dos clusters no ambiente.

Etapa 1: Configuração inicial do cluster

Neste estágio, você deve executar algumas etapas iniciais de configuração do cluster.

Passos

1. Navegue até o `playbooks/inventory/group_vars/all/tutorial-requests.yml` local e reveja a `cluster_initial` solicitação no arquivo. Faça as alterações necessárias para o seu ambiente.
2. Crie um arquivo `logic-tasks` na pasta para a solicitação de serviço. Por exemplo, crie um arquivo `cluster_initial.yml` chamado .

   Copie as seguintes linhas para o novo arquivo:

```
- 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. Defina a `raw_service_request` variável.

   Você pode usar uma das seguintes opções para definir a `raw_service_request` variável no `cluster_initial.yml` arquivo que você criou na `logic-tasks` pasta:

   - **Opção 1**: Defina manualmente a `raw_service_request` variável.

     Abra o `tutorial-requests.yml` arquivo usando um editor e copie o conteúdo da linha 11 para a linha 165. Cole o conteúdo sob a `raw service request` variável no novo `cluster_initial.yml`

arquivo, como mostrado nos exemplos a seguir:

Mostrar exemplo

Ficheiro de exemplo `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 }}"
```

- **Opção 2**: Use um modelo Jinja para definir a solicitação:

  Você também pode usar o seguinte formato de modelo Jinja para obter o `raw_service_request` valor.

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

4. Execute a configuração inicial do cluster para o primeiro cluster:

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

Verifique se não existem erros antes de prosseguir.

5. Repita o comando para o segundo cluster:

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

Verifique se não há erros para o segundo cluster.

Ao rolar para cima em direção ao início da saída do Ansible, você verá a solicitação que foi enviada para a estrutura, como mostrado no exemplo a seguir:

Mostrar exemplo

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

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

6. Faça login em cada instância do ONTAP e verifique se a solicitação foi bem-sucedida.

Etapa 2: Configurar os LIFs entre clusters

Agora você pode configurar as LIFs entre clusters adicionando as definições de LIF à `cluster_initial` solicitação e definindo o `ontap_interface` microservice.

A definição do serviço e a solicitação trabalham em conjunto para determinar a ação:

- Se você fornecer uma solicitação de serviço para um microservice que não esteja nas definições de serviço, a solicitação não será executada.
- Se você fornecer uma solicitação de serviço com um ou mais microsserviços definidos nas definições de serviço, mas omitido da solicitação, a solicitação não será executada.

O `execution.yml` manual de estratégia avalia a definição do serviço digitalizando a lista de microsserviços na ordem listada:

- Se houver uma entrada na solicitação com uma chave de dicionário que corresponda à `args` entrada contida nas definições de microservices, a solicitação será executada.
- Se não houver nenhuma entrada correspondente na solicitação de serviço, a solicitação será ignorada sem erro.

Passos

1. Navegue até o `cluster_initial.yml` arquivo que você criou anteriormente e modifique a solicitação adicionando as seguintes linhas às definições da solicitação:

```
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. Execute o comando:

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

3. Faça login em cada instância para verificar se os LIFs foram adicionados ao cluster:

Mostrar exemplo

```
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.
```

A saída mostra que os LIFs foram **not** adicionados. Isso ocorre porque o `ontap_interface` microservice ainda precisa ser definido no `services.yml` arquivo.

4. Verifique se os LIFs foram adicionados à `raw_service_request` variável.

Mostrar exemplo

O exemplo a seguir mostra que os LIFs foram adicionados à solicitação:


```
            "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. Defina o `ontap_interface` microservice em `cluster_initial` no `services.yml` arquivo.

   Copie as seguintes linhas para o arquivo para definir o microservice:

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

6. Agora que o `ontap_interface` microservice foi definido na solicitação e no `services.yml` arquivo, execute a solicitação novamente:

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

7. Faça login em cada instância do ONTAP e verifique se os LIFs foram adicionados.

Etapa 3: Opcionalmente, configure vários clusters

Se necessário, você pode configurar vários clusters na mesma solicitação. Você deve fornecer nomes de variáveis para cada cluster quando definir a solicitação.

Passos

1. Adicione uma entrada para o segundo cluster `cluster_initial.yml` no arquivo para configurar ambos os clusters na mesma solicitação.

   O exemplo a seguir exibe o `ontap_aggr` campo depois que a segunda entrada é adicionada.

```
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. Aplique as alterações para todos os outros itens em `cluster_initial`.
3. Adicione peering de cluster à solicitação copiando as seguintes linhas para o arquivo:

```
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. Execute a solicitação do Ansible:

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

Etapa 4: Configuração inicial da SVM

Nesta etapa do procedimento, você configura os SVMs no cluster.

Passos

1. Atualize a `svm_initial` solicitação no `tutorial-requests.yml` arquivo para configurar um relacionamento de pares SVM e SVM.

   Você deve configurar o seguinte:

   - O SVM

- O relacionamento entre pares SVM
- A interface SVM para cada SVM

2. Atualize as definições de variáveis nas `svm_initial` definições de solicitação. Você deve modificar as seguintes definições de variáveis:
   - `cluster_name`
   - `vserver_name`
   - `peer_cluster_name`
   - `peer_vserver`

   Para atualizar as definições, remova o * depois `req_details` para a `svm_initial` definição e adicione a definição correta.

3. Crie um arquivo `logic-tasks` na pasta para a solicitação de serviço. Por exemplo, crie um arquivo `svm_initial.yml` chamado .

   Copie as seguintes linhas para o arquivo:

```
- 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. Defina a `raw_service_request` variável.

   Pode utilizar uma das seguintes opções para definir a `raw_service_request` variável `svm_initial logic-tasks` na pasta:

   - **Opção 1**: Defina manualmente a `raw_service_request` variável.

     Abra o `tutorial-requests.yml` arquivo usando um editor e copie o conteúdo da linha 179 para a

linha 222. Cole o conteúdo sob a `raw service request` variável no novo `svm_initial.yml` arquivo, como mostrado nos exemplos a seguir:

Mostrar exemplo

Ficheiro de exemplo `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
```

- **Opção 2**: Use um modelo Jinja para definir a solicitação:

  Você também pode usar o seguinte formato de modelo Jinja para obter o `raw_service_request` valor.

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

5. Execute a solicitação:

```
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. Faça login em cada instância do ONTAP e valide a configuração.
7. Adicione as interfaces SVM.

   Defina `ontap_interface` o serviço em `svm_initial` `services.yml` no arquivo e execute a solicitação novamente:

```
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. Faça login em cada instância do ONTAP e verifique se as interfaces SVM foram configuradas.

Etapa 5: Opcionalmente, defina uma solicitação de serviço dinamicamente

Nas etapas anteriores, a `raw_service_request` variável é codificada por hardware. Isso é útil para aprendizado, desenvolvimento e teste. Você também pode gerar dinamicamente uma solicitação de serviço.

A seção a seguir fornece uma opção para produzir dinamicamente o necessário `raw_service_request` se você não quiser integrá-lo com sistemas de nível superior.

- Se a `logic_operation` variável não estiver definida no comando, o `logic.yml` arquivo não importa nenhum arquivo da `logic-tasks` pasta. Isso significa que o `raw_service_request` precisa ser definido fora do Ansible e fornecido à estrutura em execução.
- Um nome de arquivo de tarefa `logic-tasks` na pasta deve corresponder ao valor da `logic_operation` variável sem a extensão .yml.
- Os arquivos de tarefa na `logic-tasks` pasta definem dinamicamente um `raw_service_request`. o único requisito é que um válido `raw_service_request` seja definido como a última tarefa no arquivo relevante.

Como definir dinamicamente uma solicitação de serviço

Há várias maneiras de aplicar uma tarefa lógica para definir dinamicamente uma solicitação de serviço. Algumas destas opções estão listadas abaixo:

- Usando um arquivo de tarefa Ansible `logic-tasks` da pasta
- Invocando uma função personalizada que retorna dados adequados para converter para um `raw_service_request` varaible.
- Invocando outra ferramenta fora do ambiente Ansible para fornecer os dados necessários. Por exemplo, uma chamada de API REST para o Active IQ Unified Manager.

Os comandos de exemplo a seguir definem dinamicamente uma solicitação de serviço para cada cluster usando o `tutorial-requests.yml` arquivo:

```
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
```

Etapa 6: Implante a solução ONTAP Day 0/1

Nesta fase, você já deve ter completado o seguinte:

- Revisou e modificou todos os arquivos `playbooks/inventory/group_vars/all` de acordo com suas necessidades. Há comentários detalhados em cada arquivo para ajudá-lo a fazer as alterações.
- Adicionado todos os arquivos de tarefa necessários ao `logic-tasks` diretório.
- Adicionado todos os arquivos de dados necessários ao `playbook/vars` diretório.

Use os comandos a seguir para implantar a solução ONTAP day 0/1 e verificar a integridade da implantação:

> Nesta fase, você já deve ter descriptografado e modificado o `vault.yml` arquivo e ele deve ser criptografado com sua nova senha.

- Execute o serviço ONTAP Day 0:

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

- Execute o serviço ONTAP Day 1:

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

- Aplicar definições de largura do cluster:

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

- Executar verificações de integridade:

```
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>
```

Personalize a solução ONTAP Day 0/1

Para personalizar a solução do dia 0/1 do ONTAP de acordo com seus requisitos, você pode adicionar ou alterar as funções do Ansible.

As funções representam os microsserviços na estrutura do Ansible. Cada microservice executa uma operação. Por exemplo, o ONTAP Day 0 é um serviço que contém vários microsserviços.

Adicione funções do Ansible

Você pode adicionar funções do Ansible para personalizar a solução para seu ambiente. As funções necessárias são definidas pelas definições de serviço na estrutura do Ansible.

Uma função deve atender aos seguintes requisitos para ser usada como microsserviço:

- Aceite uma lista de argumentos na `args` variável.
- Use a estrutura "bloco, resgate, sempre" do Ansible com certos requisitos para cada bloco.
- Use um único módulo do Ansible e defina uma única tarefa dentro do bloco.
- Implemente todos os parâmetros do módulo disponíveis de acordo com os requisitos detalhados nesta secção.

Estrutura de microsserviço necessária

Cada função deve suportar as seguintes variáveis:

- `mode`: Se o modo estiver definido para `test` a função tenta importar o `test.yml` que mostra o que a função faz sem realmente executá-la.

  Nem sempre é possível implementar isso por causa de certas interdependências.

- `status`: O status geral da execução do playbook. Se o valor não estiver definido para `success` a função não será executado.
- `args` : Uma lista de dicionários específicos da função com chaves que correspondem aos nomes dos parâmetros da função.
- `global_log_messages`: Reúne mensagens de log durante a execução do manual de estratégia. Há uma entrada gerada cada vez que a função é executada.
- `log_name`: O nome usado para se referir à função dentro das `global_log_messages` entradas.
- `task_descr`: Uma breve descrição do que o papel faz.
- `service_start_time`: O carimbo de data/hora usado para rastrear a hora em que cada função é executada.
- `playbook_status`: O status do manual de estratégia do Ansible.
- `role_result`: A variável que contém a saída de função e é incluída em cada mensagem dentro das `global_log_messages` entradas.

Exemplo de estrutura de função

O exemplo a seguir fornece a estrutura básica de uma função que implementa um microservice. Você deve alterar as variáveis neste exemplo para sua configuração.

Mostrar exemplo

Estrutura básica da função:

```
- 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 ] }}"
```

Variáveis usadas na função de exemplo:

- `<NAME>`: Um valor substituível que deve ser fornecido para cada microsserviço.
- `<LOG_NAME>`: O nome do formulário curto da função usada para fins de Registro. Por exemplo, `ONTAP_VOLUME`.
- `<TASK_DESCRIPTION>`: Uma breve descrição do que o microservice faz.
- `<MODULE_NAME>`: O nome do módulo Ansible para a tarefa.

> O manual de estratégia de nível superior `execute.yml` especifica a `netapp.ontap` coleção. Se o módulo faz parte da `netapp.ontap` coleção, não há necessidade de especificar completamente o nome do módulo.

- `<MODULE_SPECIFIC_PARAMETERS>`: Parâmetros do módulo Ansible que são específicos do módulo usado para implementar o microservice. A lista a seguir descreve os tipos de parâmetros e como eles devem ser agrupados.
  - Parâmetros necessários: Todos os parâmetros necessários são especificados sem valor padrão.
  - Parâmetros que têm um valor padrão específico para o microservice (não o mesmo que um valor padrão especificado pela documentação do módulo).
  - Todos os parâmetros restantes usam `default(omit)` como valor padrão.

Usando dicionários de vários níveis como parâmetros do módulo

Alguns módulos do NetApp fornecem o Ansible que usam dicionários de vários níveis para parâmetros do módulo (por exemplo, grupos de políticas de QoS fixos e adaptáveis).

Usar `default(omit)` sozinho não funciona quando esses dicionários são usados, especialmente quando há mais de um e eles são mutuamente exclusivos.

Se você precisa usar dicionários de vários níveis como parâmetros de módulo, você deve dividir a funcionalidade em vários microsserviços (funções) para que cada um tenha a garantia de fornecer pelo menos um valor de dicionário de segundo nível para o dicionário relevante.

Os exemplos a seguir mostram grupos de políticas de QoS fixos e adaptáveis divididos em dois microsserviços.

O primeiro microservice contém valores fixos de grupo de políticas de QoS:

```
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)
}}"
```

O segundo microservice contém os valores do grupo de políticas de QoS adaptáveis:

```
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) }}"
```

Informações sobre direitos autorais

Informações sobre marcas comerciais