NICE DCV 会话管理器

开发人员指南

亚马逊云科技

NICE DCV 会话管理器: 开发人员指南

Table of Contents

什么是 NICE DCV 会话管理器？

NICE DCV Session Manager 是一套可安装的软件包（代理和经纪商）和应用程序编程接口 (API)，使开发人员和独立软件供应商 (ISV) 能够轻松构建以编程方式创建和管理 NICE DCV 生命周期的前端应用程序跨越 NICE DCV 服务器的会话。

本指南介绍了如何使用会话管理器 API 来管理 NICE DCV 会话的生命周期。有关如何安装和配置会话管理器代理和代理的更多信息，请参阅NICE DCV 会话管理器管理员指南.

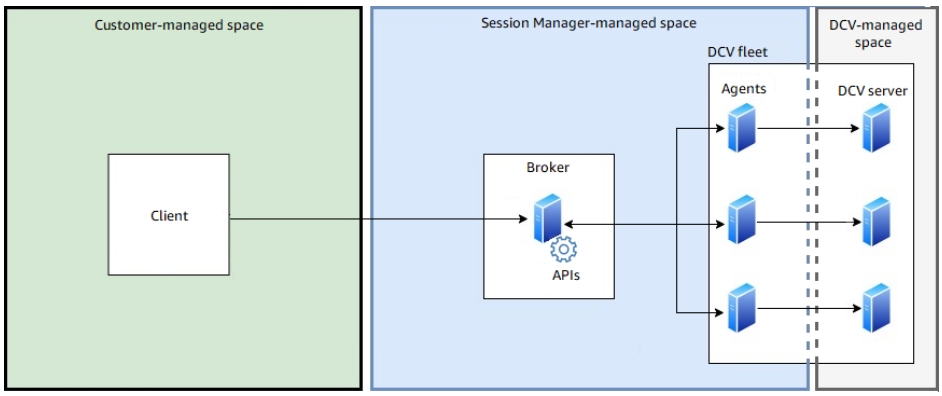
Prerequisites

在开始使用会话管理器 API 之前，请确保熟悉 NICE DCV 和 NICE DCV 会话。有关更多信息，请参阅 。NICE DCV 管理员指南.

主题

如何运行会话管理器

下图显示了会话管理器的高级组件。



代理

经纪商是托管和公开会话管理器 API 的 Web 服务器。它接收和处理API请求管理 NICE DCV 会话客户，然后将说明传递给相关人员客服. 代理必须安装在独立于 NICE DCV 服务器的主机上，但客户端必须可以访问该代理，并且必须能够访问代理。

代理

代理程序安装在队列中的每台 NICE DCV 服务器上。代理接收来自代理然后在各自的 NICE DCV 服务器上运行它们。代理还监控 NICE DCV 服务器的状态，并将定期更新状态发送回经纪商。

API

会话管理器公开了一组 REST 应用程序编程接口 (API)，可用于管理 NICE DCV 服务器队列上的 NICE DCV 会话。API 托管并由代理. 开发人员可以构建自定义会话客户那叫 API。

客户端

客户端是您开发的调用会话管理器的前端应用程序或门户API被暴露的代理. 最终用户使用客户端管理队列中 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 从经纪商或外部授权服务器请求令牌。要请求和访问令牌，客户端 API 必须提供有效的凭据。

客户端 API

客户端 API 是使用 Swagger Codegen 从会话管理器 API 定义 YAML 文件生成的。客户端 API 用于发出 API 请求。

NICE DCV 会话

您必须在 NICE DCV 服务器上创建一个可以连接到的 NICE DCV 会话。如果有活动会话，客户端只能连接到 NICE DCV 服务器。NICE DCV 支持控制台和虚拟会话。您可以使用会话管理器 API 来管理 NICE DCV 会话的生命周期。NICE DCV 会话可以处于以下状态之一：

- `CREATING`-经纪人正在创建会话。
- `READY`-会话已准备好接受客户端连接。
- `DELETING`-正在删除会话。
- `DELETED`-会话已删除。
- `UNKNOWN`— 无法确定会话的状态。经纪人和代理可能无法进行沟通。

Features

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— 通过一个 API 请求跨多个 NICE DCV 服务器为多个用户创建或删除多个会话。
- 支持标签— 在创建会话时，使用自定义标签来定位一组 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— 在上使用会话管理器Amazon、本地或使用其他基于云的服务器。

开始使用

本节介绍如何开始使用会话管理器 API。

在本节中，我们将向您演示如何使用`DescribeSessions`以 API 为例。

主题

生成客户端 API

会话管理器 API 在单个 YAML 文件中定义。这些 API 基于 OpenAPI3.0 规范，该规范定义了 RESTful API 的标准、与语言无关的接口。有关更多信息，请参阅 。OpenAPI 规范.

使用 YAML 文件，您可以使用其中一种受支持的语言生成 API 客户端。为此，必须使用 Swagger Codegen 3.0 或更高版本。有关支持的语言的更多信息，请参阅swagger-codegen 回购.

生成 API 客户端

1. 从会话管理器代理下载会话管理器 API YAML 文件。YAML 文件可在以下 URL 中找到。

```
https://broker_host_ip:port/dcv-session-manager-api.yaml
```

2. 安装 Swagger Codegen。

   - macOS

   ```
   $ brew install swagger-codegen
   ```

   - 其他平台

   ```
   $ git clone https://github.com/swagger-api/swagger-codegen --branch 3.0.0
   ```

   ```
   $ cd swagger-codegen
   ```

3. 生成 API 客户端。

   - macOS

   ```
   $ swagger-codegen generate -i /path_to/yaml_file -l language -o $output_folder
   ```

   - 其他平台

   ```
   $ mvn clean package
   ```

   ```
   $ java -jar modules/swagger-codegen-cli/target/swagger-codegen-cli.jar generate -i /
   path_to/yaml_file -l language -o output_folder
   ```

注册 API 客户端

要发出 API 请求，您必须首先从经纪商处检索访问令牌。要从经纪商那里获取访问令牌，您必须向经纪商提供客户端 API 的凭证。凭证基于客户在经纪商注册时生成的客户 ID 和客户密码。如果您没有客户的客户 ID 和客户密码，则必须向经纪商管理员申请。有关向经纪商注册客户端 API 以及获取客户 ID 和密码的更多信息，请参阅注册 api-Client.

获取访问令牌并发出 API 请求

首先我们导入应用程序所需的模型。

然后我们为客户端 ID 声明变量 (__CLIENT_ID)、客户端密码 (__CLIENT_SECRET) 和经纪人 URL，包括端口号 (__PROTOCOL_HOST_PORT)。

接下来，我们创建函数build_client_credentials这将生成客户端凭证。要生成客户端凭据，必须首先连接客户端 ID 和客户端密码，然后用冒号 (*client_ID*:*client_password*) ，然后 Base64 对整个字符串进行编码。

```
import swagger_client
import base64
import requests
import json
from swagger_client.models.describe_sessions_request_data import
 DescribeSessionsRequestData
from swagger_client.models.key_value_pair import KeyValuePair
from swagger_client.models.delete_session_request_data import DeleteSessionRequestData
from swagger_client.models.update_session_permissions_request_data import
 UpdateSessionPermissionsRequestData
from swagger_client.models.create_session_request_data import CreateSessionRequestData

__CLIENT_ID = '794b2dbb-bd82-4707-a2f7-f3d9899cb386'
__CLIENT_SECRET = 'MzcxNzJhN2UtYjEzNS00MjNjLTg2N2YtMjFlZmRlZWNjMDU1'
__PROTOCOL_HOST_PORT = 'https://<broker-hostname>:8443'

def build_client_credentials():
    client_credentials = '{client_id}:{client_secret}'.format(client_id=__CLIENT_ID,
 client_secret=__CLIENT_SECRET)
    return base64.b64encode(client_credentials.encode('utf-8')).decode('utf-8')
```

现在我们已经有了客户凭证，我们可以用它向经纪商请求访问令牌。为此，我们创建一个函数get_access_token. 你必须打电话给POST上 https://*Broker_IP*:*8443*/oauth2/token?grant_type=client_credentials，并提供授权标头，其中包括基本编码的客户端凭据和内容类型application/x-www-form-urlencoded.

```
def get_access_token():
    client_credentials = build_client_credentials()
    headers = {
        'Authorization': 'Basic {}'.format(client_credentials),
        'Content-Type': 'application/x-www-form-urlencoded'
    }
    endpoint = __PROTOCOL_HOST_PORT + '/oauth2/token?grant_type=client_credentials'
    print('Calling', endpoint, 'using headers', headers)
    res = requests.post(endpoint, headers=headers, verify=True)
    if res.status_code != 200:
        print('Cannot get access token:', res.text)
        return None
```

```
    access_token = json.loads(res.text)['access_token']
    print('Access token is', access_token)
    return access_token
```

现在，我们创建了实例化客户端 API 所需的函数。要实例化客户端 API，您必须指定客户端配置和要用于请求的标头。这些区域有：`get_client_configuration`函数创建一个配置对象，其中包括经纪商的 IP 地址和端口以及您应该从经纪商管理员那里收到的自签名证书的路径。这些区域有：`set_request_headers`函数创建一个包含客户端凭据和访问令牌的请求头对象。

```
def get_client_configuration():
    configuration = swagger_client.Configuration()
    configuration.host = __PROTOCOL_HOST_PORT
    configuration.verify_ssl = True
    # configuration.ssl_ca_cert = cert_file.pem
    return configuration

def set_request_headers(api_client):
    access_token = get_access_token()
    api_client.set_default_header(header_name='Authorization',
                                  header_value='Bearer {}'.format(access_token))

def get_sessions_api():
    api_instance =
 swagger_client.SessionsApi(swagger_client.ApiClient(get_client_configuration()))
    set_request_headers(api_instance.api_client)
    return api_instance
```

最后，我们创建了一个调用`DescribeSessions`API。有关更多信息，请参阅DescribeSessions (p. 20)。

```
def describe_sessions(session_ids=None, next_token=None, tags=None, owner=None):
    filters = list()
    if tags:
        for tag in tags:
            filter_key_value_pair = KeyValuePair(key='tag:' + tag['Key'],
 value=tag['Value'])
            filters.append(filter_key_value_pair)
    if owner:
        filter_key_value_pair = KeyValuePair(key='owner', value=owner)
        filters.append(filter_key_value_pair)

    request = DescribeSessionsRequestData(session_ids=session_ids, filters=filters,
 next_token=next_token)
    print('Describe Sessions Request:', request)
    api_instance = get_sessions_api()
    api_response = api_instance.describe_sessions(body=request)
    print('Describe Sessions Response', api_response)

def main():
    describe_sessions(
        session_ids=['SessionId1895', 'SessionId1897'],
        owner='an owner 1890',
        tags=[{'Key': 'ram', 'Value': '4gb'}])
```

参考会话管理器 API

本节提供每个会话管理器 API 操作的描述、语法和用法示例。

主题

CloseServers

关闭一台或多台 NICE DCV 服务器。关闭 NICE DCV 服务器时，将其设置为不可用于 NICE DCV 会话放置。你不能在上创建 NICE DCV 会话关闭服务器。关闭服务器可确保服务器上没有会话运行，并且用户无法在其上创建新会话。

主题

请求参数

`ServerId`

要关闭的服务器的 ID。

类型: 字符串

：必需 是

`Force`

强制执行关闭行动。如果你指定`true`，即使服务器有正在运行的会话，也会关闭。会话将继续运行。

类型: Boolean

：必需 否

响应参数

`RequestId`

请求的唯一 ID。

SuccessfulList

关于成功关闭的 NICE DCV 服务器的信息。此数据结构包括以下嵌套响应参数：

ServerId

成功关闭的服务器的 ID。

UnsuccessfulList

有关无法关闭的 NICE DCV 服务器的信息。此数据结构包括以下嵌套响应参数：

CloseServerRequestData

有关失败的原始请求的信息。此数据结构包括以下嵌套响应参数：

ServerId

无法关闭的 NICE DCV 服务器的 ID。

Force

请求的 force 参数。

FailureCode

失败的代码。

FailureReason

失败的原因。

Example

Python

Request

以下示例关闭两个 NICE DCV 服务器（`serverId1`和`serverId2`）。服务器`serverId2`不存在并导致失败。

```
from swagger_client.models import CloseServerRequestData

def get_servers_api():
    api_instance =
 swagger_client.ServersApi(swagger_client.ApiClient(get_client_configuration()))
    set_request_headers(api_instance.api_client)
    return api_instance

def close_servers(server_ids):
    request = [CloseServerRequestData(server_id=server_id) for server_id in server_ids]
    print('Close Servers Request:', request)
    api_instance = get_servers_api()
    api_response = api_instance.close_servers(body=request)
    print('Close Servers Response:', api_response)
    open_servers(server_ids)

def main():
    close_servers(["serverId1", "serverId2"])
```

Response

下面是示例输出。

```
{
```

```
    "RequestId": "4d7839b2-a03c-4b34-a40d-06c8b21099e6",
    "SuccessfulList": [
        {
            "ServerId": "serverId1"
        }
    ],
    "UnsuccessfulList": [
        {
            "OpenServerRequestData": {
                "ServerId": "serverId2"
            },
            "FailureCode": "DCV_SERVER_NOT_FOUND",
            "FailureReason": "Dcv server not found."
        }
    ]
}
```

CreateSessions

使用指定的详细信息创建一个新的 NICE DCV 会话。

API 操作

- 请求参数 (p. 4)
- 响应参数 (p. 6)
- Example (p. 7)

请求参数

Name

会话名称。

类型: 字符串

：必需 是

Owner

会话所有者的名称。这必须是目标 NICE DCV 服务器上现有用户的名称。

类型: 字符串

：必需 是

Type

会话类型。有关会话类型的更多信息，请参阅NICE DCV 会议简介中的NICE DCV 管理员指南.

有效值：控制台 | 虚拟

类型: 字符串

：必需 是

InitFile

在 Linux NICE DCV 服务器上支持将虚拟会话一起使用。Windows 和 Linux NICE DCV 服务器上的控制台会话不支持控制台会话使用。在 NICE DCV 服务器上运行的自定义脚本的路径，以便在创建会话时初

始化会话。文件路径相对于为`agent.init_folder`代理配置参数。如果文件位于指定的 init 目录中，请仅指定文件名。如果文件不在指定的 init 目录中，请指定相对路径。有关详细信息，请参阅代理配置文件中的NICE DCV 会话管理器管理员指南.

类型: 字符串

：必需 否

`MaxConcurrents`

最大并行 NICE DCV 客户端数。

类型: 整数

：必需 否

`DcvGlEnabled`

指示虚拟会话是否配置为使用基于硬件的 OpenGL。仅支持虚拟会话。Windows NICE DCV 服务器不支持将此参数与否使用。

有效值：true | false

类型: Boolean

：必需 否

`PermissionsFile`

权限文件的 base64 编码内容。如果省略，则默认为服务器默认值。有关更多信息，请参阅 。配置 NICE DCV 授权中的NICE DCV 管理员指南.

类型: 字符串

：必需 否

`EnqueueRequest`

指示如果无法立即履行请求，是否将请求排队。

类型: Boolean

默认值：false

：必需 否

`AutorunFile`

在 Windows NICE DCV 服务器上支持控制台会话和（在 Linux NICE DCV 服务器上）支持控制台会话。Linux NICE DCV 服务器上的控制台会话不支持控制台会话使用。

主机服务器上要在会话内运行的文件的路径。文件路径相对于为`agent.autorun_folder`代理配置参数。如果文件位于指定的自动运行目录中，请仅指定文件名。如果文件不在指定的自动运行目录中，请指定相对路径。有关更多信息，请参阅 。代理配置文件中的NICE DCV 会话管理器管理员指南.

该文件是代表指定的所有者. 指定的所有者必须具有在服务器上运行文件的权限。在 Windows NICE DCV 服务器上，当所有者登录会话时，该文件将运行。在 Linux NICE DCV 服务器上，文件将在创建会话时运行。

类型: 字符串

：必需 否

AutorunFileArguments

在 Linux NICE DCV 服务器上支持将虚拟会话一起使用。Windows 和 Linux NICE DCV 服务器上的控制台会话中不支持此选项卡访问。传递到的命令行参数自动运行文件在会话内执行时。参数按照出现在给定数组中的顺序传递。可以配置允许的最大参数数和每个参数的最大允许长度。有关更多信息，请参阅 。代理配置文件中的NICE DCV 会话管理器管理员指南.

类型: 字符串数组

：必需 否

DisableRetryOnFailure

指示在创建会话请求出于任何原因在 NICE DCV 主机上失败后是否不重试该请求。有关创建会话重试机制的更多信息，请参阅代理配置文件中的NICE DCV 会话管理器管理员指南.

类型: Boolean

默认值：false

：必需 否

Requirements

为了放置会话，服务器必须满足的要求。这些要求可以包括服务器标记和/或服务器属性，通过调用DescribeServersAPI。

要求条件表达式：

- *##* ！ = *b*如果 true*##*不等于*b*
- *##* = *b*如果 true*##*等于*b*
- *##* > *b*如果 true*##*大于*b*
- *##* >= *b*如果 true*##*大于或等于*b*
- *##* < *b*如果 true*##*小于*b*
- *##* <= *b*如果 true*##*小于或等于*b*
- *##* = *b*如果 true*##*包含字符串*b*

要求布尔运算符：

- *##* 和 *b*如果 true*##*和*b*is true
- *##* 要么 *b*如果 true*##*要么*b*is true
- 不 *##*如果 true*##*is false

标签密钥的前缀必须是`tag:`，服务器属性必须以`server:`。要求表达式支持括号`()`.

要求示例：

- `tag:color = 'pink' and (server:Host.Os.Family = 'windows' or tag:color := 'red')`
- `"server:Host.Aws.Ec2InstanceType := 't2' and server:Host.CpuInfo.NumberOfCpus >= 2"`

数值可以使用指数表示法指定，例如：`"server:Host.Memory.TotalBytes > 1024E6"`.

支持的服务器属性包括：

- `Id`
- `Hostname`
- `Version`

- SessionManagerAgentVersion
- Host.Os.BuildNumber
- Host.Os.Family
- Host.Os.KernelVersion
- Host.Os.Name
- Host.Os.Version
- Host.Memory.TotalBytes
- Host.Memory.UsedBytes
- Host.Swap.TotalBytes
- Host.Swap.UsedBytes
- Host.CpuLoadAverage.OneMinute
- Host.CpuLoadAverage.FiveMinutes
- Host.CpuLoadAverage.FifteenMinutes
- Host.Aws.Ec2InstanceId
- Host.Aws.Ec2InstanceType
- Host.Aws.Region
- Host.Aws.Ec2ImageId
- Host.CpuInfo.Architecture
- Host.CpuInfo.ModelName
- Host.CpuInfo.NumberOfCpus
- Host.CpuInfo.PhysicalCoresPerCpu
- Host.CpuInfo.Vendor

类型: 字符串

：必需 否

StorageRoot

指定用于会话存储的文件夹的路径。有关 NICE DCV 会话存储的更多信息，请参阅启用会话存储中的NICE DCV 管理员指南.

类型: 字符串

：必需 否

响应参数

Id

会话的唯一 ID。

Name

会话名称。

Owner

会话所有者。

Type

会话的类型。

State

会话的状态。如果请求成功完成，会话将输入CREATING状态。

Example

Python

Request

以下示例创建了三个会话。

```
from swagger_client.models.create_session_request_data import CreateSessionRequestData

def get_sessions_api():
    api_instance =
 swagger_client.SessionsApi(swagger_client.ApiClient(get_client_configuration()))
    set_request_headers(api_instance.api_client)
    return api_instance

def create_sessions(sessions_to_create):
    create_sessions_request = list()
    for name, owner, session_type, init_file_path, autorun_file,
 autorun_file_arguments, max_concurrent_clients,\
            dcv_gl_enabled, permissions_file, requirements, storage_root in
 sessions_to_create:
        a_request = CreateSessionRequestData(
            name=name, owner=owner, type=session_type,
            init_file_path=init_file_path, autorun_file=autorun_file,
 autorun_file_arguments=autorun_file_arguments,
 max_concurrent_clients=max_concurrent_clients,
            dcv_gl_enabled=dcv_gl_enabled, permissions_file=permissions_file,
 requirements=requirements, storage_root=storage_root)
        create_sessions_request.append(a_request)

    api_instance = get_sessions_api()
    print('Create Sessions Request:', create_sessions_request)
    api_response = api_instance.create_sessions(body=create_sessions_request)
    print('Create Sessions Response:', api_response)

def main():
    create_sessions([
    ('session1', 'user1', 'CONSOLE', None, None, None, 1, None, '/dcv/
permissions.file', "tag:os = 'windows' and server:Host.Memory.TotalBytes > 1024", "/
storage/root"),
    ('session2', 'user1', 'VIRTUAL', None, 'myapp.sh', None, 1, False, None, "tag:os =
 'linux'", None),
    ('session3', 'user1', 'VIRTUAL', '/dcv/script.sh', 'myapp.sh', ['argument1',
 'argument2'], 1, False, None, "tag:os = 'linux'", None),
])
```

Response

下面是示例输出。

```
{
              "RequestId": "e32d0b83-25f7-41e7-8c8b-e89326ecc87f",
    "SuccessfulList": [
    {
            "Id": "78b45deb-1163-46b1-879b-7d8fcbe9d9d6",
```

```
            "Name": "session1",
            "Owner": "user1",
            "Type": "CONSOLE",
            "State": "CREATING"
    },
    {
            "Id": " a0c743c4-9ff7-43ce-b13f-0c4d55a268dd",
            "Name": "session2",
            "Owner": "user1",
            "Type": "VIRTUAL",
            "State": "CREATING"
    },
    {
            "Id": " 10311636-df90-4cd1-bcf7-474e9675b7cd",
            "Name": "session3",
            "Owner": "user1",
            "Type": "VIRTUAL",
            "State": "CREATING"
    }
    ],
    "UnsuccessfulList": [
    ]
}
```

DescribeServers

描述一台或多台 NICE DCV 服务器。

主题

请求参数

`ServerIds`

要描述的 NICE DCV 服务器的 ID。如果未指定 ID，则以分页输出形式返回所有服务器。

类型: 字符串数组

：必需 否

`NextToken`

用于检索下一页结果的令牌。

类型: 字符串

：必需 否

`MaxResults`

分页输出中请求返回的最大结果数。使用此参数时，请求只会在单个页面中返回指定数量的结果以及`NextToken`响应元素。可以通过发送另一个返回的请求来查看初始请求的剩余结果`NextToken`值。

有效范围：1-1000

默认值：1000

类型: 整数

：必需 否

响应参数

RequestId

请求的唯一 ID。

Servers

有关 NICE DCV 服务器的信息。此数据结构包括以下嵌套响应参数：

Id

NICE DCV 服务器的唯一 ID。

Ip

NICE DCV 服务器的 IP 地址。

Hostname

NICE DCV 服务器的主机名。

Endpoints

有关 NICE DCV 服务器终端节点的信息。此数据结构包括以下嵌套响应参数：

Port

服务器终端节点的端口。

WebUrlPath

服务器终端节点的 Web URL 路径。仅适用于 HTTP 协议。

Protocol

服务器终端节点使用的协议。可能的值包括：

- `HTTP`— 终端节点使用 WebSocket (TCP) 协议。
- `QUIC`— 终端节点使用 QUIC (UDP) 协议。

Version

NICE DCV 服务器的版本。

SessionManagerAgentVersion

在 NICE DCV 服务器上运行的版本会话管理器代理。

Availability

NICE DCV 服务器的可用性。可能的值包括：

- `AVAILABLE`— 服务器可用并准备好进行会话放置。
- `UNAVAILABLE`— 服务器不可用且无法接受会话放置。

UnavailabilityReason

NICE DCV 服务器不可用的原因。可能的值包括：

- `SERVER_FULL`— NICE DCV 服务器已达到它可以运行的最大并发会话数。
- `SERVER_CLOSED`— NICE DCV 服务器已使用关闭服务器API。
- `UNREACHABLE_AGENT`— 会话管理器代理无法与 NICE DCV 服务器上的会话管理器代理进行通信。

- `UNHEALTHY_DCV_SERVER`— 会话管理器代理无法与 NICE DCV 服务器通信。
- `EXISTING_LOGGED_IN_USER`—（仅限 Windows NICE DCV 服务器）用户当前正在使用 RDP 登录到 NICE DCV 服务器。
- `UNKNOWN`— 会话管理器经纪人无法确定原因。

`ConsoleSessionCount`

NICE DCV 服务器上的控制台会话数。

`VirtualSessionCount`

NICE DCV 服务器上的虚拟会话数。

`Host`

有关运行 NICE DCV 服务器的主机服务器的信息。此数据结构包括以下嵌套响应参数：

`Os`

有关主机服务器操作系统的信息。此数据结构包括以下嵌套响应参数：

`Family`

操作系统系列。可能的值包括：

- `windows`— 主机服务器正在运行 Windows 操作系统。
- `linux`— 主机服务器正在运行 Linux 操作系统。

`Name`

操作系统的名称。

`Version`

操作系统的版本。

`KernelVersion`

（仅限 Linux）操作系统的内核版本。

`BuildNumber`

（仅限 Windows）操作系统的内部版本号。

`Memory`

有关主机服务器内存的信息。此数据结构包括以下嵌套响应参数：

`TotalBytes`

主机服务器上的总内存（以字节为单位）。

`UsedBytes`

主机服务器上使用的内存（以字节为单位）。

`Swap`

有关主机服务器的交换文件的信息。此数据结构包括以下嵌套响应参数：

`TotalBytes`

主机服务器上的交换文件总大小（以字节为单位）。

`UsedBytes`

主机服务器上使用的交换文件大小（以字节为单位）。

`Aws`

仅适用于在 Amazon EC2 实例上运行的 NICE DCV 服务器。Amazon特定信息。此数据结构包括以下嵌套响应参数：

Region

这些区域有：AmazonAmazon EC2 实例的区域。

Ec2InstanceType

Amazon EC2 实例的类型。

Ec2InstanceId

Amazon EC2 实例的 ID。

Ec2ImageId

Amazon EC2 映像的 ID。

CpuInfo

有关主机服务器 CPU 的信息。此数据结构包括以下嵌套响应参数：

Vendor

主机服务器 CPU 的供应商。

ModelName

主机服务器 CPU 的型号名称。

Architecture

主机服务器 CPU 的体系结构。

NumberOfCpus

主机服务器上的 CPU 数量。

PhysicalCorePerCpu

每个 CPU 的 CPU 核心数。

CpuLoadAverage

有关主机服务器 CPU 负载的信息。此数据结构包括以下嵌套响应参数：

OneMinute

前 1 分钟时间段的平均 CPU 负载。

FiveMinutes

前 5 分钟时间段的平均 CPU 负载。

FifteenMinutes

前 15 分钟时间段的平均 CPU 负载。

Gpus

有关主机服务器的 GPU 的信息。此数据结构包括以下嵌套响应参数：

Vendor

主机服务器 GPU 的供应商。

ModelName

主机服务器 GPU 的型号名称。

LoggedInUsers

当前登录到主机服务器的用户。此数据结构包括以下嵌套响应参数：

Username

登录用户的用户名。

Tags

分配给服务器的标签。此数据结构包括以下嵌套响应参数：

Key

标签键。

Value

标签值。

Example

Python

Request

以下示例介绍了所有可用的 NICE DCV 服务器。对结果进行分页以显示每页两个结果。

```
from swagger_client.models.describe_servers_request_data import
 DescribeServersRequestData

def get_servers_api():
    api_instance =
 swagger_client.ServersApi(swagger_client.ApiClient(get_client_configuration()))
    set_request_headers(api_instance.api_client)
    return api_instance

def describe_servers(server_ids=None, next_token=None, max_results=None):
    request = DescribeServersRequestData(server_ids=server_ids, next_token=next_token,
 max_results=max_results)
    print('Describe Servers Request:', request)
    api_instance = get_servers_api()
    api_response = api_instance.describe_servers(body=request)
    print('Describe Servers Response', api_response)

def main():
    describe_servers(max_results=2)
```

Response

下面是示例输出。

```
{
    "RequestId": "request-id-123",
    "Servers": [
        {
            "Id": "ServerId123",
            "Ip": "1.1.1.123",
            "Hostname": "node001",
            "Endpoints": [
                {
                    "Port": 8443,
                    "WebUrlPath": "/",
                    "Protocol": "HTTP"
                }
            ],
            "Version": "2021.0.10000",
```

```
            "SessionManagerAgentVersion": "2021.0.300",
            "Availability": "UNAVAILABLE",
            "UnavailabilityReason": "SERVER_FULL",
            "ConsoleSessionCount": 1,
            "VirtualSessionCount": 0,
            "Host": {
                "Os": {
                    "Family": "windows",
                    "Name": "Windows Server 2016 Datacenter",
                    "Version": "10.0.14393",
                    "BuildNumber": "14393"
                },
                "Memory": {
                    "TotalBytes": 8795672576,
                    "UsedBytes": 1743886336
                },
                "Swap": {
                    "TotalBytes": 0,
                    "UsedBytes": 0
                },
                "Aws": {
                    "Region": "us-west-2b",
                    "EC2InstanceType": "t2.large",
                    "EC2InstanceId": "i-123456789",
                    "EC2ImageId": "ami-12345678987654321"
                },
                "CpuInfo": {
                    "Vendor": "GenuineIntel",
                    "ModelName": "Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz",
                    "Architecture": "x86_64",
                    "NumberOfCpus": 2,
                    "PhysicalCoresPerCpu": 3
                },
                "CpuLoadAverage": {
                    "OneMinute": 0.04853546,
                    "FiveMinutes": 0.21060601,
                    "FifteenMinutes": 0.18792416
                },
                "Gpus": [],
                "LoggedInUsers": [
                    {
                        "Username": "Administrator"
                    }
                ]
            },
            "Tags": [
                {
                    "Key": "color",
                    "Value": "pink"
                },
                {
                    "Key": "dcv:os-family",
                    "Value": "windows"
                },
                {
                    "Key": "size",
                    "Value": "small"
                },
                {
                    "Key": "dcv:max-virtual-sessions",
                    "Value": "0"
                }
            ]
        },
        {
            "Id": "server-id-12456897",
```

```
            "Ip": "1.1.1.145",
            "Hostname": "node002",
            "Endpoints": [
                {
                    "Port": 8443,
                    "WebUrlPath": "/",
                    "Protocol": "HTTP"
                },
                {
                    "Port": 8443,
                    "Protocol": "QUIC"
                }
            ],
            "Version": "2021.0.10000",
            "SessionManagerAgentVersion": "2021.0.0",
            "Availability": "AVAILABLE",
            "ConsoleSessionCount": 0,
            "VirtualSessionCount": 5,
            "Host": {
                "Os": {
                    "Family": "linux",
                    "Name": "Amazon Linux",
                    "Version": "2",
                    "KernelVersion": "4.14.203-156.332.amzn2.x86_64"
                },
                "Memory": {
                    "TotalBytes": 32144048128,
                    "UsedBytes": 2184925184
                },
                "Swap": {
                    "TotalBytes": 0,
                    "UsedBytes": 0
                },
                "Aws": {
                    "Region": "us-west-2a",
                    "EC2InstanceType": "g3s.xlarge",
                    "EC2InstanceId": "i-123456789",
                    "EC2ImageId": "ami-12345678987654321"
                },
                "CpuInfo": {
                    "Vendor": "GenuineIntel",
                    "ModelName": "Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz",
                    "Architecture": "x86_64",
                    "NumberOfCpus": 4,
                    "PhysicalCoresPerCpu": 2
                },
                "CpuLoadAverage": {
                    "OneMinute": 2.24,
                    "FiveMinutes": 0.97,
                    "FifteenMinutes": 0.74
                },
                "Gpus": [
                    {
                        "Vendor": "NVIDIA Corporation",
                        "ModelName": "GM204GL [Tesla M60]"
                    }
                ],
                "LoggedInUsers": [
                    {
                        "Username" : "user45687"
                    },
                    {
                        "Username" : "user789"
                    }
                ]
            },
```

```
            "Tags": [
                {
                    "Key": "size",
                    "Value": "big"
                },
                {
                    "Key": "dcv:os-family",
                    "Value": "linux"
                },
                {
                    "Key": "dcv:max-virtual-sessions",
                    "Value": "10"
                },
                {
                    "Key": "color",
                    "Value": "blue"
                }
            ]
        }
    ]
}
```

DescribeSessions

描述一个或多个 NICE DCV 会话。

主题

请求参数

`SessionIds`

要描述的会话的 ID。

类型: 字符串

: 必需 否

`NextToken`

用于检索下一页结果的令牌。

类型: 字符串

: 必需 否

`Filters`

要应用于请求的其他筛选器。支持的筛选条件有：

- 标签：Key-分配给会话的标签。
- 所有者-会话所有者。

类型: 字符串

: 必需 否

响应参数

Id

会话的唯一 ID。

Name

会话的名称。

Owner

会话的所有者。

Server

有关运行会话的服务器的信息。此数据结构包括以下嵌套响应参数：

- **Ip**

  NICE DCV 服务器主机的 IP 地址。
- **Hostname**

  NICE DCV 服务器主机的主机名。
- **Port**

  NICE DCV 服务器与 NICE DCV 客户端通信的端口。
- **Tags**

  分配给会话的标签。此数据结构包括以下嵌套响应参数：
  - **Key**

    标签键。
  - **Value**

    标签值。

Type

会话的类型。

State

会话的当前状态。可能的值有：

- CREATING-经纪商正在创建会话。
- READY-会话已准备好接受客户端连接。
- DELETING-正在删除会话。
- DELETED-会话已删除。
- UNKNOWN-无法确定会话的状态。经纪人和代理可能无法进行沟通。

CreationTime

创建会话的日期和时间。

LastDisconnectionTime

上次客户端断开连接的日期和时间。

NumOfConnections

活动客户端连接数。

StorageRoot

指定用于会话存储的文件夹的路径。有关 NICE DCV 会话存储的更多信息，请参阅启用会话存储中的NICE DCV 管理员指南.

类型: 字符串

：必需 否

Example

Python

Request

以下示例描述由其拥有的会话：user1并有一个标签os=windows.

```
from swagger_client.models.describe_sessions_request_data import
 DescribeSessionsRequestData
from swagger_client.models.key_value_pair import KeyValuePair

def get_sessions_api():
    api_instance =
 swagger_client.SessionsApi(swagger_client.ApiClient(get_client_configuration()))
    set_request_headers(api_instance.api_client)
    return api_instance

def describe_sessions(session_ids=None, next_token=None, tags=None, owner=None):
    filters = list()
    if tags:
        for tag in tags:
            filter_key_value_pair = KeyValuePair(key='tag:' + tag['Key'],
 value=tag['Value'])
            filters.append(filter_key_value_pair)
    if owner:
        filter_key_value_pair = KeyValuePair(key='owner', value=owner)
        filters.append(filter_key_value_pair)

    request = DescribeSessionsRequestData(session_ids=session_ids, filters=filters,
 next_token=next_token)
    print('Describe Sessions Request:', request)
    api_instance = get_sessions_api()
    api_response = api_instance.describe_sessions(body=request)
    print('Describe Sessions Response', api_response)

def main():
    describe_sessions(
        owner='user1',
        tags=[{'Key': 'os', 'Value': 'windows'}])
```

Response

下面是示例输出。

```
{
    "Sessions": [
        {
            "Id": "SessionId1897",
            "Name": "a session name",
            "Owner": "an owner 1890",
            "Server": {
                "Ip": "1.1.1.123",
                "Hostname": "server hostname",
                "Port": "1222",
                "Tags": [
                    {
                        "Key": "os",
                        "Value": "windows"
                    },
                    {
                        "Key": "ram",
                        "Value": "4gb"
                    }
                ]
            },
            "Type": "VIRTUAL",
            "State": "READY",
            "CreationTime": "2020-10-06T10:15:31.633Z",
            "LastDisconnectionTime": "2020-10-06T10:15:31.633Z",
            "NumOfConnections": 2,
            "StorageRoot" : "/storage/root"
        },
        {
            "Id": "SessionId1895",
            "Name": "a session name",
            "Owner": "an owner 1890",
            "Server": {
                "Ip": "1.1.1.123",
                "Hostname": "server hostname",
                "Port": "1222",
                "Tags": [
                    {
                        "Key": "os",
                        "Value": "windows"
                    },
                    {
                        "Key": "ram",
                        "Value": "4gb"
                    }
                ]
            },
            "Type": "VIRTUAL",
            "State": "DELETING",
            "CreationTime": "2020-10-06T10:15:31.633Z",
            "LastDisconnectionTime": "2020-10-06T10:15:31.633Z",
            "NumOfConnections": 2,
            "StorageRoot" : "/storage/root"
        }
    ]
}
```


DeleteSessions

删除指定的 NICE DCV 会话并将其从经纪商的缓存中删除。

主题

请求参数

`SessionId`

要删除的会话的 ID。

类型: 字符串

：必需 是

`Owner`

要删除的会话的所有者。

类型: 字符串

：必需 是

`Force`

从经纪商缓存中删除会话，同时尝试将其从 NICE DCV 服务器中删除。这对于从经纪商缓存中删除过时的会话非常有用。例如，如果 NICE DCV 服务器已停止，但会话仍在 Broker 上注册，请使用此标志从经纪商缓存中清除会话。

请记住，如果会话仍处于活动状态，则经纪商会重新缓存该会话。

有效值：`true` | `false`

类型: Boolean

：必需 否

响应参数

`SessionId`

会话 ID

`State`

只有在成功删除会话时才返回。指示会话的当前状态。如果请求成功完成，会话将转换为`DELETING`状态。删除会话可能需要几分钟。删除后，状态从`DELETING`到`DELETED`.

`FailureReason`

仅在无法删除某些会话时返回。指示为什么无法删除会话。

Example

Python

Request

以下示例删除两个会话（ ID 为的会话）`SessionId123`这是由`user1`，以及 ID 为的会话`SessionIdabc`这是由`user99`.

```
from swagger_client.models.delete_session_request_data import DeleteSessionRequestData

def get_sessions_api():
    api_instance =
 swagger_client.SessionsApi(swagger_client.ApiClient(get_client_configuration()))
    set_request_headers(api_instance.api_client)
    return api_instance

def delete_sessions(sessions_to_delete, force=False):
    delete_sessions_request = list()
    for session_id, owner in sessions_to_delete:
        a_request = DeleteSessionRequestData(session_id=session_id, owner=owner,
 force=force)
        delete_sessions_request.append(a_request)

    print('Delete Sessions Request:', delete_sessions_request)
    api_instance = get_sessions_api()
    api_response = api_instance.delete_sessions(body=delete_sessions_request)
    print('Delete Sessions Response', api_response)

def main():
    delete_sessions([('SessionId123', 'an owner user1'), ('SessionIdabc', 'user99')])
```

Response

下面是示例输出。SessionId123已成功删除，而SessionIdabc无法删除。

```
{
    "RequestId": "10311636-df90-4cd1-bcf7-474e9675b7cd",
    "SuccessfulList": [
        {
            "SessionId": "SessionId123",
            "State": "DELETING"
        }
    ],
    "UnsuccessfulList": [
        {
            "SessionId": "SessionIdabc",
            "FailureReason": "The requested dcvSession does not exist"
        }
    ]
}
```

GetSessionConnectionData

获取特定用户连接到特定 NICE DCV 会话的连接信息。

主题

请求参数

`SessionId`

要查看其连接信息的会话的 ID。

类型: 字符串

：必需 是

User

要查看其连接信息的用户的名称。

类型: 字符串

：必需 是

响应参数

Id

会话的唯一 ID。

Name

会话的名称。

Owner

会话的所有者。

Server

有关运行会话的服务器的信息。此数据结构包括以下嵌套响应参数：

- **Ip**

  NICE DCV 服务器主机的 IP 地址。
- **Hostname**

  NICE DCV 服务器主机的主机名。
- **Port**

  NICE DCV 服务器与 NICE DCV 客户端通信的端口。
- **WebUrlPath**

  NICE DCV 服务器的配置文件的路径。
- **Tags**

  分配给会话的标签。此数据结构包括以下嵌套响应参数：
  - **Key**

    标签键。
  - **Value**

    标签值。

Type

会话的类型。

State

会话的当前状态。可能的值有：

- CREATING-经纪商正在创建会话。
- READY-会话已准备好接受客户端连接。
- DELETING-正在删除会话。
- DELETED-会话已删除。
- UNKNOWN-无法确定会话的状态。经纪人和代理可能无法进行沟通。

CreationTime

创建会话的日期和时间。

LastDisconnectionTime

上次客户端断开连接的日期和时间。

NumOfConnections

用户与会话的并发连接数。

ConnectionToken

用于连接到会话的身份验证令牌。

Example

Python

Request

以下示例获取用户名为user1和 ID 为的会话sessionId12345.

```
def get_session_connection_api():
    api_instance =
 swagger_client.GetSessionConnectionDataApi(swagger_client.ApiClient(get_client_configuration()))
    set_request_headers(api_instance.api_client)
    return api_instance

def get_session_connection_data(session_id, user):
    api_response =
 get_session_connection_api().get_session_connection_data(session_id=session_id,
 user=user)
    print('Get Session Connection Data Response:', api_response)

def main():
    get_session_connection_data('sessionId12345', 'user1')
```

Response

下面是示例输出。

```
{
```

```
    "Session": {
        "Id": "sessionId12345",
        "Name": "a session name",
        "Owner": "an owner 1890",
        "Server": {
            "Ip": "1.1.1.123",
            "Hostname": "server hostname",
            "Port": "1222",
            "WebUrlPath": "/path",
            "Tags": [
                {
                    "Key": "os",
                    "Value": "windows"
                },
                {
                    "Key": "ram",
                    "Value": "4gb"
                }
            ]
        },
        "Type": "VIRTUAL",
        "State": "UNKNOWN",
        "CreationTime": "2020-10-06T10:15:31.633Z",
        "LastDisconnectionTime": "2020-10-06T10:15:31.633Z",
        "NumOfConnections": 2
    },
    "ConnectionToken":
 "EXAMPLEiOiJmOWM1YTRhZi1jZmU0LTQ0ZjEtYjZlOC04ZjY0YjM4ZTE2ZDkiLCJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.
tngiKXevUxhhJvm3BPJYRs9NPE4GCJRTc13EXAMPLEIxNEPPh5IMcVmROfU1WKPnry4ypPTp3rsZ7YWjCTSfs1GoN3R_nLFyAxf
Kqtpd5GH0D-E8FwsedV-
Q2bRQ4y9y1q0MgFU4QjaSMypUuYR0YjkCaoainjmEZew4A33fG40wATrBvoivBiNWdNpytHX2CDOuk_k0k_DWeZjMvv9jF1f5EX
h_GaMgHmltqBIA4jdPD7i0CmC2e7413KFy-
EQ4Ej1cM7RjLwhFuWpKWAVJxogJjYpfoKKaPo4KxvJjJIPYhksck1INQpe2W5rnlxCq7sC7ptcGw17DUobP7egRv9H37VD8SrkL
hK1G4G8erHvl9HIrTR9_c884fNrTCC8DvC062e4KYdLkAhhJmboN9CAGIGFyd2c1AY_CzzvDL0EXAMLE"
}
```


GetSessionScreenshots

获取一个或多个 NICE DCV 会话的屏幕截图。

屏幕截图的图像文件类型和分辨率取决于会话管理器 Broker 配置。要修改图像文件类型，请配置`session-screenshot-format`参数。要修改分辨率，请配置`session-screenshot-max-width`和`session-screenshot-max-height`参数。有关更多信息，请参阅 。代理配置文件中的NICE DCV 会话管理器管理员指南.

主题

请求参数

`SessionId`

要从中获取屏幕截图的 NICE DCV 会话的 ID。

类型: 字符串

：必需 是

响应参数

RequestId

请求的唯一 ID。

SuccessfulList

有关成功屏幕截图的信息。此数据结构包括以下嵌套响应参数：

SessionScreenshot

截图的相关信息。此数据结构包括以下嵌套响应参数：

SessionId

截图从中拍摄的 NICE DCV 会话的 ID。

Images

有关图像的信息。此数据结构包括以下嵌套响应参数：

Format

图像的格式。可能的值包括：`jpeg`和`png`.

Data

屏幕截图图像 base64 编码格式。

CreationTime

截图的日期和时间。

Primary

指示屏幕截图是否为 NICE DCV 会话的主显示屏。

UnsuccessfulList

有关失败的屏幕截图的信息。此数据结构包括以下嵌套响应参数：

GetSesionScreenshotRequestData

失败的原始请求。

SessionId

要从中截取屏幕截图的 NICE DCV 会话的 ID。

FailureReason

失败的原因。

Example

Python

Request

以下示例从两个会话中获取屏幕截图（`sessionId1`和`sessionId2`）。会话`sessionId2`不存在并导致失败。

```
from swagger_client.models.describe_servers_request_data import
 DescribeServersRequestData

def get_sessions_api():
    api_instance =
 swagger_client.ServersApi(swagger_client.ApiClient(get_client_configuration()))
    set_request_headers(api_instance.api_client)
    return api_instance

def get_session_screenshots(session_ids):
    request = [GetSessionScreenshotRequestData(session_id=session_id) for session_id in
 session_ids]
    print('Get Session Screenshots Request:', request)
    api_instance = get_sessions_api()
    api_response = api_instance.get_session_screenshots(body=request)
    print('Get Session Screenshots Response:', api_response)

def main():
    get_session_screenshots(["sessionId1", "sessionId2"])
```

Response

下面是示例输出。

```
{
    "RequestId": "542735ef-f6ab-47d8-90e5-23df31d8d166",
    "SuccessfulList": [
        {
            "SessionScreenshot": {
                "SessionId": "sessionId1",
                "Images": [
                    {
                        "Format": "png",
                        "Data": "iVBORw0KGgoAAAANSUhEUgAAAEXAMPLE",
                        "CreationTime": "2021-03-30T15:47:06.822Z",
                        "Primary": true
                    }
                ]
            }
        }
    ],
    "UnsuccessfulList": [
        {
            "GetSessionScreenshotRequestData": {
                "SessionId": "sessionId2"
            },
            "FailureReason": "Dcv session not found."
        }
    ]
}
```

OpenServers

打开一个或多个 NICE DCV 服务器。在 NICE DCV 服务器上创建 NICE DCV 会话之前，必须将服务器的状态更改为打开. NICE DCV 服务器打开，你可以在服务器上创建 NICE DCV 会话。

主题

请求参数

`ServerId`

要打开的服务器的 ID。

类型: 字符串

：必需 是

响应参数

`RequestId`

请求的唯一 ID。

`SuccessfulList`

有关成功打开的 NICE DCV 服务器的信息。此数据结构包括以下嵌套响应参数：

`ServerId`

成功打开的服务器的 ID。

`UnsuccessfulList`

有关无法打开的 NICE DCV 服务器的信息。此数据结构包括以下嵌套响应参数：

`OpenServerRequestData`

有关失败的原始请求的信息。此数据结构包括以下嵌套响应参数：

`ServerId`

无法打开的 NICE DCV 服务器的 ID。

`FailureCode`

失败的代码。

`FailureReason`

失败的原因。

Example

Python

Request

以下示例打开两个 NICE DCV 服务器（`serverId1`和`serverId2`）。

```
from swagger_client.models import OpenServerRequestData


def get_servers_api():
    api_instance =
 swagger_client.ServersApi(swagger_client.ApiClient(get_client_configuration()))
```

```
    set_request_headers(api_instance.api_client)
    return api_instance

def open_servers(server_ids):
    request = [OpenServerRequestData(server_id=server_id) for server_id in server_ids]
    print('Open Servers Request:', request)
    api_instance = get_servers_api()
    api_response = api_instance.open_servers(body=request)
    print('Open Servers Response:', api_response)

def main():
    open_servers(["serverId1", "serverId2"])
```

Response

下面是示例输出。

```
{
    "RequestId": "1e64830f-0a27-41bf-8147-0f3411791b64",
    "SuccessfulList": [
        {
            "ServerId": "serverId1"
        }
    ],
    "UnsuccessfulList": [
        {
            "OpenServerRequestData": {
                "ServerId": "serverId2"
            },
            "FailureCode": "DCV_SERVER_NOT_FOUND",
            "FailureReason": "Dcv server not found."
        }
    ]
}
```

UpdateSessionPermissions

更新特定 NICE DCV 会话的用户权限。

主题

请求参数

`SessionId`

要更改权限的会话的 ID。

类型: 字符串

: 必需 是

`Owner`

要更改权限的会话的所有者。

类型: 字符串

：必需 是

`PermissionFile`

要使用的权限文件的 base64 编码内容。有关更多信息，请参阅 。配置 NICE DCV 授权中的NICE DCV 管理员指南.

类型: 字符串

：必需 是

响应参数

`SessionId`

会话 ID。

Example

Python

Request

以下示例为的会话 ID 为的会话设置新权限。`SessionId1897`.

```
from swagger_client.models.update_session_permissions_request_data import
 UpdateSessionPermissionsRequestData

def get_session_permissions_api():
    api_instance =
 swagger_client.SessionPermissionsApi(swagger_client.ApiClient(get_client_configuration()))
    set_request_headers(api_instance.api_client)
    return api_instancedef update_session_permissions(session_permissions_to_update):
    update_session_permissions_request = list()
    for session_id, owner, permissions_base64_encoded in session_permissions_to_update:
        a_request = UpdateSessionPermissionsRequestData(
            session_id=session_id, owner=owner,
 permissions_file=permissions_base64_encoded)
        update_session_permissions_request.append(a_request)
    print('Update Session Permissions Request:', update_session_permissions_request)
    api_instance = get_session_permissions_api()
    api_response =
 api_instance.update_session_permissions(body=update_session_permissions_request)
    print('Update Session Permissions Response:', api_response)

def main():
    update_session_permissions([('SessionId1897', 'an owner 1890',
 'file_base64_encoded')])
```

Response

下面是示例输出。

```
{
   'request_id': 'd68ebf66-4022-42b5-ba65-99f89b18c341',
    'successful_list': [
        {'
```

```
            session_id': 'SessionId1897'
        }
    ],
    'unsuccessful_list': []
}
```

NICE DCV 会话管理的发布说明和文档历史记录

本页面提供了 NICE DCV 会话管理的发布说明和文档历史记录。

主题

NICE DCV 会话管理器发行说明

本节概述了 NICE DCV 会话管理器的主要更新、功能版本和错误修复。所有更新都按发布日期组织。我们经常更新文档来处理您发送给我们的反馈意见。

主题

2021.3-11591— 2021 年 12 月 20 日

内部版本号	新 功能
• Broker：307 • 代理：453 • CLI：92	• 添加了对与 NICE DCV 连接网关集成的支持。 • 增加了对 Ubuntu 18.04 和 Ubuntu 20.04 的代理支持。

2021.2-11445— 2021 年 11 月 18 日

内部版本号	更改和错误修复
• 代理：288 • 代理：413 • CLI：54	• 修复了验证包含 Windows 域的登录名的问题。

2021.2-11190— 2021 年 10 月 11 日

内部版本号	更改和错误修复
• 代理：254 • 代理：413 • CLI：54	• 修复了命令行界面中无法启动 Windows 会话的问题。

2021.2-11042— 2021 年 9 月 1 日

内部版本号	新 功能	更改和错误修复
• 代理：254 • 代理：413 • CLI：37	• NICE DCV 会话管理器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 会话管理器引入了经纪商数据持久性 为了提高可用性，经纪商可以在外部数据存储中保存服务器状态信息，并在启动时恢复数据。	• 注册外部授权服务器时，您现在可以指定授权服务器用于签署 JSON 格式的 Web 令牌的算法。通过此更改，你可以使用 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
• 代理：214 • 代理：365	• NICE DCV 会话管理器添加了对传递给 Linux 上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创建会话API。	• 我们修复了 Windows 上自动运行文件的问题。

2021.0-10242— 2021 年 4 月 12 日

内部版本号	更改和错误修复

• 代理：183
• 代理：318
	• NICE DCV 会话管理器推出了以下新 API：
	  • 开放服务器
	  • 关闭服务器
	  • DescribeServers
	  • 获取会话截图
	• 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-`

内部版本号	更改和错误修复
	size, 和create-sessions-queue-max-time-seconds. • 客服参数：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
• 代理：114 • 代理：211	• 我们修复了阻止经纪商启动的自动生成的 TLS 证书的问题。

2020.2-9508— 2020 年 11 月 11 日

内部版本号	更改和错误修复
• 代理：78 • 代理：183	• NICE DCV 会话管理器的初始版本。

文档历史记录

下表介绍此版本的 NICE DCV 会话管理的文档。

更改	描述	日期
不错的 DCV 版本 2021.3	NICE DCV 会话管理器已更新 NICE DCV 2021.3。有关更多信息，请参阅2021.3-11591— 2021 年 12 月 20 日 (p. 35)。	2021 年 12 月 20 日
不错的 DCV 版本 2021.2	NICE DCV 会话管理器已针对 NICE DCV 2021.2 进行了更新。有关更多信息，请参阅2021.2-11042— 2021 年 9 月 1 日 (p. 36)。	2021 年 9 月 1 日
不错的 DCV 版本 2021.1	NICE DCV 会话管理器已针对 NICE DCV 2021.1 进行了更新。有关更多信息，请参阅2021.1-10557— 2021 年 5 月 31 日 (p. 36)。	2021 年 5 月 31 日
不错的 DCV 版本 2021.0	NICE DCV 会话管理器已更新为与 NICE DCV 版本 2021.0 兼容。有关更多信息，请参阅2021.0-10242— 2021 年 4 月 12 日 (p. 36)。	2021 年 4 月 12 日

更改	描述	日期
NICE DCV 会话管理的初始版本。	此内容的第一版。	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。