aws

Guide du développeur

# NICEDCVGestionnaire de sessions

# NICEDCVGestionnaire de sessions: Guide du développeur

# Table of Contents

# Qu'est-ce que le gestionnaire de session NICE DCV ?

Le gestionnaire de session NICE DCV est un ensemble de progiciels installables (un agent et un courtier) et une interface de programmation d'applications (API) qui permet aux développeurs et aux éditeurs de logiciels indépendants (ISV) de créer facilement des applications frontales qui créent et gèrent par programmation le cycle de vie des sessions NICE DCV sur un parc de serveurs NICE DCV.

Ce guide explique comment utiliser les API du gestionnaire de session pour gérer le cycle de vie des sessions NICE DCV. Pour plus d'informations sur l'installation et la configuration du Session Manager Broker and Agents, consultez le guide de l'administrateur du gestionnaire de session NICE DCV.

Prérequis

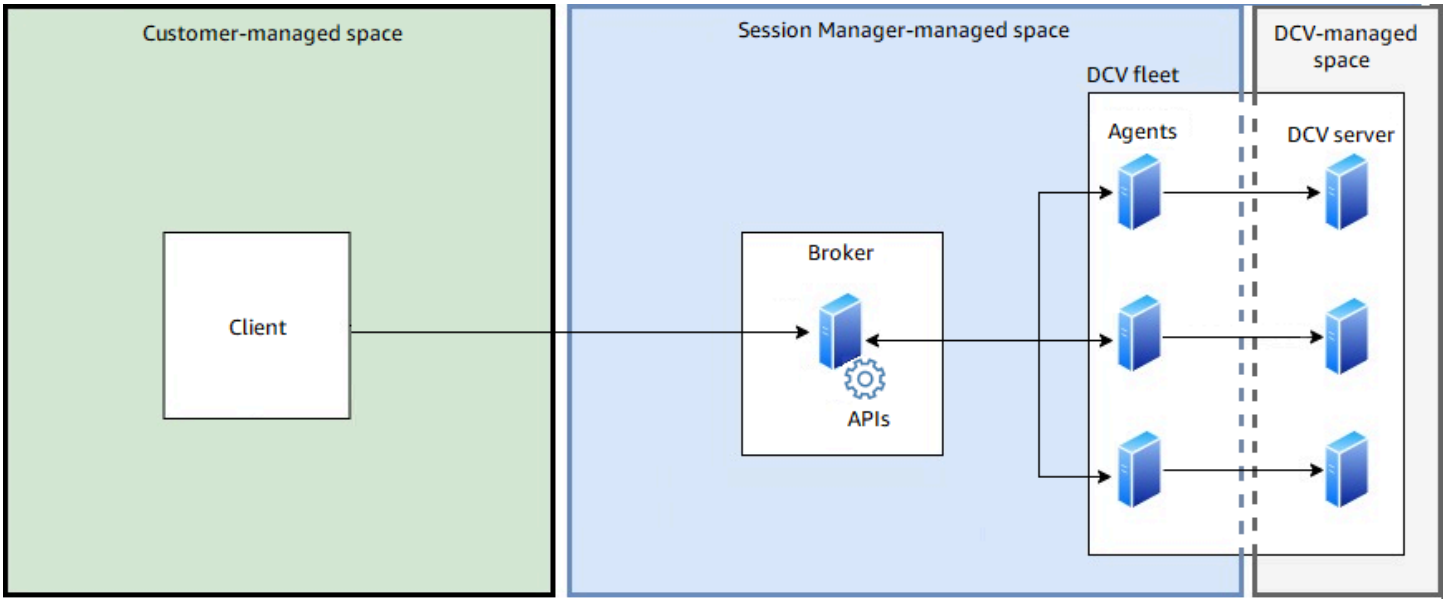
Avant de commencer à utiliser les API du gestionnaire de sessions, assurez-vous de bien connaître les sessions NICE DCV et NICE DCV. Pour plus amples informations, veuillez consulter le Guide de l'administrateur NICE DCV pour plus amples informations, veuillez consulter

Rubriques



# Fonctionnement du Gestionnaire de session

Le diagramme suivant illustre les composants principaux de Session Manager.




Agent

Le Broker est un serveur Web qui héberge et expose les API du gestionnaire de session. Il reçoit et traite les demandes d'API pour gérer les sessions NICE DCV du client, puis transmet les instructions aux agents concernés. Le Broker doit être installé sur un hôte distinct de vos serveurs NICE DCV, mais il doit être accessible au client et doit pouvoir accéder aux agents.

Agent

L'agent est installé sur chaque serveur NICE DCV de la flotte. Les Agents reçoivent des instructions du Broker et les exécutent sur leurs serveurs NICE DCV respectifs. Les agents surveillent également l'état des serveurs NICE DCV et envoient des mises à jour périodiques au courtier.

API

Session Manager expose un ensemble d'interfaces de programmation d'applications (API) REST qui peuvent être utilisées pour gérer les sessions NICE DCV sur un parc de serveurs NICE DCV. Les API sont hébergées et exposées par le Broker. Les développeurs peuvent créer des clients de gestion de session personnalisés qui appellent les API.

Client

Le client est l'application frontale ou le portail que vous développez pour appeler les API du gestionnaire de session exposées par le Broker. Les utilisateurs finaux utilisent le client pour gérer les sessions hébergées sur les serveurs NICE DCV de la flotte.

Jeton d'accès

Pour effectuer une demande d'API, vous devez fournir un jeton d'accès. Les jetons peuvent être demandés au Broker ou à un serveur d'autorisation externe par les API clientes enregistrées. Pour demander un jeton et y accéder, l'API client doit fournir des informations d'identification valides.

API client

L'API client est générée à partir du fichier YAML de définition de l'API Session Manager, à l'aide de Swagger Codegen. L'API client est utilisée pour effectuer des demandes d'API.

Séance NICE DCV

Vous devez créer une session NICE DCV sur votre serveur NICE DCV à laquelle vos clients peuvent se connecter. Les clients ne peuvent se connecter à un serveur NICE DCV que s'il existe une session active. NICE DCV prend en charge les consoles et les sessions virtuelles. Vous utilisez les API du gestionnaire de sessions pour gérer le cycle de vie des sessions NICE DCV. NICE DCV de l'un des états suivants :

- `CREATING`—Le courtier est en train de créer la session.
- `READY`—la session est prête à accepter les connexions client.
- `DELETING`—la session est en cours de suppression.
- `DELETED`—la session a été supprimée.
- `UNKNOWN`: impossible de déterminer l'état de la session. Il se peut que le courtier et l'agent ne puissent pas communiquer.

# Fonctions

Le Gestionnaire de l'peut avoir les fonctions suivantes :

- Fournit des informations sur les sessions NICE DCV : permet d'obtenir des informations sur les sessions exécutées sur plusieurs serveurs NICE DCV.
- Gérez le cycle de vie de plusieurs sessions NICE DCV : créez ou supprimez plusieurs sessions pour plusieurs utilisateurs sur plusieurs serveurs NICE DCV à l'aide d'une seule demande d'API.
- Supporte les balises : utilisez des balises personnalisées pour cibler un groupe de serveurs NICE DCV lors de la création de sessions.

- Gère les autorisations pour plusieurs sessions NICE DCV : modifiez les autorisations utilisateur pour plusieurs sessions à l'aide d'une seule demande d'API.
- Fournit des informations de connexion : permet de récupérer les informations de connexion client pour les sessions NICE DCV.
- Supports pour le cloud et sur site : utilisez le gestionnaire de session sur siteAWS, sur site ou avec d'autres serveurs basés sur le cloud.

# Commencer à utiliser le gestionnaire de session API

Le gestionnaire de NICE DCV session API fournit une interface automatisée pour gérer les sessions de bureau à distance. Grâce à celaAPI, les développeurs peuvent créer, répertorier, démarrer, arrêter et contrôler DCV des sessions par programmation. Cela permet d'intégrer des NICE DCV fonctionnalités dans des applications et des flux de travail personnalisés. En tirant parti de celaAPI, les entreprises peuvent rationaliser la gestion des charges de travail de visualisation à distance, en automatisant de nombreuses tâches courantes.

Avant de pouvoir passer des appels vers le NICE DCVAPI, vous devez obtenir un jeton d'accès qui authentifie votre application et l'autorise à accéder aux ressources nécessaires. Il NICE DCV API utilise la OAuth version 2.0 pour l'authentification, vous devrez donc enregistrer votre application et récupérer les informations d'identification nécessaires. Une fois que vous avez votre jeton d'accès, vous pouvez commencer à envoyer des demandes aux NICE DCV API points de terminaison pour commencer à traiter les données.

Rubriques



# Étape 1 : Générez votre API client

Le gestionnaire de APIs session est défini dans un seul YAML fichier. Ils APIs sont basés sur la spécification Open API3 .0, qui définit une interface standard indépendante de la langue pour. RESTful APIs Pour plus d'informations, consultez Open API Specification.

À l'aide du YAML fichier, vous pouvez générer un API client dans l'une des langues prises en charge. Pour ce faire, vous devez utiliser Swagger Codegen 3.0 ou version ultérieure. Pour plus d'informations sur les langues prises en charge, consultez le dépôt swagger-codegen.

Pour générer le API client

1. Téléchargez le API YAML fichier du gestionnaire de session depuis le courtier du gestionnaire de session. Le YAML fichier est disponible à l'adresse suivanteURL.

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

2. Installez Swagger Codegen.

   - macOS

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

   - Autres plateformes

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

     ```
     $ cd swagger-codegen
     ```

3. Générez le API client.

   - macOS

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

   - Autres plateformes

     ```
     $ 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
     ```

# Étape 2 : Enregistrez votre client API

APIles demandes utilisent un jeton d'accès pour vérifier vos informations d'identification. Ces informations d'identification sont basées sur un identifiant client et un mot de passe client générés lorsque votre client est inscrit auprès du courtier.

Pour accéder à ce jeton, vous devez vous inscrire auprès du courtier. register-api-clientÀ utiliser pour enregistrer le clientAPI.

Si vous n'avez pas d'identifiant client ni de mot de passe client pour votre client, vous devez les demander à l'administrateur de votre courtier.

# Étape 3 : Obtenir un jeton d'accès et faire une API demande

Cet exemple décrit les étapes à suivre pour configurer votre jeton d'accès, puis vous montre comment effectuer une API demande de base. Cela vous donnera les connaissances de base nécessaires pour commencer à créer des applications plus avancées basées sur le NICE DCVAPI.

Dans cet exemple, nous allons vous montrer comment procéder à l'aide du `DescribeSessions`API.

Example

Nous importons d'abord les modèles nécessaires à l'application.

Ensuite, nous déclarons des variables pour l'ID du client (`__CLIENT_ID`), le mot de passe du client (`__CLIENT_SECRET`) et le BrokerURL, y compris le numéro de port (`__PROTOCOL_HOST_PORT`).

Ensuite, nous créons une fonction appelée `build_client_credentials` qui génère les informations d'identification du client. Pour générer les informations d'identification du client, vous devez d'abord concaténer l'ID client et le mot de passe du client et séparer les valeurs par deux points (*`client_ID`*`:`*`client_password`*), puis encoder la chaîne entière en Base64.

```
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')
```

Maintenant que nous avons les informations d'identification de nos clients, nous pouvons les utiliser pour demander un jeton d'accès au courtier. Pour ce faire, nous créons une fonction appeléeget_access_token. Vous devez appeler un POST on `https://Broker_IP:8443/oauth2/token?grant_type=client_credentials` et fournir un en-tête d'autorisation, qui inclut les informations d'identification du client codées en Basic et un type de contenu de. `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
```

Maintenant, nous créons les fonctions nécessaires pour instancier un client. API Pour instancier un clientAPI, vous devez spécifier la configuration du client et les en-têtes à utiliser pour les demandes. La `get_client_configuration` fonction crée un objet de configuration qui inclut l'adresse IP et le port du courtier ainsi que le chemin d'accès au certificat auto-signé du courtier, que vous auriez dû recevoir de l'administrateur du courtier. La `set_request_headers` fonction crée un objet d'en-tête de demande qui inclut les informations d'identification du client et le jeton d'accès.

```
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
```

Enfin, nous créons une méthode principale qui appelle le `DescribeSessions`API. Pour de plus amples informations, veuillez consulter DescribeSessions.

```
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'}])
```

# APIRéférence du gestionnaire de session

Cette référence fournit des détails sur les API actions disponibles, les paramètres requis et les formats de réponse pour vous permettre d'exploiter efficacement le gestionnaire de session API dans vos propres systèmes. À l'aide du gestionnaire de sessionAPI, vous pouvez démarrer, arrêter et obtenir des informations sur les sessions interactives. Cela vous permet d'automatiser et d'intégrer des fonctionnalités dans vos applications et vos flux de travail.

Rubriques



## CloseServers

Ferme un ou plusieurs NICE DCV serveurs. Lorsque vous fermez un NICE DCV serveur, vous le rendez indisponible pour le placement de NICE DCV session. Vous ne pouvez pas créer NICE DCV de sessions sur des serveurs fermés. La fermeture d'un serveur garantit qu'aucune session n'est en cours d'exécution sur celui-ci et que les utilisateurs ne peuvent pas y créer de nouvelles sessions.

Rubriques

## Paramètres de demande

**`ServerId`**

ID du serveur à fermer.

Type : String

Obligatoire : oui

**`Force`**

Force l'opération rapprochée. Si vous le spécifiez`true`, le serveur est fermé même s'il a des sessions en cours d'exécution. Les sessions continuent de se dérouler.

Type : booléen

Obligatoire : non

## Paramètres de réponse

**`RequestId`**

L'identifiant unique de la demande.

**`SuccessfulList`**

Informations sur les NICE DCV serveurs qui ont été fermés avec succès. Cette structure de données inclut le paramètre de réponse imbriqué suivant :

**`ServerId`**

ID du serveur qui a été fermé avec succès.

**`UnsuccessfulList`**

Informations sur les NICE DCV serveurs qui n'ont pas pu être fermés. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**`CloseServerRequestData`**

Informations relatives à la demande initiale qui a échoué. Cette structure de données inclut le paramètre de réponse imbriqué suivant :

**`ServerId`**

ID du NICE DCV serveur qui n'a pas pu être fermé.

**`Force`**

Le paramètre de force demandé.

**`FailureCode`**

Le code de l'échec.

**`FailureReason`**

Raison de l'échec.

## Exemple

Python

Demande

L'exemple suivant ferme deux NICE DCV serveurs (`serverId1`et`serverId2`). Le serveur `serverId2` n'existe pas et entraîne une panne.

```
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"])
```

Réponse

Voici un exemple de sortie.

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

# CreateSessions

Crée une nouvelle NICE DCV session avec les détails spécifiés.

APIactions

- Paramètres de demande
- Paramètres de réponse
- Exemple

## Paramètres de demande

**`Name`**

Le nom de la session.

Type : String

Obligatoire : oui

**`Owner`**

Nom du propriétaire de la session. Il doit s'agir du nom d'un utilisateur existant sur le NICE DCV serveur cible.

Type : String

Obligatoire : oui

**`Type`**

Le type de session. Pour plus d'informations sur les types de sessions, voir Présentation des NICE DCV sessions dans le guide de l'NICEDCVadministrateur.

Valeurs valides : CONSOLE | VIRTUAL

Type : String

Obligatoire : oui

**`InitFile`**

Pris en charge par les sessions virtuelles sur NICE DCV les serveurs Linux. Il n'est pas pris en charge avec les sessions de console sur les NICE DCV serveurs Windows et Linux. Le chemin d'accès au script personnalisé sur le NICE DCV serveur à exécuter pour initialiser la session lors de sa création. Le chemin du fichier est relatif au répertoire d'initialisation spécifié pour le paramètre de configuration de l'`agent.init_folder`agent. Si le fichier se trouve dans le répertoire d'initialisation spécifié, spécifiez uniquement le nom du fichier. Si le fichier ne se trouve pas dans le répertoire d'initialisation spécifié, spécifiez le chemin relatif. Pour plus d'informations, voir le fichier de configuration de l'agent dans le guide de l'administrateur de NICE DCV Session Manager.

Type : chaîne

Obligatoire : non

**`MaxConcurrents`**

Le nombre maximum de NICE DCV clients simultanés.

Type : entier

Obligatoire : non

**`DcvGlEnabled`**

Indique si la session virtuelle est configurée pour utiliser OpenGL basé sur le matériel. Pris en charge uniquement avec les sessions virtuelles. Ce paramètre n'est pas pris en charge par NICE DCV les serveurs Windows.

Valeurs valides : true | false

Type : booléen

Obligatoire : non

**`PermissionsFile`**

Le contenu codé en Base64 du fichier d'autorisations. Si elle est omise, il s'agit par défaut des valeurs par défaut du serveur. Pour plus d'informations, consultez la section Configuration de NICE DCV l'autorisation dans le guide de NICE DCV l'administrateur.

Type : chaîne

Obligatoire : non

**`EnqueueRequest`**

Indique si la demande doit être mise en file d'attente si elle ne peut pas être traitée immédiatement.

Type : booléen

Valeur par défaut : false

Obligatoire : non

**`AutorunFile`**

Compatible avec les sessions de console sur NICE DCV les serveurs Windows et les sessions virtuelles sur NICE DCV les serveurs Linux. Il n'est pas pris en charge avec les sessions de console sur NICE DCV les serveurs Linux.

Le chemin d'accès à un fichier sur le serveur hôte qui doit être exécuté dans le cadre de la session. Le chemin du fichier est relatif au répertoire autorun spécifié pour le paramètre de configuration de l'`agent.autorun_folder`agent. Si le fichier se trouve dans le répertoire autorun spécifié, spécifiez uniquement le nom du fichier. Si le fichier ne se trouve pas dans le répertoire autorun spécifié, spécifiez le chemin relatif. Pour plus d'informations, consultez le fichier de configuration de l'agent dans le guide de l'administrateur de NICE DCV Session Manager.

Le fichier est exécuté pour le compte du propriétaire spécifié. Le propriétaire indiqué doit être autorisé à exécuter le fichier sur le serveur. Sur NICE DCV les serveurs Windows, le fichier est exécuté lorsque le propriétaire se connecte à la session. Sur NICE DCV les serveurs Linux, le fichier est exécuté lors de la création de la session.

Type : chaîne

Obligatoire : non

**`AutorunFileArguments`**

Pris en charge par les sessions virtuelles sur NICE DCV les serveurs Linux. Il n'est pas pris en charge dans les sessions de console sur les NICE DCV serveurs Windows et Linux. Arguments de ligne de commande transmis AutorunFilelors de son exécution dans la session. Les arguments sont transmis dans l'ordre dans lequel ils apparaissent dans le tableau donné. Le nombre maximum autorisé d'arguments et la longueur maximale autorisée de chaque argument peuvent être configurés. Pour plus d'informations, consultez le fichier de configuration du courtier dans le guide de l'administrateur de NICE DCV Session Manager.

Type : tableau de chaînes

Obligatoire : non

**`DisableRetryOnFailure`**

Indique s'il ne faut pas réessayer la demande de création de session après qu'elle ait échoué sur un NICE DCV hôte pour une raison quelconque. Pour plus d'informations sur le mécanisme de création de nouvelles tentatives de session, consultez le fichier de configuration de Broker dans le guide de l'administrateur du gestionnaire de NICE DCV session.

Type : booléen

Valeur par défaut : false

Obligatoire : non

**`Requirements`**

Les exigences auxquelles le serveur doit satisfaire pour pouvoir ouvrir la session. Les exigences peuvent inclure des balises de serveur et/ou des propriétés de serveur, les balises de serveur et les propriétés de serveur sont récupérées en appelant le DescribeServersAPI.

Expressions de conditions d'exigences :

- *a* != *b* vrai si *a* n'est pas égal à *b*

- *a* = *b* vrai si *a* est égal à *b*
- *a* > *b* vrai si *a* est supérieur à *b*
- *a* >= *b* vrai si *a* est supérieur ou égal à *b*
- *a* < *b* vrai si *a* est inférieur à *b*
- *a* <= *b* vrai si *a* est inférieur ou égal à *b*
- *a* = *b* vrai si *a* contient la chaîne *b*

Opérateurs booléens des exigences :

- *a* et *b* vrai si *a* and *b* sont vrais
- *a* ou *b* vrai si *a* or *b* sont vrais
- pas *a* vrai si *a* est faux

Les clés de balise doivent être préfixées par`tag:`, les propriétés du serveur doivent être préfixées par`server:`. Les expressions d'exigences acceptent les parenthèses. ( )

Exemples d'exigences :

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

Les valeurs numériques peuvent être spécifiées à l'aide de la notation exponentielle, par exemple :`"server:Host.Memory.TotalBytes > 1024E6"`.

Les propriétés de serveur prises en charge sont les suivantes :

- `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 : chaîne

Obligatoire : non

**`StorageRoot`**

Indique le chemin du dossier utilisé pour le stockage de session. Pour plus d'informations sur le stockage de NICE DCV session, consultez la section Activation du stockage de session dans le guide de l'NICEDCVadministrateur.

Type : chaîne

Obligatoire : non

## Paramètres de réponse

**`Id`**

L'identifiant unique de la session.

**`Name`**

Nom de la session.

**`Owner`**

Le propriétaire de la session.

**`Type`**

Type de session.

**`State`**

État de la session. Si la demande aboutit, la session passe à l'`CREATING`état.

**`Substate`**

Sous-état de la session. Si la demande aboutit, le sous-état entre dans le `SESSION_PLACING` sous-état.

## Exemple

Python

Demande

L'exemple suivant crée trois 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),
])
```

### Réponse

Voici un exemple de sortie.

```
{
              "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

Décrit un ou plusieurs NICE DCV serveurs.

Rubriques



## Paramètres de demande

**`ServerIds`**

Les IDs NICE DCV serveurs à décrire. Si aucun n'IDsest spécifié, tous les serveurs sont renvoyés dans une sortie paginée.

Type : tableau de chaînes

Obligatoire : non

**`NextToken`**

Le jeton à utiliser pour récupérer la page de résultats suivante.

Type : chaîne

Obligatoire : non

**`MaxResults`**

Le nombre maximum de résultats à renvoyer par la demande dans une sortie paginée. Lorsque ce paramètre est utilisé, la demande renvoie uniquement le nombre de résultats spécifié sur une seule page, ainsi qu'un élément de `NextToken` réponse. Les résultats restants de la demande initiale peuvent être consultés en envoyant une autre demande avec la `NextToken` valeur renvoyée.

Plage valide : 1 - 1 000

Par défaut: 1000

Type : entier

Obligatoire : non

## Paramètres de réponse

**`RequestId`**

L'identifiant unique de la demande.

**`Servers`**

Informations sur les NICE DCV serveurs. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**`Id`**

L'identifiant unique du NICE DCV serveur.

**`Ip`**

Adresse IP du NICE DCV serveur.

**`Hostname`**

Le nom d'hôte du NICE DCV serveur.

**`Endpoints`**

Informations sur les points de terminaison NICE DCV du serveur. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**IpAddress**

Adresse IP du point de terminaison du serveur.

**Port**

Port du point de terminaison du serveur.

**Protocol**

Protocole utilisé par le point de terminaison du serveur. Les valeurs possibles incluent :

- `HTTP`— Le point de terminaison utilise le protocole WebSocket (TCP).
- `QUIC`— Le point de terminaison utilise le protocole QUIC (UDP).

**WebUrlPath**

Le URL chemin Web du point de terminaison du serveur. Disponible uniquement pour le HTTP protocole.

**Version**

Version du NICE DCV serveur.

**SessionManagerAgentVersion**

Version de l'agent Session Manager exécutée sur le NICE DCV serveur.

**Availability**

La disponibilité du NICE DCV serveur. Les valeurs possibles incluent :

- `AVAILABLE`— Le serveur est disponible et prêt pour le placement des sessions.
- `UNAVAILABLE`— Le serveur n'est pas disponible et ne peut pas accepter le placement de session.

**UnavailabilityReason**

La raison de l'indisponibilité du NICE DCV serveur. Les valeurs possibles incluent :

- `SERVER_FULL`— Le NICE DCV serveur a atteint le nombre maximum de sessions simultanées qu'il peut exécuter.
- `SERVER_CLOSED`— Le NICE DCV serveur a été rendu indisponible à l'aide du CloseServerAPI.
- `UNREACHABLE_AGENT`— Le Session Manager Broker ne peut pas communiquer avec l'agent Session Manager sur le NICE DCV serveur.

- `UNHEALTHY_DCV_SERVER`— L'agent du gestionnaire de session ne peut pas communiquer avec le NICE DCV serveur.
- `EXISTING_LOGGED_IN_USER`— (NICEDCVServeurs Windows uniquement) Un utilisateur est actuellement connecté au NICE DCV serveur à l'aide deRDP.
- `UNKNOWN`— Le courtier du gestionnaire de session n'est pas en mesure d'en déterminer la raison.

**`ConsoleSessionCount`**

Le nombre de sessions de console sur le NICE DCV serveur.

**`VirtualSessionCount`**

Le nombre de sessions virtuelles sur le NICE DCV serveur.

**`Host`**

Informations relatives au serveur hôte sur lequel le NICE DCV serveur est exécuté. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**`Os`**

Informations sur le système d'exploitation du serveur hôte. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**`Family`**

La famille de systèmes d'exploitation. Les valeurs possibles incluent :

- `windows`— Le serveur hôte exécute un système d'exploitation Windows.
- `linux`— Le serveur hôte exécute un système d'exploitation Linux.

**`Name`**

Le nom du système d'exploitation.

**`Version`**

La version du système d'exploitation.

**`KernelVersion`**

(Linux uniquement) Version du noyau du système d'exploitation.

**`BuildNumber`**

(Windows uniquement) Numéro de version du système d'exploitation.

**Memory**

Informations sur la mémoire du serveur hôte. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**TotalBytes**

Mémoire totale, en octets, sur le serveur hôte.

**UsedBytes**

Mémoire utilisée, en octets, sur le serveur hôte.

**Swap**

Informations sur le fichier d'échange du serveur hôte. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**TotalBytes**

Taille totale du fichier d'échange, en octets, sur le serveur hôte.

**UsedBytes**

Taille du fichier d'échange utilisé, en octets, sur le serveur hôte.

**Aws**

Uniquement pour NICE DCV les serveurs exécutés sur une EC2 instance Amazon. AWS-informations spécifiques. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**Region**

Le AWS Région de l'EC2instance Amazon.

**Ec2InstanceType**

Type d'EC2instance Amazon.

**Ec2InstanceId**

L'ID de l'EC2instance Amazon.

**Ec2ImageId**

L'ID de l'EC2image Amazon.

**CpuInfo**

Informations sur le serveur hôteCPUs. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**Vendor**

Le fournisseur du serveur hôteCPU.

**ModelName**

Le nom du modèle du serveur hôteCPU.

**Architecture**

Architecture du serveur hôteCPU.

**NumberOfCpus**

Le numéro de CPUs sur le serveur hôte.

**PhysicalCorePerCpu**

Le nombre de CPU cœurs parCPU.

**CpuLoadAverage**

Informations sur la CPU charge du serveur hôte. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**OneMinute**

CPUCharge moyenne au cours de la dernière minute.

**FiveMinutes**

CPUCharge moyenne au cours des 5 dernières minutes.

**FifteenMinutes**

CPUCharge moyenne au cours des 15 dernières minutes.

**Gpus**

Informations sur le serveur hôteGPUs. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**Vendor**

Le fournisseur du serveur hôteGPU.

**ModelName**

Le nom du modèle du serveur hôteGPU.

**LoggedInUsers**

Les utilisateurs actuellement connectés au serveur hôte. Cette structure de données inclut le paramètre de réponse imbriqué suivant :

**Username**

Le nom d'utilisateur de l'utilisateur connecté.

**Tags**

Les balises attribuées au serveur. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**Key**

Identification de balise.

**Value**

Valeur de balise.

## Exemple

Python

Demande

L'exemple suivant décrit tous les NICE DCV serveurs disponibles. Les résultats sont paginés pour afficher deux résultats par 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)
```

### Réponse

Voici un exemple de sortie.

```
{
    "RequestId": "request-id-123",
    "Servers": [
        {
            "Id": "ServerId123",
            "Ip": "1.1.1.123",
            "Hostname": "node001",
            "DefaultDnsName": "node001",
            "Endpoints": [
                {
                    "IpAddress": "x.x.x.x",
                    "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",
            "DefaultDnsName": "node002",
            "Endpoints": [
                {
                    "IpAddress": "x.x.x.x",
                    "Port": 8443,
                    "WebUrlPath": "/",
                    "Protocol": "HTTP"
                },
                {
                    "IpAddress": "x.x.x.x",
                    "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

Décrit une ou plusieurs NICE DCV sessions.

Rubriques

- Paramètres de demande
- Paramètres de réponse
- Exemple

## Paramètres de demande

**`SessionIds`**

Les IDs sessions à décrire.

Type : chaîne

Obligatoire : non

**`NextToken`**

Le jeton à utiliser pour récupérer la page de résultats suivante.

Type : chaîne

Obligatoire : non

**`Filters`**

Filtres supplémentaires à appliquer à la demande. Les filtres pris en charge incluent :

- tag:key : balises attribuées à la session.
- Propriétaire : propriétaire de la session.

Type : chaîne

Obligatoire : non

## Paramètres de réponse

**`Id`**

L'identifiant unique de la session.

**`Name`**

Le nom de la session.

**`Owner`**

Le propriétaire de la session.

**`Server`**

Informations sur le serveur sur lequel s'exécute la session. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**`Ip`**

Adresse IP de l'hôte NICE DCV du serveur.

**`Hostname`**

Le nom d'hôte de l'hôte du NICE DCV serveur.

**`Port`**

Port par lequel le NICE DCV serveur communique avec les NICE DCV clients.

**`Endpoints`**

Informations sur les points de terminaison NICE DCV du serveur. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**IpAddress**

Adresse IP du point de terminaison du serveur.

**Port**

Port du point de terminaison du serveur.

**Protocol**

Protocole utilisé par le point de terminaison du serveur. Les valeurs possibles incluent :

- `HTTP`— Le point de terminaison utilise le protocole WebSocket (TCP).
- `QUIC`— Le point de terminaison utilise le protocole QUIC (UDP).

**WebUrlPath**

Le URL chemin Web du point de terminaison du serveur. Disponible uniquement pour le HTTP protocole.

**Tags**

Les balises attribuées au serveur. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**Key**

Identification de balise.

**Value**

Valeur de balise.

**Type**

Type de session.

**State**

État actuel de la session. Les valeurs possibles sont :

- `CREATING`- le Broker est en train de créer la session.
- `READY`- la session est prête à accepter les connexions des clients.
- `DELETING`- la session est en cours de suppression.
- `DELETED`- la session a été supprimée.

- `UNKNOWN`- impossible de déterminer l'état de la session. Le courtier et l'agent peuvent ne pas être en mesure de communiquer.

**`Substate`**

Sous-état actuel de la session. Les valeurs possibles sont :

- `SESSION_PLACING`- la session est en attente d'être placée sur un DCV serveur disponible.
- `PENDING_PREPARATION`- la session est créée mais n'est pas utilisable ; elle est liée à un DCV serveur.

**`CreationTime`**

Date et heure de création de la session.

**`LastDisconnectionTime`**

Date et heure de la dernière déconnexion du client.

**`NumOfConnections`**

Le nombre de connexions client actives.

**`StorageRoot`**

Indique le chemin du dossier utilisé pour le stockage de session. Pour plus d'informations sur le stockage de NICE DCV session, consultez la section Activation du stockage de session dans le guide de l'NICEDCVadministrateur.

Type : chaîne

Obligatoire : non

## Exemple

Python

Demande

L'exemple suivant décrit les sessions détenues par `user1` et dotées d'une balise de`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'}])
```

### Réponse

Voici un exemple de sortie.

```
{
    "Sessions": [
        {
            "Id": "SessionId1897",
            "Name": "a session name",
            "Owner": "an owner 1890",
```

```
            "Server": {
                "Ip": "1.1.1.123",
                "Hostname": "server hostname",
                "Port": "1222",
                "Endpoints": [
                    {
                        "IpAddress": "x.x.x.x",
                        "Port": 8443,
                        "WebUrlPath": "/",
                        "Protocol": "HTTP"
                    },
                    {
                        "IpAddress": "x.x.x.x",
                        "Port": 9443,
                        "WebUrlPath": "/",
                        "Protocol": "HTTP"
                    },
                    {
                        "IpAddress": "x.x.x.x",
                        "Port": 8443,
                        "WebUrlPath": "",
                        "Protocol": "QUIC"
                    }
                ],
                "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",
                "Endpoints": [
                    {
                        "IpAddress": "x.x.x.x",
                        "Port": 8443,
                        "WebUrlPath": "/",
                        "Protocol": "HTTP"
                    },
                    {
                        "IpAddress": "x.x.x.x",
                        "Port": 9443,
                        "WebUrlPath": "/",
                        "Protocol": "HTTP"
                    },
                    {
                        "IpAddress": "x.x.x.x",
                        "Port": 8443,
                        "WebUrlPath": "",
                        "Protocol": "QUIC"
                    }
                ],
                "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

Supprime la NICE DCV session spécifiée et la retire du cache du broker.

Rubriques



## Paramètres de demande

**`SessionId`**

ID de la session à supprimer.

Type : String

Obligatoire : oui

**`Owner`**

Le propriétaire de la session à supprimer.

Type : String

Obligatoire : oui

**`Force`**

Supprime une session du cache du courtier en tentant de la supprimer du NICE DCV serveur. Cela est utile pour supprimer les sessions obsolètes du cache du broker. Par exemple, si un NICE DCV serveur a été arrêté, mais que les sessions sont toujours enregistrées sur le broker, utilisez cet indicateur pour purger les sessions du cache du broker.

N'oubliez pas que si la session est toujours active, elle est à nouveau mise en cache par le courtier.

Valeurs valides : `true` | `false`

Type : booléen

Obligatoire : non

## Paramètres de réponse

**`SessionId`**

L'ID de la session

**`State`**

Renvoyé uniquement si les sessions ont été correctement supprimées. Indique l'état actuel de la session. Si la demande est terminée avec succès, la session passe à l'`DELETING`état actuel. La suppression de la session peut prendre quelques minutes. Lorsqu'il a été supprimé, l'état passe de `DELETING` à`DELETED`.

**`FailureReason`**

Renvoyé uniquement si certaines sessions n'ont pas pu être supprimées. Indique pourquoi la session n'a pas pu être supprimée.

## Exemple

Python

Demande

L'exemple suivant supprime deux sessions : une session dont l'`SessionId`123identifiant appartient à et une session dont l'`SessionId`abcidentifiant appartient à. `user1` `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')])
```

Réponse

Voici un exemple de sortie. `SessionId123`a été correctement supprimé, mais n'`SessionIdabc`a pas pu être supprimé.

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

# GetSessionConnectionData

Obtient les informations de connexion d'un utilisateur spécifique à une NICE DCV session spécifique.

Rubriques



## Paramètres de demande

**`SessionId`**

ID de la session pour laquelle les informations de connexion doivent être affichées.

Type : String

Obligatoire : oui

**`User`**

Nom de l'utilisateur pour lequel les informations de connexion doivent être affichées.

Type : String

Obligatoire : oui

## Paramètres de réponse

**`Id`**

L'identifiant unique de la session.

**`Name`**

Le nom de la session.

**`Owner`**

Le propriétaire de la session.

**`Server`**

Informations sur le serveur sur lequel s'exécute la session. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**`Ip`**

Adresse IP de l'hôte NICE DCV du serveur.

**`Hostname`**

Le nom d'hôte de l'hôte du NICE DCV serveur.

**`Port`**

Port par lequel le NICE DCV serveur communique avec les NICE DCV clients.

**`Endpoints`**

Informations sur les points de terminaison NICE DCV du serveur. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**`IpAddress`**

Adresse IP du point de terminaison du serveur.

**`Port`**

Port du point de terminaison du serveur.

**`Protocol`**

Protocole utilisé par le point de terminaison du serveur. Les valeurs possibles incluent :

- `HTTP`— Le point de terminaison utilise le protocole WebSocket (TCP).
- `QUIC`— Le point de terminaison utilise le protocole QUIC (UDP).

**`WebUrlPath`**

Le URL chemin Web du point de terminaison du serveur. Disponible uniquement pour le HTTP protocole.

**`WebUrlPath`**

Le chemin d'accès au fichier de configuration du NICE DCV serveur.

**`Tags`**

Les balises attribuées au serveur. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**Key**

Identification de balise.

**Value**

Valeur de balise.

**Type**

Type de session.

**State**

État actuel de la session. Les valeurs possibles sont :

- `CREATING`- le Broker est en train de créer la session.
- `READY`- la session est prête à accepter les connexions des clients.
- `DELETING`- la session est en cours de suppression.
- `DELETED`- la session a été supprimée.
- `UNKNOWN`- impossible de déterminer l'état de la session. Le courtier et l'agent peuvent ne pas être en mesure de communiquer.

**CreationTime**

Date et heure de création de la session.

**LastDisconnectionTime**

Date et heure de la dernière déconnexion du client.

**NumOfConnections**

Le nombre de connexions simultanées de l'utilisateur à la session.

**ConnectionToken**

Le jeton d'authentification utilisé pour se connecter à la session.

## Informations supplémentaires

Les informations ainsi obtenues API peuvent être transmises à un NICE DCV client afin de se connecter à la NICE DCV session.

Dans le cas du client NICE DCV Web, vous pouvez en créer un URL qui peut être ouvert dans le navigateur. Le format URL est le suivant :

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

Dans le cas du client NICE DCV natif, vous pouvez créer un URL avec le `dcv://` schéma. Lorsque le client NICE DCV natif est installé, il s'enregistre auprès du système en tant que gestionnaire de `dcv://`URLs. Le format URL est le suivant :

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

> **Note**
>
> Si vous utilisez AmazonEC2, l'adresse IP doit être publique. Si votre configuration comporte NICE DCV des hôtes derrière une passerelle, spécifiez l'adresse de la passerelle plutôt que celle renvoyée par le SessionConnectionData API.

## Exemple

Python

Demande

L'exemple suivant obtient les informations de connexion d'un utilisateur dont le nom d'utilisateur est `user1` et d'une session dont l'ID est égal à`sessionId12345`.

```
def get_session_connection_api():
    api_instance =
 swagger_client.GetSessionConnectionDataApi(swagger_client.ApiClient(get_client_configuratio
    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')
```

### Réponse

Voici un exemple de sortie.

```
{
    "Session": {
        "Id": "sessionId12345",
        "Name": "a session name",
        "Owner": "an owner 1890",
        "Server": {
            "Ip": "1.1.1.123",
            "Hostname": "server hostname",
            "Port": "1222",
            "endpoints": [
                {
                    "port": 8443,
                    "web_url_path": "/",
                    "protocol": "HTTP"
                },
                {
                    "port": 9443,
                    "web_url_path": "/",
```

```
                    "protocol": "HTTP"
                },
                {
                    "port": 8443,
                    "web_url_path": "",
                    "protocol": "QUIC"
                }
            ],
            "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":
 "EXAMPLEiOiJmOWM1YTRhZi1jZmU0LTQ0ZjEtYjZlOC04ZjY0YjM4ZTE2ZDkiLCJ0eXAiOiJKV1QiLCJhbGciOiJSUz
tngiKXevUxhhJvm3BPJYRs9NPE4GCJRTc13EXAMPLEIxNEPPh5IMcVmROfU1WKPnry4ypPTp3rsZ7YWjCTSfs1GoN3R_
Kqtpd5GH0D-E8FwsedV-
Q2bRQ4y9y1q0MgFU4QjaSMypUuYR0YjkCaoainjmEZew4A33fG40wATrBvoivBiNWdNpytHX2CDOuk_k0k_DWeZjMvv9
h_GaMgHmltqBIA4jdPD7i0CmC2e7413KFy-
EQ4Ej1cM7RjLwhFuWpKWAVJxogJjYpfoKKaPo4KxvJjJIPYhkscklINQpe2W5rnlxCq7sC7ptcGw17DUobP7egRv9H37
hK1G4G8erHvl9HIrTR9_c884fNrTCC8DvC062e4KYdLkAhhJmboN9CAGIGFyd2c1AY_CzzvDL0EXAMLE"
}
```


# GetSessionScreenshots

Permet d'obtenir des captures d'écran d'une ou de plusieurs NICE DCV sessions.

Le type de fichier image et la résolution de la capture d'écran dépendent de la configuration de Session Manager Broker. Pour modifier le type de fichier image, configurez le `session-`

`screenshot-format` paramètre. Pour modifier la résolution, configurez les `session-screenshot-max-height` paramètres `session-screenshot-max-width` et. Pour plus d'informations, consultez le fichier de configuration du courtier dans le guide de l'administrateur de NICE DCV Session Manager.

Rubriques



## Paramètres de demande

**`SessionId`**

ID de la NICE DCV session à partir de laquelle vous souhaitez obtenir la capture d'écran.

Type : String

Obligatoire : oui

## Paramètres de réponse

**`RequestId`**

L'identifiant unique de la demande.

**`SuccessfulList`**

Informations sur les captures d'écran réussies. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**`SessionScreenshot`**

Informations sur les captures d'écran. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**`SessionId`**

ID de la NICE DCV session à partir de laquelle la capture d'écran a été prise.

**Images**

Informations sur les images. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**Format**

Format de l'image. Les valeurs possibles incluent `jpeg` et `png`.

**Data**

Le format codé base64 de l'image de capture d'écran.

**CreationTime**

Date et heure auxquelles la capture d'écran a été prise.

**Primary**

Indique si la capture d'écran est celle de l'écran principal de la NICE DCV session.

**UnsuccessfulList**

Informations sur les captures d'écran infructueuses. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**GetSesionScreenshotRequestData**

La demande initiale qui a échoué.

**SessionId**

ID de la NICE DCV session à partir de laquelle la capture d'écran devait être prise.

**FailureReason**

Raison de l'échec.

## Exemple

Python

Demande

L'exemple suivant obtient des captures d'écran de deux sessions (`sessionId1`et`sessionId2`). La session `sessionId2` n'existe pas et entraîne un échec.

```
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"])
```

### Réponse

Voici un exemple de sortie.

```
{
    "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

Ouvre un ou plusieurs NICE DCV serveurs. Avant de créer des NICE DCV sessions sur un NICE DCV serveur, vous devez modifier l'état du serveur pour qu'il soit ouvert. Une fois le NICE DCV serveur ouvert, vous pouvez créer des NICE DCV sessions sur le serveur.

Rubriques



## Paramètres de demande

**`ServerId`**

ID du serveur à ouvrir.

Type : String

Obligatoire : oui

## Paramètres de réponse

**`RequestId`**

L'identifiant unique de la demande.

**`SuccessfulList`**

Informations sur les NICE DCV serveurs qui ont été ouverts avec succès. Cette structure de données inclut le paramètre de réponse imbriqué suivant :

**`ServerId`**

L'ID du serveur qui a été ouvert avec succès.

**`UnsuccessfulList`**

Informations sur les NICE DCV serveurs qui n'ont pas pu être ouverts. Cette structure de données inclut les paramètres de réponse imbriqués suivants :

**`OpenServerRequestData`**

Informations relatives à la demande initiale qui a échoué. Cette structure de données inclut le paramètre de réponse imbriqué suivant :

**`ServerId`**

ID du NICE DCV serveur qui n'a pas pu être ouvert.

**`FailureCode`**

Le code de l'échec.

**`FailureReason`**

Raison de l'échec.

## Exemple

Python

Demande

L'exemple suivant ouvre deux NICE DCV serveurs (`serverId1`et`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"])
```

Réponse

Voici un exemple de sortie.

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

# UpdateSessionPermissions

Met à jour les autorisations utilisateur pour une NICE DCV session spécifique.

Rubriques

## Paramètres de demande

**`SessionId`**

ID de la session pour laquelle vous souhaitez modifier les autorisations.

Type : String

Obligatoire : oui

**`Owner`**

Le propriétaire de la session pour laquelle vous souhaitez modifier les autorisations.

Type : String

Obligatoire : oui

**`PermissionFile`**

Le contenu codé en Base64 du fichier d'autorisations à utiliser. Pour plus d'informations, consultez la section Configuration de NICE DCV l'autorisation dans le guide de NICE DCV l'administrateur.

Type : String

Obligatoire : oui

## Paramètres de réponse

**`SessionId`**

L'ID de la séance.

## Exemple

Python

Demande

L'exemple suivant définit de nouvelles autorisations pour une session dont l'ID de session est`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')])
```

### Réponse

Voici un exemple de sortie.

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

# Notes de mise à jour et historique des documents pour NICE DCV Session Manager

Cette page fournit les notes de publication et l'historique des documents pour NICE DCV Session Manager.

Rubriques



## NICEDCVNotes de mise à jour de Session Manager

Cette section fournit un aperçu des principales mises à jour, des versions de fonctionnalités et des corrections de bogues du gestionnaire de NICE DCV session. Toutes les mises à jour sont organisées par date de sortie. Nous mettons fréquemment à jour la documentation pour répondre aux commentaires que vous nous envoyez.

Rubriques

- 2020.2-9662 — 04 décembre 2020
- 2020.2-9508 — 11 novembre 2020

## 2023.1-17652 — 1er août 2024

| Numéros de version | Modifications et correctifs de bogues |
| --- | --- |
| • Courtier : 426<br>• Agent : 748<br>• CLI: 140 | • Correctifs de bogues et améliorations de performances |

## 2023.1-16388 — 26 juin 2024

| Numéros de version | Modifications et correctifs de bogues |
| --- | --- |
| • Courtier : 417<br>• Agent : 748<br>• CLI: 140 | • Correction d'un bug qui affichait incorrectement la mémoire en To, et non en Go.<br>• Correctifs de bogues et améliorations de performances |

## 2023.1 — 9 novembre 2023

| Numéros de version | Modifications et correctifs de bogues |
| --- | --- |
| • Courtier : 410<br>• Agent : 732<br>• CLI: 140 | • Correctifs de bogues et améliorations de performances |

## 2023.0-15065— 4 mai 2023

| Numéros de version | Modifications et correctifs de bogues |
| --- | --- |
| • Courtier : 392<br>• Agent : 675<br>• CLI: 132 | • Ajout du support pour Red Hat Enterprise Linux 9, Rocky Linux 9 et CentOS Stream 9 sur les ARM plateformes. |

## 2023.0-14852 — 28 mars 2023

| Numéros de version | Modifications et correctifs de bogues |
| --- | --- |
| • Courtier : 392<br>• Agent : 642<br>• CLI: 132 | • Ajout du support pour Red Hat Enterprise Linux 9, Rocky Linux 9 et CentOS Stream 9. |

## 2022.2-13907 — 11 novembre 2022

| Numéros de version | Modifications et correctifs de bogues |
| --- | --- |
| • Courtier : 382<br>• Agent : 612<br>• CLI: 123 | • Ajout d'un `Substate` champ en `DescribeSessions` réponse.<br>• Correction d'un problème qui empêchait CLI le broker de se connecter au broker en fonction de URL celui utilisé. |

## 2022.1-13067 — 29 juin 2022

| Numéros de version | Modifications et correctifs de bogues |
| --- | --- |
| • Courtier : 355<br>• Agent : 592 | • Ajout du support pour exécuter le broker sur des instances AWS Graviton.<br>• Ajout du support des agents et des courtiers pour Ubuntu 22.04. |

| Numéros de version | Modifications et correctifs de bogues |
| --- | --- |
| • CLI: 114 | |

## 2022.0-11952 — 23 février 2022

| Numéros de version | Modifications et correctifs de bogues |
| --- | --- |
| • Courtier : 341<br>• Agent : 520<br>• CLI: 112 | • Ajout de la fonctionnalité de rotation des journaux à l'agent.<br>• Ajout d'un paramètre de configuration pour définir Java Home dans le Broker.<br>• Amélioration du transfert des données du cache vers le disque dans le Broker.<br>• URLValidation fixe dans leCLI. |

## 2021.3-11591 — 20 décembre 2021

| Numéros de version | Nouvelles fonctionnalités |
| --- | --- |
| • Courtier : 307<br>• Agent : 453<br>• CLI: 92 | • Ajout de la prise en charge de l'intégration à la passerelle de NICE DCV connexion.<br>• Ajout du support Broker pour Ubuntu 18.04 et Ubuntu 20.04. |

## 2021.2-11445 — 18 novembre 2021

| Numéros de version | Modifications et correctifs de bogues |
| --- | --- |
| • Courtier : 288<br>• Agent : 413<br>• CLI: 54 | • Correction d'un problème lié à la validation des noms de connexion incluant un domaine Windows. |

## 2021.2-11190 — 11 octobre 2021

| Numéros de version | Modifications et correctifs de bogues |
| --- | --- |
| • Courtier : 254<br>• Agent : 413<br>• CLI: 54 | • Correction d'un problème dans l'interface de ligne de commande qui empêchait le lancement de sessions Windows. |

## 2021.2-11042 — 1er septembre 2021

| Numéros de version | Nouvelles fonctionnalités | Modifications et correctifs de bogues |
| --- | --- | --- |
| • Courtier : 254<br>• Agent : 413<br>• CLI: 37 | • NICEDCVLe gestionnaire de session prend désormais en charge l'interfa ce de ligne de commande (CLI). Vous pouvez créer et gérer NICE DCV des sessions dans leCLI, au lieu d'appeler APIs.<br>• NICEDCVLe gestionnaire de session a introduit la persistance des données Broker. Pour une disponibilité accrue, les courtiers peuvent conserver les informations sur l'état du serveur dans un magasin de données externe et restaurer les données au démarrage. | • Lorsque vous enregistrez un serveur d'autorisation externe, vous pouvez désormais spécifier l'algorithme utilisé par le serveur d'autorisation pour signer les jetons Web JSON formatés. Avec cette modification, vous pouvez utiliser Azure AD comme serveur d'autorisation externe. |

## 2021.1-10557 — 31 mai 2021

| Numéros de version | Nouvelles fonctionnalités | Modifications et correctifs de bogues |
| --- | --- | --- |
| • Courtier : 214 | • NICEDCVLe gestionnaire de session a ajouté la prise en charge des | • Nous avons résolu un problème avec le fichier autorun sous Windows. |

| Numéros de version | Nouvelles fonctionnalités | Modifications et correctifs de bogues |
| --- | --- | --- |
| • Agent : 365 | paramètres d'entrée transmis au fichier autorun sous Linux.<br>• Les propriétés du serveur peuvent désormais être transmises en tant qu'exigences au CreateSessionsAPI. | |

## 2021.0-10242 — 12 avril 2021

| Numéros de version | Modifications et correctifs de bogues |
| --- | --- |
| • Courtier : 183<br>• Agent : 318 | • NICEDCVLe gestionnaire de session a introduit les nouveautés suivantes APIs :<br>• OpenServers<br>• CloseServers<br>• DescribeServers<br>• GetSessionScreenshots<br>• Il a également introduit les nouveaux paramètres de configuration suivants :<br>• Paramètres du courtier : `session-screenshot-max-width session-screenshot-max-height ,session-screenshot-format ,create-sessions-queue-max-size` , et`create-sessions-queue-max-time-seconds` .<br>• Paramètres de l'agent : `agent.autorun_folder max_virtual_sessions` , et`max_concurrent_sessions_per_user` .<br>Paramètres de l'agent : `agent.autorun_folder max_virtual_sessions` , et`max_concurrent_sessions_per_user` . |

| Numéros de version | Modifications et correctifs de bogues |
| --- | --- |
| | Paramètres de l'agent : `agent.autorun_folder` `max_virtual_sessions` , et`max_concurrent_sessions_per_user` . |

## 2020.2-9662 — 04 décembre 2020

| Numéros de version | Modifications et correctifs de bogues |
| --- | --- |
| • Courtier : 114<br>• Agent : 211 | • Nous avons résolu un problème lié aux TLS certificats générés automatiquement qui empêchait le Broker de démarrer. |

## 2020.2-9508 — 11 novembre 2020

| Numéros de version | Modifications et correctifs de bogues |
| --- | --- |
| • Courtier : 78<br>• Agent : 183 | • La version initiale de NICE DCV Session Manager. |

# Historique de la documentation

Le tableau suivant décrit la documentation de cette version de NICE DCV Session Manager.

| Modification | Description | Date |
| --- | --- | --- |
| NICEDCVVersion 2023.1-17652 | NICEDCVLe gestionnaire de session a été mis à jour pour NICE DCV 2023.1-17652. Pour de plus amples informations, veuillez consulter 2023.1-17652 — 1er août 2024. | 1er août 2024 |

| Modification | Description | Date |
| --- | --- | --- |
| NICEDCVVersion 2023.1-16388 | NICEDCVLe gestionnaire de session a été mis à jour pour NICE DCV 2023.1-16388. Pour de plus amples informations, veuillez consulter 2023.1-16388 — 26 juin 2024. | 26 juin 2024 |
| NICEDCVLaversion 2023.1 | NICEDCVLe gestionnaire de session a été mis à jour pour NICE DCV 2023.1. Pour de plus amples informations, veuillez consulter 2023.1 — 9 novembre 2023. | 9 novembre 2023 |
| NICEDCVLaversion 2023.0 | NICEDCVLe gestionnaire de session a été mis à jour pour NICE DCV 2023.0. Pour de plus amples informations, veuillez consulter 2023.0-14852 — 28 mars 2023. | 28 mars 2023 |
| NICEDCVLaversion 2022.2 | NICEDCVLe gestionnaire de session a été mis à jour pour NICE DCV 2022.2. Pour de plus amples informations, veuillez consulter 2022.2-13907 — 11 novembre 2022. | 11 novembre 2022 |
| NICEDCVLaversion 2022.1 | NICEDCVLe gestionnaire de session a été mis à jour pour NICE DCV 2022.1. Pour de plus amples informations, veuillez consulter 2022.1-13067 — 29 juin 2022. | 29 juin 2022 |
| NICEDCVLaversion 2022.0 | NICEDCVLe gestionnaire de session a été mis à jour pour NICE DCV 2022.0. Pour de plus amples informations, veuillez consulter 2022.0-11952 — 23 février 2022. | 23 février 2022 |

| Modification | Description | Date |
| --- | --- | --- |
| NICEDCVLa version 2021.3 | NICEDCVLe gestionnaire de session a été mis à jour pour NICE DCV 2021.3. Pour de plus amples informations, veuillez consulter 2021.3-11591 — 20 décembre 2021. | 20 décembre 2021 |
| NICEDCVLa version 2021.2 | NICEDCVLe gestionnaire de session a été mis à jour pour NICE DCV 2021.2. Pour de plus amples informations, veuillez consulter 2021.2-11042 — 1er septembre 2021. | 01 septembre 2021 |
| NICEDCVLa version 2021.1 | NICEDCVLe gestionnaire de session a été mis à jour pour NICE DCV 2021.1. Pour de plus amples informations, veuillez consulter 2021.1-10557 — 31 mai 2021. | 31 mai 2021 |
| NICEDCVLa version 2021.0 | NICEDCVLe gestionnaire de session a été mis à jour pour NICE DCV 2021.0. Pour de plus amples informations, veuillez consulter 2021.0-10242 — 12 avril 2021. | 12 avril 2021 |
| Version initiale du gestionnaire de NICE DCV session | Il s'agit de la première publication de ce contenu. | 11 novembre 2020 |

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.