Guía de Configuración de CSR1000v HA Versión 2 en Microsoft Azure

Contenido

Introducción

Este documento sirve como guía de configuración complementaria para High Availability Version 2 (HAv2) en Azure. Encontrará detalles completos en la Guía de implementación de Cisco CSR 1000v para Microsoft Azure. HAv2 se soporta primero en Cisco IOS-XE® Denali 16.9.1s.

En HAv2, la implementación de HA se ha movido fuera del código Cisco IOS XE y se ejecuta en el contenedor del shell de invitado. Para obtener más información sobre el shell de invitado, vea la sección *Shell de invitado* en la Guía de Configuración de Programabilidad. En HAv2, la configuración de nodos de redundancia se realiza en el shell de invitado con un conjunto de scripts Python.

Prerequisites

Requirements

Cisco recomienda que tenga conocimiento sobre estos temas:

- Una cuenta de Microsoft Azure.
- Dos routers CSR1000v con dos interfaces gigabit. La interfaz de cara externa debe estar en GigabitEthernet1 (eth0).
- Un mínimo de Cisco IOS-XE® Denali 16.9.1s.

Componentes Utilizados

La información de este documento se basa en Cisco IOS-XE® Denali 16.9.1s implementado

nativamente desde Azure Marketplace.

Los recursos implementados en Azure a partir de los pasos de este documento pueden incurrir en un costo.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. Si tiene una red en vivo, asegúrese de entender el posible impacto de cualquier comando.

Restricciones

- La interfaz de cara pública externa debe configurarse en eth0, que corresponde a GigabitEthernet1. El acceso al servidor de metadatos de Azure sólo se puede lograr a través de la interfaz principal de una máquina virtual.
- Si existe la configuración del IOS de HAv1, debe eliminarse antes de la configuración de HAv2 en el shell de invitado. La configuración HAv1 consta de los comandos **redundancy** y **cloud Provider**.

Configurar

Paso 1. Configure IOX para el alojamiento de aplicaciones.

1. Habilite el alojamiento de aplicaciones IOX. Asigne una dirección ip privada a VirtualPortGroup0. NAT VirtualPortGroup0 con la interfaz de cara pública para permitir que el invitado llegue a Internet. En este ejemplo, la ip de GigabitEthernet1 es 10.3.0.4.

```
vrf definition GS
!
iox
app-hosting appid guestshell
app-vnic gateway1 virtualportgroup 0 guest-interface 0
guest-ipaddress 192.168.35.102 netmask 255.255.255.0
app-default-gateway 192.168.35.101 guest-interface 0
name-server0 8.8.8.8
!
interface VirtualPortGroup0
vrf forwarding GS
ip address 192.168.35.101 255.255.255.0
ip nat inside
!
interface GigabitEthernet1
ip nat outside
!
ip access-list standard GS_NAT_ACL
permit 192.168.35.0 0.0.0.255
!
ip nat inside source list GS_NAT_ACL interface GigabitEthernet1 vrf GS overload
!
! The static route points to the gig1 private ip address gateway
ip route vrf GS 0.0.0.0 0.0.0.0 GigabitEthernet1 10.1.0.1 global
```

> **Nota:** Las nuevas instancias implementadas desde Azure Marketplace pueden tener iox preconfigurado.

Paso 2. Instale los paquetes Python en Guestshell.

1. Habilite el shell de invitado e login.
```
csr-1#guestshell enable
csr-1#guestshell
```
2. Ping www.google.com para verificar que el invitado puede llegar a internet. Si no se puede alcanzar, verifique la configuración del servidor de nombres en la configuración del IOS de alojamiento de aplicaciones o agregue un servidor en resolv.conf en el shell de invitado.
```
[guestshell@guestshell ~]$ ping www.google.com
PING www.google.com (172.217.14.228) 56(84) bytes of data.
64 bytes from sea30s02-in-f4.1e100.net (172.217.14.228): icmp_seq=1 ttl=51 time=4.89 ms
64 bytes from sea30s02-in-f4.1e100.net (172.217.14.228): icmp_seq=2 ttl=51 time=5.02 ms
```
Ejecute curl para verificar que los metadatos se puedan recuperar. La interfaz de cara externa debe ser Gig1 (eth0). De lo contrario, verifique los grupos de seguridad de Azure, el enrutamiento u otras características que podrían bloquear 169.254.169.254. 169.254.169.254 no es una dirección que se puede hacer ping.
```
[guestshell@guestshell ~]$ curl -H Metadata:true
"http://169.254.169.254/metadata/instance?api-version=2018-04-02"
{"compute":{"location":"westus2","name":"csr-david-2","offer":"cisco-csr-
1000v","osType":"Linux","placementGroupId":"","plan":{"name":"16_7","product":"cisco-csr-
1000v","publisher":"cisco"},"platformFaultDomain":"0","platformUpdateDomain":"0","publicKey
s":[],"publisher":"cisco","resourceGroupName":"RG-David-
2","sku":"16_7","subscriptionId":"09e13fd4-def2-46aa-a056-
xxxxxxxxxxx","tags":"","version":"16.7.120171201","vmId":"f8f32b48-daa0-4053-8ba4-
xxxxxxxxxxxxx","vmScaleSetName":"","vmSize":"Standard_DS2_v2","zone":""},"network":{"interf
ace":[{"ipv4":{"ipAddress":[{"privateIpAddress":"10.3.0.5","publicIpAddress":"21.53.135.210
"}],"subnet":[{"address":"10.3.0.0","prefix":"24"}]},"ipv6":{"ipAddress":[]},"macAddress":"
000D3A93F"},{"ipv4":{"ipAddress":[{"privateIpAddress":"10.3.1.5","publicIpAddress":""}],"su
bnet":[{"address":"10.3.1.0","prefix":"24"}]},"ipv6":{"ipAddress":[]},"macAddress":"000D3A9
61"}]}}[
```

3. Instale los paquetes python. **Nota:** No utilice el modo sudo para instalar paquetes. Asegúrese de utilizar la opción **—user**. Si no se realizan los tres pasos, se instalarán los paquetes en la carpeta incorrecta. Esto puede dar lugar a ImportErrors. Para reparar paquetes instalados incorrectamente, puede que necesite ejecutar el comando IOS **guestshell detect** y comenzar de nuevo.
```
[guestshell@guestshell ~]$ pip install csr_azure_guestshell~=1.1 --user
[guestshell@guestshell ~]$ pip install csr_azure_ha~=1.0 --user
[guestshell@guestshell ~]$ source ~/.bashrc
```
4. Asegúrese de que los paquetes estén correctamente instalados en **/home/guestshell/.local/lib/python2.7/site-packages**.
```
[guestshell@guestshell ~]$ which show_node.py
~/.local/lib/python2.7/site-packages/csr_azure_ha/client_api/show_node.py
```

Paso 3. Configuración de la autenticación para llamadas API CSR1000v.

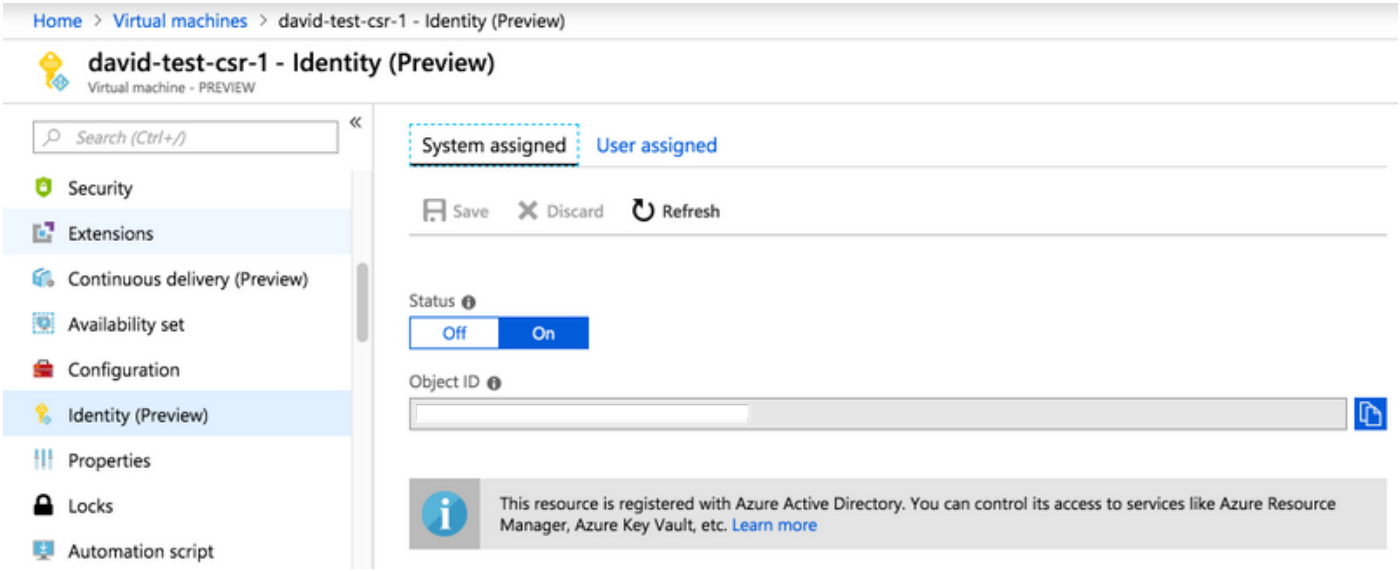
Hay dos métodos para permitir que el CSR1000v realice llamadas de API a Azure.

1. Azure Active Directory (AAD): Método estándar HAv1 que también se puede utilizar en HAv2. Anote la **ID del arrendatario, la ID de la aplicación, la clave de la aplicación** que se utilizará en la secuencia de comandos create_node.py. Visite Crear una aplicación en un Active Directory de Microsoft Azure para obtener más detalles. **Nota:** La clave de aplicación utilizada en HAv1 es la clave codificada. La clave de aplicación utilizada en HAv2 es la clave

no codificada. Si no ha anotado la clave no codificada, es posible que deba crear una nueva, ya que las claves no se pueden recuperar.

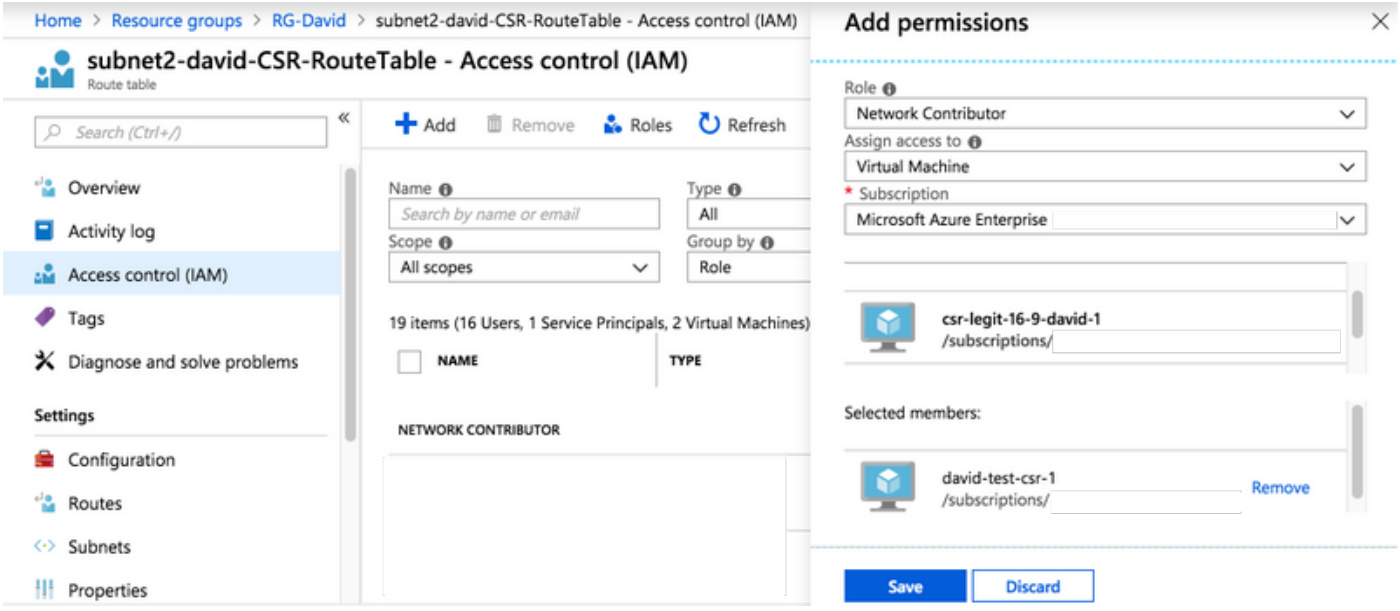
2. Microsoft dispone de un servicio de identidad de servicio gestionado (MSI) que automatiza la creación de una aplicación para una máquina virtual. Para obtener más información sobre MSI, visite https://docs.microsoft.com/en-us/azure/active-directory/managed-service-identity/overview. La versión 2 de HA puede utilizar el servicio MSI para autenticar Cisco CSR 1000v. La versión 1 de HA no puede utilizar MSI.

Paso 1. Habilite MSI para cada una de las máquinas virtuales CSR1000v. Navegue hasta la VM en Azure Portal. Navegue hasta **Identidad** y haga clic en **Sistema asignado > En > Guardar**.



Paso 2. En **Subnet Route Table,** para permitir llamadas de API desde el router CSR1000v, elija **Access Control (IAM)** y haga clic en **Add**.

Paso 3. Elija **Role - Network Contributor**. Elija **Asignar acceso a - Máquina virtual**. Elija la **suscripción** adecuada. Seleccione la VM de la lista que tiene su MSI activado.



Paso 4. Configure HAv2 en Guestshell.

1. Utilice el script **create_node.py** para agregar las configuraciones HA. Para comprobar todas las definiciones de parámetros del indicador, vea las Tablas 3 y 4 de la Guía de implementación de Cisco CSR 1000v para Microsoft Azure. Este ejemplo utiliza la autenticación AAD que requiere los indicadores **app-id (a)**, **tenant-id (d)** y **app-key (k)**. Si utiliza autenticación MSI, estos indicadores adicionales no son necesarios. El **indicador node [-i]** es un número arbitrario. Utilice números de nodo únicos para crear varios nodos si se requieren actualizaciones en varias tablas de ruta.
   ```
   create_node.py -i 100 -p azure -s 09e13fd4-def2-46aa-a056-xxxxxxxxxx -g RG-David -t
   subnet2-david-CSR-RouteTable -r 8.8.8.8/32 -n 10.3.1.4 -a 1e0f69c3-b6aa-46cf-b5f9-
   xxxxxxxxx -d ae49849c-2622-4d45-b95e-xxxxxxxxx -k bDEN1k8batJqpeqjAuUvaUCZn5Md6rWEi=
   ```
2. Utilice **set_params.py** para agregar o cambiar parámetros individuales.
   ```
   set_params.py -i 100 [option1] [option2]
   ```
3. Utilice **clear_params.py** para borrar parámetros individuales.
   ```
   clear_params.py -i 100 [option1] [option2]
   ```
4. Utilice **delete_node.py** para eliminar el nodo.
   ```
   delete_node.py -i 100
   ```

Paso 5. Configure EEM para activar los fallos.

La opción **node_event.py** con peerFail es cómo HAv2 activa una conmutación por fallas y actualiza la Azure Route Table. Aquí es donde tiene la flexibilidad para programar su propia lógica. Puede utilizar EEM dentro del IOS para ejecutar **node_event.py**, o escribir una secuencia de comandos python dentro del shell de invitado.

Un ejemplo es detectar un estado inactivo de una interfaz con EEM para activar **node_event.py**.

```
event manager applet HAv2_interface_flap
 event syslog pattern "Interface GigabitEthernet2, changed state to down"
 action 1 cli command "enable"
 action 2 cli command "guestshell run node_event.py -i 100 -e peerFail"
```

Puede ejecutar manualmente **node_event.py** en el shell de invitado para probar un failover real.

```
[guestshell@guestshell ~]$ node_event.py -i 100 -e peerFail
```

El HAv2 también puede revertir la ruta nuevamente al router original con la opción **revert**. Esta es una configuración opcional que simula la prevención. El indicador **-m primario** en **create_node.py** se requiere para ser configurado en el router primario**.** Este es un ejemplo que utiliza BFD para monitorear el estado de la interfaz.

```
event manager applet bfd_session_up
 event syslog pattern ".*BFD_SESS_UP.*"
 action 1 cli command "enable"
 action 2 cli command "guestshell run node_event.py -i 100 -e revert"

[guestshell@guestshell ~]$ set_params.py -i 100 -m
```

Verificación

1. Asegúrese de que los tres procesos estén activos.
   ```
   systemctl status auth-token
   systemctl status azure-ha
   systemctl status waagent
   ```

2. Reinicie los que hayan fallado.
```
sudo systemctl start waagent
sudo systemctl start azure-ha
sudo systemctl start auth-token
```
3. Dos métodos para verificar la configuración agregada por **create_node.py**.
```
show_node.py -i 100

[guestshell@guestshell ~]$ cat azure/HA/node_file
{'appKey': 'bDEN1k8batJqWEiGXAxSR4Y=', 'index': '100', 'routeTableName': 'subnet2-david-
CSR-RouteTable', 'route': '8.8.8.8/32', 'nextHop': '10.3.1.4', 'tenantId': 'ae49849c-2622-
4d45-b95e-xxxxxxxxxx', 'resourceGroup': 'RG-David', 'appId': '1e0f69c3-b6aa-46cf-b5f9-
xxxxxxxxxx', 'subscriptionId': '09e13fd4-def2-46aa-a056-xxxxxxxxxxx', 'cloud': 'azure'}
```
4. Simula un failover en el router en espera. Esto en realidad no causa una conmutación por fallas pero verifica que la configuración sea válida. Verifique los registros en el paso 6.
```
node_event.py -i 100 -e verify
```
5. Activa un evento de failover real en el router en espera. En Azure, verifique si la tabla de rutas actualizó la ruta al nuevo salto. Verifique los registros en el paso 6.

```
node_event.py -i 100 -e peerFail
```
6. **node_event.py** genera 2 tipos de registros cuando se activa. Esto es útil para verificar si el failover fue exitoso o para resolver problemas. Cada vez se generan nuevos archivos de eventos. Sin embargo, **routeTableGetRsp** se sobrescribe cada vez, por lo que generalmente hay un archivo.
```
[guestshell@guestshell ~]$ ls -latr /home/guestshell/azure/HA/events/
total 5
drwxr-xr-x 3 guestshell root 1024 Sep 18 23:01 ..
drwxr-xr-x 2 guestshell root 1024 Sep 19 19:40 .
-rw-r--r-- 1 guestshell guestshell 144 Sep 19 19:40 routeTableGetRsp
-rw-r--r-- 1 guestshell guestshell 390 Sep 19 19:40 event.2018-09-19 19:40:28.341616
-rw-r--r-- 1 guestshell guestshell 541 Sep 18 23:09 event.2018-09-18 23:09:58.413523
```

Troubleshoot

Paso 1. Los paquetes Python se instalan erróneamente en **/usr/lib/python2.7/site-packages/.** Destruya el shell de invitado y siga los pasos de configuración.

```
[guestshell@guestshell ~]$ create_node.py -h
bash: create_node.py: command not found

[guestshell@guestshell ~]$ ls /usr/lib/python2.7/site-packages/
```
La trayectoria de instalación correcta es **~/.local/lib/python2.7/site-packages/**.

```
[guestshell@guestshell ~]$ which show_node.py
~/.local/lib/python2.7/site-packages/csr_azure_ha/client_api/show_node.py
```
Paso 2. Si la autenticación no se configuró correctamente en el paso 3, se pueden generar errores de token. Para la autenticación AAD, si la **clave de aplicación** utilizada no es válida, o la URL codificada, se pueden ver errores de autenticación después de que **node_event.py** se active.

```
[guestshell@guestshell ~]$ cat /home/guestshell/azure/HA/events/routeTableGetRsp
{"error":{"code":"AuthenticationFailedMissingToken","message":"Authentication failed. The
'Authorization' header is missing the access token."}}
```

```
[guestshell@guestshell ~]$ cat /home/guestshell/azure/HA/events/event.2018-09-19\
23\:02\:55.581684

Event type is verify
appKey zGuYMyXQha5Kqe8xdufhUJ9eX%2B1zIhLsuw%3D
index 100
routeTableName subnet2-david-CSR-RouteTable
route 8.8.8.8/32
nextHop 10.3.1.4
tenantId ae49849c-2622-4d45-b95e-xxxxxxxxxx
resourceGroup RG-David
appId 1e0f69c3-b6aa-46cf-b5f9-xxxxxxxxxx
subscriptionId 09e13fd4-def2-46aa-a056-xxxxxxxxxx
cloud azure
All required parameters have been provided
Requesting token using Azure Active Directory
Token=
Failed to obtain token
Reading route table
Route GET request failed with code 401
```

Paso 3. Si el **ID de arrendatario** o **app-id** es incorrecto.

```
[guestshell@guestshell ~]$ cat azure/tools/TokenMgr/token_get_rsp
{"error":"invalid_request","error_description":"AADSTS90002: Tenant 1e0f69c3-b6aa-46cf-b5f9-
xxxxxxxxxx not found. This may happen if there are no active subscriptions for the tenant. Check
with your subscription administrator.\r\nTrace ID: 8bc80efc-f086-46ec-83b9-
xxxxxxxxxxx\r\nCorrelation ID: 2c6062f8-3a40-4b0e-83ec-xxxxxxxxxxx\r\nTimestamp: 2018-09-19
23:58:02Z","error_codes":[90002],"timestamp":"2018-09-19 23:58:02Z","trace_id":"8bc80efc-f086-
46ec-83b9-xxxxxxxxxxx","correlation_id":"2c6062f8-3a40-4b0e-83ec-xxxxxxxxxx"}
```

Paso 4. Durante la instalación del paquete, se pudo haber utilizado el modo **sudo**, **—el usuario** no estaba incluido o **origen ~/.bashrc** no se ejecutó. Esto hace que **create_node.py** falle o genere un ImportError.

```
[guestshell@guestshell ~]$ create_node.py -i 1 -p azure -s d91490ec -g RG -t RT -r 10.12.0.0/11
-n 10.2.0.31 -m secondary
/usr/lib64/python2.7/site-packages/cryptography/hazmat/primitives/constant_time.py:26:
CryptographyDeprecationWarning: Support for your Python version is deprecated. The next version
of cryptography will remove support. Please upgrade to a 2.7.x release that supports
hmac.compare_digest as soon as possible.
utils.DeprecatedIn23,
create_node -i 1 -p azure -s d91490ec -g RG -t RT -r 10.12.0.0/11 -n 10.2.0.31 -m secondary
failed

[guestshell@guestshell ~]$ create_node.py -i 1 -p azure -s d91490ec -g RG -t RT -r 10.1.0.0/18 -
n 10.2.0.31 -m secondary
Traceback (most recent call last):
  File "/usr/bin/create_node.py", line 5, in
    import ha_api
ImportError: No module named ha_api
```

Paso 5. Verifique el historial de instalación del paquete.

```
[guestshell@guestshell ~]$ cat azure/HA/install.log
Installing the Azure high availability package
Show the current PATH
/usr/local/bin:/usr/bin:/home/guestshell/.local/lib/python2.7/site-
packages/csr_azure_ha/client_api
Show the current PYTHONPATH
```

```
:/home/guestshell/.local/lib/python2.7/site-
packages/csr_azure_guestshell:/home/guestshell/.local/lib/python2.7/site-
packages/csr_azure_guestshell/TokenMgr:/home/guestshell/.local/lib/python2.7/site-
packages/csr_azure_guestshell/MetadataMgr:/home/guestshell/.local/lib/python2.7/site-
packages/csr_azure_guestshell/bin:/home/guestshell/.local/lib/python2.7/site-
packages/csr_azure_ha/client_api:/home/guestshell/.local/lib/python2.7/site-
packages/csr_azure_ha/server
```

Paso 6. Verifique los registros de configuración de HA.

```
[guestshell@guestshell ~]$ cat azure/HA/azha.log
2018-09-24 16:56:29.215743 High availability server started with pid=7279
2018-09-24 17:03:20.602579 Server processing create_node command
2018-09-24 17:03:20.602729 Created new node with index 100
```

Paso 6. Ejecute el script debug_ha.sh para recopilar todos los archivos de registro en un único archivo .tar.

```
[guestshell@guestshell ~]$ bash ~/azure/HA/debug_ha.sh
```

El archivo se coloca en la memoria flash de inicialización a la que se puede acceder desde el shell de invitado y desde el IOS.

```
[guestshell@guestshell ~]$ ls /bootflash/ha_debug.tar
/bootflash/ha_debug.tar

csr-david-2#dir | i debug
   28  -rw-            92160  Sep 27 2018 22:42:54 +00:00  ha_debug.tar
```