亚马逊云科技

Administrator Guide

NICE DCV Session Manager

NICE DCV Session Manager: Administrator Guide

Table of Contents

What is NICE DCV Session Manager?

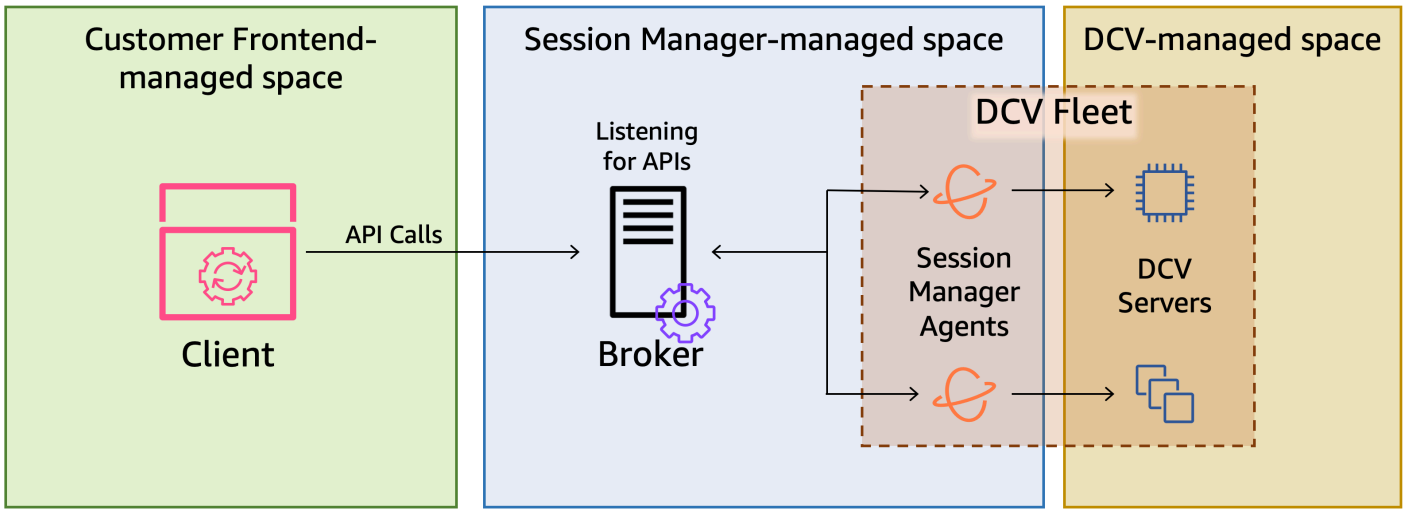
NICE DCV Session Manager is set of installable software packages (an Agent and a Broker) and an application programming interface (API) that makes it easy for developers and independent software vendors (ISVs) to build front-end applications that programmatically create and manage the lifecycle of NICE DCV sessions across a fleet of NICE DCV servers.

This guide explains how to install and configure the Session Manager Agent and Broker. For more information about using the Session Manager APIs, see the *NICE DCV Session Manager Developer Guide*.

Topics

How Session Manager works

The following diagram shows the high-level components of Session Manager.



Broker

The Broker is a web server that hosts and exposes the Session Manager APIs. It receives and processes *API* requests to manage NICE DCV sessions from the *client*, and then passes the instructions to the relevant *Agents*. The Broker must be installed on a host that is separate from your NICE DCV servers, but it must be accessible to the client, and it must be able to access the Agents.

Agent

The Agent is installed on each NICE DCV server in the fleet. The Agents receive instructions from the *Broker* and run them on their respective NICE DCV servers. The Agents also monitor the state of the NICE DCV servers, and send periodic status updates back to the Broker.

APIs

Session Manager exposes a set of REST application programming interfaces (APIs) that can be used to manage NICE DCV sessions on a fleet of NICE DCV servers. The APIs are hosted on and exposed by the *Broker*. Developers can build custom session management *clients* that call the APIs.

Client

The client is the front-end application or portal that you develop to call the Session Manager *APIs* that are exposed by the *Broker*. End users use the client to manage the sessions hosted on the NICE DCV servers in the fleet.

Access token

In order to make an API request, you must provide an access token. Tokens can be requested from the Broker, or an external authorization server, by registered client APIs. To request and access token, the client API must provide valid credentials.

Client API

The client API is generated from the Session Manager API definition YAML file, using Swagger Codegen. The client API is used to make API requests.

NICE DCV session

You must create a NICE DCV session on your NICE DCV server that your clients can connect to. Clients can only connect to a NICE DCV server if there is an active session. NICE DCV supports console and virtual sessions. You use the Session Manager APIs to manage the lifecycle of NICE DCV sessions. NICE DCV sessions can be in one of the following states:

- `CREATING`—the Broker is in the process of creating the session.
- `READY`—the session is ready to accept client connections.
- `DELETING`—the session is being deleted.
- `DELETED`—the session has been deleted.
- `UNKNOWN`—unable to determine the session's state. The Broker and the Agent might be unable to communicate.

Features

DCV Session Manager offers the following features:

- **Provides NICE DCV session information**—get information about the sessions running on multiple NICE DCV servers.
- **Manage the lifecycle for multiple NICE DCV sessions**—create or delete multiple sessions for multiple users across multiple NICE DCV servers with one API request.
- **Supports tags**—use custom tags to target a group of NICE DCV servers when creating sessions.
- **Manages permissions for multiple NICE DCV sessions**—modify user permissions for multiple sessions with one API request.
- **Provides connection information**—retrieve client connection information for NICE DCV sessions.
- **Supports for cloud and on-premises**—use Session Manager on Amazon, on-premises, or with alternative cloud-based servers.

Limitations

Session Manager does not provide resource provisioning capabilities. If you are running NICE DCV on Amazon EC2 instances, you might need to use additional Amazon services, such as Amazon EC2 Auto Scaling to manage the scaling of your infrastructure.

Pricing

Session Manager is available at no cost for Amazon customers running EC2 instances.

On-premises customers require a NICE DCV Plus or DCV Professional Plus license. For information about how to purchase a NICE DCV Plus or NICE DCV Professional Plus license, see How to Buy on

the NICE website and find a NICE distributor or reseller in your region. To allow all on-premises customers to experiment with the DCV Session Manager, the licensing requirements will only be enforced starting with NICE DCV version 2021.0.

For more information, see Licensing the NICE DCV Server in the *NICE DCV Administrator Guide*.

NICE DCV Session Manager requirements

The NICE DCV Session Manager Agent and Broker have the following requirements.

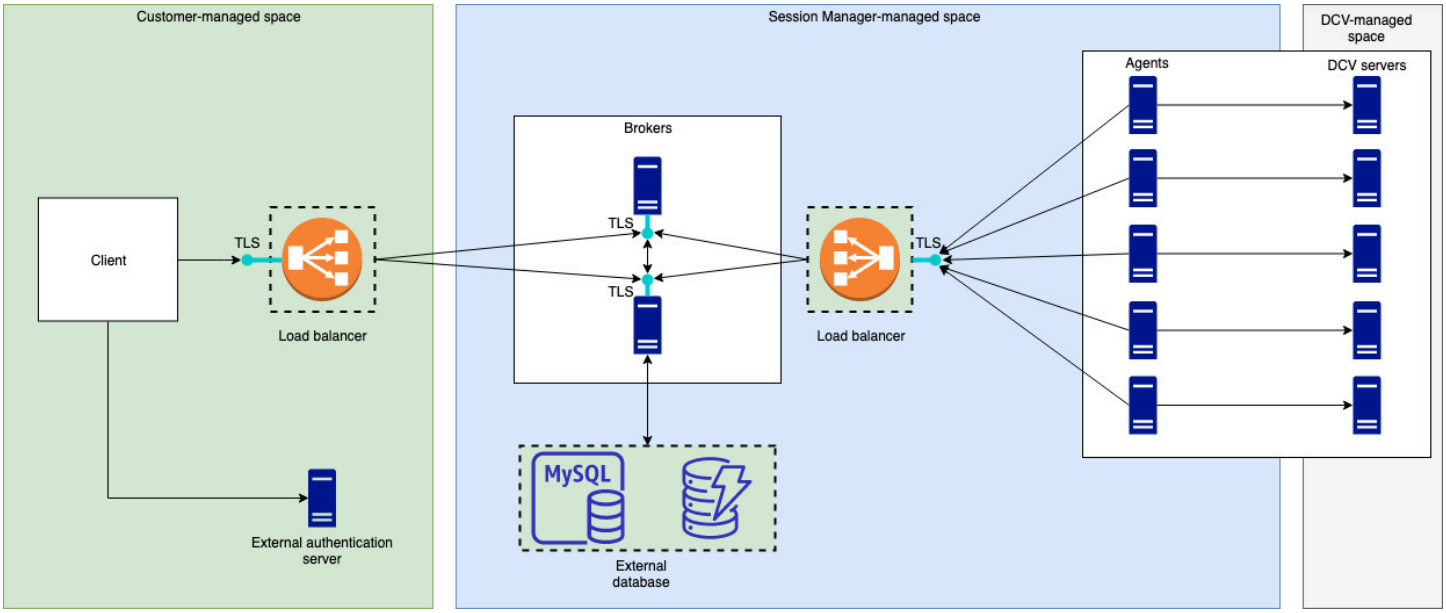
	Broker	Agent

Operating system	• Amazon Linux 2
	• CentOS 7.6 or later
	• CentOS Stream 8
	• CentOS Stream 9
	• RHEL 7.6 or later
	• RHEL 8.x
	• RHEL 9.x
	• Rocky Linux 8.5 or later
	• Rocky Linux 9.x
	• Ubuntu 20.04
	• Ubuntu 22.04
		• Windows
		  • Windows Server 2022
		  • Windows Server 2019
		  • Windows Server 2016
		• Linux server
		  • Amazon Linux 2
		  • CentOS 7.6 or later
		  • CentOS Stream 8
		  • CentOS Stream 9
		  • RHEL 7.6 or later
		  • RHEL 8.x
		  • RHEL 9.x
		  • Rocky Linux 8.5 or later
		  • Rocky Linux 9.x
		  • Ubuntu 20.04
		  • Ubuntu 22.04
		  • SUSE Linux Enterprise 12 with SP4 or later
		  • SUSE Linux Enterprise 15

	Broker	Agent
Architecture	• 64-bit x86 • 64-bit ARM	• 64-bit x86 • 64-bit ARM (Amazon Linux 2, CentOS 7.x/8.x/9.x, RHEL 7.x/8.x/9.x and Rocky 8.x/9.x only) • 64-bit ARM (Ubuntu 22.04)
Memory	8 GB	4 GB
NICE DCV version	NICE DCV 2020.2 and later	NICE DCV 2020.2 and later
Additional requirements	Java 11	-

Networking and connectivity requirements

The following diagram provides a high-level overview of the Session Manager networking and connectivity requirements.



The **Broker** must be installed on a separate host, but it must have network connectivity with the Agents on the NICE DCV servers. If you choose to have multiple Brokers to improve availability,

then you must install and configure each broker on a separate host, and use one or more load balancers to manage the traffic between the client and the Brokers, and the Brokers and the Agents. The Brokers should also be able to communicate with each other in order to exchange information about the NICE DCV servers and sessions. The Brokers can store their keys and status data on an external database and have this information available after reboot or termination. This helps mitigate the risk of losing important Broker information by persisting it on the external database. You can retrieve it later. If you choose to have it, then you must set up the external database and configure the brokers. DynamoDB, MariaDB, and MySQL are supported. You can find configuration parameters are listed on the Broker Configuration File.

The **Agents** must be able to initiate secure, persistent, bi-directional HTTPs connections with the Broker.

Your **client**, or frontend application, must be able to access the Broker in order to call the APIs. The client should also be able to access your authentication server.

Setting up NICE DCV Session Manager

The following section explains how to install Session Manager with a single Broker and multiple Agents. You can use multiple Brokers to improve scalability and performance. For more information, see Scaling Session Manager.

To set up NICE DCV Session Manager, do the following:

Steps

Step 1: Prepare the NICE DCV servers

You must have a fleet of NICE DCV servers with which you intend to use Session Manager. For more information about installing NICE DCV servers, see Installing the NICE DCV server in the *NICE DCV Administrator Guide*.

On Linux NICE DCV servers, Session Manager uses a local service user named `dcvsmagent`. This user is automatically created when the Session Manager Agent is installed. You must grant this service user administrator privileges for NICE DCV so that it can perform actions on behalf of other users. To grant the Session Manager service user administrator privileges, do the following:

To add the local service user for Linux NICE DCV servers

1. Open `/etc/dcv/dcv.conf` using your preferred text editor.
2. Add the `administrators` parameter to the `[security]` section and specify the Session Manager user. For example:

   ```
   [security]
   administrators=["dcvsmagent"]
   ```

3. Save and close the file.
4. Stop and restart the NICE DCV server.

Session Manager is only able to create NICE DCV sessions on behalf of users that already exist on the NICE DCV server. If a request is made to create a session for a user that doesn't exist, the request fails. Therefore, you must ensure that each intended end user has a valid system user on the NICE DCV server.

> **Tip**
>
> If you intend to use multiple Broker hosts or NICE DCV servers with Agents, we recommend that you configure only one Broker and one NICE DCV server with an Agent by performing the following steps, creating Amazon Machine Images (AMI) of the hosts with the completed configurations, and then using the AMIs to launch the remaining Brokers and NICE DCV servers. Alternatively, you can use Amazon Systems Manager to run the commands on multiple instances remotely.

Step 2: Set up the NICE DCV Session Manager Broker

The Broker must be installed on a Linux host. For more information about the supported Linux distributions, see NICE DCV Session Manager requirements. Install the Broker on a host that is separate from the Agent and the NICE DCV server host. The host can be installed on a different private network, but it must be able to connect to and communicate with the Agent.

To install and start the Broker

1. Connect to the host on which you intend to install the Broker.
2. The packages are digitally signed with a secure GPG signature. To allow the package manager to verify the package signature, you must import the NICE GPG key. Run the following command to import the NICE GPG key.

   - Amazon Linux 2, RHEL, CentOS, and Rocky Linux

     ```
     $ sudo rpm --import https://d1uj6qtbmh3dt5.cloudfront.net/NICE-GPG-KEY
     ```

   - Ubuntu

     ```
     $ wget https://d1uj6qtbmh3dt5.cloudfront.net/NICE-GPG-KEY gpg --import NICE-GPG-KEY
     ```

3. Download the installation package.

- Amazon Linux 2, RHEL 7.x, and CentOS 7.x

```
$ wget https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerBrokers/nice-
dcv-session-manager-broker-2023.1.410-1.el7.noarch.rpm
```

- RHEL 8.x, CentOS Stream 8, and Rocky Linux 8.x

```
$ wget https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerBrokers/nice-
dcv-session-manager-broker-2023.1.410-1.el8.noarch.rpm
```

- Ubuntu 20.04

```
$ wget https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerBrokers/nice-
dcv-session-manager-broker_2023.1.410-1_all.ubuntu2004.deb
```

- Ubuntu 22.04

```
$ wget https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerBrokers/nice-
dcv-session-manager-broker_2023.1.410-1_all.ubuntu2204.deb
```

4. Install the package.

- Amazon Linux 2, RHEL 7.x, and CentOS 7.x

```
$ sudo yum install -y ./nice-dcv-session-manager-
broker-2023.1.410-1.el7.noarch.rpm
```

- RHEL 8.x, Stream CentOS 8, and Rocky Linux 8.x

```
$ sudo yum install -y ./nice-dcv-session-manager-
broker-2023.1.410-1.el8.noarch.rpm
```

- Ubuntu 20.04

```
$ sudo apt install -y ./nice-dcv-session-manager-
broker_2023.1.410-1_all.ubuntu2004.deb
```

- Ubuntu 22.04

```
$ sudo apt install -y ./nice-dcv-session-manager-
broker_2023.1.410-1_all.ubuntu2204.deb
```

5. Check that the default Java environment version is *11*

   ```
   $ java -version
   ```

   If not, you can explicitly set the Java home directory that the Broker will use to target the right Java version. This is done setting the parameter `broker-java-home` in the Broker configuration file. For more information, see Broker Configuration File.

6. Start the Broker service and ensure that it starts automatically every time the instance starts.

   ```
   $  sudo systemctl start dcv-session-manager-broker && sudo systemctl enable dcv-session-manager-broker
   ```

7. Place a copy of the Broker's self-signed certificate in your user directory. You'll need it when you install the Agents in the next step.

   ```
   sudo cp /var/lib/dcvsmbroker/security/dcvsmbroker_ca.pem $HOME
   ```

Step 3: Set up the NICE DCV Session Manager Agent

The Agent must be installed on all of the NICE DCV server hosts in the fleet. The Agent can be installed on both Windows and Linux servers. For more information about the supported operating systems, see NICE DCV Session Manager requirements.

Prerequisites

The NICE DCV server must be installed on the host before installing the Agent.

Linux host

> **Note**
>
> The Session Manager Agent is available for the Linux distributions and architectures listed in Requirements:
> The following instructions are for installing the Agent on 64-bit x86 hosts. To install the Agent on 64-bit ARM hosts replace *x86_64* with `aarch64`. For Ubuntu, replace *amd64* with `arm64`.

To install the Agent on a Linux host

1. The packages are digitally signed with a secure GPG signature. To allow the package manager to verify the package signature, you must import the NICE GPG key. Run the following command to import the NICE GPG key.

   - Amazon Linux 2, RHEL, CentOS, and SUSE Linux Enterprise

     ```
     $ sudo rpm --import https://d1uj6qtbmh3dt5.cloudfront.net/NICE-GPG-KEY
     ```

   - Ubuntu

     ```
     $ wget https://d1uj6qtbmh3dt5.cloudfront.net/NICE-GPG-KEY
     ```

     ```
     $ gpg --import NICE-GPG-KEY
     ```

2. Download the installation package.

   - Amazon Linux 2, RHEL 7.x, and CentOS 7.x

     ```
     $  wget https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerAgents/
     nice-dcv-session-manager-agent-2023.1.732-1.el7.x86_64.rpm
     ```

   - RHEL 8.x, CentOS Stream 8, and Rocky Linux 8.x

     ```
     $  wget https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerAgents/
     nice-dcv-session-manager-agent-2023.1.732-1.el8.x86_64.rpm
     ```

   - Ubuntu 20.04

     ```
     $  wget https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerAgents/
     nice-dcv-session-manager-agent_2023.1.732-1_amd64.ubuntu2004.deb
     ```

   - Ubuntu 22.04

     ```
     $  wget https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerAgents/
     nice-dcv-session-manager-agent_2023.1.732-1_amd64.ubuntu2204.deb
     ```

   - SUSE Linux Enterprise 12

```
$ curl -O https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerAgents/
nice-dcv-session-manager-agent-2023.1.732-1.sles12.x86_64.rpm
```

- SUSE Linux Enterprise 15

```
$ curl -O https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerAgents/
nice-dcv-session-manager-agent-2023.1.732-1.sles15.x86_64.rpm
```

3. Install the package.

- Amazon Linux 2, RHEL 7.x, and CentOS 7.x

```
$  sudo yum install -y ./nice-dcv-session-manager-
agent-2023.1.732-1.el7.x86_64.rpm
```

- RHEL 8.x, CentOS Stream 8, and Rocky Linux 8.x

```
$  sudo yum install -y ./nice-dcv-session-manager-
agent-2023.1.732-1.el8.x86_64.rpm
```

- Ubuntu 20.04

```
$  sudo apt install ./nice-dcv-session-manager-
agent_2023.1.732-1_amd64.ubuntu2004.deb
```

- Ubuntu 22.04

```
$  sudo apt install ./nice-dcv-session-manager-
agent_2023.1.732-1_amd64.ubuntu2204.deb
```

- SUSE Linux Enterprise 12

```
$ sudo zypper install ./nice-dcv-session-manager-
agent-2023.1.732-1.sles12.x86_64.rpm
```

- SUSE Linux Enterprise 15

```
$ sudo zypper install ./nice-dcv-session-manager-
agent-2023.1.732-1.sles15.x86_64.rpm
```

4. Place a copy of the Broker's self-signed certificate (that you copied in the previous step) in the `/etc/dcv-session-manager-agent/` directory on the Agent.
5. Open `/etc/dcv-session-manager-agent/agent.conf` using your preferred text editor and do the following.

   - For `broker_host`, specify the DNS name of the host on which the Broker is installed.

     > **Important**
     >
     > If the Broker is running on an Amazon EC2 instance, for `broker_host` you must specify the instance's private Ipv4 address.

   - (Optional) For `broker_port`, specify the port over which to communicate with the Broker. By default the Agent and the Broker communicate over port 8445. Only change this if you need to use a different port. If you do change it, ensure that the Broker is configured to use the same port.
   - For `ca_file`, specify the full path the certificate file that you copied in the previous step. For example:

     ```
     ca_file = '/etc/dcv-session-manager-agent/broker_cert.pem'
     ```

     Alternatively, if you want to disable TLS verification, set `tls_strict` to `false`.
6. Save and close the file.
7. Run the following command to start the Agent.

   ```
   $ sudo systemctl start dcv-session-manager-agent
   ```

Windows host

To install the Agent on a Windows host

1. Download the Agent installer.
2. Run the installer. On the Welcome screen, choose **Next**.
3. On the EULA screen, carefully read the license agreement, and if you agree, select **I accept the terms** and choose **Next**.
4. To begin the installation, choose **Install**.

5. Place a copy of the Broker's self-signed certificate (that you copied in the previous step) in the `C:\Program Files\NICE\DCVSessionManagerAgent\conf\` folder on the Agent.
6. Open `C:\Program Files\NICE\DCVSessionManagerAgent\conf\agent.conf` using your preferred text editor, and then do the following:

   - For `broker_host`, specify the DNS name of the host on which the Broker is installed.

     > **Important**
     >
     > If the Broker is running on an Amazon EC2 instance, for `broker_host` you must specify the instance's private IPv4 address.

   - (Optional) For `broker_port`, specify the port over which to communicate with the Broker. By default the Agent and the Broker communicate over port 8445. Only change this if you need to use a different port. If you do change it, ensure that the Broker is configured to use the same port.
   - For `ca_file`, specify the full path the certificate file that you copied in the previous step. For example:

     ```
     ca_file = 'C:\Program Files\NICE\DCVSessionManagerAgent\conf\broker_cert.pem'
     ```

     Alternatively, if you want to disable TLS verification, set `tls_strict` to `false`.
7. Save and close the file.
8. Stop and restart the Agent service for the changes to take effect. Run the following commands at the command prompt.

   ```
   C:\> sc stop DcvSessionManagerAgentService
   ```

   ```
   C:\> sc start DcvSessionManagerAgentService
   ```

Step 4: Configure the NICE DCV server to use the Broker as the authentication server

Configure the NICE DCV server to use the Broker as the external authentication server for validating client connection tokens. You must also configure the NICE DCV server to trust the Broker's self-signed CA.

Linux NICE DCV server

To add the local service user for Linux NICE DCV servers

1. Open `/etc/dcv/dcv.conf` using your preferred text editor.
2. Add the `ca-file` and `auth-token-verifier` parameters to the `[security]` section.

   For `ca-file`, specify the path to the Broker's self-signed CA that you copied to the host in the previous step.

   For `auth-token-verifier`, specify the URL for the token verifier on the Broker in the following format: `https://`*`broker_ip_or_dns`*`:`*`port`*`/agent/validate-authentication-token`. Specify the port used for Broker-Agent communication, which is 8445 by default. If you are running the Broker on an Amazon EC2 instance, you must use the private DNS or private IP address.

   For example

   ```
   [security]
   ca-file="/etc/dcv-session-manager-agent/broker_cert.pem"
   auth-token-verifier="https://my-sm-broker.com:8445/agent/validate-
   authentication-token"
   ```

3. Save and close the file.
4. Stop and restart the NICE DCV server. For more information, see Stopping the NICE DCV Server and Starting the NICE DCV Server in the *NICE DCV Administrator Guide*.

Windows NICE DCV server

On Windows NICE DCV servers

1. Open the Windows Registry Editor and navigate to the **HKEY_USERS/S-1-5-18/Software/GSettings/com/nicesoftware/dcv/security/** key.
2. Open the **ca-file** parameter. For **Value data**, specify the path to the Broker's self-signed CA that you copied to the host in the previous step.

   > **Note**
   >
   > If the parameter does not exist, create a new string parameter and name it `ca-file`.

3. Open the **auth-token-verifier** parameter. For **Value data**, specify the URL for the token verifier on the Broker in the following format: `https://`*`broker_ip_or_dns`*`:`*`port`*`/agent/validate-authentication-token`. Specify the port used for Broker-Agent communication, which is 8445 by default. If you are running the Broker on an Amazon EC2 instance, you must use the private DNS or private IP address.

   > **Note**
   >
   > If the parameter does not exist, create a new string parameter and name it `auth-token-verifier`.

4. Choose **OK** and close the Windows Registry Editor.
5. Stop and restart the NICE DCV server. For more information, see Stopping the NICE DCV Server and Starting the NICE DCV Server in the *NICE DCV Administrator Guide*.

Step 5: Verify the installations

Topics

Verify the Agent

After you have installed the Broker and the Agent, make sure that the Agent is running and that it's able to connect to the Broker.

Linux Agent host

The command to run depends on the version.

- Since version 2022.0

  From the Agent host, run the following command:

  ```
  $ grep 'sessionsUpdateResponse' /var/log/dcv-session-manager-agent/agent.log | tail
   -1 | grep -o success
  ```

- Versions prior to 2022.0

  From the Agent host, run the following command, and specify the current year, month, and day.

  ```
  $ grep 'sessionsUpdateResponse' /var/log/dcv-session-manager-agent/
  agent.log.yyyy-mm-dd | tail -1 | grep -o success
  ```

  For example

  ```
  $ grep 'sessionsUpdateResponse' /var/log/dcv-session-manager-agent/
  agent.log.2020-11-19 | tail -1 | grep -o success
  ```

If the Agent is running and it's able to connect to the broker, the command should return `success`.

If the command returns different output, inspect the Agent log file for more information. The log files are located here: `/var/log/dcv-session-manager-agent/`.

Windows Agent host

Open the Agent log file, which is located in `C:\ProgramData\NICE\DCVSessionManagerAgent\log`.

If the log file includes a line similar to the one below, the Agent is running and it's able to connect to the Broker.

```
2020-11-02 12:38:03,996919  INFO ThreadId(05) dcvsessionmanageragent::agent:Processing
 broker message "{\n  \"sessionsUpdateResponse\" : {\n    \"requestId\" :
 \"69c24a3f5f6d4f6f83ffbb9f7dc6a3f4\",\n    \"result\" : {\n      \"success\" : true\n
   }\n  }\n}"
```

If your log file doesn't have a similar line, inspect the log file for errors.

Verify the Broker

After you have installed the Broker and Agent, make sure that your Broker is running and that it's reachable from your users and front-end applications.

From a computer that should be able to reach the Broker, run the following command:

```
$ curl -X GET https://broker_host_ip:port/sessionConnectionData/aSession/aOwner --
insecure
```

If the verification is successful, the Broker returns the following:

```
{
    "error": "No authorization header"
}
```

Configuring NICE DCV Session Manager

This section explains how to perform advanced configuration for Session Manager.

Topics

Scaling Session Manager

To enable high availability and improve performance, you can configure Session Manager to use multiple Agents and Brokers. If you do intend to use multiple Agents and Brokers, we recommend that you install and configure only one Agent and Broker host, create Amazon Machines Images (AMI) from those hosts, and then launch the remaining hosts from the AMIs.

By default, Session Manager supports the use of multiple Agents without any additional configuration. However, if you intend to use multiple Brokers, you must use a load balancer to balance the traffic between the frontend client and the Brokers, and between the Brokers and the Agents. Load balancer setup and configuration is entirely owned and managed by you.

The following section explains how to configure Session Manager to use multiple hosts with an Application Load Balancer.

Steps

Step 1: Create an instance profile

You must attach an instance profile to the Broker and Agent hosts that give them permission to use the Elastic Load Balancing APIs. For more information, see IAM roles for Amazon EC2 in the *Amazon EC2 User Guide for Linux Instances*.

To create an instance profile

1. Create an Amazon Identity and Access Management (IAM) role that defines the permissions to use in the instance profile. Use the following trust policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

   Then attach the following policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ec2:DescribeInstances"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Action": [
        "elasticloadbalancing:DescribeTargetHealth"
      ],
      "Effect": "Allow",
      "Resource": "*"
```

```
    }
  ]
}
```

For more information, see Creating an IAM role in the *IAM User Guide*.

2. Create a new instance profile. For more information, see create-instance-profile in the *Amazon CLI Command Reference*.
3. Add the IAM role to the instance profile. For more information, see add-role-to-instance-profile in the *Amazon CLI Command Reference*.
4. Attach the instance profile to the Broker hosts. For more information, see Attaching an IAM role to an instance in the *Amazon EC2 User Guide for Linux Instances*.

Step 2: Prepare the SSL certificate for the load balancer

When you use HTTPS for your load balancer listener, you must deploy an SSL certificate on your load balancer. The load balancer uses this certificate to terminate the connection and decrypt requests from clients before sending them to the targets.

To prepare the SSL certificate

1. Create a private certificate authority (CA) Amazon Certificate Manager Private Certificate Authority (ACM PCA). For more information, see Procedures for Creating a CA in the Amazon *Certificate Manager Private Certificate Authority User Guide*.
2. Install the CA. For more information, see Installing a Root CA Certificate in the Amazon *Certificate Manager Private Certificate Authority User Guide*.
3. Request a new private certificate signed by the CA. For the domain name, use `*.region.elb.amazonaws.com` and specify the Region in which you intend to create the load balancer. For more information, see Requesting a Private Certificate in the Amazon *Certificate Manager Private Certificate Authority User Guide*.

Step 3: Create the Broker application load balancer

Create an application load balancer to balance the traffic between your front-end clients and the Brokers.

To create the load balancer

1. Open the Amazon EC2 console at https://console.amazonaws.cn/ec2/.

   In the navigation pane, choose **Load Balancers** and then choose **Create Load Balancer**. For load balancer type, choose **Application Load Balancer**.

2. For **Step 1: Configure Load Balancer**, do the following:

   a. For **Name**, enter a descriptive name for the load balancer.

   b. For **Scheme**, select **internet-facing**.

   c. For **Load Balancer Protocol**, select **HTTPS**, and for **Load Balancer Port**, enter 8443.

   d. For **VPC**, select the VPC to use and then select all of the subnets in that VPC.

   e. Choose **Next**.

3. For **Step 2: Configure Security Settings**, do the following:

   a. For **Certificate type**, choose **Choose a certificate from ACM**.

   b. For **Certificate name**, select the private certificate that you requested earlier.

   c. Choose **Next**.

4. For **Step 3: Configure Security Groups**, create a new security group, or select an existing security group that allows inbound and outbound traffic between your frontend client and the Brokers over HTTPS and port 8443.

   Choose **Next**.

5. For **Step 4: Configure Routing**, do the following:

   a. For **Target group**, select **New target group**.

   b. For **Name**, enter a name for the target group.

   c. For **Target type**, choose **Instance**.

   d. For **Protocol**, select **HTTPS**. For **Port**, enter 8443. For **Protocol version**, choose **HTTP1**.

   e. For the health check **Protocol**, choose HTTPS, and for **Path**, enter `/health`.

   f. Choose **Next**.

6. For **Step 5: Register Targets**, choose **Next**.

7. Choose **Create**.

Step 4: Launch the Brokers

Create an initial Broker and configure it to use the load balancer, create an AMI from the Broker, and then use the AMI to launch the remaining Brokers. This ensures that all of the Brokers are configure to use the same CA and the same load balancer configuration.

To launch the Brokers

1. Launch and configure the initial Broker host. For more information about installing and configuring the Broker, see Step 2: Set up the NICE DCV Session Manager Broker.

   > **Note**
   >
   > Broker's self signed certificate is not needed since we are using an application load balancer.

2. Connect to the Broker, open `/etc/dcv-session-manager-broker/session-manager-broker.properties` using your preferred text editor, and do the following:

   a. Comment out the `broker-to-broker-discovery-addresses` parameter by placing a hash (#) at the start of the line.

   b. For `broker-to-broker-discovery-aws-region`, enter the Region in which you created the application load balancer.

   c. For `broker-to-broker-discovery-aws-alb-target-group-arn`, enter the ARN of the target group associated with the Broker load balancer.

   d. Save and close the file.

3. Stop the Broker instance.

4. Create an AMI from the stopped Broker instance. For more information, see Creating a Linux AMI from an instance in the *Amazon EC2 User Guide for Linux Instances*.

5. Use the AMI to launch the remaining Brokers.

6. Assign the instance profile that you created to all of the Broker instances.

7. Assign a security group which allows Broker to Broker and Broker to load balancer network traffic to all of the Broker instances. For more information about network ports, see Broker Configuration File.

8. Register all of the Broker instances as targets for the Broker load balancer. For more information, see Register targets with your target group in the *User Guide for Application Load Balancers*.

Step 5: Create the Agent application load balancer

Create an application load balancer to balance the Agents and the Brokers.

To create the load balancer

1. Open the Amazon EC2 console at https://console.amazonaws.cn/ec2/.

   In the navigation pane, choose **Load Balancers** and then choose **Create Load Balancer**. For load balancer type, choose **Application Load Balancer**.
2. For **Step 1: Configure Load Balancer**, do the following:

   a. For **Name**, enter a descriptive name for the load balancer.

   b. For **Scheme**, select **internet-facing**.

   c. For **Load Balancer Protocol**, select **HTTPS**, and for **Load Balancer Port**, enter 8445.

   d. For **VPC**, select the VPC to use and then select all of the subnets in that VPC.

   e. Choose **Next**.
3. For **Step 2: Configure Security Settings**, do the following:

   a. For **Certificate type**, choose **Choose a certificate from ACM**.

   b. For **Certificate name**, select the private certificate that you requested earlier.

   c. Choose **Next**.
4. For **Step 3: Configure Security Groups**, create a new security group, or select an existing security group that allows inbound and outbound traffic the Agents and the Brokers over HTTPS and port 8445.

   Choose **Next**.
5. For **Step 4: Configure Routing**, do the following:

   a. For **Target group**, select **New target group**.

   b. For **Name**, enter a name for the target group.

   c. For **Target type**, choose **Instance**.

d. For **Protocol**, select **HTTPS**. For **Port**, enter 8445. For **Protocol version**, choose **HTTP1**.

e. For the health check **Protocol**, choose **HTTPS**, and for **Path**, enter `/health`.

f. Choose **Next**.

6. For **Step 5: Register Targets**, select all of the Broker instances and choose **Add to registered**. Choose **Next: Review**.

7. Choose **Create**.

Step 6: Launch the Agents

Create an initial Agent and configure it to use the load balancer, create an AMI from the Agent, and then use the AMI to launch the remaining Agents. This ensures that all of the Agents are configured to use the same load balancer configuration.

To launch the Agents

1. Prepare the NICE DCV server. For more information, see Step 1: Prepare the NICE DCV servers.

2. Place a copy of the CA public key created in Step 2: Prepare the SSL certificate for the load balancer. Choose or create a directory readable by any user. The CA public key file must be readable by any user as well.

3. Install and configure the Agent. For more information about installing and configuring the Agent, see Step 3: Set up the NICE DCV Session Manager Agent.

   > **Important**
   >
   > When modifying the Agent configuration file:
   >
   > - for the `broker_host` parameter, enter the Agent load balancer's DNS
   > - for the `ca_file` parameter, enter the path to the CA public key file created in the previous step

4. Configure the NICE DCV server to use the Broker as the authentication server. For more information, see Step 4: Configure the NICE DCV server to use the Broker as the authentication server.

> **Important**
>
> When modifying the NICE DCV server configuration file:
>
> - for the `ca-file` parameter, enter the same path to the CA public key file used in the previous step
> - for the `auth-token-verifier` parameter, use the Agent load balancer's DNS for *`broker_ip_or_dns`*

5. Stop the Agent instance.
6. Create an AMI from the stopped Agent instance. For more information, see Creating a Linux AMI from an instance in the *Amazon EC2 User Guide for Linux Instances*.
7. Use the AMI to launch the remaining Agents and assign the instance profile that you created to all of them.
8. Assign a security group which allows Agent to load balancer network traffic to all of the Agent instances. For more information about network ports, see Agent Configuration File.

Using tags to target NICE DCV servers

You can assign custom tags to Session Manager Agents to help identify and categorize them and the NICE DCV servers with which they are associated. When creating a new NICE DCV session, you can target a group of NICE DCV servers based on the tags that are assigned to their respective Agents. For more information about how to target NICE DCV servers based on Agent tags, see CreateSessionRequests in the *Session Manager Developer Guide*.

A tag consists of a tag key and value pair, and you can use any information pair that makes sense for your use case or environment. You can choose to tag Agents based on their host's hardware configuration. For example, you can tag all Agents with hosts that have 4 GB of memory with `ram=4GB`. Or you can tag Agents based on purpose. For example, you can tag all Agents running on production hosts with `purpose=production`.

To assign tags to an Agent

1. Using your preferred text editor, create a new file and give it a descriptive name, for example `agent_tags.toml`. The file type must be `.toml`, and the file contents must be specified in the TOML file format.

2. In the file, add each new tag key and value pair on a new line using the `key=value` format. For example:

```
tag1="abc"
tag2="xyz"
```

3. Open the Agent configuration file (`/etc/dcv-session-manager-agent/agent.conf` for Linux or `C:\Program Files\NICE\DCVSessionManagerAgent\conf\agent.conf` for Windows). For `tags_folder`, and specify the path to the directory in which the tag file is located.

   If the directory contains multiple tag files, all of the tags defined across the files apply the Agent. The files are read in alphabetical order. If multiple files contain a tag with the same key, the value is overwritten with the value from the last read file.

4. Save and close the file.
5. Stop and restart the Agent.

   - Windows

   ```
   C:\> sc stop DcvSessionManagerAgentService
   ```

   ```
   C:\> sc start DcvSessionManagerAgentService
   ```

   - Linux

   ```
   $ sudo systemctl stop dcv-session-manager-agent
   ```

   ```
   $ sudo systemctl start dcv-session-manager-agent
   ```

Configuring an external authorization server

The authorization server is the server that is responsible for authenticating and authorizing the client SDKs and Agents.

By default, Session Manager uses the Broker as the authorization server to generate OAuth 2.0 access tokens for client SDKs and software statements for Agents. If you use the Broker as the authorization server, no additional configuration is required.

You can configure Session Manager to use Amazon Cognito as an external authorization server instead of the Broker. For more information, about Amazon Cognito, see the Amazon Cognito Developer Guide.

To use Amazon Cognito as the authorization server

1. Create a new Amazon Cognito user pool. For more information about user pools, see Features of Amazon Cognito in the *Amazon Cognito Developer Guide*.

   Use the create-user-pool command, and specify a pool name and the Region in which to create it.

   In this example, we name the pool `dcv-session-manager-client-app` and we create it in `us-east-1`.

   ```
   $ aws cognito-idp create-user-pool --pool-name dcv-session-manager-client-app --
   region us-east-1
   ```

   Example output

   ```
   {
       "UserPoolClient": {
           "UserPoolId": "us-east-1_QLEXAMPLE",
           "ClientName": "dcv-session-manager-client-app",
           "ClientId": "15hhd8jij74hf32f24uEXAMPLE",
           "LastModifiedDate": 1602510048.054,
           "CreationDate": 1602510048.054,
           "RefreshTokenValidity": 30,
           "AllowedOAuthFlowsUserPoolClient": false
       }
   }
   ```

   Make a note of the `userPoolId`, you'll need it in the next step.

2. Create a new domain for your user pool. Use the create-user-pool-domain command, and specify a domain name and the `userPoolId` of the user pool that you created in the previous step.

   In this example, the domain name is `mydomain-544fa30f-c0e5-4a02-8d2a-a3761EXAMPLE` and we create it in `us-east-1`.

```
$ aws cognito-idp create-user-pool-domain --domain mydomain-544fa30f-
c0e5-4a02-8d2a-a3761EXAMPLE --user-pool-id us-east-1_QLEXAMPLE --region us-east-1
```

Example output

```
{
    "DomainDescription": {
        "UserPoolId": "us-east-1_QLEXAMPLE",
        "AWSAccountId": "123456789012",
        "Domain": "mydomain-544fa30f-c0e5-4a02-8d2a-a3761EXAMPLE",
        "S3Bucket": "aws-cognito-prod-pdx-assets",
        "CloudFrontDistribution": "dpp0gtexample.cloudfront.net",
        "Version": "20201012133715",
        "Status": "ACTIVE",
        "CustomDomainConfig": {}
    }
}
```

The format of the user pool domain is as follows: `https://domain_name.auth.region.amazoncognito.com`. In this example, the user pool domain is `https://mydomain-544fa30f-c0e5-4a02-8d2a-a3761EXAMPLE.auth.us-east-1.amazoncognito.com`.

3. Create a user pool client. Use the create-user-pool-client command and specify the `userPoolId` of the user pool that you created, a name for the client, and the Region in which to create it. Also, include the `--generate-secret` option to specify that you want to generate a secret for the user pool client being created.

   In this case, the client name is `dcv-session-manager-client-app` and we create it in the `us-east-1` Region.

```
$ aws cognito-idp create-user-pool-client --user-pool-id us-east-1_QLEXAMPLE --
client-name dcv-session-manager-client-app --generate-secret --region us-east-1
```

Example output

```
{
    "UserPoolClient": {
        "UserPoolId": "us-east-1_QLEXAMPLE",
```

```
        "ClientName": "dcv-session-manager-client-app",
        "ClientId": "2l9273hp6k2ut5cugg9EXAMPLE",
        "ClientSecret": "1vp5e8nec7cbf4m9me55mbmht91u61hlh0a78rq1qki1lEXAMPLE",
        "LastModifiedDate": 1602510291.498,
        "CreationDate": 1602510291.498,
        "RefreshTokenValidity": 30,
        "AllowedOAuthFlowsUserPoolClient": false
    }
}
```

> **Note**
>
> Make a note of the `ClientId` and `ClientSecret`. You'll need to provide this information to the developers for when they request access tokens for the API requests.

4. Create a new OAuth2.0 resource server for the user pool. A resource server is a server for access-protected resources. It handles authenticated requests for access tokens.

   Use the create-resource-server command and specify the `userPoolId` of the user pool, a unique identifier and name for the resource server, the scope, and the Region in which to create it.

   In this example, we use `dcv-session-manager` as the identifier and the name, and we use `sm_scope` as the scope name and description.

```
$ aws cognito-idp create-resource-server --user-pool-id us-east-1_QLEXAMPLE
 --identifier dcv-session-manager --name dcv-session-manager --scopes
 ScopeName=sm_scope,ScopeDescription=sm_scope --region us-east-1
```

   Example output

```
{
    "ResourceServer": {
        "UserPoolId": "us-east-1_QLEXAMPLE",
        "Identifier": "dcv-session-manager",
        "Name": "dcv-session-manager",
        "Scopes": [
        {
            "ScopeName": "sm_scope",
```

```
            "ScopeDescription": "sm_scope"
        }]
    }
}
```

5. Update the user pool client.

   Use the update-user-pool-client command. Specify the `userPoolId` of the user pool, the `ClientId` of the user pool client, and the Region. For `--allowed-o-auth-flows`, specify `client_credentials` to indicate that the client should get access tokens from the token endpoint by using a combination of a client ID and a client secret. For `--allowed-o-auth-scopes`, specify the resource server identifier and the scope name as follows: *`resource_server_identifier`*/*`scope_name`*. Include the `--allowed-o-auth-flows-user-pool-client` to indicate that the client is allowed to follow the OAuth protocol when interacting with Cognito user pools.

```
$ aws cognito-idp update-user-pool-client --user-pool-id us-east-1_QLEXAMPLE --
client-id 2l9273hp6k2ut5cugg9EXAMPLE --allowed-o-auth-flows client_credentials --
allowed-o-auth-scopes dcv-session-manager/sm_scope --allowed-o-auth-flows-user-
pool-client --region us-east-1
```

   Example output

```
{
    "UserPoolClient": {
        "UserPoolId": "us-east-1_QLEXAMPLE",
        "ClientName": "dcv-session-manager-client-app",
        "ClientId": "2l9273hp6k2ut5cugg9EXAMPLE",
        "ClientSecret": "1vp5e8nec7cbf4m9me55mbmht91u61hlh0a78rq1qki1lEXAMPLE",
        "LastModifiedDate": 1602512103.099,
        "CreationDate": 1602510291.498,
        "RefreshTokenValidity": 30,
        "AllowedOAuthFlows": [
            "client_credentials"
        ],
        "AllowedOAuthScopes": [
            "dcv-session-manager/sm_scope"
        ],
        "AllowedOAuthFlowsUserPoolClient": true
    }
}
```

> **Note**
>
> The user pool is now ready to provide and authenticate access tokens. In this example, the URL for the authorization server is `https://cognito-idp.us-east-1.amazonaws.com/us-east-1_QLEXAMPLE/.well-known/jwks.json`.

6. Test the configuration.

```
$ curl -H "Authorization: Basic `echo -
n 2l9273hp6k2ut5cugg9EXAMPLE:1vp5e8nec7cbf4m9me55mbmht91u61hlh0a78rq1qki1lEXAMPLE
 | base64`" -H "Content-Type: application/x-www-form-urlencoded" -X
 POST "https://mydomain-544fa30f-c0e5-4a02-8d2a-a3761EXAMPLE.auth.us-
east-1.amazoncognito.com/oauth2/token?grant_type=client_credentials&scope=dcv-
session-manager/sm_scope"
```

Example output

```
{
"access_token":"eyJraWQiOiJGQ0VaRFpJUUptT3NSaW41MmtqaDdEbTZYb0RnSTQ5b2VUT0cxUUI1Q2VJPSIsImF
Zkfi0HIDsd6audjTXKzHlZGScr6ROdZtId5dThkpEZiSx0YwiiWe9crAlqoazlDcCsUJHIXDtgKW64pSj3-
uQQGg1Jv_tyVjhrA4JbD0k67WS2V9NW-
uZ7t4zwwaUmOi3KzpBMi54fpVgPaewiVlUm_aS4LUFcWT6hVJjiZF7om7984qb2gOa14iZxpXPBJTZX_gtG9EtvnS9u
"expires_in":3600,
"token_type":"Bearer"
}
```

7. Register the external authorization server for use with the broker by using the register-auth-server command.

```
$ sudo -u root dcv-session-manager-broker register-auth-server --url  https://
cognito-idp.us-east-1.amazonaws.com/us-east-1_QLEXAMPLE/.well-known/jwks.json
```

Developers can now use the server to request access tokens. When requesting access tokens, provide the client ID, client secret, and server URL generated here. For more information about requesting access tokens, see Create get an access token and make an API request in the*NICE DCV Session Manager Developer Guide*.

Configuring broker persistence

Session Manager brokers support integration with external databases. The external database allows Session Manager to persist status data and keys so they are available afterwards. In fact the broker data is distributed over the cluster, making it susceptible to data loss if a host needs to reboot or a cluster is terminated. With this feature enabled, you can add and remove broker nodes. Also, you can stop a cluster and restart it, without the need to regenerate keys or lose information about which NICE DCV Server is open or closed.

The following types of information can be set to persist:

- Keys for setting up sessions to establish connection with clients
- In-flight sessions data
- NICE DCV server status

NICE DCV Session Manager supports DynamoDB, MariaDB, and MySQL databases. You must set up and manage one of these databases to use this feature. If your broker machines are hosted on Amazon EC2, we recommend using DynamoDB as the external database, since it does not require any additional setup.

> **Note**
>
> You may incur additional costs when running an external database. To see information on DynamoDB pricing, see Pricing for Provisioned Capacity.

Configure the broker to persist on DynamoDB

Configure the brokers to start storing their data on DynamoDB:

1. Open `/etc/dcv-session-manager-broker/session-manager-broker.properties` using your preferred text editor and make the following edits:
   - Set `enable-persistence = true`
   - Set `persistence-db = dynamodb`

- For `dynamodb-region` specify the &aws; Region where you want to store the tables containing the broker data. For the list of supported Regions, see DynamoDB service endpoints.
- For `dynamodb-table-rcu` specify the amount of Read Capacity Units (RCU) that each table supports. For more information about RCU, see DynamoDB provisioned capacity.
- For `dynamodb-table-wcu` specify the amount of Write Capacity Units (WCU) that each table supports. For more info on WCU, see DynamoDB provisioned capacity.
- For dynamodb-table-name-prefix specify the prefix that is added to each DynamoDB table (useful to distinguish multiple broker clusters using the same account). Only alphanumeric characters, dot, dash, and underscore are allowed.

2. Stop all the brokers in the cluster. For each broker, run the following command:

```
sudo systemctl stop dcv-session-manager-broker
```

3. Ensure all brokers in the cluster are stopped and then restart all of them. Start each broker by running the following command:

```
sudo systemctl start dcv-session-manager-broker
```

The broker host must have permission to call the DynamoDB APIs. On Amazon EC2 instances, the credentials are automatically retrieved using the Amazon EC2 metadata service. If you need to specify different credentials, you can set them using one of the supported credential retrieval techniques (such as Java system properties or environment variables). For more information, see Supplying and Retrieving &aws; Credentials.

Configure the broker to persist on MariaDB/MySQL

> **Note**
>
> The `/etc/dcv-session-manager-broker/session-manager-broker.properties` file contains sensitive data. By default, its write access is restricted to root and its read access is restricted to root and to the user running the Broker. By default, this is the `dcvsmbroker` user. The Broker checks at startup that the file has the expected permissions.

Configure the brokers to start persisting their data on MariaDB/MySQL:

1. Open `/etc/dcv-session-manager-broker/session-manager-broker.properties` with your preferred text editor and make the following edits:

   - Set `enable-persistence = true`
   - Set `persistence-db = mysql`
   - Set `jdbc-connection-url = jdbc:mysql://<db_endpoint>:<db_port>/<db_name>?createDatabaseIfNotExist=true`

     In this configuration, <db_endpoint> is the database endpoint, <db_port> is the database port, and <db_name> is the database name.
   - For `jdbc-user` specify the name of the user that has access to the database.
   - For `jdbc-password` specify the password of the user that has access to the database.
2. Stop all the brokers in the cluster. For each broker, run the following command:

   ```
   sudo systemctl stop dcv-session-manager-broker
   ```
3. Ensure all brokers in the cluster are stopped, and then restart all of them. For each broker, run the following command:

   ```
   sudo systemctl start dcv-session-manager-broker
   ```

Integrating with the NICE DCV Connection Gateway

NICE DCV Connection Gateway is an installable software package that enables users to access a fleet of NICE DCV servers through a single access point to a LAN or VPC.

If your infrastructure includes NICE DCV servers that are accessible through the NICE DCV Connection Gateway, you can configure the Session Manager to integrate the NICE DCV Connection Gateway. By following the steps outlined in the following section, the broker will act as a Session Resolver for the Connection Gateway. In other words, the broker will expose an additional HTTP endpoint. The Connection Gateway will make API calls to the endpoint to retrieve the information needed to route NICE DCV connections to the host selected by the broker.

Set up the Session Manager Broker as a Session Resolver for the NICE DCV Connection Gateway

Session Manager Broker side

1. Open `/etc/dcv-session-manager-broker/session-manager-broker.properties` using your preferred text editor and apply the following changes:
   - Set `enable-gateway = true`
   - Set `gateway-to-broker-connector-https-port` to a free TCP port (default is 8447)
   - Set `gateway-to-broker-connector-bind-host` to the IP address of the host where the Broker binds for NICE DCV Connection Gateway connections (default is 0.0.0.0)
2. Then run the following commands to stop and restart the Broker:

   ```
   sudo systemctl stop dcv-session-manager-broker
   ```

   ```
   sudo systemctl start dcv-session-manager-broker
   ```
3. Retrieve a copy of the Broker's self-signed certificate and place it in your user directory.

   ```
   sudo cp /var/lib/dcvsmbroker/security/dcvsmbroker_ca.pem $HOME
   ```

   You'll need it when you install the NICE DCV Connection Gateway in the next step.

NICE DCV Connection Gateway side

- Please follow the section in the NICE DCV Connection Gateway documentation.

  Since the NICE DCV Connection Gateway makes HTTP API calls to the broker, if the broker is using a self-signed certificate, you will need to copy the broker certificate to the NICE DCV Connection Gateway host (retrieved in the previous step) and set the `ca-file` parameter in the `[resolver]` section of the NICE DCV Connection Gateway configuration.

Optional - Enable TLS client authentication

Once you have completed the previous step, the Session Manager and the Connection Gateway can communicate over a secure channel, where the Connection Gateway can verify the identity of the Session Manager Brokers. If you require that also the Session Manager Brokers validate the identity of the Connection Gateway before establishing the secure channel, you need to enable the TLS client authentication feature, following the steps in the next section.

> **Note**
>
> If the Session Manager is behind a load balancer, TLS client authentication cannot be enabled with load balancers that have TLS connection termination, such as Application Load Balancers (ALBs) or Gateway Load Balancers (GLBs). Only load balancers without TLS termination can be supported, such as Network Load Balancers (NLBs). If you use ALBs or GLBs, you can enforce that only specific security groups can contact the load balancers, ensuring an additional level of security; more info about security groups here: Security groups for your VPC

Session Manager Broker side

1. To enable the TLS client authentication for the communication between the Session Manager Brokers and the NICE DCV Connection Gateway, please follow the next steps:
2. Generate the required keys and certificates by running: The output of the command will tell you the folder where the credentials have been generated and the password used for creating the TrustStore file.

```
sudo /usr/share/dcv-session-manager-broker/bin/gen-gateway-certificates.sh
```

3. Place a copy of the NICE DCV Connection Gateway's private key and self-signed certificate in your user directory. You'll need it when you enable the TLS client authentication in the NICE DCV Connection Gateway in the next step.

```
sudo cp /etc/dcv-session-manager-broker/resolver-creds/dcv_gateway_key.pem $HOME
```

```
sudo cp /etc/dcv-session-manager-broker/resolver-creds/dcv_gateway_cert.pem $HOME
```

4. Then open /etc/dcv-session-manager-broker/session-manager-broker.properties using your preferred text editor and do the following:

   - Set `enable-tls-client-auth-gateway` to `true`
   - Set `gateway-to-broker-connector-trust-store-file` to the path of the TrustStore file created in the previous step
   - Set `gateway-to-broker-connector-trust-store-pass` to the password used for creating the TrustStore file in the previous step
5. Then run the following command to stop and restart the Broker:

```
sudo systemctl stop dcv-session-manager-broker
```

```
sudo systemctl start dcv-session-manager-broker
```

NICE DCV Connection Gateway side

- Please follow the section in the NICE DCV Connection Gateway documentation.

  - use the full path of the certificate file that you copied in the previous step when setting the `cert-file` parameter in the `[resolver]` section
  - use the full path of the key file that you copied in the previous step when setting the `cert-key-file` parameter in the `[resolver]` section

NICE DCV Session Manager NICE DCV server - DNS mapping

The NICE DCV Connection Gateway requires the NICE DCV servers' DNS names in order to connect to the DCV server instances. This section illustrates how you can define a JSON file containing the mapping between each DCV Server and its associated DNS name.

File structure

The mapping consists of a list of JSON objects with the following fields:

```
[
 {
  “ServerIdType”: "Ip",
  “ServerId”: "192.168.0.1",
```

```
  “DnsNames”:
  {
   “InternalDnsName”: "internal"
  }
 },
 ...
]
```

Where:

`ServerIdType:`

Identifies which type of id the value refers to; currently the available values are ipAddress, agentServerId, and instanceId:

`Ip:`

Available for both Amazon EC2 and on premise infrastructures; can be quickly retrieved by system administrators with an ifconfig (Linux) or ipconfig (Windows) command. This info is also available in the DescribeServers API response.

`Id:`

Available for both Amazon EC2 and on premise infrastructures; the Session Manager Agent creates a new UUID every time the hostname or the ip address changes. This info is available in the DescribeServers API response.

`Host.Aws.Ec2InstanceId:`

Available only for Amazon EC2 instances, it uniquely identifies a machine; it does not change after an instance reboot. Can be retrieved on the host by contacting http://169.254.169.254/latest/meta-data/instance-id. This info is also available in the DescribeServers API response.

`ServerId:`

An Id of the specified type that uniquely identifies each NICE DCV server in the network.

`DnsNames:`

The object containing the DNS names associated with the NICE DCV server this object will contain:

`InternalDnsNames:`

The DNS name used by the NICE DCV Connection Gateway to connect to the instance.

Please use the Session Manager Broker CLI commands `register-server-dns-mapping` to load the mapping from a file (command page reference: register-server-dns-mapping) and `describe-server-dns-mappings` to list the mappings currently loaded in the Session Manager Broker (command page reference: describe-server-dns-mappings).

Persistence

We strongly recommend that you enable the persistence feature of the Session Manager Broker, to protect against the mapping loss when multiple brokers or the entire cluster go down. For more information about enabling data persistence, see Configure Broker Persistence

Integrating with Amazon CloudWatch

Session Manager supports integration with Amazon CloudWatch for Brokers running on Amazon EC2 instances, and also Brokers running on on-premises hosts.

Amazon CloudWatch monitors your Amazon Web Services (Amazon) resources and the applications you run on Amazon in real time. You can use CloudWatch to collect and track metrics, which are variables you can measure for your resources and applications. For more information, see the Amazon CloudWatch User Guide.

You can configure the Session Manager Broker to send the following metric data to Amazon CloudWatch:

- `Number of DCV servers`—The number of DCV servers managed by the Broker.
- `Number of ready DCV servers`—The number of DCV servers that are in the `READY` state managed by the Broker.
- `Number of DCV sessions`—The number of DCV sessions managed by the Broker.
- `Number of DCV console sessions`—The number of DCV console sessions managed by the Broker.
- `Number of DCV virtual sessions`—The number of DCV virtual sessions managed by the Broker.
- `Heap memory used`—The amount of heap memory used by the Broker.
- `Off-heap memory used`—The amount of off-heap memory used by the Broker.
- `Describe sessions request time`—The amount of time taken to complete DescribeSessions API requests.

- `Delete sessions request time`—The amount of time taken to complete DeleteSessions API requests.
- `Create sessions request time`—The amount of time taken to complete CreateSessions API requests.
- `Get session connection data request time`—The amount of time taken to complete GetSessionConnectionData API requests.
- `Update session permissions sequest time`—The amount of time taken to complete UpdateSessionPermissions API requests.

To configure the Broker to send metric data to the Amazon CloudWatch

1. Open `/etc/dcv-session-manager-broker/session-manager-broker.properties` using your preferred text editor and do the following:

   - Set `enable-cloud-watch-metrics` to `true`
   - For `cloud-watch-region`, specify the Region in which to collect the metric data.

   > **Note**
   >
   > If your Broker is running on an Amazon EC2 instance, this parameter is optional. The Region is automatically retrieved from the Instance Metadata Service (IMDS). If you are running the Broker on an on-premises host, this parameter is mandatory.

2. Stop and restart the Broker.

```
$  sudo systemctl stop dcv-session-manager-broker
```

```
$  sudo systemctl start dcv-session-manager-broker
```

The Broker host must also have permission to call the `cloudwatch:PutMetricData` API. Amazon credentials can be retrieved using one of the supported credential retrieval techniques. For more information, see Supplying and Retrieving Amazon Credentials.

Upgrading the NICE DCV Session Manager

The following topic describes how to upgrade the Session Manager.

> **Note**
>
> We strongly recommend that you upgrade all the Session Manager Agents before upgrading the Session Manager Brokers, to avoid incompatibility issues in case new features are introduced.

Topics

Upgrading the NICE DCV Session Manager Agent

Linux host

> **Note**
>
> The following instructions are for installing the Agent on 64-bit x86 hosts. To install the Agent on 64-bit ARM hosts, for Amazon Linux, RHEL, and Centos, replace *x86_64* with `aarch64`, and for Ubuntu, replace *amd64* with `arm64`.

To update the Agent on a Linux host

1. Run the following command to stop the Agent.

   ```
   $  sudo systemctl stop dcv-session-manager-agent
   ```

2. Download the installation package.

   - Amazon Linux 2, RHEL 7.x, and CentOS 7.x

```
$  wget https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerAgents/
nice-dcv-session-manager-agent-2023.1.732-1.el7.x86_64.rpm
```

- RHEL 8.x, CentOS Stream 8, and Rocky Linux 8.x

```
$  wget https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerAgents/
nice-dcv-session-manager-agent-2023.1.732-1.el8.x86_64.rpm
```

- Ubuntu 20.04

```
$  wget https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerAgents/
nice-dcv-session-manager-agent_2023.1.732-1_amd64.ubuntu2004.deb
```

- SUSE Linux Enterprise 12

```
$  curl -O https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerAgents/
nice-dcv-session-manager-agent-2023.1.732-1.sles12.x86_64.rpm
```

- SUSE Linux Enterprise 15

```
$  curl -O https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerAgents/
nice-dcv-session-manager-agent-2023.1.732-1.sles15.x86_64.rpm
```

3. Install the package.

- Amazon Linux 2, RHEL 7.x, and CentOS 7.x

```
$  sudo yum install -y nice-dcv-session-manager-
agent-2023.1.732-1.el7.x86_64.rpm
```

- RHEL 8.x, CentOS Stream 8, and Rocky Linux 8.x

```
$  sudo yum install -y nice-dcv-session-manager-
agent-2023.1.732-1.el8.x86_64.rpm
```

- Ubuntu 20.04

```
$  sudo apt install ./nice-dcv-session-manager-
agent_2023.1.732-1_amd64.ubuntu2004.deb
```

- SUSE Linux Enterprise 12

```
$  sudo zypper install nice-dcv-session-manager-
agent-2023.1.732-1.sles12.x86_64.rpm
```

- SUSE Linux Enterprise 15

```
$  sudo zypper install nice-dcv-session-manager-
agent-2023.1.732-1.sles15.x86_64.rpm
```

4. Run the following command to start the Agent.

```
$  sudo systemctl start dcv-session-manager-agent
```

Windows host

To update the Agent on a Windows host

1. Stop the Agent service. Run the following commands at the command prompt.

```
C:\>  sc start DcvSessionManagerAgentService
```

2. Download the Agent installer.
3. Run the installer. On the Welcome screen, choose **Next**.
4. On the EULA screen, carefully read the license agreement, and if you agree, select **I accept the terms** and choose **Next**.
5. To begin the installation, choose **Install**.
6. Restart the Agent service. Run the following commands at the command prompt.

```
C:\>  sc start DcvSessionManagerAgentService
```

Upgrading the NICE DCV Session Manager Broker

To upgrade the Broker

1. Connect to the host on which you intend to upgrade the Broker.
2. Stop the Broker service.

```
$  sudo systemctl stop dcv-session-manager-broker
```

3. Download the installation package.

   - Amazon Linux 2, RHEL 7.x, and CentOS 7.x

     ```
     $  wget https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerBrokers/nice-dcv-session-manager-broker-2023.1.410-1.el7.noarch.rpm
     ```

   - RHEL 8.x, CentOS Stream 8, and Rocky Linux 8.x

     ```
     $  wget https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerBrokers/nice-dcv-session-manager-broker-2023.1.410-1.el8.noarch.rpm
     ```

   - Ubuntu 20.04

     ```
     $  wget https://d1uj6qtbmh3dt5.cloudfront.net/2023.1/SessionManagerBrokers/nice-dcv-session-manager-broker-2023.1.410-1_all.ubuntu2004.deb
     ```

4. Install the package.

   - Amazon Linux 2, RHEL 7.x, and CentOS 7.x

     ```
     $  sudo yum install -y nice-dcv-session-manager-broker-2023.1.410-1.el7.noarch.rpm
     ```

   - RHEL 8.x, CentOS Stream 8, and Rocky Linux 8.x

     ```
     $  sudo yum install -y nice-dcv-session-manager-broker-2023.1.410-1.el8.noarch.rpm
     ```

   - Ubuntu 20.04

     ```
     $  sudo apt install -y nice-dcv-session-manager-broker-2023.1.410-1_all.ubuntu2004.deb
     ```

5. Start the Broker service and ensure that it starts automatically every time the instance starts.

   ```
   $  sudo systemctl start dcv-session-manager-broker && sudo systemctl enable dcv-session-manager-broker
   ```

Broker CLI reference

This section explains how to use the Broker command line interface (CLI) commands.

Use the following commands if you use an external authentication server to generate OAuth 2.0 access tokens:

- register-auth-server
- list-auth-servers
- unregister-auth-server

Use the following commands if you use the Session Manager Broker as the OAuth 2.0 authentication server.

- register-api-client
- describe-api-clients
- unregister-api-client
- renew-auth-server-api-key

Use the following commands to manage the Session Manager Agent.

- generate-software-statement
- describe-software-statements
- deactivate-software-statement
- describe-agent-clients
- unregister-agent-client

Use the following commands to manage the DCV server - DNS names mapping file.

- register-server-dns-mappings
- describe-server-dns-mappings

register-auth-server

Registers an external authentication server for use with the Broker.

By default, Session Manager uses the Broker as the authentication server to generate OAuth 2.0 access tokens. If you use the Broker as the authentication server, no additional configuration is required.

However, if you choose to use an external authentication server, such as Active Directory or Amazon Cognito, you must use this command to register the external authentication server.

Topics

Syntax

```
sudo -u root dcv-session-manager-broker register-auth-server --url  server_url.well-known/jwks.json
```

Options

`--url`

The URL of the external authentication server to be used. You must append `.well-known/jwks.json` to the authentication server URL.

Type: String

Required: Yes

Example

The following example registers an external authentication server with a URL of `https://my-auth-server.com/`.

Command

```
sudo -u root dcv-session-manager-broker register-auth-server --url https://my-auth-server.com/.well-known/jwks.json
```

Output

```
Jwk url registered.
```

list-auth-servers

Lists the external authentication servers that have been registered.

Topics

Syntax

```
sudo -u root dcv-session-manager-broker list-auth-servers
```

Output

`Urls`

The URLs of the registered external authentication servers.

Example

The following example lists all external authentication servers that have been registered.

Command

```
sudo -u root dcv-session-manager-broker list-auth-servers
```

Output

```
Urls: [ "https://my-auth-server.com/.well-known/jwks.json" ]
```

unregister-auth-server

Unregisters an external authentication server. After you unregister an external authentication server, it can no longer be used to generate OAuth 2.0 access tokens.

Topics

Syntax

```
sudo -u root dcv-session-manager-broker unregister-auth-server --url  server_url.well-known/jwks.json
```

Options

`--url`

The URL of the external authentication server to unregister. You must append `.well-known/jwks.json` to the authentication server URL.

Type: String

Required: Yes

Output

`Url`

The URL of the unregistered external authentication server.

Example

The following example registers an external authentication server with a URL of `https://my-auth-server.com/`.

Command

```
sudo -u root dcv-session-manager-broker unregister-auth-server --url https://my-auth-server.com/.well-known/jwks.json
```

Output

```
Jwk urlhttps://my-auth-server.com/.well-known/jwks.json unregistered
```

register-api-client

Registers a Session Manager client with the Broker and generates client credentials that can be used by the client to retrieve an OAuth 2.0 access token, which is needed to make API requests.

> **Important**
>
> Ensure that you store the credentials in a safe place. They can't be recovered later.

This command is used only if the Broker is used as the OAuth 2.0 authentication server.

Topics

Syntax

```
sudo -u root dcv-session-manager-broker register-api-client --client-name client_name
```

Options

`--name`

A unique name used to identify the Session Manager client.

Type: String

Required: Yes

Output

`client-id`

The unique client ID to be used by the Session Manager client to retrieve an OAuth 2.0 access token.

`client-password`

The password to be used by the Session Manager client to retrieve an OAuth 2.0 access token.

Example

The following example registers a client named `my-sm-client`.

Command

```
sudo -u root dcv-session-manager-broker register-api-client --client-name my-sm-client
```

Output

```
client-id: 21cfe9cf-61d7-4c53-b1b6-cf248EXAMPLE
client-password: NjVmZDRlN2ItNjNmYS00M2QxLWFlZmMtZmNmMDNkMEXAMPLE
```

describe-api-clients

Lists the Session Manager clients that have been registered with the Broker.

Topics

Syntax

```
sudo -u root dcv-session-manager-broker describe-api-clients
```

Output

`name`

The unique name of the Session Manager client.

`id`

The unique ID of the Session Manager client.

`active`

Indicates the status of the Session Manager client. If the client is active, the value is `true`; otherwise, it's `false`.

Example

The following example lists the registered Session Manager clients.

Command

```
sudo -u root dcv-session-manager-broker describe-api-clients
```

Output

```
Api clients
[ {
"name" : "client-abc",
"id" : "f855b54b-40d4-4769-b792-b727bEXAMPLE",
"active" : false
```

```
}, {
"name" : "client-xyz",
"id" : "21cfe9cf-61d7-4c53-b1b6-cf248EXAMPLE",
"active" : true
}]
```

unregister-api-client

Deactivates a registered Session Manager client. A deactivated Session Manager client can no longer use its credentials to retrieve OAuth 2.0 access tokens.

Topics

Syntax

```
sudo -u root dcv-session-manager-broker unregister-api-client --client-id client_id
```

Options

`--client -id`

The client ID of the Session Manager client to deactivate.

Type: String

Required: Yes

Example

The following example deactivates a Session Manager client with a client ID of `f855b54b-40d4-4769-b792-b727bEXAMPLE`.

Command

```
sudo -u root dcv-session-manager-broker unregister-api-client --client-id
 f855b54b-40d4-4769-b792-b727bEXAMPLE
```

Output

```
Client f855b54b-40d4-4769-b792-b727bEXAMPLE unregistered.
```

renew-auth-server-api-key

Renews the public and private keys used by the Broker to sign the OAuth 2.0 access tokens that are vended to the Session Manager client. If you renew the keys, then you must provide the new private key to the developer, as it is needed to make API requests.

Topics

Syntax

```
sudo -u root dcv-session-manager-broker renew-auth-server-api-key
```

Example

The following example renews the public and private keys.

Command

```
sudo -u root dcv-session-manager-broker renew-auth-server-api-key
```

Output

```
Keys renewed.
```

generate-software-statement

Generates a software statement.

Agents must be registered with the Broker to enable communication. Agents need a software statement in order to register with the Broker. After the Agent has a software statement, it can automatically register itself with the Broker by using the OAuth 2.0 Dynamic Client Registration Protocol. After the Agent has registered with the Broker, it receives a client ID and client secret that it uses to authenticate with the Broker.

The Broker and Agent receive and use a default software statement when they're first installed. You can continue to use the default software statement, or you can choose to generate a new one. If you generate a new software statement, you must place the software statement into a new file on the Agent, and then add the file path to the `agent.software_statement_path` parameter in the `agent.conf` file. After you have done this, stop and restart the Agent so that it can use the new software statement to register with the Broker.

Topics

Syntax

```
sudo -u root dcv-session-manager-broker generate-software-statement
```

Output

`software-statement`

The software statement.

Example

The following example generates a software statement.

Command

```
sudo -u root dcv-session-manager-broker generate-software-statement
```

Output

```
software-statement:
 ewogICJpZCIgOiAiYjc1NTVhN2QtNWI0MC00OTJhLWJjOTUtNmUzOWNhYzkxMDcxIiwKICAiYWN0aXZlIiA6IHRydWUsCi
```

describe-software-statements

Describes the existing software statements.

Topics

Syntax

```
sudo -u root dcv-session-manager-broker describe-software-statements
```

Output

`software-statement`

The software statement.

`issued-at`

The date and time the software was generated.

`is-active`

The current state of the software statement. `true` if the software statement is active; otherwise it's `false`.

Example

The following example generates a software statement.

Command

```
sudo -u root dcv-session-manager-broker describe-software-statements
```

Output

```
Software Statements
[ {
"software-statement" :
 "ewogICJpZCIgOiAiYmEEXAMPLEYtNzUwNy00YmFhLTliZWItYTA1MmJjZTE3NDJjIiwKICAiaXNzdWVkQXQiIDogMTU5N
"issued-at" : "2020.08.05 15:38:32 +0000",
"is-active" : "true"
}, {
"software-statement" :
 "EXAMPLEpZCIgOiAiYjc1NTVhN2QtNWI0MC00OTJhLWJjOTUtNmUzOWNhYzkxMDcxIiwKICAiaXNzdWEXAMPLEDogMTU5N
"issued-at" : "2020.08.07 10:24:41 +0000",
"is-active" : "true"
} ]
```

deactivate-software-statement

Deactivates a software statement. When you deactivate a software statement, it can no longer be used for Agent registrations.

Topics

Syntax

```
sudo -u root dcv-session-manager-broker deactivate-software-statement --software-
statement software_statement
```

Options

`--software-statement`

The software statement to deactivate.

Type: String

Required: Yes

Example

The following example deactivates a software statement.

Command

```
sudo -u root dcv-session-manager-broker deactivate-software-statement --software-
statement
 EXAMPLEpZCIgOiAiYjc1NTVhN2QtNWI0MC00OTJhLWJjOTUtNmUzOWNhYzkxMDcxIiwKICAiaXNEXAMPLEQiIDogMTU5Nj
```

Output

```
Software statement
 EXAMPLEpZCIgOiAiYjc1NTVhN2QtNWI0MC00OTJhLWJjOTUtNmUzOWNhYzkxMDcxIiwKICAiaXNEXAMPLEQiIDogMTU5Nj
 deactivated
```

describe-agent-clients

Describes the Agents that are registered with the Broker.

Topics

Syntax

```
sudo -u root dcv-session-manager-broker describe-agent-clients
```

Output

`name`

The name of the Agent.

`id`

The unique ID of the Agent.

`active`

The state of the Agent. `true` if the Agent is active; otherwise it's `false`.

Example

The following example describes the Agents.

Command

```
sudo -u root dcv-session-manager-broker describe-agent-clients
```

Output

```
Session manager agent clients
[ {
"name" : "test",
"id" : "6bc05632-70cb-4410-9e54-eaf9bEXAMPLE",
"active" : true
}, {
"name" : "test",
"id" : "27131cc2-4c71-4157-a4ca-bde38EXAMPLE",
"active" : true
}, {
"name" : "test",
"id" : "308dd275-2b66-443f-95af-33f63EXAMPLE",
"active" : false
}, {
"name" : "test",
"id" : "ce412d1b-d75c-4510-a11b-9d9a3EXAMPLE",
"active" : true
} ]
```

unregister-agent-client

Unregister an Agent from the Broker.

Topics

Syntax

```
sudo -u root dcv-session-manager-broker unregister-agent-client --client-id client_id
```

Options

`--client-id`

The ID of the Agent to unregister.

Type: String

Required: Yes

Example

The following example unregisters an Agent.

Command

```
sudo -u root dcv-session-manager-broker unregister-agent-client --client-id
 3b0d7b1d-78c7-4e79-b2e1-b976dEXAMPLE
```

Output

```
Agent client 3b0d7b1d-78c7-4e79-b2e1-b976dEXAMPLE unregistered
```

register-server-dns-mappings

Register the DCV Servers - DNS names mappings coming from a JSON file.

Syntax

```
sudo -u root dcv-session-manager-broker register-server-dns-mappings --file-
path file_path
```

Options

`--file-path`

The path of the file containing the DCV Servers - DNS names mappings.

Type: String

Required: Yes

Example

The following example registers the DCV Servers - DNS names mappings from file /tmp/mappings.json.

Command

```
sudo -u root dcv-session-manager-broker register-server-dns-mappings --file-path /tmp/mappings.json
```

Output

```
Successfully loaded 2 server id - dns name mappings from file /tmp/mappings.json
```

describe-server-dns-mappings

Describe the currently available DCV Servers - DNS names mappings.

Syntax

```
sudo -u root dcv-session-manager-broker describe-server-dns-mappings
```

Output

`serverIdType`

The type of the server Id.

serverId

The unique ID of the Server.

dnsNames

The internal and external dns names

internalDnsNames

The internal dns names

externalDnsNames

The external dns names

Example

The following example lists the registered DCV Servers - DNS names mappings.

Command

```
sudo -u root dcv-session-manager-broker describe-server-dns-mappings
```

Output

```
[
{
 "serverIdType" : "Id",
 "serverId" : "192.168.0.1",
 "dnsNames" : {
  "internalDnsName" : "internal1",
  "externalDnsName" : "external1"
 }
},
{
 "serverIdType" : "Host.Aws.Ec2InstanceId",
 "serverId" : "i-0648aee30bc78bdff",
 "dnsNames" : {
  "internalDnsName" : "internal2",
  "externalDnsName" : "external2"
 }
}
```

```
]
```

Configuration File Reference

This section provides information about the Agent and Broker configuration files.

Topics

Broker configuration file

The Broker configuration file (`/etc/dcv-session-manager-broker/session-manager-broker.properties`) includes parameters that can be configured to customize the Session Manager functionality. You can edit the configuration file using your preferred text editor.

> **Note**
>
> The `/etc/dcv-session-manager-broker/session-manager-broker.properties` file contains sensitive data. By default, its write access is restricted to root and its read access is restricted to root and to the user running the Broker. By default, this is the `dcvsmbroker` user. The Broker checks at startup that the file has the expected permissions.

The following table lists the parameters in the Broker configuration file.

Paramet	Required	Default value	Description
`broker-java-home`	No		Specifies the path to the Java home directory the Broker will use instead of the system default one. If set, the Broker will use `<broker-java-home>/bin/java` at startup.

Paramet	Required	Default value	Description
			Tip: the Broker requires Java Runtime Environme nt 11 and it is installed if missing as a dependency upon successfull installat ion. If version 11 is not set as default Java environme nt, its home directory can be grabbed using the following command: `$ sudo alternatives --display java`
sessior s creensh - max- widt h	No	160	Specifies the maximum width, in pixels, of session screenshots taken using the **GetSessionScreensh ots** API.
sessior s creensh - max- heig ht	No	100	Specifies the maximum height, in pixels, of session screenshots taken using the **GetSessionScreensh ots** API.
sessior s creensh - format	No	png	The image file format of session screenshots taken using the **GetSessio nScreenshots** API.

Paramet	Required	Default value	Description
create-sessions-queue-max-size	No	1000	The maximum number of unfulfilled **CreateSessions** API requests that can be queued. When the queue is full, new unfulfilled requests are rejected.
create-sessions-queue-max-time-seconds	No	1800	The maximum of time, in seconds, that an unfulfilled **CreateSessions** API request can remain in the queue. If the request cannot be fulfilled within the specified amount of time, it fails.
sessionmanager-working-path	Yes	/tmp	Specifies the path to the directory where the Broker writes the files needed to operate. This directory must be accessible only to the Broker.
enable-authorization-server	Yes	true	Specifies whether the broker is the authentication server used to generate OAuth 2.0 access tokens for client APIs.

Paramet	Required	Default value	Description
enable-authorization	Yes	true	Enables or disables client authorization. if you enable client authorization, the client API must provide an access token when making API requests. If you disable client authorization, client APIs can make requests without access tokens.
enable-agent-authorizatic	Yes	true	Enables or disables Agent authorization. If you enable Agent authorization, the Agent must provide an access token when communicating with the Broker.
delete-session-duration-hours	No	1	Specifies the number of hours after which deleted sessions become invisible and are no longer returned by DescribeSession API calls.

Paramet	Required	Default value	Description
connect session-token-token-duration-minutes	No	60	Specifies the number of minutes for which the ConnectSession token remains valid.
client-to-broker-connecto https-port	Yes	8443	Specifies the HTTPS port where the Broker listens for client connections.
client-to-broker-connecto bind-host	No	0.0.0.0	Specifies the IP address of the host where the Broker binds for client connections.
client-to-broker-connecto key-store-file	Yes		Specifies the key store used for TLS client connections.

Paramet	Required	Default value	Description
client-to-broker-connecto key-store-pass	Yes		Specifies the key store pass.
agent-to-broker-connecto https-port	Yes	8445	Specifies the HTTPS port where the Broker listens for Agent connections.
agent-to-broker-connecto bind-host	No	0.0.0.0	Specifies the IP address of the host where the Broker binds for Agent connections.

Paramet	Required	Default value	Description
agent-to-broker-connector-key-store-file	Yes		Specifies the key store used for TLS Agent connections.
agent-to-broker-connector-key-store-pass	Yes		Specifies the key store pass.
broker-to-broker-port	Yes	47100	Specifies the port used for Broker-to-Broker connections.
broker-to-broker-bind-host	No	0.0.0.0	Specifies the IP address of the host where the Broker binds for Broker-to-Broker connections.

Paramet	Required	Default value	Description
broker-to-broker-discover port	Yes	47500	Specifies the port used by Brokers to discover each other.
broker-to-broker-discover address	No		Specifies the IP addresses and ports of the other brokers in the fleet in the *ip_address* :*port* format. If there are multiple Brokers, separate the values with a comma. If you specify broker-to-broker-discovery-multicast-group , broker-to-broker-discovery-multicast-port , broker-to-broker-discovery-Amazon-region , or broker-to-broker-discovery-Amazon-alb-target-group-arn , omit this parameter.

Paramet	Required	Default value	Description
broker-to-broker-discovery-multica-group	No		Specifies the multicast group for Broker-to-roker discovery. If you specify broker-to-broker-discovery-addresses , broker-to-broker-discovery-aws-region , or broker-to-broker-discovery-Amazon-alb-target-group-arn , omit this parameter.
broker-to-broker-discovery-multica-port	No		Specifies the multicast port for Broker-o-oker discovery. If you specify broker-to-broker-discovery-addresses , broker-to-broker-discovery-Amazon-region , or broker-to-broker-discovery-Amazon-alb-target-group-arn , omit this parameter.

Paramet	Required	Default value	Description
broker-to-broker-discovery-Amazon-region	No		Specifies the Amazon Region of the application load balancer used for Broker to Broker discovery. If you specify broker-to-broker-discovery-multicast-group, broker-to-broker-discovery-multicast-port, or broker-to-broker-discovery-addresses, omit this parameter.
broker-to-broker-discovery-Amazon-alb-target-group-arn	No		The ARN of the application load balancer target group user for Broker-to-Broker discovery. If you specify broker-to-broker-discovery-multicast-group, broker-to-broker-discovery-multicast-port, or broker-to-broker-discovery-addresses, omit this parameter.

Paramet	Required	Default value	Description
broker-to-broker-distributed-memory-max-size-mb	No	4096	Specifies the maximum amount of off -heap memory to be used by a single Broker to store NICE DCV session data.
broker-to-broker-key-store-file	Yes		Specifies the key store used for TLS Broker connections.
broker-to-broker-key-store-pass	Yes		Specifies the key store pass.
enable-cloud-watch-metrics	No	false	Enables or disables Amazon CloudWatch metrics. If you enable CloudWatch Metrics, you might need to specify a value for cloud-watch-region .

Paramet	Required	Default value	Description
cloud-watch-region	No	Only required if enable-cloud-watch-metrics is set to true. If the Broker is installed on an Amazon EC2 instance, the region is retrieved from the IMDS.	The Amazon region where the CloudWatch metrics are posted.
max-api-requests-per-second	No	1000	Specifies the maximum number of requests that the Broker api can process each second before being throttled.
enable-throttling-forwarded-for-header	No	false	If set to true the throttling retrieves the caller ip from the X-Forwared-For header if present.
create-sessions-number-of-retries-on-failure	No	2	Specifies the maximum number of retries to be performed after a create session request fails on a NICE DCV server host. Set to 0 to never perform retries on failures.

Paramet	Required	Default value	Description
autorun file-arguments-max-size	No	50	Specifies the maximum number of arguments that can be passed to the autorun file.
autorun file-arguments-max-argumer length	No	150	Specifies the maximum length in chars of each autorun file argument.
enable- pe rsister	Yes	false	If set to true, the broker status data is persisted on an external database.
persist ce-db	No	Only required if enable-persistence is set to true.	Specifies which database is used for persistence. The only supported values are: dynamodb and mysql.
dynamoc region	No	Only required if enable-persistence is set to true and persistence-db is set to dynamodb.	Specifies the region where the DynamoDB tables are created and accessed.

Paramet	Required	Default value	Description
dynamoc table-rcu	No	Only required if enable-persistence is set to true and persistence-db is set to dynamodb.	Specifies the Read Capacity Units (RCU) for each DynamoDB table For more information on RCU, see Pricing for Provisioned Capacity.
dynamoc table-wcu	No	Only required if enable-persistence is set to true and persistence-db is set to dynamodb.	Specifies the Write Capacity Units (WCU) for each DynamoDB table. For more information on WCU, see Pricing for Provisioned Capacity.
dynamoc table-nam e-prefix	No	Only required if enable-persistence is set to true and persistence-db is set to dynamodb.	Specifies the prefix that is added to each DynamoDB table (useful to distingui sh multiple broker clusters using the same Amazon account). Only alphanume ric characters, dot, dash and underscore are allowed.

Paramet	Required	Default value	Description
jdbc-connection-url	No	Only required if enable-persistence is set to true and persistence-db is set to mysql.	Specifies the connection URL to the MariaDB/MySQL database; it contains the endpoint and the database name. The url should have this format: `jdbc:mysql://<db_endpoint>:<db_port>/<db_name>?createDatabaseIfNotExist=true` Where <db_endpoint> is the MariaDB/MySQL database endpoint, <db_port> is the database port and <db_name> is the database name.
jdbc-user	No	Only required if enable-persistence is set to true and persistence-db is set to mysql.	Specifies the name of the user that has access to the MariaDB/MySQL database.
jdbc-password	No	Only required if enable-persistence is set to true and persistence-db is set to mysql.	Specifies the password of the user that has access to the MariaDB/MySQL database.

Paramet	Required	Default value	Description
seconds b efore- del eting- unr eachabl dcv- serve r	No	1800	Specifies the amount of seconds after which an unreachable server is deleted from the system.

Agent configuration file

The Agent configuration file (`/etc/dcv-session-manager-agent/agent.conf` for Linux and `C:\Program Files\NICE\DCVSessionManagerAgent\conf\agent.conf` for Windows) includes parameters that can be configured to customize the Session Manager functionality. You can edit the configuration file using your preferred text editor.

The following table lists the parameters in the Agent configuration file.

Paramet	Required	Default value	Description
agent.b ker_hos	Yes		Specifies the DNS name of the Broker host.
agent.b ker_por	Yes	8445	Specifies the port over which to communicate with the Broker.
agent.c file	No		Only needed if `tls_strict` is set to `true`. Specifies the path to the certificate (.pem) file needed to validate the TLS certificate. Copy

Paramet	Required	Default value	Description
			the self-signed certifica te from the Broker to the Agent.
agent.i t_folde	No	• /var/lib/dcv-session-manager-agent/init (Linux)	Specifies the path to a folder on the host server used to store custom scripts allowed to initialize NICE DCV server sessions when they are created. You must specify an absolute path. The folder must be accessibl e and the files must be executable by users who make use of the **InitFile** request parameter of the **CreateSessions** API.
agent.t _strict	No	true	Indicates whether strict TLS validation should be used.
agent.s tware_s tement_ th	No		Only needed if the default software statement is not used. Specifies the path to the software statement file. For more information, see generate-software-statement.

Paramet	Required	Default value	Description
agent.t s_folde	No	• /etc/dcv-session-manager-agent (Linux) • C:\Program Files \NICE\DCVSess ionManagerAgent \conf\tags (Windows)	Specifies the path to the folder in which the tag files are located. For more information, see Using tags to target NICE DCV servers.
agent.a orun_fo er	No	• /var/lib/dcv-session-manage r-agent/autorun (Linux) • C:\ProgramData \NICE\DcvSess ionManagerAgent \autorun (Windows)	Specifies the path to a folder on the host server used to store scripts and apps that are allowed to be automatically run at session startup. You must specify an absolute path. The folder must be accessible and the files must be executabl e by users who make use of the **AutorunFile** request parameter of the **CreateSessions** API.
agent.m _virtua sessior	No	-1 (no limit)	The maximum number of virtual sessions that can be created on a NICE DCV server using NICE DCV Session Manager.

Paramet	Required	Default value	Description
agent.m_concurrent_sessions_per_user	No	1	The maximum number of virtual sessions that can be created on a NICE DCV server by a single user using NICE DCV Session Manager.
agent.broker_update_interval	No	30	Specifies the amount of seconds to wait before sending updated data to the Broker. Sent data includes NICE DCV server and host status, as well as updated session information. Lower values make the Session Manager more aware of changes happening on the system where the Agent runs, but increase system load and network traffic. Higher values decrease system and network load, but the Session Manager becomes less responsive to system changes, thus values higher than 120 are not recommended.

Paramet	Required	Default value	Description
log.lev	No	info	Specifies the verbosity level of the log files. The following verbosity levels are available: • error—Provides the least detail. Includes errors only. • warning—Includes errors and warnings. • info—The default verbosity level. Includes errors, warnings, and information messages. • debug—Provides the most detail. Provides detailed informati on that is useful for debugging issues.
log.dir tory	No	• /var/log/dcv-session-manager-agent/ (Linux) • C:\ProgramData \NICE\DCVSess ionManagerAgent \log (Windows)	Specifies the directory in which to create log files.

Paramet	Required	Default value	Description
log.rot ion	No	daily	Specifies the log file rotation. Valid values are: • hourly—Log files are rotated hourly. • daily—Log files are rotated daily.
log.max f ile- size	No	10485760	When a log file size reaches the specified size in bytes, it will be rotated. A new log file will be created and further log events will be placed in the new file.
log.rot e	No	9	The maximum number of log files preserved in the rotation. Each time a rotation happens and this number is reached, the oldest log file will be deleted.

Release notes and document history for NICE DCV Session Manager

This page provides the release notes and document history for NICE DCV Session Manager.

Topics

NICE DCV Session Manager release notes

This section provides an overview of the major updates, feature releases, and bug fixes for NICE DCV Session Manager. All the updates are organized by release date. We update the documentation frequently to address the feedback that you send us.

Topics

2023.1— November 9, 2023

Build numbers	Changes and bug fixes
• Broker: 410 • Agent: 732 • CLI: 140	• Bug fixes and performance improvements

2023.0-15065— May 4, 2023

Build numbers	Changes and bug fixes
• Broker: 392 • Agent: 675 • CLI: 132	• Added support for Red Hat Enterprise Linux 9, Rocky Linux 9, and CentOS Stream 9 on ARM platforms.

2023.0-14852— March 28, 2023

Build numbers	Changes and bug fixes
• Broker: 392 • Agent: 642 • CLI: 132	• Added support for Red Hat Enterprise Linux 9, Rocky Linux 9, and CentOS Stream 9.

2022.2-13907— November 11, 2022

Build numbers	Changes and bug fixes
• Broker: 382 • Agent: 612 • CLI: 123	• Added a `Substate` field in `DescribeSessions` response. • Fixed a problem that could cause the CLI to fail to connect to the broker depending on the URL in use.

2022.1-13067— June 29, 2022

Build numbers	Changes and bug fixes
• Broker: 355 • Agent: 592 • CLI: 114	• Added support to run the broker on Amazon Graviton instances. • Added agent and broker support for Ubuntu 22.04.

2022.0-11952— February 23, 2022

Build numbers	Changes and bug fixes
• Broker: 341 • Agent: 520 • CLI: 112	• Added log rotation capability to the Agent. • Added configuration parameter to set Java home in the Broker. • Improved data flushing from cache to disk in the Broker. • Fixed URL validation in the CLI.

2021.3-11591— December 20, 2021

Build numbers	New features
• Broker: 307 • Agent: 453 • CLI: 92	• Added support for integrating with the NICE DCV Connection Gateway. • Added Broker support for Ubuntu 18.04 and Ubuntu 20.04.

2021.2-11445— November 18, 2021

Build numbers	Changes and bug fixes
• Broker: 288 • Agent: 413 • CLI: 54	• Fixed a problem with the validation of login names which include a Windows domain.

2021.2-11190— October 11, 2021

Build numbers	Changes and bug fixes
• Broker: 254 • Agent: 413 • CLI: 54	• Fixed a problem in the command line interface which prevented from launching Windows sessions.

2021.2-11042— September 01, 2021

Build numbers	New features	Changes and bug fixes
• Broker: 25 • Agent: 413 • CLI: 37	• NICE DCV Session Manager now offers command line interface (CLI) support. You can create and manage NICE DCV sessions in the CLI, instead of calling APIs. • NICE DCV Session Manager introduce d Broker data persistence. For higher availability, brokers can persist server state information on an external data store and restore the data at startup.	• When registering an external authorization server, you can now specify the algorithm that the authorization server uses to sign JSON-formatted Web Tokens. With this change, you can use Azure AD as an external authorization server.

2021.1-10557— May 31, 2021

Build numbers	New features	Changes and bug fixes
• Broker: 21 • Agent: 365	• NICE DCV Session Manager added support for input parameters passed to the autorun file on Linux.	• We fixed a problem with the autorun file on Windows.

Build numbers	New features	Changes and bug fixes
	• Server properties can now be passed as requirements to the CreateSessions API.	

2021.0-10242— April 12, 2021

Build numbers	Changes and bug fixes

• Broker: 183
• Agent: 318
	• NICE DCV Session Manager introduced the following new APIs:
	  • OpenServers
	  • CloseServers
	  • DescribeServers
	  • GetSessionScreenshots
	• It also introduced the following new configuration parameters:
	  • Broker parameters: `session-screenshot-max-width` , `session-screenshot-max-height` , `session-screenshot-format` , `create-sessions-queue-max-size` , and `create-sessions-queue-max-time-seconds` .
	  • Agent parameters: `agent.autorun_folder` , `max_virtual_sessions` , and `max_concurrent_sessions_per_user` .
	
	  Agent parameters: `agent.autorun_folder` ,`max_virtual_sessions` , and `max_concurrent_sessions_per_user` .
	
	  Agent parameters: `agent.autorun_folder` ,`max_virtual_sessions` , and `max_concurrent_sessions_per_user` .

2020.2-9662— December 04, 2020

Build numbers	Changes and bug fixes
• Broker: 114 • Agent: 211	• We fixed a problem with the auto-generated TLS certificates that prevented the Broker from starting.

2020.2-9508— November 11, 2020

Build numbers	Changes and bug fixes
• Broker: 78 • Agent: 183	• The initial release of NICE DCV Session Manager.

Document history

The following table describes the documentation for this release of NICE DCV Session Manager.

Change	Description	Date
NICE DCV Version 2023.1	NICE DCV Session Manager has been updated for NICE DCV 2023.1. For more information, see 2023.1— November 9, 2023.	November 9, 2023
NICE DCV Version 2023.0	NICE DCV Session Manager has been updated for NICE DCV 2023.0. For more information, see 2023.0-14852— March 28, 2023.	March 28, 2023
NICE DCV Version 2022.2	NICE DCV Session Manager has been updated for NICE DCV 2022.2. For more information, see 2022.2-13907— November 11, 2022.	November 11, 2022

Change	Description	Date
NICE DCV Version 2022.1	NICE DCV Session Manager has been updated for NICE DCV 2022.1. For more information, see 2022.1-13067—June 29, 2022.	June 29, 2022
NICE DCV Version 2022.0	NICE DCV Session Manager has been updated for NICE DCV 2022.0. For more information, see 2022.0-11952—February 23, 2022.	February 23, 2022
NICE DCV Version 2021.3	NICE DCV Session Manager has been updated for NICE DCV 2021.3. For more information, see 2021.3-11591—December 20, 2021.	December 20, 2021
NICE DCV Version 2021.2	NICE DCV Session Manager has been updated for NICE DCV 2021.2. For more information, see 2021.2-11042—September 01, 2021.	September 01, 2021
NICE DCV Version 2021.1	NICE DCV Session Manager has been updated for NICE DCV 2021.1. For more information, see 2021.1-10557—May 31, 2021.	May 31, 2021
NICE DCV Version 2021.0	NICE DCV Session Manager has been updated for NICE DCV 2021.0. For more information, see 2021.0-10242—April 12, 2021.	April 12, 2021
Initial release of NICE DCV Session Manager	The first publication of this content.	November 11, 2020