NICE DCV Session Manager

Developer Guide

NICE DCV Session Manager: Developer Guide

Table of Contents

What is NICE DCV Session Manager?

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

This guide explains how to use the Session Manager APIs to manage the lifecycle of NICE DCV sessions. For more information about how to install and configure the Session Manager Broker and Agents, see the *NICE DCV Session Manager Administrator Guide*.

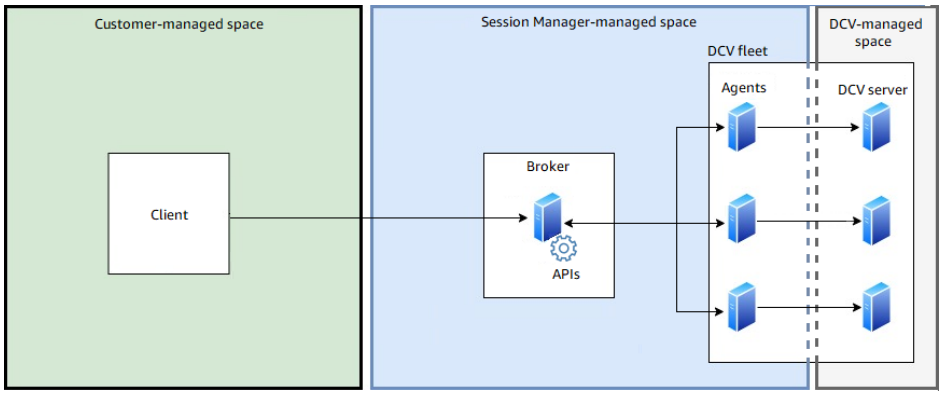
Prerequisites

Before you start working with the Session Manager APIs, ensure that you're familiar with NICE DCV and NICE DCV sessions. For more information, see the NICE DCV Administrator Guide.

Topics

How Session Manager works

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



Broker

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

Agent

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

APIs

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

Client

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

Access token

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

Client API

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

NICE DCV session

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

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

Features

DCV Session Manager offers the following features:

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

Getting started

The section explains how to get started with the Session Manager APIs.

In this section, we'll show you how to do this by using the `DescribeSessions` API as an example.

Topics

Generating the client API

The Session Manager APIs are defined in a single YAML file. The APIs are based on the OpenAPI3.0 specification, which defines a standard, language-agnostic interface to RESTful APIs. For more information, see OpenAPI Specification.

Using the YAML file, you can generate API clients in one of the supported languages. To do this, you must use Swagger Codegen 3.0 or later. For more information about the supported languages, see the swagger-codegen repo.

To generate the API client

1. Download the Session Manager API YAML file from the Session Manager Broker. The YAML file is available at the following URL.

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

2. Install Swagger Codegen.

   - macOS

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

   - Other platforms

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

     ```
     $ cd swagger-codegen
     ```

3. Generate the API client.

   - macOS

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

   - Other platforms

     ```
     $ 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
```

Register your API client

To make an API request, you must first retrieve an access token from the Broker. To get an access token from the Broker, you must supply the Broker with the credentials for your client API. The credentials are based on a client ID and client password that are generated when your client is registered with the Broker. If you don't have a client ID and client password for your client, you must request them from your Broker administrator. For more information about registering your client API with the Broker and getting a client ID and password, see register-api-client.

Get an access token and make an API request

First we import the models needed for the application.

Then we declare variables for the client ID (`__CLIENT_ID`), client password (`__CLIENT_SECRET`), and the Broker URL, including the port number (`__PROTOCOL_HOST_PORT`).

Next, we create a function called `build_client_credentials` that generates the client credentials. To generate the client credentials, you must first concatenate the client ID and client password and separate the values with a colon (*`client_ID`*`:`*`client_password`*), and then Base64 encode the entire string.

```
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')
```

Now that we have our client credentials, we can use it to request an access token from the Broker. To do this, we create a function called `get_access_token`. You must call a `POST` on `https://`*`Broker_IP`*`:`*`8443`*`/oauth2/token?grant_type=client_credentials`, and provide an authorization header, which includes the Basic-encoded client credentials, and a content type of `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
```

Now, we create the functions needed to instantiate a client API. To instantiate a client API, you must specify the client configuration and the headers to be used for requests. The `get_client_configuration` function creates a configuration object that includes the Broker's IP address and port and the path to Broker's self-signed certificate, which you should have received from the Broker administrator. The `set_request_headers` function creates a request header object that includes the client credentials and the access token.

```
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
```

Finally, we create a main method that calls the `DescribeSessions` API. For more information, see DescribeSessions (p. 21).

```
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'}])
```

Session Manager API reference

This section provides descriptions, syntax, and usage examples for each of the Session Manager API actions.

Topics

CloseServers

Closes one or more NICE DCV servers. When you close a NICE DCV server, you make it unavailable for NICE DCV session placement. You cannot create NICE DCV sessions on *closed* servers. Closing a server ensures that no sessions are running on it and that users cannot create new sessions on it.

Topics

Request parameters

`ServerId`

The ID of the server to close.

Type: String

Required: Yes

`Force`

Forces the close operation. If you specify `true`, the server is closed even if it has running sessions. The sessions continue to run.

Type: Boolean

Required: No

Response parameters

`RequestId`

The unique ID of the request.

`SuccessfulList`

Information about the NICE DCV servers that were successfully closed. This data structure includes the following nested response parameter:

`ServerId`

The ID of the server that was successfully closed.

`UnsuccessfulList`

Information about the NICE DCV servers that could not be closed. This data structure includes the following nested response parameters:

`CloseServerRequestData`

Information about the original request that failed. This data structure includes the following nested response parameter:

`ServerId`

The ID of the NICE DCV server that could not be closed.

`Force`

The requested force parameter.

`FailureCode`

The code of the failure.

`FailureReason`

The reason for the failure.

Example

Python

Request

The following example closes two NICE DCV servers (`serverId1` and `serverId2`). Server `serverId2` doesn't exist and results in a failure.

```
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

The following is the sample output.

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

CreateSessions

Creates a new NICE DCV session with the specified details.

API actions

- Request parameters (p. 4)
- Response parameters (p. 8)
- Example (p. 8)

Request parameters

`Name`

The name for the session.

Type: String

Required: Yes

`Owner`

The name of the session owner. This must be the name of an existing user on the target NICE DCV server.

Type: String

Required: Yes

`Type`

The session type. For more information about the types of sessions, see Introduction to NICE DCV Sessions in the *NICE DCV Administrator Guide*.

Valid values: CONSOLE | VIRTUAL

Type: String

Required: Yes

`InitFile`

Supported with virtual sessions on Linux NICE DCV servers. It is not supported with console sessions on Windows and Linux NICE DCV servers. The path to custom script on the NICE DCV server to run for initializing the session when it is created. The file path is relative to the init directory specified for the `agent.init_folder` Agent configuration parameter. If the file is in the specified init directory, specify the file name only. If the file is not in the specified init directory, specify the relative path. For more information, see see Agent configuration file in the *NICE DCV Session Manager Administrator Guide*.

Type: String

Required: No

`MaxConcurrents`

The maximum number of concurrent NICE DCV clients.

Type: Integer

Required: No

`DcvGlEnabled`

Indicates whether the virtual session is configured to use hardware-based OpenGL. Supported with virtual sessions only. This parameter is not supported with Windows NICE DCV servers.

Valid values: true | false

Type: Boolean

Required: No

`PermissionsFile`

The Base64-encoded contents of the permissions file. Defaults to the server defaults if omitted. For more information, see Configuring NICE DCV Authorization in the *NICE DCV Administrator Guide*.

Type: String

Required: No

`EnqueueRequest`

Indicates whether to queue the request if it can't be immediately fulfilled.

Type: Boolean

Default: false

Required: No

`AutorunFile`

Supported with console sessions on Windows NICE DCV servers and virtual sessions on Linux NICE DCV servers. It is not supported with console sessions on Linux NICE DCV servers.

The path to a file on the host server that is to be run inside the session. The file path is relative to the autorun directory specified for the `agent.autorun_folder` Agent configuration parameter. If the file is in the specified autorun directory, specify the file name only. If the file is not in the specified autorun directory, specify the relative path. For more information, see Agent configuration file in the *NICE DCV Session Manager Administrator Guide*.

The file is run on behalf of the specified **Owner**. The specified owner must have permission to run the file on the server. On Windows NICE DCV servers, the file is run when the owner logs into the session. On Linux NICE DCV servers, the file is run when the session is created.

Type: String

Required: No

`AutorunFileArguments`

Supported with virtual sessions on Linux NICE DCV servers. It is not supported in console sessions on Windows and Linux NICE DCV servers. Command-line arguments passed to **AutorunFile** upon its execution inside the session. Arguments are passed in the order they appear into the given array. Maximum allowed number of arguments and maximum allowed length of each argument can be configured. For more information, see Broker configuration file in the *NICE DCV Session Manager Administrator Guide*.

Type: Array of strings

Required: No

`DisableRetryOnFailure`

Indicates whether to not retry the create session request after it fails on a NICE DCV host for any reason. For more information about create session retry mechanism, see Broker configuration file in the *NICE DCV Session Manager Administrator Guide*.

Type: Boolean

Default: false

Required: No

`Requirements`

The requirements that the server must satisfy in order to place the session. The requirements can include server tags and/or server properties, both server tags and server properties are retrieved by calling the **DescribeServers** API.

Requirements condition expressions:

- *a* **!=** *b* true if *a* is not equal to *b*
- *a* **=** *b* true if *a* is equal to *b*
- *a* **>** *b* true if *a* is greater than *b*
- *a* **>=** *b* true if *a* is greater than or equal to *b*
- *a* **<** *b* true if *a* is less than *b*
- *a* **<=** *b* true if *a* is less than or equal to *b*
- *a* **=** *b* true if *a* contains the string *b*

Requirements boolean operators:

- *a* **and** *b* true if *a* and *b* are true
- *a* **or** *b* true if *a* or *b* are true
- **not** *a* true if *a* is false

The tag keys must be prefixed by `tag:`, the server properties must be prefixed by `server:`.The requirements expressions supports parenthesis `()`.

Requirements examples:

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

Numerical values can be specified using the exponential notation, for example: `"server:Host.Memory.TotalBytes > 1024E6"`.

The supported server properties are:

- `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`

Type: String

Required: No

`StorageRoot`

Specifies the path to the folder used for session storage. For more information about the NICE DCV session storage, see Enabling Session Storage in the *NICE DCV Administrator Guide*.

Type: String

Required: No

Response parameters

`Id`

The unique ID of the session.

`Name`

The session name.

`Owner`

The session owner.

`Type`

The type of session.

`State`

The state of the session. If the request completes successfully, the session enters the `CREATING` state.

Example

Python

Request

The following example creates three sessions.

```
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

The following is the sample output.

```
{
              "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

Describes one or more NICE DCV servers.

Topics

Request parameters

`ServerIds`

The IDs of the NICE DCV servers to describe. If no IDs are specified, all servers are returned in paginated output.

Type: Array of strings

Required: No

`NextToken`

The token to use to retrieve the next page of results.

Type: String

Required: No

`MaxResults`

The maximum number of results to be returned by the request in paginated output. When this parameter is used, the request returns only the specified number of results in a single page along with a `NextToken` response element. The remaining results of the initial request can be seen by sending another request with the returned `NextToken` value.

Valid range: 1 - 1000

Default: 1000

Type: Integer

Required: No

Response parameters

`RequestId`

The unique ID of the request.

`Servers`

Information about the NICE DCV servers. This data structure includes the following nested response parameters:

`Id`

The unique ID of the NICE DCV server.

`Ip`

The IP address of the NICE DCV server.

`Hostname`

The hostname of the NICE DCV server.

`Endpoints`

Information about the NICE DCV server endpoints. This data structure includes the following nested response parameters:

`Port`

The port of the server endpoint.

`WebUrlPath`

The web URL path of the server endpoint. Available for the HTTP protocol only.

`Protocol`

The protocol used by the server endpoint. Possible values include:

- `HTTP` — The endpoint uses the WebSocket (TCP) protocol.

- `QUIC` — The endpoint uses the QUIC (UDP) protocol.

`Version`

The version of the NICE DCV server.

`SessionManagerAgentVersion`

The version Session Manager Agent running on the NICE DCV server.

`Availability`

The availability of the NICE DCV server. Possible values include:

- `AVAILABLE` — The server is available and ready for session placement.
- `UNAVAILABLE` — The server is unavailable and can't accept session placement.

`UnavailabilityReason`

The reason for the NICE DCV server's unavailability. Possible values include:

- `SERVER_FULL` — The NICE DCV server has reached the maximum number of concurrent sessions that it can run.
- `SERVER_CLOSED` — The NICE DCV server has been made unavailable using the **CloseServer** API.
- `UNREACHABLE_AGENT` — The Session Manager Broker can't communicate with the Session Manager Agent on the NICE DCV server.
- `UNHEALTHY_DCV_SERVER` — The Session Manager Agent can't communicate with the NICE DCV server.
- `EXISTING_LOGGED_IN_USER` — (Windows NICE DCV servers only) A user is currently logged in to the NICE DCV server using RDP.
- `UNKNOWN` — The Session Manager Broker is unable to determine the reason.

`ConsoleSessionCount`

The number of console sessions on the NICE DCV server.

`VirtualSessionCount`

The number of virtual sessions on the NICE DCV server.

`Host`

Information about the host server on which the NICE DCV server is running. This data structure includes the following nested response parameters:

`Os`

Information about host server's operating system. This data structure includes the following nested response parameters:

`Family`

The operating system family. Possible values include:

- `windows` — The host server is running a Windows operating system.
- `linux` — The host server is running a Linux operating system.

`Name`

The name of the operating system.

`Version`

The version of the operating system.

`KernelVersion`

(Linux only) The kernel version of the operating system.

BuildNumber

(Windows only) The build number of the operating system.

Memory

Information about the host server's memory. This data structure includes the following nested response parameters:

TotalBytes

The total memory, in bytes, on the host server.

UsedBytes

The used memory, in bytes, on the host server.

Swap

Information about the host server's swap file. This data structure includes the following nested response parameters:

TotalBytes

The total swap file size, in bytes, on the host server.

UsedBytes

The used swap file size, in bytes, on the host server.

Aws

Only for NICE DCV servers running on an Amazon EC2 instance. AWS-specific information. This data structure includes the following nested response parameters:

Region

The AWS Region of the Amazon EC2 instance.

Ec2InstanceType

The type of Amazon EC2 instance.

Ec2InstanceId

The ID of the Amazon EC2 instance.

Ec2ImageId

The ID of the Amazon EC2 image.

CpuInfo

Information about the host server's CPUs. This data structure includes the following nested response parameters:

Vendor

The vendor of the host server's CPU.

ModelName

The model name of the host server's CPU.

Architecture

The architecture of the host server's CPU.

NumberOfCpus

The number of CPUs on the host server.

PhysicalCorePerCpu

The number of CPU cores per CPU.

CpuLoadAverage

Information about the host server's CPU load. This data structure includes the following nested response parameters:

OneMinute

The average CPU load over the last 1-minute period.

FiveMinutes

The average CPU load over the last 5-minute period.

FifteenMinutes

The average CPU load over the last 15-minute period.

Gpus

Information about the host server's GPUs. This data structure includes the following nested response parameters:

Vendor

The vendor of the host server's GPU.

ModelName

The model name of the host server's GPU.

LoggedInUsers

The users that are currently logged in to the host server. This data structure includes the following nested response parameter:

Username

The user name of the logged in user.

Tags

The tags assigned to the server. This data structure includes the following nested response parameters:

Key

The tag key.

Value

The tag value.

Example

Python

Request

The following example describes all available NICE DCV servers. The results are paginated to show two results per page.

```
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

The following is the sample output.

```
{
    "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

Describes one or more NICE DCV sessions.

Topics

Request parameters

`SessionIds`

The IDs of the sessions to describe.

Type: String

Required: No

`NextToken`

The token to use to retrieve the next page of results.

Type: String

Required: No

`Filters`

Additional filters to apply to the request. Supported filters include:

- tag:key—The tags assigned to the session.
- owner—The session owner.

Type: String

Required: No

Response parameters

`Id`

The unique ID of the session.

`Name`

The name of the session.

`Owner`

The owner of the session.

`Server`

Information about the server on which the session is running. This data structure includes the following nested response parameters:

- **`Ip`**

The IP address of the NICE DCV server host.

- **`Hostname`**

  The hostname of the NICE DCV server host.

- **`Port`**

  The port over which the NICE DCV server communicates with NICE DCV clients.

- **`Tags`**

  The tags assigned to the session. This data structure includes the following nested response parameters:

  - **`Key`**

    The tag key.

  - **`Value`**

    The tag value.

`Type`

The type of session.

`State`

The current state of the session. Possible values are:

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

`CreationTime`

The date and time the session was created.

`LastDisconnectionTime`

The date and time of the last client disconnection.

`NumOfConnections`

The number of active client connections.

`StorageRoot`

Specifies the path to the folder used for session storage. For more information about the NICE DCV session storage, see Enabling Session Storage in the *NICE DCV Administrator Guide*.

Type: String

Required: No

Example

Python

Request

The following example describes sessions that are owned by `user1` and have a tag of `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

The following is the sample output.

```
{
    "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

Deletes the specified NICE DCV session and removes it from the Broker's cache.

Topics

Request parameters

`SessionId`

The ID of the session to delete.

Type: String

Required: Yes

`Owner`

The owner of the session to delete.

Type: String

Required: Yes

`Force`

Removes a session from the Broker's cache with attempting to delete it from the NICE DCV server. This is useful for removing outdated sessions from the Broker's cache. For example, if a NICE DCV server was stopped, but the sessions are still registered on the Broker, use this flag to purge the sessions from the Broker's cache.

Keep in mind that if the session is still active, it is re-cached by the Broker.

Valid values: `true` | `false`

Type: Boolean

Required: No

Response parameters

`SessionId`

The ID of the session

`State`

Only returned if the sessions were successfully deleted. Indicates the current state of the session. If the request completes successfully, the session transitions to the `DELETING` state. It could take a few minutes for the session to be deleted. When it has been deleted, the state transitions from `DELETING` to `DELETED`.

`FailureReason`

Only returned if some sessions could not be deleted. Indicates why the session could not be deleted.

Example

Python

Request

The following example deletes two sessions—a session with an ID of `SessionId123` that is owned by `user1`, and a session with an ID of `SessionIdabc` that is owned by `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

The following is the sample output. `SessionId123` was successfully deleted, while `SessionIdabc` could not be deleted.

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

GetSessionConnectionData

Gets connection information for a specific user's connection to a specific NICE DCV session.

Topics

Request parameters

`SessionId`

The ID of the session for which to view connection information.

Type: String

Required: Yes

`User`

The name of the user for which to view connection information.

Type: String

Required: Yes

Response parameters

`Id`

The unique ID of the session.

`Name`

The name of the session.

`Owner`

The owner of the session.

`Server`

Information about the server on which the session is running. This data structure includes the following nested response parameters:

- **`Ip`**

  The IP address of the NICE DCV server host.
- **`Hostname`**

  The hostname of the NICE DCV server host.
- **`Port`**

  The port over which the NICE DCV server communicates with NICE DCV clients.
- **`WebUrlPath`**

  The path to the NICE DCV server's configuration file.
- **`Tags`**

  The tags assigned to the session. This data structure includes the following nested response parameters:
  - **`Key`**

    The tag key.
  - **`Value`**

    The tag value.

`Type`

The type of session.

`State`

The current state of the session. Possible values are:

- `CREATING` - the Broker is in the process of creating the session.

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

`CreationTime`

The date and time the session was created.

`LastDisconnectionTime`

The date and time of the last client disconnection.

`NumOfConnections`

The number of concurrent connections the user has to the session.

`ConnectionToken`

The authentication token used to connect to the session.

Additional info

The information obtained from this API can be passed to a NICE DCV client in order to connect to the NICE DCV session.

In the case of the NICE DCV Web client, you can build an URL that can be opened in the browser. The URL has the following format:

```
https://{Ip}:{Port}{WebUrlPath}?authToken={ConnectionToken}#{SessionId}.
```

In the case of the NICE DCV native client, you can build an URL with the `dcv://` schema. When the NICE DCV native client is installed, it registers itself with the system as the handler for `dcv://` URLs. The URL has the following format:

```
dcv://{Ip}:{Port}{WebUrlPath}?authToken={ConnectionToken}#{SessionId}.
```

Note
If you're using Amazon EC2, the IP address should be the public one. If your configuration has NICE DCV hosts behind a gateway, specify the gateway address rather than the one returned by the SessionConnectionData API.

Example

Python

Request

The following example gets connection information for a user with a user name of `user1` and a session with an ID of `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_url_to_connect(api_response):
    ip_address = api_response.session.server.ip
    port = api_response.session.server.port
    web_url_path = api_response.session.server.web_url_path
    connection_token = api_response.connection_token
    session_id = api_response.session.id
    url = f'https://{ip_address}:{port}{web_url_path}?
authToken={connection_token}#{session_id}'
    return url


def get_session_connection_data(session_id, user):
    api_response =
 get_session_connection_api().get_session_connection_data(session_id=session_id,
 user=user)
    url_to_connect = get_url_to_connect(api_response)
    print('Get Session Connection Data Response:', api_response)
    print('URL to connect: ', url_to_connect)


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

Response

The following is the sample output.

```
{
    "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

Gets screenshots of one or more NICE DCV sessions.

The image file type and resolution of the screenshot depends on the Session Manager Broker configuration. To modify the image file type, configure the `session-screenshot-format` parameter. To modify the resolution, configure the `session-screenshot-max-width` and `session-screenshot-max-height` parameters. For more information, see Broker configuration file in the *NICE DCV Session Manager Administrator Guide*.

Topics

Request parameters

`SessionId`

The ID of the NICE DCV session from which to get the screenshot.

Type: String

Required: Yes

Response parameters

`RequestId`

The unique ID of the request.

`SuccessfulList`

Information about the successful screenshots. This data structure includes the following nested response parameters:

`SessionScreenshot`

Information about the screenshots. This data structure includes the following nested response parameters:

`SessionId`

The ID of the NICE DCV session from which the screenshot was taken.

`Images`

Information about the images. This data structure includes the following nested response parameters:

Format

The format of the image. Possible values include: `jpeg` and `png`.

Data

The screenshot image base64 encoded format.

CreationTime

The date and time the screenshot was taken.

Primary

Indicates whether the screenshot is of the NICE DCV session's primary display.

UnsuccessfulList

Information about the unsuccessful screenshots. This data structure includes the following nested response parameters:

GetSesionScreenshotRequestData

The original request that failed.

SessionId

The ID of the NICE DCV session from which the screenshot was to be taken.

FailureReason

The reason for the failure.

Example

Python

Request

The following example gets screenshots from two sessions (`sessionId1` and `sessionId2`). Session `sessionId2` doesn't exist and results in a failure.

```
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

The following is the sample output.

```
{
    "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

Opens one or more NICE DCV servers. Before you can create NICE DCV sessions on a NICE DCV server, you must change the server's state to *open*. After the NICE DCV server is *open*, you can create NICE DCV sessions on the server.

Topics

Request parameters

`ServerId`

The ID of the server to open.

Type: String

Required: Yes

Response parameters

`RequestId`

The unique ID of the request.

SuccessfulList

Information about the NICE DCV servers that were successfully opened. This data structure includes the following nested response parameter:

ServerId

The ID of the server that was successfully opened.

UnsuccessfulList

Information about the NICE DCV servers that could not be opened. This data structure includes the following nested response parameters:

OpenServerRequestData

Information about the original request that failed. This data structure includes the following nested response parameter:

ServerId

The ID of the NICE DCV server that could not be opened.

FailureCode

The code of the failure.

FailureReason

The reason for the failure.

Example

Python

Request

The following example opens two NICE DCV servers (`serverId1` and `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

The following is the sample output.

```
{
    "RequestId": "1e64830f-0a27-41bf-8147-0f3411791b64",
```

```
    "SuccessfulList": [
        {
            "ServerId": "serverId1"
        }
    ],
    "UnsuccessfulList": [
        {
            "OpenServerRequestData": {
                "ServerId": "serverId2"
            },
            "FailureCode": "DCV_SERVER_NOT_FOUND",
            "FailureReason": "Dcv server not found."
        }
    ]
}
```

UpdateSessionPermissions

Updates the user permissions for a specific NICE DCV session.

Topics

Request parameters

`SessionId`

The ID of the session for which to change the permissions.

Type: String

Required: Yes

`Owner`

The owner of the session for which to change the permissions.

Type: String

Required: Yes

`PermissionFile`

The Base64-encoded content of the permissions file to use. For more information, see Configuring NICE DCV Authorization in the *NICE DCV Administrator Guide*.

Type: String

Required: Yes

Response parameters

`SessionId`

The ID of the session.

Example

Python

Request

The following example sets new permissions for a session with a session ID of `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

The following is the sample output.

```
{
   'request_id': 'd68ebf66-4022-42b5-ba65-99f89b18c341',
    'successful_list': [
        {'
            session_id': 'SessionId1897'
        }
    ],
    'unsuccessful_list': []
}
```

Release notes and document history for NICE DCV Session Manager

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

Topics

NICE DCV Session Manager release notes

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

Topics

2022.0-11952— February 23, 2022

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

2021.3-11591— December 20, 2021

Build numbers	New features
• Broker: 307 • Agent: 453	• Added support for integrating with the NICE DCV Connection Gateway.

Build numbers	New features
• CLI: 92	• Added Broker support for Ubuntu 18.04 and Ubuntu 20.04.

2021.2-11445— November 18, 2021

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

2021.2-11190— October 11, 2021

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

2021.2-11042— September 01, 2021

Build numbers	New features	Changes and bug fixes
• Broker: 254 • Agent: 413 • CLI: 37	• NICE DCV Session Manager now offers command line interface (CLI) support. You can create and manage NICE DCV sessions in the CLI, instead of calling APIs. • NICE DCV Session Manager introduced Broker data persistence. For higher availability, brokers can persist server state information on an external data store and restore the data at startup.	• When registering an external authorization server, you can now specify the algorithm that the authorization server uses to sign JSON-formatted Web Tokens. With this change, you can use Azure AD as an external authorization server.

2021.1-10557— May 31, 2021

Build numbers	New features	Changes and bug fixes
• Broker: 214 • Agent: 365	• NICE DCV Session Manager added support for input parameters passed to the autorun file on Linux.	• We fixed a problem with the autorun file on Windows.

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

2021.0-10242— April 12, 2021

Build numbers	Changes and bug fixes

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

2020.2-9662— December 04, 2020

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

2020.2-9508— November 11, 2020

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

Document history

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

Change	Description	Date
NICE DCV Version 2022.0	NICE DCV Session Manager has been updated for NICE DCV 2022.0. For more information, see 2022.0-11952— February 23, 2022 (p. 37).	February 23, 2022
NICE DCV Version 2021.3	NICE DCV Session Manager has been updated for NICE DCV 2021.3. For more information, see 2021.3-11591— December 20, 2021 (p. 37).	December 20, 2021
NICE DCV Version 2021.2	NICE DCV Session Manager has been updated for NICE DCV 2021.2. For more information, see 2021.2-11042— September 01, 2021 (p. 38).	September 01, 2021
NICE DCV Version 2021.1	NICE DCV Session Manager has been updated for NICE DCV 2021.1. For more information, see 2021.1-10557— May 31, 2021 (p. 38).	May 31, 2021
NICE DCV Version 2021.0	NICE DCV Session Manager was updated to be compatible with NICE DCV version 2021.0. For more information, see 2021.0-10242— April 12, 2021 (p. 39).	April 12, 2021
Initial release of NICE DCV Session Manager	The first publication of this content.	November 11, 2020