aws

管理员指南

NICE DCV Session Manager

NICE DCV Session Manager: 管理员指南

Table of Contents

什么是 NICE DCV Session Manager？

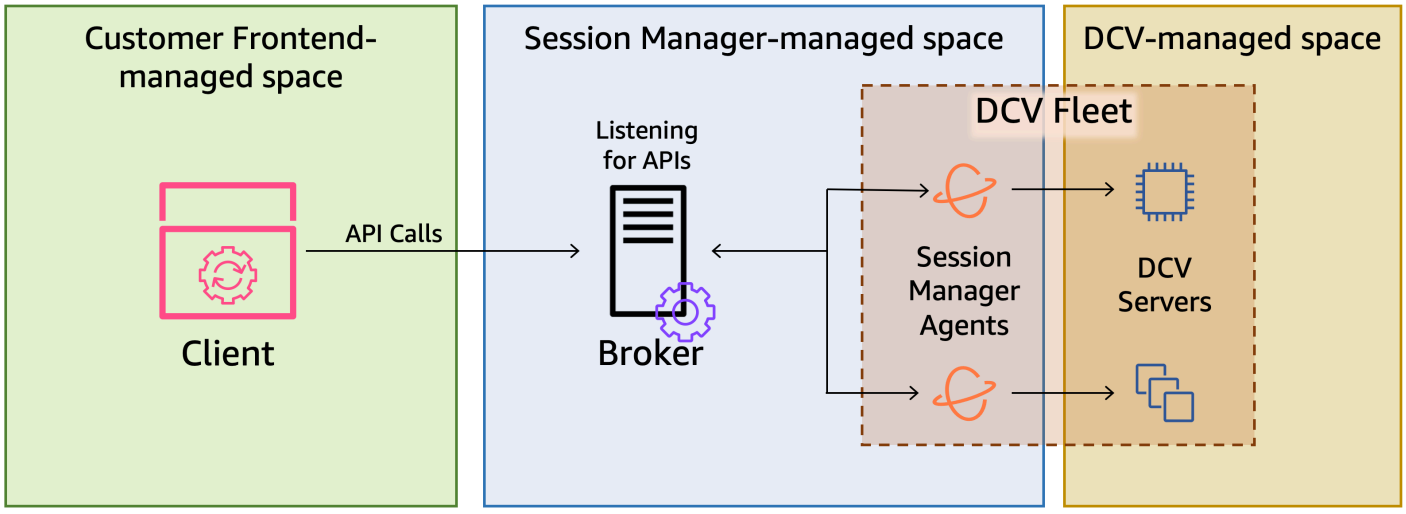
NICE DCV Session Manager 是一组可安装的软件包（Agent 和 Broker）和一个应用程序编程接口 (API)，使开发人员和独立软件供应商 (ISV) 可以轻松构建前端应用程序，从而以编程方式创建和管理一组 NICE DCV 服务器中的 NICE DCV 会话的生命周期。

本指南介绍了如何安装和配置 Session Manager Agent 和 Broker。有关使用 Session Manager API 的更多信息，请参阅《NICE DCV Session Manager 开发人员指南》。

主题

Session Manager 的工作方式

下图简要显示了 Session Manager 组件。



代理

Broker 是一个托管并公开 Session Manager API 的 Web 服务器。它接收并处理来自客户端的 API 请求以管理 NICE DCV 会话，然后将指令传送到相关的 Agent。Broker 必须安装在与 NICE DCV 服务器分开的主机上，但客户端必须可以访问 Broker，并且 Broker 必须能够访问 Agent。

Agent

Agent 安装在一组 NICE DCV 服务器中的每个服务器上。Agent 从 Broker 接收指令，并在相应的 NICE DCV 服务器上运行这些指令。Agent 还监控 NICE DCV 服务器的状态，并将定期状态更新发回到 Broker。

API

Session Manager 公开一组 REST 应用程序编程接口 (API)，这些 API 可用于管理一组 NICE DCV 服务器上的 NICE DCV 会话。这些 API 在 Broker 上托管并由 Broker 公开。开发人员可以构建调用这些 API 的自定义会话管理客户端。

客户端

客户端是您开发的前端应用程序或门户，用于调用 Broker 公开的 Session Manager API。最终用户使用客户端以管理一组 NICE DCV 服务器上托管的会话。

访问令牌

要发出 API 请求，您必须提供访问令牌。可以通过注册的客户端 API 从 Broker 或外部授权服务器中请求令牌。要请求和访问令牌，客户端 API 必须提供有效的凭证。

客户端 API

客户端 API 是使用 Swagger Codegen 从 Session Manager API 定义 YAML 文件中生成的。客户端 API 用于发出 API 请求。

NICE DCV 会话

您必须在客户端可以连接到的 NICE DCV 服务器上创建 NICE DCV 会话。只有在存在活动会话时，客户端才能连接到 NICE DCV 服务器。NICE DCV 支持控制台会话和虚拟会话。您可以使用 Session Manager API 管理 NICE DCV 会话的生命周期。NICE DCV 会话可以处于以下状态之一：

- `CREATING` - Broker 正在创建会话。
- `READY` - 会话准备好接受客户端连接。
- `DELETING` - 正在删除会话。
- `DELETED` - 已删除会话。
- `UNKNOWN` - 无法确定会话的状态。Broker 和 Agent 可能无法通信。

特征

DCV Session Manager 提供以下功能：

- 提供 NICE DCV 会话信息 - 获取有关在多个 NICE DCV 服务器上运行的会话的信息。
- 管理多个 NICE DCV 会话的生命周期 - 使用一个 API 请求为多个 NICE DCV 服务器中的多个用户创建或删除多个会话。
- 支持标签 - 在创建会话时，使用自定义标签定位一组 NICE DCV 服务器。
- 管理多个 NICE DCV 会话的权限 - 使用一个 API 请求修改多个会话的用户权限。
- 提供连接信息 - 检索 NICE DCV 会话的客户端连接信息。
- 支持云和本地 - 在 AWS、本地或其他基于云的服务器上使用 Session Manager。

限制

Session Manager 不提供资源预置功能。如果在 Amazon EC2 实例上运行 NICE DCV，您可能需要使用其他 AWS 服务（例如 Amazon EC2 Auto Scaling）管理基础设施扩缩。

定价

运行 EC2 实例的 AWS 客户可以免费使用 Session Manager。

本地客户需要具有 NICE DCV Plus 或 DCV Professional Plus 许可证。有关如何购买 NICE DCV Plus 或 NICE DCV Professional Plus 许可证的信息，请参阅 NICE 网站上的 How to Buy 并查找您所在区域的 NICE 分销商或经销商。为了允许所有本地客户试用 DCV Session Manager，仅从 NICE DCV 版本 2021.0 开始强制实施许可要求。

有关更多信息，请参阅《NICE DCV 管理员指南》中的为 NICE DCV 服务器授予许可。

NICE DCV Session Manager 要求

NICE DCV Session Manager Agent 和 Broker 具有以下要求。

	代理	Agent
操作系统	• Amazon Linux 2	• Windows

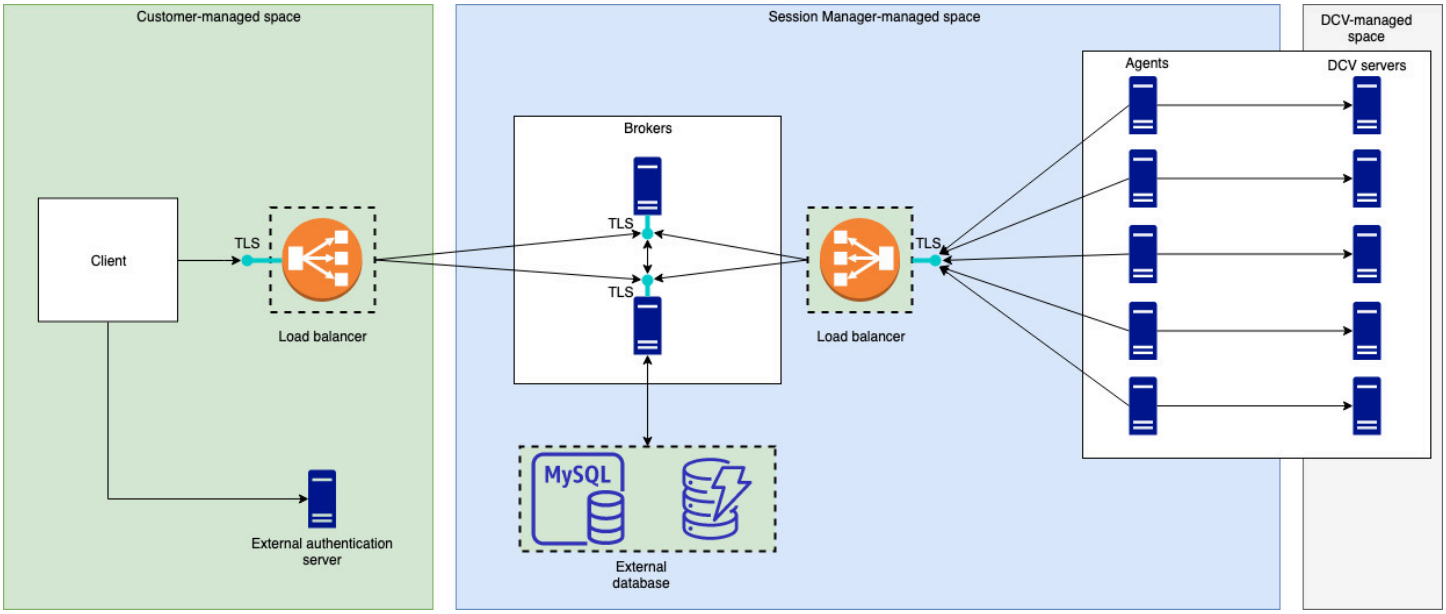
	代理	Agent

	• CentOS 7.6 或更高版本
	• CentOS Stream 8
	• RHEL 7.6 或更高版本
	• RHEL 8.x
	• Rocky Linux 8.5 或更高版本
	• Ubuntu 20.04
	• Ubuntu 22.04
		• Windows Server 2019
		• Windows Server 2016
		• Windows Server 2012 R2
		• Linux 服务器
		  • Amazon Linux 2
		  • CentOS 7.6 或更高版本
		  • CentOS Stream 8
		  • RHEL 7.6 或更高版本
		  • RHEL 8.x
		  • Rocky Linux 8.5 或更高版本
		  • Ubuntu 20.04
		  • Ubuntu 22.04
		  • SUSE Linux Enterprise 12（带 SP4）或更高版本
		  • SUSE Linux Enterprise 15

架构	• 64 位 x86 • 64 位 ARM	• 64 位 x86 • 64 位 ARM（仅限 Amazon Linux 2、CentOS 7.x/8.x 和 RHEL 7.x/8.x） • 64 位 ARM (Ubuntu 22.04)
内存	8GB	4 GB
NICE DCV 版本	NICE DCV 2020.2 和更高版本	NICE DCV 2020.2 和更高版本
其他要求	Java 11	-

网络和连接要求

下图简要说明了 Session Manager 网络和连接要求。



Broker 必须安装在单独的主机上，但它必须具有到 NICE DCV 服务器上的 Agent 的网络连接。如果您选择使用多个 Broker 以提高可用性，则必须在单独的主机上安装和配置每个 Broker，并使用一个或多个负载均衡器管理客户端和 Broker 之间以及 Broker 和 Agent 之间的流量。Broker 还应该能够相互通信，以便交换有关 NICE DCV 服务器和会话的信息。Broker 可以将其密钥和状态数据存储在外部数据库上，并在重新引导或终止后能够使用该信息。通过将重要 Broker 信息持久保留在外部数据库上，这有助于降低丢失这些信息的风险。您以后可以检索这些信息。如果您选择使用该功能，则必须设置外部数据库并配置这些 Broker。支持 DynamoDB、MariaDB 和 MySQL。您可以找到 Broker 配置文件中列出的配置参数。

Agent 必须能够启动到 Broker 的安全持久双向 HTTPs 连接。

您的客户端或前端应用程序必须能够访问 Broker 才能调用这些 API。客户端还应该能够访问您的身份验证服务器。

设置 NICE DCV Session Manager

下一节介绍了如何安装具有单个 Broker 和多个 Agent 的 Session Manager。您可以使用多个 Broker 以提高可扩展性和性能。有关更多信息，请参阅 扩展 Session Manager。

要设置 NICE DCV Session Manager，请执行以下操作：

步骤

步骤 1：准备 NICE DCV 服务器

您必须具有一组打算使用 Session Manager 的 NICE DCV 服务器。有关安装 NICE DCV 服务器的更多信息，请参阅《NICE DCV 管理员指南》中的安装 NICE DCV 服务器。

在 Linux NICE DCV 服务器上，Session Manager 使用名为 `dcvsmagent` 的本地服务用户。在安装 Session Manager Agent 时，将自动创建该用户。您必须为该服务用户授予 NICE DCV 管理员权限，以使该用户可以代表其他用户执行操作。要为 Session Manager 服务用户授予管理员权限，请执行以下操作：

为 Linux NICE DCV 服务器添加本地服务用户

1. 使用所需的文本编辑器打开 `/etc/dcv/dcv.conf`。
2. 将 `administrators` 参数添加到 `[security]` 部分并指定 Session Manager 用户。例如：

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

3. 保存并关闭文件。
4. 停止并重新启动 NICE DCV 服务器。

Session Manager 只能代表 NICE DCV 服务器上已存在的用户创建 NICE DCV 会话。如果发出请求为不存在的用户创建会话，该请求将失败。因此，您必须确保每个预期最终用户在 NICE DCV 服务器上具有有效的系统用户。

> **Tip**
>
> 如果您打算使用多个 Broker 主机或带有 Agent 的 NICE DCV 服务器，我们建议您仅配置一个 Broker 和一个带有 Agent 的 NICE DCV 服务器，方法是执行以下步骤，创建具有完成的配置的主机的亚马逊机器映像（AMI），然后使用该 AMI 启动其余 Broker 和 NICE DCV 服务器。或者，您可以使用 AWS Systems Manager 在多个实例上远程运行命令。

步骤 2：设置 NICE DCV Session Manager Broker

Broker 必须安装在 Linux 主机上。有关支持的 Linux 发行版的更多信息，请参阅NICE DCV Session Manager 要求。将 Broker 安装在与 Agent 和 NICE DCV 服务器主机不同的主机上。该主机可以安装在不同的私有网络上，但它必须能够连接到 Agent 并与其通信。

安装并启动 Broker

1. 连接到您打算在其中安装 Broker 的主机。
2. 程序包使用安全 GPG 签名进行数字签名。要允许软件包管理器验证软件包签名，您必须导入 NICE GPG 密钥。运行以下命令以导入 NICE GPG 密钥。

   - Amazon Linux 2、RHEL、CentOS 和 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. 下载安装软件包。

   - Amazon Linux 2、RHEL 7.x 和 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 和 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. 安装 软件包。

- Amazon Linux 2、RHEL 7.x 和 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 和 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. 检查默认 Java 环境版本是否为 11。

```
$ java -version
```

如果不是，您可以明确设置 Broker 用于定位正确 Java 版本的 Java 主目录。这是通过在 Broker 配置文件中设置 `broker-java-home` 参数完成的。有关更多信息，请参阅 Broker 配置文件。

6. 启动 Broker 服务，并确保该服务在每次实例启动时自动启动。

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

7. 将 Broker 的自签名证书副本放置在您的用户目录中。在下一步中安装 Agent 时，您需要使用该证书。

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

步骤 3：设置 NICE DCV Session Manager Agent

Agent 必须安装在一组 NICE DCV 服务器主机中的所有主机上。Agent 可以安装在 Windows 和 Linux 服务器上。有关支持的操作系统的更多信息，请参阅NICE DCV Session Manager 要求。

先决条件

在安装 Agent 之前，必须在主机上安装 NICE DCV 服务器

Linux host

> **Note**
>
> 会话管理器代理可用于 “要求” 中列出的 Linux 发行版和架构：
> 以下说明适用于在 64 位 x86 主机上安装 Agent。要在 64 位 ARM 主机上安装代理，请将 *x8* 6_64 替换为。`aarch64`对于 Ubuntu，将 *am* d64 替换为。`arm64`

在 Linux 主机上安装 Agent

1. 程序包使用安全 GPG 签名进行数字签名。要允许软件包管理器验证软件包签名，您必须导入 NICE GPG 密钥。运行以下命令以导入 NICE GPG 密钥。

   - Amazon Linux 2、RHEL、CentOS 和 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. 下载安装软件包。

- Amazon Linux 2、RHEL 7.x 和 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 和 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. 安装 软件包。

- Amazon Linux 2、RHEL 7.x 和 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 和 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. 将 Broker 的自签名证书副本（您在上一步中复制的副本）放置在 Agent 上的 `/etc/dcv-session-manager-agent/` 目录中。
5. 使用常用的文本编辑器打开 `/etc/dcv-session-manager-agent/agent.conf`，然后执行以下操作。

   - 对于 `broker_host`，指定在其中安装 Broker 的主机的 DNS 名称。

     Important

     如果 Broker 在 Amazon EC2 实例上运行，您必须为 `broker_host` 指定实例的私有 IPv4 地址。

- （可选）对于 `broker_port`，指定用于与 Broker 通信的端口。默认情况下，Agent 和 Broker 通过 8445 端口进行通信。只有在您需要使用不同的端口时，才需要更改该设置。如果您确实要更改该设置，请确保 Broker 配置为使用相同的端口。
- 对于 `ca_file`，指定您在上一步中复制的证书文件的完整路径。例如：

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

或者，如果您要禁用 TLS 验证，请将 `tls_strict` 设置为 `false`。

6. 保存并关闭该文件。
7. 运行以下命令以启动 Agent。

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

Windows host

在 Windows 主机上安装 Agent

1. 下载 Agent 安装程序。
2. 运行安装程序。在欢迎屏幕上，选择 Next。
3. 在 EULA 屏幕上，仔细阅读许可协议；如果同意，请选择 I accept the terms 并选择 Next。
4. 要开始安装，请选择 Install。
5. 将 Broker 的自签名证书副本（您在上一步中复制的副本）放置在 Agent 上的 `C:\Program Files\NICE\DCVSessionManagerAgent\conf\` 文件夹中。
6. 使用常用的文本编辑器打开 `C:\Program Files\NICE\DCVSessionManagerAgent\conf\agent.conf`，然后执行以下操作：

- 对于 `broker_host`，指定在其中安装 Broker 的主机的 DNS 名称。

> **Important**
>
> 如果 Broker 在 Amazon EC2 实例上运行，您必须为 `broker_host` 指定实例的私有 IPv4 地址。

- （可选）对于 broker_port，指定用于与 Broker 通信的端口。默认情况下，Agent 和 Broker 通过 8445 端口进行通信。只有在您需要使用不同的端口时，才需要更改该设置。如果您确实要更改该设置，请确保 Broker 配置为使用相同的端口。
- 对于 ca_file，指定您在上一步中复制的证书文件的完整路径。例如：

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

或者，如果您要禁用 TLS 验证，请将 tls_strict 设置为 false。

7. 保存并关闭该文件。
8. 停止并重新启动 Agent 服务以使更改生效。在命令提示符下运行以下命令。

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

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

步骤 4：配置 NICE DCV 服务器以将 Broker 作为身份验证服务器

配置 NICE DCV 服务器以将 Broker 作为外部身份验证服务器，以验证客户端连接令牌。您还必须将 NICE DCV 服务器配置为信任 Broker 的自签名 CA。

Linux NICE DCV server

为 Linux NICE DCV 服务器添加本地服务用户

1. 使用所需的文本编辑器打开 /etc/dcv/dcv.conf。
2. 将 ca-file 和 auth-token-verifier 参数添加到 [security] 部分中。

   对于 ca-file，指定您在上一步中复制到主机的 Broker 自签名 CA 的路径。

   对于 auth-token-verifier，按以下格式指定 Broker 上的令牌验证程序的 URL：https://*broker_ip_or_dns*:*port*/agent/validate-authentication-token。指定用于 Broker-Agent 通信的端口，默认为 8445。如果在 Amazon EC2 实例上运行 Broker，您必须使用私有 DNS 或私有 IP 地址。

   例如

```
[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. 保存并关闭文件。
4. 停止并重新启动 NICE DCV 服务器。有关更多信息，请参阅《NICE DCV 管理员指南》中的停止 NICE DCV 服务器和启动 NICE DCV 服务器。

Windows NICE DCV server

在 Windows NICE DCV 服务器上

1. 打开 Windows 注册表编辑器，并导航到 HKEY_USERS/S-1-5-18/Software/GSettings/com/nicesoftware/dcv/security/ 项。
2. 打开 ca-file 参数。对于值数据，指定您在上一步中复制到主机的 Broker 自签名 CA 的路径。

> **Note**
>
> 如果该参数不存在，则创建一个新的字符串参数并将其命名为 `ca-file`。

3. 打开auth-token-verifier参数。对于值数据，按以下格式指定 Broker 上的令牌验证程序的 URL：`https://broker_ip_or_dns:port/agent/validate-authentication-token`。指定用于 Broker-Agent 通信的端口，默认为 8445。如果在 Amazon EC2 实例上运行 Broker，您必须使用私有 DNS 或私有 IP 地址。

> **Note**
>
> 如果该参数不存在，则创建一个新的字符串参数并将其命名为 `auth-token-verifier`。

4. 选择确定，并关闭 Windows 注册表编辑器。
5. 停止并重新启动 NICE DCV 服务器。有关更多信息，请参阅《NICE DCV 管理员指南》中的停止 NICE DCV 服务器和启动 NICE DCV 服务器。

步骤 5：验证安装

主题

验证 Agent

在安装 Broker 和 Agent 后，确保 Agent 正在运行并且能够连接到 Broker。

Linux Agent 主机

要运行的命令取决于版本。

- 自版本 2022.0 起

  从 Agent 主机中，运行以下命令：

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

- 2022.0 之前的版本

  从 Agent 主机中，运行以下命令并指定当前年份、月份和日期。

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

  例如

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

如果 Agent 正在运行并且能够连接到 Broker，该命令应返回 `success`。

如果该命令返回不同的输出，请检查 Agent 日志文件以获取更多信息。这些日志文件位于以下位置中： `/var/log/dcv-session-manager-agent/`。

Windows Agent 主机

打开 Agent 日志文件，该文件位于 `C:\ProgramData\NICE\DCVSessionManagerAgent\log` 中。

如果该日志文件包含类似下面的行，则 Agent 正在运行并且能够连接到 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}"
```

如果您的日志文件没有类似的行，请检查日志文件以查找错误。

验证 Broker

在安装 Broker 和 Agent 后，请确保 Broker 正在运行，并且您的用户和前端应用程序可以访问 Broker。

从应该能够访问 Broker 的计算机中，运行以下命令：

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

如果验证成功，Broker 将返回以下内容：

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

配置 NICE DCV Session Manager

本节介绍了如何为 Session Manager 执行高级配置。

主题

扩展 Session Manager

为了实现高可用性并提高性能，您可以将 Session Manager 配置为使用多个 Agent 和 Broker。如果您确实打算使用多个 Agent 和 Broker，我们建议您仅安装并配置一个 Agent 和 Broker 主机，从这些主机中创建亚马逊机器映像（AMI），然后通过该 AMI 启动其余主机。

默认情况下，Session Manager 支持使用多个 Agent，而无需进行任何额外的配置。不过，如果您打算使用多个 Broker，您必须使用负载均衡器均衡前端客户端和 Broker 之间的流量以及 Broker 和 Agent 之间的流量。负载均衡器的设置和配置完全由您拥有和管理。

下一节介绍了如何配置 Session Manager 以将多个主机与 Application Load Balancer 一起使用。

步骤

步骤 1：创建实例配置文件

您必须将实例配置文件附加到 Broker 和 Agent 主机，以授予它们使用 Elastic Load Balancing API 的权限。有关更多信息，请参阅《Amazon EC2 用户指南》中的适用于 Amazon EC2 的 IAM 角色。

要创建实例配置文件

1. 创建一个 AWS Identity and Access Management (IAM) 角色来定义要在实例配置文件中使用的权限。使用以下信任策略：

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

然后，附加以下策略：

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

```
  ]
}
```

有关更多信息，请参阅《IAM 用户指南》中的创建 IAM 角色。

2. 创建一个新的实例配置文件。有关更多信息，请参阅《AWS CLI 命令参考》中的 create-instance-profile。
3. 将 IAM 角色添加到实例配置文件。有关更多信息，请参阅《AWS CLI 命令参考》中的 add-role-to-instance-profile。
4. 将该实例配置文件附加到 Broker 主机。有关更多信息，请参阅 A mazon EC2 用户指南中的将 IAM 角色附加到实例。

步骤 2：为负载均衡器准备 SSL 证书

当您对负载均衡器侦听器使用 HTTPS 时，必须在负载均衡器上部署 SSL 证书。负载均衡器先使用此证书终止连接，然后解密来自客户端的请求，最后再将请求发送到目标。

准备 SSL 证书

1. 创建私有证书颁发机构 (CA) Certif AWS ice Manager 私有证书颁发机构 (ACM PCA)。有关更多信息，请参阅《Certif ice Manager 私有 AWS 证书颁发机构用户指南》中的创建 CA 的程序。
2. 安装该 CA。有关更多信息，请参阅 Certificate Manager 私有 AWS 证书颁发机构用户指南中的安装根 CA 证书。
3. 申请由该 CA 签名的新私有证书。对于域名，请使用 `*.region.elb.amazonaws.com` 并指定您打算在其中创建负载均衡器的区域。有关更多信息，请参阅 Cer tificate Manager 私有 AWS 证书颁发机构用户指南中的申请私有证书。

步骤 3：创建 Broker Application Load Balancer

创建一个 Application Load Balancer 以均衡前端客户端和 Broker 之间的流量。

创建负载均衡器

1. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。

   在导航窗格中，选择负载均衡器，然后选择创建负载均衡器。对于负载均衡器类型，选择 Application Load Balancer。
2. 对于步骤 1：配置负载均衡器，执行以下操作：

a. 对于名称，输入负载均衡器的描述性名称。

b. 对于模式，选择面向互联网。

c. 对于负载均衡器协议，选择 HTTPS；对于负载均衡器端口，输入 8443。

d. 对于 VPC，选择要使用的 VPC，然后选择该 VPC 中的所有子网。

e. 选择下一步。

3. 对于步骤 2: 配置安全设置，请执行以下操作：

a. 对于证书类型，选择从 ACM 中列表中选择证书。

b. 对于证书名称，选择您以前申请的私有证书。

c. 选择下一步。

4. 对于步骤 3: 配置安全组，创建新的安全组或选择现有的安全组，以允许在前端客户端和 Broker 之间通过 HTTPS 和端口 8443 传输入站和出站流量。

选择下一步。

5. 对于步骤 4: 配置路由，请执行以下操作：

a. 对于目标组，选择新建目标组。

b. 对于名称，输入目标组的名称。

c. 对于目标类型，选择实例。

d. 对于协议，选择 HTTPS。对于端口，输入 8443。对于协议版本，选择 HTTP1。

e. 对于运行状况检查协议，选择 HTTPS；对于路径，输入 `/health`。

f. 选择下一步。

6. 对于步骤 5: 注册目标，选择下一步。

7. 选择创建。

步骤 4：启动 Broker

创建一个初始 Broker 并将其配置为使用负载均衡器，从该 Broker 中创建一个 AMI，然后使用该 AMI 启动其余 Broker。这确保所有 Broker 都配置为使用相同的 CA 和相同的负载均衡器配置。

启动 Broker

1. 启动并配置初始 Broker 主机。有关安装和配置 Broker 的更多信息，请参阅步骤 2：设置 NICE DCV Session Manager Broker。

> **Note**
>
> 由于我们使用 Application Load Balancer，因此，不需要使用 Broker 的自签名证书。

2. 连接到 Broker，使用常用的文本编辑器打开 `/etc/dcv-session-manager-broker/session-manager-broker.properties`，然后执行以下操作：

   a. 在行首放置井号（#）以注释掉 `broker-to-broker-discovery-addresses` 参数。

   b. 对于 `broker-to-broker-discovery-aws-region`，输入您在其中创建 Application Load Balancer 的区域。

   c. 对于 `broker-to-broker-discovery-aws-alb-target-group-arn`，输入与 Broker 负载均衡器关联的目标组的 ARN。

   d. 保存并关闭文件。

3. 停止 Broker 实例。

4. 从停止的 Broker 实例中创建一个 AMI。有关更多信息，请参阅《适用于 Linux 实例的 Amazon EC2 用户指南》中的从实例创建 Linux AMI。

5. 使用该 AMI 启动其余 Broker。

6. 将您创建的实例配置文件分配给所有 Broker 实例。

7. 分配一个安全组，以允许将 Broker 到 Broker 的网络流量以及 Broker 到负载均衡器的网络流量传输到所有 Broker 实例。有关网络端口的更多信息，请参阅 Broker 配置文件。

8. 将所有 Broker 实例注册为 Broker 负载均衡器的目标。有关更多信息，请参阅《Application Load Balancer 用户指南》中的向您的目标组注册目标。

步骤 5：创建 Agent Application Load Balancer

创建一个 Application Load Balancer 以均衡 Agent 和 Broker。

创建负载均衡器

1. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。

   在导航窗格中，选择负载均衡器，然后选择创建负载均衡器。对于负载均衡器类型，选择 Application Load Balancer。

2. 对于步骤 1：配置负载均衡器，执行以下操作：

a. 对于名称，输入负载均衡器的描述性名称。

b. 对于模式，选择面向互联网。

c. 对于负载均衡器协议，选择 HTTPS；对于负载均衡器端口，输入 8445。

d. 对于 VPC，选择要使用的 VPC，然后选择该 VPC 中的所有子网。

e. 选择下一步。

3. 对于步骤 2: 配置安全设置，请执行以下操作：

a. 对于证书类型，选择从 ACM 中列表中选择证书。

b. 对于证书名称，选择您以前申请的私有证书。

c. 选择下一步。

4. 对于步骤 3: 配置安全组，创建新的安全组或选择现有的安全组，以允许在 Agent 和 Broker 之间通过 HTTPS 和端口 8445 传输入站和出站流量。

   选择下一步。

5. 对于步骤 4: 配置路由，请执行以下操作：

a. 对于目标组，选择新建目标组。

b. 对于名称，输入目标组的名称。

c. 对于目标类型，选择实例。

d. 对于协议，选择 HTTPS。对于端口，输入 8445。对于协议版本，选择 HTTP1。

e. 对于运行状况检查协议，选择 HTTPS；对于路径，输入 `/health`。

f. 选择下一步。

6. 对于步骤 5: 注册目标，选择所有 Broker 实例并选择添加到已注册。选择 下一步: 审核。

7. 选择创建。

步骤 6：启动 Agent

创建一个初始 Agent 并将其配置为使用负载均衡器，从该 Agent 中创建一个 AMI，然后使用该 AMI 启动其余 Agent 这确保所有 Agent 都配置为使用相同的负载均衡器配置。

启动 Agent

1. 准备 NICE DCV 服务器。有关更多信息，请参阅 步骤 1：准备 NICE DCV 服务器。

2. 放置在步骤 2：为负载均衡器准备 SSL 证书中创建的 CA 公有密钥的副本。选择或创建一个任何用户都可以读取的目录。CA 公有密钥文件也必须可供任何用户读取。
3. 安装并配置 Agent。有关安装和配置 Agent 的更多信息，请参阅步骤 3：设置 NICE DCV Session Manager Agent。

> **Important**
>
> 在修改 Agent 配置文件时：
>
> - 对于 `broker_host` 参数，输入 Agent 负载均衡器的 DNS
> - 对于 `ca_file` 参数，输入在上一步中创建的 CA 公有密钥文件的路径

4. 配置 NICE DCV 服务器以将 Broker 作为身份验证服务器。有关更多信息，请参阅 步骤 4：配置 NICE DCV 服务器以将 Broker 作为身份验证服务器。

> **Important**
>
> 在修改 NICE DCV 服务器配置文件时：
>
> - 对于 `ca-file` 参数，输入上一步中使用的 CA 公有密钥文件的相同路径
> - 对于 `auth-token-verifier` 参数，将 Agent 负载均衡器的 DNS 作为 *`broker_ip_or_dns`*

5. 停止 Agent 实例。
6. 从停止的 Agent 实例中创建一个 AMI。有关更多信息，请参阅《适用于 Linux 实例的 Amazon EC2 用户指南》中的从实例创建 Linux AMI。
7. 使用该 AMI 启动其余 Agent，并将您创建的实例配置文件分配给所有这些 Agent。
8. 分配一个安全组，以允许将 Agent 到负载均衡器的网络流量传输到所有 Agent 实例。有关网络端口的更多信息，请参阅 Agent 配置文件。

使用标签定位 NICE DCV 服务器

您可以将自定义标签分配给 Session Manager Agent，以帮助识别 Agent 及其关联的 NICE DCV 服务器并对它们进行分类。在创建新的 NICE DCV 会话时，您可以根据分配给相应 Agent 的标签定位一组 NICE DCV 服务器。有关如何根据 Agent 标签定位 NICE DCV 服务器的更多信息，请参阅《Session Manager 开发人员指南》中的 CreateSessionRequests。

标签由标签键和值对组成，可以使用对您的使用案例或环境有意义的任何信息对。您可以选择根据主机的硬件配置标记 Agent。例如，您可以使用 `ram=4GB` 标记具有 4 GB 内存的主机的所有 Agent。或者，您可以根据用途标记 Agent。例如，您可以使用 `purpose=production` 标记生产主机上运行的所有 Agent。

为 Agent 分配标签

1. 通过使用常用的文本编辑器，创建一个新文件并为其指定一个描述性名称，例如 `agent_tags.toml`。文件类型必须为 `.toml`，并且必须使用 TOML 文件格式指定文件内容。
2. 在该文件中，使用 `key=value` 格式将每个新标签键和值对添加到一个新行中。例如：

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

3. 打开 Agent 配置文件（Linux 为 `/etc/dcv-session-manager-agent/agent.conf`，Windows 为 `C:\Program Files\NICE\DCVSessionManagerAgent\conf\agent.conf`）。对于 `tags_folder`，指定标签文件所在目录的路径。

   如果目录包含多个标签文件，在这些文件中定义的所有标签都适用于 Agent。这些文件是按字母顺序读取的。如果多个文件包含具有相同键的标签，则该值被最后读取的文件中的值覆盖。
4. 保存并关闭该文件。
5. 停止并重新启动 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
     ```

配置外部授权服务器

授权服务器是负责对客户端 SDK 和 Agent 进行身份验证和授权的服务器。

默认情况下，Session Manager 将 Broker 作为授权服务器，为客户端 SDK 生成 OAuth 2.0 访问令牌并为 Agent 生成软件声明。如果将 Broker 作为授权服务器，则无需进行额外的配置。

您可以配置 Session Manager 以将 Amazon Cognito 作为外部授权服务器，而不是将 Broker 作为授权服务器。有关 Amazon Cognito 的更多信息，请参阅《Amazon Cognito 开发人员指南》。

将 Amazon Cognito 作为授权服务器

1. 创建一个新的 Amazon Cognito 用户池。有关用户池的更多信息，请参阅《Amazon Cognito 开发人员指南》中的 Amazon Cognito 的功能。

   使用 create-user-pool 命令，并指定池名称以及要在其中创建池的区域。

   在该示例中，我们将池命名为 `dcv-session-manager-client-app` 并在 `us-east-1` 中创建该池。

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

   输出示例

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

   记下 `userPoolId`，您需要在下一步中使用该 ID。

2. 为您的用户池创建一个新的域。使用 create-user-pool-domain 命令，并指定域名以及在上一步中创建的用户池的 `userPoolId`。

在该示例中，域名为 mydomain-544fa30f-c0e5-4a02-8d2a-a3761EXAMPLE，我们在 us-east-1 中创建该域。

```
$ 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
```

输出示例

```
{
    "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": {}
    }
}
```

用户池域的格式如下所示：https://*domain_name*.auth.*region*.amazoncognito.com。在该示例中，用户池域为 https://mydomain-544fa30f-c0e5-4a02-8d2a-a3761EXAMPLE.auth.us-east-1.amazoncognito.com。

3. 创建用户池客户端。使用 create-user-pool-client 命令，并指定您创建的用户池的 userPoolId、客户端的名称以及要在其中创建客户端的区域。此外，还包括 --generate-secret 选项以指定您要为创建的用户池客户端生成密钥。

   在该示例中，客户端名称为 dcv-session-manager-client-app，我们在 us-east-1 区域中创建该客户端。

   ```
   $ 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
   ```

   输出示例

   ```
   {
       "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**
>
> 记下 `ClientId` 和 `ClientSecret`。在开发人员请求用于 API 请求的访问令牌时，您需要为他们提供该信息。

4. 为用户池创建一个新的 OAuth2.0 资源服务器。资源服务器是用于访问受保护的资源的服务器。它处理经过身份验证的访问令牌请求。

   使用 create-resource-server 命令，并指定用户池的 `userPoolId`、资源服务器的唯一标识符和名称、范围以及在其中创建资源服务器的区域。

   在该示例中，我们将 `dcv-session-manager` 作为标识符和名称，并将 `sm_scope` 作为范围名称和描述。

   ```
   $ 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
   ```

   输出示例

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

```
}
```

5. 更新用户池客户端。

   使用 update-user-pool-client 命令。指定用户池的 `userPoolId`、用户池客户端的 `ClientId` 以及区域。对于 `--allowed-o-auth-flows`，指定 `client_credentials` 以指示客户端应使用客户端 ID 和客户端密钥组合从令牌终端节点中获取访问令牌。对于 `--allowed-o-auth-scopes`，指定资源服务器标识符和范围名称，如下所示：*`resource_server_identifier`*/*`scope_name`*。包括 `--allowed-o-auth-flows-user-pool-client` 以表示允许客户端在与 Cognito 用户池交互时采用 OAuth 协议。

```
$ 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
```

   输出示例

```
{
    "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**
>
> 用户池现在已准备好提供访问令牌并进行身份验证。在该示例中，授权服务器的 URL 为 https://cognito-idp.*us-east-1*.amazonaws.com/*us-east-1_QLEXAMPLE*/.well-known/jwks.json。

6. 测试配置。

```
$ 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"
```

输出示例

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

7. 使用 register-auth-server 命令注册外部授权服务器以供 Broker 使用。

```
$ 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
```

开发人员现在可以使用服务器请求访问令牌。在请求访问令牌时，请提供此处生成的客户端 ID、客户端密钥和服务器 URL。有关请求访问令牌的更多信息，请参阅《NICE DCV Session Manager 开发人员指南》中的 Create get an access token and make an API request。

配置 Broker 持久性

Session Manager Broker 支持与外部数据库集成在一起。Session Manager 可以在外部数据库中持久保留状态数据和密钥，以便以后使用这些数据和密钥。事实上，Broker 数据分布在集群上，如果一个主机需要重新引导或终止了集群，则很容易丢失数据。在启用该功能后，您可以添加和删除 Broker 节点。此外，您可以停止并重新启动集群，而无需重新生成密钥或丢失有关打开或关闭的 NICE DCV 服务器的信息。

可以将以下类型的信息设置为持久保留：

- 用于设置会话以与客户端建立连接的密钥
- 正在传输的会话数据
- NICE DCV 服务器状态

NICE DCV Session Manager 支持 DynamoDB、MariaDB 和 MySQL 数据库。您必须设置和管理这些数据库之一才能使用该功能。如果在 Amazon EC2 上托管您的 Broker 计算机，我们建议将 DynamoDB 作为外部数据库，因为它不需要进行任何额外的设置。

> **Note**
>
> 在运行外部数据库时，您可能会产生额外费用。要查看有关 DynamoDB 定价的信息，请参阅预置容量的定价。

配置 Broker 以在 DynamoDB 上持久保留数据

配置 Broker 以开始在 DynamoDB 上存储其数据：

1. 使用常用的文本编辑器打开 `/etc/dcv-session-manager-broker/session-manager-broker.properties` 并进行以下编辑：
   - Set `enable-persistence = true`
   - Set `persistence-db = dynamodb`
   - 对于 `dynamodb-region`，指定您希望存储包含 Broker 数据的表的 &aws; 区域。有关支持的区域列表，请参阅 DynamoDB service endpoints。
   - 对于 `dynamodb-table-rcu`，指定每个表支持的读取容量单位 (RCU) 数量。有关 RCU 的更多信息，请参阅 DynamoDB 预置容量。

- 对于 dynamodb-table-wcu，指定每个表支持的写入容量单位 (WCU) 数量。有关 WCU 的更多信息，请参阅 DynamoDB 预置容量。
- 对于 dynamodb-table-name-prefix，指定添加到每个 DynamoDB 表的前缀（用于区分使用同一账户的多个 Broker 集群）。仅允许使用字母数字字符、圆点、短划线和下划线。

2. 停止集群中的所有 Broker。对于每个 Broker，运行以下命令：

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

3. 确保集群中的所有 Broker 已停止，然后重新启动所有 Broker。运行以下命令以启动每个 Broker：

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

Broker 主机必须有权调用 DynamoDB API。在 Amazon EC2 实例上，凭证是使用 Amazon EC2 元数据服务自动检索的。如果需要指定不同的凭证，您可以使用支持的凭证检索技术之一（例如 Java 系统属性或环境变量）设置这些凭证。有关更多信息，请参阅 Supplying and Retrieving &aws; Credentials。

配置 Broker 以在 MariaDB/MySQL 上持久保留数据

> **Note**
>
> /etc/dcv-session-manager-broker/session-manager-broker.properties 文件包含敏感数据。默认情况下，其写入访问权限限制为根用户，其读取访问权限限制为根用户和运行 Broker 的用户。默认情况下，这是 dcvsmbroker 用户。Broker 在启动时检查文件是否具有预期的权限。

配置 Broker 以开始在 MariaDB/MySQL 上持久保留其数据：

1. 使用常用的文本编辑器打开 /etc/dcv-session-manager-broker/session-manager-broker.properties 并进行以下编辑：

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

在该配置中，<db_endpoint> 是数据库终端节点，<db_port> 是数据库端口，<db_name> 是数据库名称。

- 对于 `jdbc-user`，指定有权访问数据库的用户的名称。
- 对于 `jdbc-password`，指定有权访问数据库的用户的密码。

2. 停止集群中的所有 Broker。对于每个 Broker，运行以下命令：

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

3. 确保集群中的所有 Broker 已停止，然后重新启动所有 Broker。对于每个 Broker，运行以下命令：

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

与 NICE DCV Connection Gateway 集成

NICE DCV Connection Gateway 是一个可安装的软件包，使用户能够通过到 LAN 或 VPC 的单个接入点访问一组 NICE DCV 服务器。

如果您的基础设施包括可通过 NICE DCV Connection Gateway 访问的 NICE DCV 服务器，您可以配置 Session Manager 以集成 NICE DCV Connection Gateway。通过执行下一节中概述的步骤，Broker 将充当 Connection Gateway 的会话解析器。换句话说，Broker 将公开一个额外的 HTTP 终端节点。Connection Gateway 对该终端节点进行 API 调用，以检索将 NICE DCV 连接路由到 Broker 选择的主机所需的信息。

将 Session Manager Broker 设置为 NICE DCV Connection Gateway 的会话解析器

Session Manager Broker 端

1. 使用常用的文本编辑器打开 `/etc/dcv-session-manager-broker/session-manager-broker.properties` 并应用以下更改：

   - Set `enable-gateway = true`
   - 将 `gateway-to-broker-connector-https-port` 设置为空闲 TCP 端口（默认为 8447）
   - 将 `gateway-to-broker-connector-bind-host` 设置为 Broker 为 NICE DCV Connection Gateway 连接绑定的主机的 IP 地址（默认值为 0.0.0.0）

2. 然后，运行以下命令以停止并重新启动 Broker：

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

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

3. 检索 Broker 的自签名证书副本，并将其放置在您的用户目录中。

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

在下一步中安装 NICE DCV Connection Gateway 时，您需要使用该证书。

NICE DCV Connection Gateway 端

- 请按照 NICE DCV Connection Gateway 文档中的相应小节进行操作。

  由于 NICE DCV Connection Gateway 对 Broker 进行 HTTP API 调用，如果 Broker 使用自签名证书，您需要将 Broker 证书复制到 NICE DCV Connection Gateway 主机（在上一步中检索）并在 NICE DCV Connection Gateway 配置的 `[resolver]` 部分中设置 `ca-file` 参数。

可选 - 启用 TLS 客户端身份验证

在完成上一步后，Session Manager 和 Connection Gateway 可以通过安全通道进行通信，其中 Connection Gateway 可以验证 Session Manager Broker 的身份。如果您要求 Session Manager Broker 还在建立安全通道之前验证 Connection Gateway 身份，您需要按照下一节中的步骤启用 TLS 客户端身份验证功能。

> **Note**
>
> 如果 Session Manager 位于负载均衡器后面，则无法使用具有 TLS 连接终止的负载均衡器（例如 Application Load Balancer (ALB) 或 Gateway Load Balancer (GLB)）启用 TLS 客户端身份验证。仅支持没有 TLS 终止的负载均衡器，例如 Network Load Balancer (NLB)。如果您使用 ALB 或 GLB，您可以强制要求仅特定的安全组连接到负载均衡器，从而确保达到额外的安全级别；有关安全组的更多信息，请参阅此处：您的 VPC 的安全组

Session Manager Broker 端

1. 要为 Session Manager Broker 和 NICE DCV Connection Gateway 之间的通信启用 TLS 客户端身份验证，请按照以下步骤进行操作：
2. 运行以下命令以生成所需的密钥和证书：该命令的输出指示您生成凭证的文件夹以及用于创建 TrustStore 文件的密码。

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

3. 将 NICE DCV Connection Gateway 的私有密钥和自签名证书的副本放置在您的用户目录中。在下一步中在 NICE DCV Connection Gateway 中启用 TLS 客户端身份验证时，您需要使用这些密钥和证书。

```
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. 然后，使用常用的文本编辑器打开 /etc/dcv-session-manager-broker/session-manager-broker.properties，并执行以下操作：
   - 将 `enable-tls-client-auth-gateway` 设置为 `true`
   - 将 `gateway-to-broker-connector-trust-store-file` 设置为上一步中创建的 TrustStore 文件的路径
   - 将 `gateway-to-broker-connector-trust-store-pass` 设置为用于在上一步中创建 TrustStore 文件的密码
5. 然后，运行以下命令以停止并重新启动 Broker：

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

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

NICE DCV Connection Gateway 端

- 请按照 NICE DCV Connection Gateway 文档中的相应小节进行操作。
  - 在设置 `[resolver]` 部分中的 `cert-file` 参数时，使用您在上一步中复制的证书文件的完整路径

- 在设置 [resolver] 部分中的 cert-key-file 参数时，使用您在上一步中复制的密钥文件的完整路径

NICE DCV Session Manager NICE DCV 服务器 - DNS 映射

NICE DCV Connection Gateway 需要使用 NICE DCV 服务器的 DNS 名称，才能连接到 DCV 服务器实例。本节说明了如何定义包含每个 DCV 服务器与其关联 DNS 名称之间的映射的 JSON 文件。

文件结构

该映射包含具有以下字段的 JSON 对象列表：

```
[
 {
  “ServerIdType”: "Ip",
  “ServerId”: "192.168.0.1",
  “DnsNames”:
  {
   “InternalDnsName”: "internal"
  }
 },
 ...
]
```

其中：

ServerIdType:

指定值表示的 ID 的类型；目前，可用的值为 ipAddress、agentServerId 和 instanceId：

Ip:

适用于 Amazon EC2 和本地基础设施；系统管理员可以使用 ifconfig (Linux) 或 ipconfig (Windows) 命令快速进行检索。在 DescribeServers API 响应中也提供了该信息。

Id:

适用于 Amazon EC2 和本地基础设施；每次主机名或 IP 地址发生变化时，Session Manager Agent 都会创建新的 UUID。在 DescribeServers API 响应中提供了该信息。

`Host.Aws.Ec2InstanceId:`

仅适用于 Amazon EC2 实例，它唯一地标识一个计算机；在实例重新引导后，它不会发生变化。可以在主机上访问 http://169.254.169.254/latest/meta-data/instance-id 以进行检索。在 DescribeServers API 响应中也提供了该信息。

`ServerId:`

指定类型的 ID，用于唯一地标识网络中的每个 NICE DCV 服务器。

`DnsNames:`

包含与 NICE DCV 服务器关联的 DNS 名称的对象，该对象将包含：

`InternalDnsNames:`

NICE DCV Connection Gateway 用于连接到实例的 DNS 名称。

请使用 Session Manager Broker CLI 命令 `register-server-dns-mapping` 从文件中加载映射（命令页面参考：register-server-dns-mapping），并使用 `describe-server-dns-mappings` 列出当前在 Session Manager Broker 中加载的映射（命令页面参考：describe-server-dns-mappings）。

持久性

我们强烈建议您启用 Session Manager Broker 的持久性功能，以防止多个 Broker 或整个集群发生故障时映射丢失。有关启用数据持久性的更多信息，请参阅配置 Broker 持久性。

与 Amazon CloudWatch 集成

对于在 Amazon EC2 实例上运行的 Broker 以及在本地主机上运行的 Broker，Session Manager 支持与 Amazon CloudWatch 集成在一起。

Amazon CloudWatch 可实时监控您的亚马逊云科技 (AWS) 资源以及您在 AWS 上运行的应用程序。您可以使用 CloudWatch 收集和跟踪指标，这些指标是您可衡量的相关资源和应用程序的变量。有关更多信息，请参阅 Amazon CloudWatch 用户指南。

您可以配置 Session Manager Broker 以将以下指标数据发送到 Amazon CloudWatch：

- `Number of DCV servers` - Broker 管理的 DCV 服务器数量。
- `Number of ready DCV servers` - Broker 管理的处于 `READY` 状态的 DCV 服务器数量。
- `Number of DCV sessions` - Broker 管理的 DCV 会话数量。

- `Number of DCV console sessions` - Broker 管理的 DCV 控制台会话数量。
- `Number of DCV virtual sessions` - Broker 管理的 DCV 虚拟会话数量。
- `Heap memory used` - Broker 使用的堆内存量。
- `Off-heap memory used` - Broker 使用的堆外内存量。
- `Describe sessions request time` - 完成 DescribeSessions API 请求所需的时间。
- `Delete sessions request time` - 完成 DeleteSessions API 请求所需的时间。
- `Create sessions request time` - 完成 CreateSessions API 请求所需的时间。
- `Get session connection data request time` - 完成 GetSessionConnectionData API 请求所需的时间。
- `Update session permissions sequest time` - 完成 UpdateSessionPermissions API 请求所需的时间。

配置 Broker 以将指标数据发送到 Amazon CloudWatch

1. 使用常用的文本编辑器打开 `/etc/dcv-session-manager-broker/session-manager-broker.properties`，然后执行以下操作：

   - 将 `enable-cloud-watch-metrics` 设置为 `true`
   - 对于 `cloud-watch-region`，指定要在其中收集指标数据的区域。

   > **Note**
   >
   > 如果您的 Broker 在 Amazon EC2 实例上运行，则该参数是可选的。区域是从实例元数据服务 (IMDS) 中自动检索的。如果您在本地主机上运行 Broker，则该参数是必需的。

2. 停止并重新启动 Broker。

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

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

Broker 主机还必须具有调用 `cloudwatch:PutMetricData` API 的权限。可以使用支持的凭证检索方法之一检索 AWS 凭证。有关更多信息，请参阅 Supplying and Retrieving AWS Credentials。

升级 NICE DCV Session Manager

以下主题介绍了如何升级 Session Manager。

> **Note**
>
> 我们强烈建议您在升级 Session Manager Broker 之前升级所有 Session Manager Agent，以避免引入新功能时出现不兼容问题。

主题

升级 NICE DCV Session Manager Agent

Linux host

> **Note**
>
> 以下说明适用于在 64 位 x86 主机上安装 Agent。要在 64 位 ARM 主机上安装 Agent，对于 Amazon Linux、RHEL 和 Centos，请将 *x86_64* 替换为 `aarch64`，对于 Ubuntu，将 *amd64* 替换为 `arm64`。

更新 Linux 主机上的 Agent

1. 运行以下命令以停止 Agent。

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

2. 下载安装软件包。

   - Amazon Linux 2、RHEL 7.x 和 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 和 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. 安装 软件包。

- Amazon Linux 2、RHEL 7.x 和 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 和 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. 运行以下命令以启动 Agent。

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

Windows host

更新 Windows 主机上的 Agent

1. 停止 Agent 服务。在命令提示符下运行以下命令。

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

2. 下载 Agent 安装程序。
3. 运行安装程序。在欢迎屏幕上，选择 Next。
4. 在 EULA 屏幕上，仔细阅读许可协议；如果同意，请选择 I accept the terms 并选择 Next。
5. 要开始安装，请选择 Install。
6. 重新启动 Agent 服务。在命令提示符下运行以下命令。

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

升级 NICE DCV Session Manager Broker

升级 Broker

1. 连接到您打算在其中升级 Broker 的主机。
2. 停止 Broker 服务。

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

3. 下载安装软件包。

- Amazon Linux 2、RHEL 7.x 和 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 和 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. 安装 软件包。

- Amazon Linux 2、RHEL 7.x 和 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 和 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. 启动 Broker 服务，并确保该服务在每次实例启动时自动启动。

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

Broker CLI 参考

本节介绍了如何使用 Broker 命令行界面 (CLI) 命令。

如果您使用外部身份验证服务器生成 OAuth 2.0 访问令牌，请使用以下命令：

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

如果您将 Session Manager Broker 作为 OAuth 2.0 身份验证服务器，请使用以下命令。

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

可以使用以下命令管理 Session Manager Agent。

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

可以使用以下命令管理 DCV 服务器 - DNS 名称映射文件。

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

register-auth-server

注册外部身份验证服务器以与 Broker 一起使用。

默认情况下，Session Manager 将 Broker 作为身份验证服务器以生成 OAuth 2.0 访问令牌。如果将 Broker 作为身份验证服务器，则无需进行额外的配置。

不过，如果您选择使用外部身份验证服务器（例如 Active Directory 或 Amazon Cognito），您必须使用该命令注册外部身份验证服务器。

主题

语法

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

选项

`--url`

要使用的外部身份验证服务器的 URL。您必须将 `.well-known/jwks.json` 附加到身份验证服务器 URL 中。

类型：字符串

必需：是

示例

以下示例使用 URL `https://my-auth-server.com/` 注册外部身份验证服务器。

命令

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

输出

```
Jwk url registered.
```

list-auth-servers

列出已注册的外部身份验证服务器。

主题

语法

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

输出

Urls

注册的外部身份验证服务器的 URL。

示例

以下示例列出所有已注册的外部身份验证服务器。

命令

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

输出

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

unregister-auth-server

取消注册外部身份验证服务器。在取消注册外部身份验证服务器后，无法再使用该服务器生成 OAuth 2.0 访问令牌。

主题

语法

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

选项

`--url`

要取消注册的外部身份验证服务器的 URL。您必须将 `.well-known/jwks.json` 附加到身份验证服务器 URL 中。

类型：字符串

必需：是

输出

`Url`

取消注册的外部身份验证服务器的 URL。

示例

以下示例使用 URL `https://my-auth-server.com/` 注册外部身份验证服务器。

命令

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

输出

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

register-api-client

在 Broker 中注册 Session Manager 客户端并生成客户端凭证，客户端可以使用这些凭证检索 OAuth 2.0 访问令牌（需要使用该令牌以发出 API 请求）。

> **Important**
>
> 确保将这些凭证存储在安全的地方。以后无法恢复这些凭证。

只有在将 Broker 作为 OAuth 2.0 身份验证服务器时，才会使用该命令。

主题

语法

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

选项

`--name`

用于标识 Session Manager 客户端的唯一名称。

类型：字符串

必需：是

输出

`client-id`

Session Manager 客户端在检索 OAuth 2.0 访问令牌时使用的唯一客户端 ID。

`client-password`

Session Manager 客户端在检索 OAuth 2.0 访问令牌时使用的密码。

示例

以下示例注册一个名为 `my-sm-client` 的客户端。

命令

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

输出

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

describe-api-clients

列出已在 Broker 中注册的 Session Manager 客户端。

主题

语法

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

输出

`name`

Session Manager 客户端的唯一名称。

`id`

Session Manager 客户端的唯一 ID。

`active`

指示 Session Manager 客户端的状态。如果客户端处于活动状态，则值为 `true`；否则，值为 `false`。

示例

以下示例列出注册的 Session Manager 客户端。

命令

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

输出

```
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

停用注册的 Session Manager 客户端。停用的 Session Manager 客户端无法再使用其凭证检索 OAuth 2.0 访问令牌。

主题

语法

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

选项

`--client -id`

要停用的 Session Manager 客户端的客户端 ID。

类型：字符串

必需：是

示例

以下示例停用客户端 ID 为 `f855b54b-40d4-4769-b792-b727bEXAMPLE` 的 Session Manager 客户端。

命令

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

输出

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

renew-auth-server-api-key

续订 Broker 对提供给 Session Manager 客户端的 OAuth 2.0 访问令牌进行签名时使用的公有密钥和私有密钥。如果您续订这些密钥，您必须向开发人员提供新的私有密钥，因为需要使用该密钥以发出 API 请求。

主题

语法

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

示例

以下示例续订公有密钥和私有密钥。

命令

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

输出

```
Keys renewed.
```

generate-software-statement

生成软件声明。

Agent 必须在 Broker 中注册才能进行通信。Agent 需要具有软件声明才能在 Broker 中注册。在 Agent 具有软件声明后，它可以使用 OAuth 2.0 动态客户端注册协议自动在 Broker 中进行注册。在 Broker 中注册后，Agent 将收到用于在 Broker 中进行身份验证的客户端 ID 和客户端密钥。

在首次安装时，Broker 和 Agent 收到并使用默认软件声明。您可以继续使用默认软件声明，也可以选择生成新的软件声明。如果生成新的软件声明，您必须将软件声明放入 Agent 上的新文件中，然后将文件路径添加到 `agent.conf` 文件的 `agent.software_statement_path` 参数中。在完成该操作后，停止并重新启动 Agent，以使其可以使用新的软件声明在 Broker 中注册。

主题

语法

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

输出

`software-statement`

软件声明。

示例

以下示例生成一个软件声明。

命令

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

输出

```
software-statement:
 ewogICJpZCIgOiAiYjc1NTVhN2QtNWI0MC00OTJhLWJjOTUtNmUzOWNhYzkxMDcxIiwKICAiYWN0aXZlIiA6IHRydWUsCi
```

describe-software-statements

描述现有的软件声明。

主题

语法

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

输出

software-statement

软件声明。

issued-at

生成软件的日期和时间。

is-active

软件声明的当前状态。如果软件声明处于活动状态，则为 `true`，否则为 `false`。

示例

以下示例生成一个软件声明。

命令

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

输出

```
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

停用软件声明。在您停用软件声明后，则无法再使用该声明注册 Agent。

主题

语法

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

选项

`--software-statement`

要停用的软件声明。

类型：字符串

必需：是

示例

以下示例停用一个软件声明。

命令

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

输出

```
Software statement
 EXAMPLEpZCIgOiAiYjc1NTVhN2QtNWI0MC00OTJhLWJjOTUtNmUzOWNhYzkxMDcxIiwKICAiaXNEXAMPLEQiIDogMTU5Nj
 deactivated
```

describe-agent-clients

描述在 Broker 中注册的 Agent。

主题

语法

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

输出

`name`

Agent 的名称。

`id`

Agent 的唯一 ID。

`active`

Agent 状态。如果 Agent 处于活动状态；则为 `true`，否则为 `false`。

示例

以下示例描述了 Agent。

命令

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

输出

```
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

从 Broker 中取消注册 Agent。

主题

语法

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

选项

`--client-id`

要取消注册的 Agent 的 ID。

类型：字符串

必需：是

示例

以下示例取消注册一个 Agent。

命令

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

输出

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

register-server-dns-mappings

注册来自 JSON 文件的 DCV 服务器 - DNS 名称映射。

语法

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

选项

`--file-path`

包含 DCV 服务器 - DNS 名称映射的文件的路径。

类型：字符串

必需：是

示例

以下示例注册来自 /tmp/mappings.json 文件的 DCV 服务器 - DNS 名称映射。

命令

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

输出

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

describe-server-dns-mappings

描述当前可用的 DCV 服务器 - DNS 名称映射。

语法

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

输出

`serverIdType`

服务器 ID 的类型。

`serverId`

服务器的唯一 ID。

`dnsNames`

内部和外部 DNS 名称

`internalDnsNames`

内部 DNS 名称

`externalDnsNames`

外部 DNS 名称

示例

以下示例列出注册的 DCV 服务器 - DNS 名称映射。

命令

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

输出

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

配置文件参考

本节提供了有关 Agent 和 Broker 配置文件的信息。

主题

Broker 配置文件

Broker 配置文件 (`/etc/dcv-session-manager-broker/session-manager-broker.properties`) 包含一些参数，可以配置这些参数以自定义 Session Manager 功能。您可以使用常用的文本编辑器编辑配置文件。

> **Note**
>
> `/etc/dcv-session-manager-broker/session-manager-broker.properties` 文件包含敏感数据。默认情况下，其写入访问权限限制为根用户，其读取访问权限限制为根用户和运行 Broker 的用户。默认情况下，这是 `dcvsmbroker` 用户。Broker 在启动时检查文件是否具有预期的权限。

下表列出了 Broker 配置文件中的参数。

参数名称	必填	默认值	描述
`broker-java-home`	否		指定 Broker 将使用的 Java 主目录的路径，而不是系统默认目录的路径。如果已设置，Broker 将在启动时使用 `<broker-java-home>/bin/java` 。 提示：Broker 需要使用 Java Runtime Environment

参数名称	必填	默认值	描述
			11；如果缺少该环境，在成功安装 Broker 时，它将其作为依赖项进行安装。如果版本 11 未设置为默认 Java 环境，可以使用以下命令获取其主目录： `$ sudo alternatives --display java`
sessions-screenshot-max-width	否	160	指定使用 GetSessionScreenshots API 获取的会话屏幕截图的最大宽度（以像素为单位）。
sessions-screenshot-max-height	否	100	指定使用 GetSessionScreenshots API 获取的会话屏幕截图的最大高度（以像素为单位）。
sessions-screenshot-format	否	png	使用 GetSessionScreenshots API 获取的会话屏幕截图的图像文件格式。

参数名称	必填	默认值	描述
create-sessions-queue-max-size	否	1000	可以排入队列的最大未完成 CreateSessions API 请求数量。在队列已满时，将拒绝新的未完成请求。
create-sessions-queue-max-time-seconds	否	1800	未完成的 CreateSessions API 请求可以保留在队列中的最长时间（以秒为单位）。如果无法在指定时间内完成请求，请求将失败。
sessionmanager-working-path	是	/tmp	指定 Broker 将运行所需的文件写入到的目录路径。只能由 Broker 访问该目录。
enable-authorization-server	是	true	指定 Broker 是否为用于为客户端 API 生成 OAuth 2.0 访问令牌的身份验证服务器。

参数名称	必填	默认值	描述
enable-authorization	是	true	启用或禁用客户端授权。如果启用客户端授权，则客户端 API 在发出 API 请求时必须提供访问令牌。如果禁用客户端授权，客户端 API 可以在没有访问令牌的情况下发出请求。
enable-agent-authorizatio	是	true	启用或禁用 Agent 授权。如果启用 Agent 授权，Agent 在与 Broker 通信时必须提供访问令牌。
delete-session-duration-hours	否	1	指定在经过多少小时后，删除的会话变得不可见，并且 DescribeSession API 调用不再返回这些会话。
connects session-token-duration-minutes	否	60	指定 ConnectSession 令牌保持有效的分钟数。

参数名称	必填	默认值	描述
client-to-broker-connect https-port	是	8443	指定 Broker 侦听客户端连接的 HTTPS 端口。
client-to-broker-connect bind-host	否	0.0.0.0	指定 Broker 为客户端连接绑定的主机的 IP 地址。
client-to-broker-connect key-store-file	是		指定用于 TLS 客户端连接的密钥存储。

参数名称	必填	默认值	描述
client-to-broker-connector-key-store-pass	是		指定密钥存储密码。
agent-to-broker-connector-https-port	是	8445	指定 Broker 侦听 Agent 连接的 HTTPS 端口。
agent-to-broker-connector-bind-host	否	0.0.0.0	指定 Broker 为 Agent 连接绑定的主机的 IP 地址。

参数名称	必填	默认值	描述
agent-to-broker-connector-key-store-file	是		指定用于 TLS Agent 连接的密钥存储。
agent-to-broker-connector-key-store-pass	是		指定密钥存储密码。
broker-to-broker-port	是	47100	指定用于 Broker 到 Broker 连接的端口。
broker-to-broker-bind-host	否	0.0.0.0	指定 Broker 为 Broker 到 Broker 连接绑定的主机的 IP 地址。

参数名称	必填	默认值	描述
broker-to-broker-discovery-port	是	47500	指定 Broker 用于发现对方的端口。
broker-to-broker-discovery-address	否		以 *ip_address* :*port* 格式指定一组 Broker 中的其他 Broker 的 IP 地址和端口。如果有多个 Broker，请使用逗号分隔这些值。如果指定 broker-to-broker-discovery-multicast-group 、broker-to-broker-discovery-multicast-port、broker-to-broker-discovery-AWS-region 或 broker-to-broker-discovery-AWS-alb-target-group-arn ，则省略该参数。

参数名称	必填	默认值	描述
broker-to-broker-discovery-multicast-group	否		指定用于 Broker 到 Broker 发现的组播组。如果指定 broker-to-broker-discovery-addresses 、broker-to-broker-discovery-aws-region 或 broker-to-broker-discovery-AWS-alb-target-group-arn ，则省略该参数。
broker-to-broker-discovery-multicast-port	否		指定用于 Broker 到 Broker 发现的组播端口。如果指定 broker-to-broker-discovery-addresses 、broker-to-broker-discovery-AWS-region 或 broker-to-broker-discovery-AWS-alb-target-group-arn ，则省略该参数。

参数名称	必填	默认值	描述
broker-to-broker-discovery-AWS-region	否		指定用于 Broker 到 Broker 发现的 Application Load Balancer 的 AWS 区域。如果指定 broker-to-broker-discovery-multicast-group 、broker-to-broker-discovery-multicast-port 或 broker-to-broker-discovery-addresses ，则省略该参数。
broker-to-broker-discovery-AWS-alb-target-group-arn	否		用于 Broker 到 Broker 发现的 Application Load Balancer 目标组用户的 ARN。如果指定 broker-to-broker-discovery-multicast-group 、broker-to-broker-discovery-multicast-port 或 broker-to-broker-discovery-addresses ，则省略该参数。

参数名称	必填	默认值	描述
broker-to-broker-distributed-memory-max-size-mb	否	4096	指定单个 Broker 用于存储 NICE DCV 会话数据的最大堆外内存量。
broker-to-broker-key-store-file	是		指定用于 TLS Broker 连接的密钥存储。
broker-to-broker-key-store-pass	是		指定密钥存储密码。
enable-cloud-watch-metrics	否	false	启用或禁用 Amazon CloudWatch 指标。如果启用 CloudWatch 指标，您可能需要指定 cloud-watch-region 值。

参数名称	必填	默认值	描述
cloud-watch-region	否	只有在 enable-cloud-watch-metrics 设置为 true 时才需要。如果 Broker 安装在 Amazon EC2 实例上，则会从 IMDS 中检索区域。	在其中发布 CloudWatch 指标的 AWS 区域。
max-api-requests-per-second	否	1000	指定 Broker API 在受到限制之前每秒可以处理的最大请求数。
enable-throttling-forwarded-for-header	否	false	如果设置为 true，则限制功能从 X-Forwared-For 标头（如果存在）中检索调用方 IP。
create-sessions-number-of-retries-on-failure	否	2	指定 NICE DCV 服务器主机上的创建会话请求失败后执行的最大重试次数。设置为 0 表示从不在失败时执行重试。

参数名称	必填	默认值	描述
autorunfile-arguments-max-size	否	50	指定可以传递给自动运行文件的最大参数数量。
autorunfile-arguments-max-argumentlength	否	150	指定每个自动运行文件参数的最大长度（字符数）。
enable-persister	是	false	如果设置为 true，Broker 状态数据将持久保留在外部数据库中。
persistence-db	否	只有在 enable-persistence 设置为 true 时才需要。	指定用于持久保留数据的数据库。唯一支持的值是：dynamodb 和 mysql。
dynamodbregion	否	只有在 enable-persistence 设置为 true 并且 persistence-db 设置为 dynamodb 时才需要。	指定在其中创建和访问 DynamoDB 表的区域。

参数名称	必填	默认值	描述
dynamoc table-rcu	否	只有在 enable-persistence 设置为 true 并且 persistence-db 设置为 dynamodb 时才需要。	指定每个 DynamoDB 表的读取容量单位 (RCU)。有关 RCU 的更多信息，请参阅预置容量的定价。
dynamoc table-wcu	否	只有在 enable-persistence 设置为 true 并且 persistence-db 设置为 dynamodb 时才需要。	指定每个 DynamoDB 表的写入容量单位 (WCU)。有关 WCU 的更多信息，请参阅预置容量的定价。
dynamoc table-name-prefix	否	只有在 enable-persistence 设置为 true 并且 persistence-db 设置为 dynamodb 时才需要。	指定添加到每个 DynamoDB 表的前缀（用于区分使用同一 AWS 账户的多个 Broker 集群）。仅允许使用字母数字字符、圆点、短划线和下划线。
jdbc-connection-url	否	只有在 enable-persistence 设置为 true 并且 persistence-db 设置为 mysql 时才需要。	指定 MariaDB/MySQL 数据库的连接 URL；它包含终端节点和数据库名称。该 URL 应采用以下格式： `jdbc:mysql://<db_endpoint>:<db_port>/<db_name>?createDatabaseIfNotExist=true` 其中 <db_endpoint> 是 MariaDB/MySQL 数据库终端节点，<db_port> 是数据库端口，<db_name> 是数据库名称。

参数名称	必填	默认值	描述
jdbc-user	否	只有在 enable-persistence 设置为 true 并且 persistence-db 设置为 mysql 时才需要。	指定有权访问 MariaDB/MySQL 数据库的用户的名称。
jdbc-password	否	只有在 enable-persistence 设置为 true 并且 persistence-db 设置为 mysql 时才需要。	指定有权访问 MariaDB/MySQL 数据库的用户的密码。
seconds-before-deleting-unreachable-dcv-server	否	1800	指定从系统中删除无法访问的服务器之前经过的秒数。

代理配置文件

Agent 配置文件（Linux 为 `/etc/dcv-session-manager-agent/agent.conf`，Windows 为 `C:\Program Files\NICE\DCVSessionManagerAgent\conf\agent.conf`）包含一些参数，可以配置这些参数以自定义 Session Manager 功能。您可以使用常用的文本编辑器编辑配置文件。

下表列出了 Agent 配置文件中的参数。

参数名称	必填	默认值	描述
agent.broker_hos	是		指定 Broker 主机的 DNS 名称。
agent.broker_por	是	8445	指定用于与 Broker 通信的端口。
agent.file	否		只有在 tls_strict 设置为 true 时才需要。指定验证 TLS 证书所需的证书 (.pem) 文件的路径。将自签名证书从 Broker 复制到 Agent。
agent.t_folde	否	• /var/lib/dcv-session-manager-agent/init (Linux)	指定主机服务器上的文件夹的路径，用于存储在创建 NICE DCV 服务器会话时允许对其进行初始化的自定义脚本。您必须指定绝对路径。使用 CreateSessions API 的 InitFile 请求参数的用户必须能够访问该文件夹，并且可以执行这些文件。
agent._strict	否	true	指示是否应使用严格 TLS 验证。
agent.tware_stement_th	否		只有在未使用默认软件声明时才需要。指定软件声明文件的路径。有关更多信息，请参阅generate-software-statement。

参数名称	必填	默认值	描述
agent.t s_folde	否	• /etc/dcv-session-manager-agent (Linux) • C:\Program Files\NICE\DCVSessionManagerAgent\conf\tags (Windows)	指定标签文件所在文件夹的路径。有关更多信息，请参阅使用标签定位 NICE DCV 服务器。
agent.a orun_fo er	否	• /var/lib/dcv-session-manager-agent/autorun (Linux) • C:\ProgramData\NICE\DcvSessionManagerAgent\autorun (Windows)	指定主机服务器上的文件夹的路径，用于存储在会话启动时允许自动运行的脚本和应用程序。您必须指定绝对路径。使用 CreateSessions API 的 AutorunFile 请求参数的用户必须能够访问该文件夹，并且可以执行这些文件。
agent.m _virtua sessior	否	-1（无限制）	可以使用 NICE DCV Session Manager 在 NICE DCV 服务器上创建的最大虚拟会话数量。
agent.m _concur nt_sess ns_per_ er	否	1	单个用户可以使用 NICE DCV Session Manager 在 NICE DCV 服务器上创建的最大虚拟会话数量。

参数名称	必填	默认值	描述
agent.broker_update_interval	否	30	指定将更新的数据发送到 Broker 之前等待的秒数。发送的数据包括 NICE DCV 服务器和主机状态以及更新的会话信息。较低的值使 Session Manager 更早地发现运行 Agent 的系统上发生的变化，但会增加系统负载和网络流量。较高的值会降低系统和网络负载，但 Session Manager 对系统变化的响应速度变慢，因此不建议使用高于 120 的值。
log.level	否	info	指定日志文件的详细程度。提供了以下详细程度等级： • error - 提供最少的详细信息。仅包括错误。 • warning - 包括错误和警告。 • info - 默认详细程度。包括错误、警告和信息消息。 • debug - 提供最多的详细信息。提供有助于调试问题的详细信息。

参数名称	必填	默认值	描述
log.dir tory	否	• /var/log/dcv-session-manager-agent/ (Linux) • C:\ProgramData\NICE\DCVSessionManagerAgent\log (Windows)	指定在其中创建日志文件的目录。
log.rot ion	否	daily	指定日志文件轮换。有效值为： • hourly - 每小时轮换一次日志文件。 • daily - 每天轮换一次日志文件。
log.max f ile-size	否	10485760	在日志文件达到指定大小（以字节为单位）时，轮换日志文件。将创建新的日志文件，并将后续日志事件放置在新文件中。
log.rot e	否	9	轮换中保留的最大日志文件数量。每次发生轮换并达到该数量时，将删除最早的日志文件。

NICE DCV Session Manager 发行说明和文档历史记录

该页面提供 NICE DCV Session Manager 的发行说明和文档历史记录。

主题

NICE DCV Session Manager 发行说明

本节简要说明了 NICE DCV Session Manager 的主要更新、功能版本和错误修复。所有更新是按发行日期排列的。我们经常更新文档以处理您发给我们的反馈。

主题

2023.1 - 2023 年 11 月 9 日

内部版本号	更改和错误修复
• Broker：410 • Agent：732 • CLI：140	• 错误修复和性能改进

2023.0-15065 - 2023 年 5 月 4 日

内部版本号	更改和错误修复
• Broker：392 • Agent：675 • CLI：132	• 添加了对 ARM 平台上的 Red Hat Enterprise Linux 9、Rocky Linux 9 和 CentOS Stream 9 的支持。

2023.0-14852 - 2023 年 3 月 28 日

内部版本号	更改和错误修复
• Broker：392 • Agent：642 • CLI：132	• 添加了对 Red Hat Enterprise Linux 9、Rocky Linux 9 和 CentOS Stream 9 的支持。

2022.2-13907 - 2022 年 11 月 11 日

内部版本号	更改和错误修复
• Broker：382 • Agent：612 • CLI：123	• 添加了一个 `Substate` 字段以作为 `DescribeSessions` 响应。 • 修复了可能导致 CLI 无法连接到 Broker 的问题（具体取决于使用的 URL）。

2022.1-13067 - 2022 年 6 月 29 日

内部版本号	更改和错误修复
• Broker：355 • Agent：592 • CLI：114	• 添加了在 AWS Graviton 实例上运行 Broker 的支持。 • 为 Ubuntu 22.04 添加了 Agent 和 Broker 支持。

2022.0-11952 - 2022 年 2 月 23 日

内部版本号	更改和错误修复
• Broker：341 • Agent：520 • CLI：112	• 为 Agent 添加了日志轮换功能。 • 添加了配置参数以在 Broker 中设置 Java 主目录。 • 在 Broker 中改进了从缓存到磁盘的数据刷新。 • 修复了 CLI 中的 URL 验证。

2021.3-11591 - 2021 年 12 月 20 日

内部版本号	新功能
• Broker：307 • Agent：453 • CLI：92	• 为与 NICE DCV Connection Gateway 集成添加了支持。 • 为 Ubuntu 18.04 和 Ubuntu 20.04 添加了 Broker 支持。

2021.2-11445 - 2021 年 11 月 18 日

内部版本号	更改和错误修复
• Broker：288 • Agent：413 • CLI：54	• 修复了包含 Windows 域的登录名的验证问题。

2021.2-11190 - 2021 年 10 月 11 日

内部版本号	更改和错误修复
• Broker：254 • Agent：413 • CLI：54	• 修复了命令行界面中的一个问题，该问题导致无法启动 Windows 会话。

2021.2-11042 - 2021 年 9 月 1 日

内部版本号	新功能	更改和错误修复
• Broker：254 • Agent：413 • CLI：37	• NICE DCV Session Manager 现在提供命令行界面 (CLI) 支持。您可以在 CLI 中创建和管理 NICE DCV 会话，而不是调用 API。 • NICE DCV Session Manager 引入了 Broker 数据持久性。为了获得更高的可用性，Broker 可以将服务器状态信息持久保留在外部数据存储上，并在启动时恢复数据。	• 在注册外部授权服务器时，您现在可以指定授权服务器用于对 JSON 格式的 Web 令牌进行签名的算法。通过该更改，您可以将 Azure AD 作为外部授权服务器。

2021.1-10557 - 2021 年 5 月 31 日

内部版本号	新功能	更改和错误修复
• Broker：214 • Agent：365	• NICE DCV Session Manager 添加了对传递到 Linux 上的自动运行文件的输入参数的支持。 • 现在可以将服务器属性作为要求传递给 CreateSessions API。	• 我们修复了 Windows 上的自动运行文件存在的一个问题。

2021.0-10242 - 2021 年 4 月 12 日

内部版本号	更改和错误修复

• Broker：183
• Agent：318
	• NICE DCV Session Manager 引入了以下新的 API：
	  • OpenServers
	  • CloseServers
	  • DescribeServers
	  • GetSessionScreenshots
	• 它还引入了以下新的配置参数：
	  • Broker 参数：session-screenshot-max-width 、session-screenshot-max-height 、session-screenshot-format 、create-sessions-queue-max-size 和 create-sessions-queue-max-time-seconds 。
	  • Agent 参数：agent.autorun_folder 、max_virtual_sessions 和 max_concurrent_sessions_per_user 。
	
	Agent 参数：agent.autorun_folder 、max_virtual_sessions 和 max_concurrent_sessions_per_user 。
	
	Agent 参数：agent.autorun_folder 、max_virtual_sessions 和 max_concurrent_sessions_per_user 。

2020.2-9662 - 2020 年 12 月 4 日

内部版本号	更改和错误修复
• Broker：114 • Agent：211	• 我们修复了自动生成的 TLS 证书存在的一个问题，该问题导致 Broker 无法启动。

2020.2-9508 - 2020 年 11 月 11 日

内部版本号	更改和错误修复
• Broker：78 • Agent：183	• NICE DCV Session Manager 初始版本。

文档历史记录

下表介绍了该版本的 NICE DCV Session Manager 的文档。

更改	说明	日期
NICE DCV 版本 2023.1	已针对 NICE DCV 2023.1 更新了 NICE DCV Session Manager。有关更多信息，请参阅2023.1 - 2023 年 11 月 9 日。	2023 年 11 月 9 日
NICE DCV 版本 2023.0	已针对 NICE DCV 2023.0 更新了 NICE DCV Session Manager。有关更多信息，请参阅2023.0-14852 - 2023 年 3 月 28 日。	2023 年 3 月 28 日
NICE DCV 版本 2022.2	已针对 NICE DCV 2022.2 更新了 NICE DCV Session Manager。有关更多信息，请参阅2022.2-13907 - 2022 年 11 月 11 日。	2022 年 11 月 11 日
NICE DCV 版本 2022.1	已针对 NICE DCV 2022.1 更新了 NICE DCV Session Manager。有关更多信息，请参阅2022.1-13067 - 2022 年 6 月 29 日。	2022 年 6 月 29 日
NICE DCV 版本 2022.0	已针对 NICE DCV 2022.0 更新了 NICE DCV Session Manager。有关更多信	2022 年 2 月 23 日

更改	说明	日期
	息，请参阅2022.0-11952 - 2022 年 2 月 23 日。	
NICE DCV 版本 2021.3	已针对 NICE DCV 2021.3 更新了 NICE DCV Session Manager。有关更多信息，请参阅2021.3-11591 - 2021 年 12 月 20 日。	2021 年 12 月 20 日
NICE DCV 版本 2021.2	已针对 NICE DCV 2021.2 更新了 NICE DCV Session Manager。有关更多信息，请参阅2021.2-11042 - 2021 年 9 月 1 日。	2021 年 9 月 1 日
NICE DCV 版本 2021.1	已针对 NICE DCV 2021.1 更新了 NICE DCV Session Manager。有关更多信息，请参阅2021.1-10557 - 2021 年 5 月 31 日。	2021 年 5 月 31 日
NICE DCV 版本 2021.0	已针对 NICE DCV 2021.0 更新了 NICE DCV Session Manager。有关更多信息，请参阅2021.0-10242 - 2021 年 4 月 12 日。	2021 年 4 月 12 日
NICE DCV Session Manager 初始版本	该内容的第一版。	2020 年 11 月 11 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。