



开发人员指南

# Amazon DCV Session Manager



# Amazon DCV Session Manager: 开发人员指南

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆或者贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

# Table of Contents

什么是 Session Manager ? .....	1
Session Manager 的工作方式 .....	1
功能 .....	3
开始使用 Session Manager API .....	4
步骤 1 : 生成 API 客户端 .....	4
步骤 2 : 注册客户端 API .....	5
步骤 3 : 获取访问令牌并发出 API 请求 .....	5
Session Manager API 参考 .....	9
CloseServers .....	9
请求参数 .....	5
响应参数 .....	10
示例 .....	11
CreateSessions .....	12
请求参数 .....	5
响应参数 .....	10
示例 .....	11
DescribeServers .....	19
请求参数 .....	5
响应参数 .....	10
示例 .....	11
DescribeSessions .....	30
请求参数 .....	5
响应参数 .....	10
示例 .....	11
DeleteSessions .....	36
请求参数 .....	5
响应参数 .....	10
示例 .....	11
GetSessionConnectionData .....	39
请求参数 .....	5
响应参数 .....	10
其他信息 .....	42
示例 .....	11
GetSessionScreenshots .....	44

请求参数 .....	5
响应参数 .....	10
示例 .....	11
OpenServers .....	48
请求参数 .....	5
响应参数 .....	10
示例 .....	11
UpdateSessionPermissions .....	50
请求参数 .....	5
响应参数 .....	10
示例 .....	11
发布说明和文档历史记录 .....	53
发布说明 .....	53
2024.0-457 - 2024 年 10 月 1 日 .....	54
2023.1-17652 - 2024 年 8 月 1 日 .....	54
2023.1-16388 - 2024 年 6 月 26 日 .....	54
2023.1 - 2023 年 11 月 9 日 .....	54
2023.0-15065 - 2023 年 5 月 4 日 .....	55
2023.0-14852 - 2023 年 3 月 28 日 .....	55
2022.2-13907 - 2022 年 11 月 11 日 .....	55
2022.1-13067 - 2022 年 6 月 29 日 .....	55
2022.0-11952 - 2022 年 2 月 23 日 .....	56
2021.3-11591 - 2021 年 12 月 20 日 .....	56
2021.2-11445 - 2021 年 11 月 18 日 .....	56
2021.2-11190 - 2021 年 10 月 11 日 .....	56
2021.2-11042 - 2021 年 9 月 1 日 .....	57
2021.1-10557 - 2021 年 5 月 31 日 .....	57
2021.0-10242 - 2021 年 4 月 12 日 .....	57
2020.2-9662 - 2020 年 12 月 4 日 .....	58
.....	58
文档历史记录 .....	59

# 什么是 Amazon DCV Session Manager ?

## Note

Amazon DCV 以前称为 NICE DCV。

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

本指南介绍了如何使用 Session Manager API 管理 Amazon DCV 会话的生命周期。有关如何安装和配置 Session Manager Broker 和 Agent 的更多信息 , 请参阅《Amazon DCV Session Manager 管理员指南》。

## 先决条件

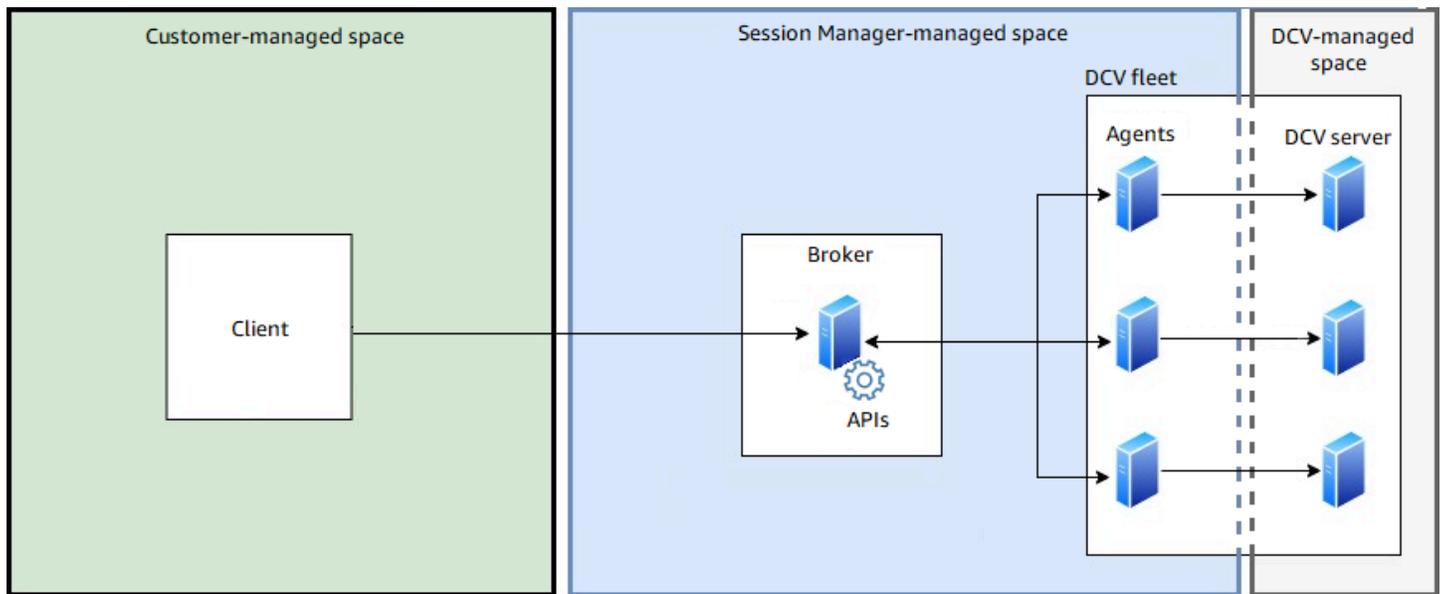
在开始使用 Session Manager API 之前 , 请确保您熟悉 Amazon DCV 和 Amazon DCV 会话。有关更多信息 , 请参阅 [《Amazon DCV 管理员指南》](#)。

## 主题

- [Session Manager 的工作方式](#)
- [功能](#)

## Session Manager 的工作方式

下图简要显示了 Session Manager 组件。



## 代理

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

## Agent

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

## API

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

## 客户端

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

## 访问令牌

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

## 客户端 API

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

## Amazon DCV 会话

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

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

## 功能

DCV Session Manager 提供以下功能：

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

# 开始使用 Session Manager API

Amazon DCV Session Manager API 提供了一个用于管理远程桌面会话的自动化界面。通过此 API，开发人员能够以编程方式创建、列出、启动、停止和控制 DCV 会话。这样便可将 Amazon DCV 功能集成到自定义应用程序和工作流中。利用此 API，组织可以精简远程可视化工作负载的管理，自动执行许多常见任务。

开始调用 Amazon DCV API 之前，您需要获取访问令牌，该令牌可对您的应用程序进行身份验证并授权其访问必要的资源。Amazon DCV API 使用 OAuth 2.0 进行身份验证，因此您需要注册应用程序并检索必要的凭证。获得访问令牌后，您就可以开始向 Amazon DCV API 端点发送请求以开始处理数据。

## 主题

- [步骤 1：生成 API 客户端](#)
- [步骤 2：注册客户端 API](#)
- [步骤 3：获取访问令牌并发出 API 请求](#)

## 步骤 1：生成 API 客户端

Session Manager API 是在单个 YAML 文件中定义的。这些 API 基于 OpenAPI3.0 规范，该规范定义了与语言无关的标准 RESTful API 接口。有关更多信息，请参阅 [OpenAPI 规范](#)。

通过使用 YAML 文件，您可以使用支持的语言之一生成 API 客户端。为此，您必须使用 Swagger Codegen 3.0 或更高版本。有关支持的语言的更多信息，请参阅 [swagger-codegen 存储库](#)。

### 生成 API 客户端

1. 从 Session Manager Broker 中下载 Session Manager API YAML 文件。在以下 URL 中提供了该 YAML 文件。

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

2. 安装 Swagger Codegen。

- macOS

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

- 其他平台

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

```
$ cd swagger-codegen
```

### 3. 生成 API 客户端。

- macOS

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

- 其他平台

```
$ mvn clean package
```

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

## 步骤 2：注册客户端 API

API 请求使用访问令牌来验证您的凭证。这些凭证基于在 Broker 中注册您的客户端时生成的客户端 ID 和客户端密码。

要访问此令牌，您需要向 Broker 注册。使用 [register-api-client](#) 注册客户端 API。

如果您没有客户端的客户端 ID 和客户端密码，必须向您的 Broker 管理员索取 ID 和密码。

## 步骤 3：获取访问令牌并发出 API 请求

本示例将介绍设置访问令牌的步骤，然后向您展示如何发出基本的 API 请求。这将为您的应用程序提供基础知识，以开始构建由 Amazon DCV API 支持的更高级的应用程序。

在本示例中，我们将通过使用 DescribeSessions API 来说明如何执行该操作。

### Example

首先，我们导入应用程序所需的模型。

然后，我们声明客户端 ID 变量 ( `__CLIENT_ID` )、客户端密码变量 ( `__CLIENT_SECRET` ) 和 Broker URL 变量，包括端口号变量 ( `__PROTOCOL_HOST_PORT` )。

接下来，我们创建一个名为 `build_client_credentials` 的函数以生成客户端凭证。要生成客户端凭证，您必须先串联客户端 ID 和客户端密码并用冒号 ( `client_id:client_password` ) 分隔这些值，然后对整个字符串进行 Base64 编码。

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

__CLIENT_ID = '794b2dbb-bd82-4707-a2f7-f3d9899cb386'
__CLIENT_SECRET = 'MzcxNzJhN2UtYjEzNS00MjNjLTg2N2YtMjF1ZmRlZWJMDU1'
__PROTOCOL_HOST_PORT = 'https://<broker-hostname>:8443'

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

现在我们生成了客户端凭证，可以使用该凭证从 Broker 中请求访问令牌。为此，我们创建一个名为 `get_access_token` 的函数。您必须对 `https://Broker_IP:8443/oauth2/token?grant_type=client_credentials` 调用 POST 并提供授权标头，其中包括 Basic 编码的客户端凭证和内容类型 `application/x-www-form-urlencoded`。

```
def get_access_token():
    client_credentials = build_client_credentials()
    headers = {
        'Authorization': 'Basic {}'.format(client_credentials),
        'Content-Type': 'application/x-www-form-urlencoded'
    }
    endpoint = __PROTOCOL_HOST_PORT + '/oauth2/token?grant_type=client_credentials'
```

```
print('Calling', endpoint, 'using headers', headers)
res = requests.post(endpoint, headers=headers, verify=True)
if res.status_code != 200:
    print('Cannot get access token:', res.text)
    return None
access_token = json.loads(res.text)['access_token']
print('Access token is', access_token)
return access_token
```

现在，我们创建实例化客户端 API 所需的函数。要实例化客户端 API，您必须指定客户端配置和用于请求的标头。get\_client\_configuration 函数创建一个配置对象，其中包括 Broker 的 IP 地址和端口以及 Broker 自签名证书的路径（您应该从 Broker 管理员处收到了该证书）。set\_request\_headers 函数创建一个请求标头对象，其中包括客户端凭证和访问令牌。

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

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

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

最后，我们创建一个调用 DescribeSessions API 的 main 方法。有关更多信息，请参阅 [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'}])
```

# Session Manager API 参考

本参考提供了有关可用 API 操作、必需参数和响应格式的详细信息，使您能够在自己的系统中有效地利用 Session Manager API。使用 Session Manager API，您可以启动或停止交互式会话，并获取有关交互式会话的详细信息。这使您可以实现功能自动化，并将功能集成到您的应用程序和工作流中。

## 主题

- [CloseServers](#)
- [CreateSessions](#)
- [DescribeServers](#)
- [DescribeSessions](#)
- [DeleteSessions](#)
- [GetSessionConnectionData](#)
- [GetSessionScreenshots](#)
- [OpenServers](#)
- [UpdateSessionPermissions](#)

## CloseServers

关闭一个或多个 Amazon DCV 服务器。在您关闭 Amazon DCV 服务器时，将使其无法放置新的 Amazon DCV 会话。您无法在关闭的服务器上创建 Amazon DCV 会话。关闭服务器可以确保在服务器上没有运行任何会话，并且用户无法在服务器上创建新的会话。

## 主题

- [请求参数](#)
- [响应参数](#)
- [示例](#)

## 请求参数

### ServerId

要关闭的服务器的 ID。

类型：字符串

必需：是

### **Force**

强制执行关闭操作。如果指定 `true`，即使服务器具有运行的会话，也会关闭服务器。这些会话继续运行。

类型：布尔值

必需：否

## 响应参数

### **RequestId**

请求的唯一 ID。

### **SuccessfulList**

有关成功关闭的 Amazon DCV 服务器的信息。该数据结构包括以下嵌套的响应参数：

#### **ServerId**

成功关闭的服务器的 ID。

### **UnsuccessfulList**

有关无法关闭的 Amazon DCV 服务器的信息。该数据结构包括以下嵌套的响应参数：

#### **CloseServerRequestData**

有关失败的原始请求的信息。该数据结构包括以下嵌套的响应参数：

#### **ServerId**

无法关闭的 Amazon DCV 服务器的 ID。

#### **Force**

请求的 `force` 参数。

#### **FailureCode**

失败代码。

## FailureReason

失败的原因。

## 示例

### Python

#### 请求

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

```
from swagger_client.models import CloseServerRequestData

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

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

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

#### 响应

以下是示例输出。

```
{
  "RequestId": "4d7839b2-a03c-4b34-a40d-06c8b21099e6",
  "SuccessfulList": [
    {
```

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

## CreateSessions

使用指定的详细信息创建新的 Amazon DCV 会话。

### API 操作

- [请求参数](#)
- [响应参数](#)
- [示例](#)

## 请求参数

### Name

会话的名称。

类型：字符串

必需：是

### Owner

会话所有者的名称。这必须是目标 Amazon DCV 服务器上的现有用户的名称。

类型：字符串

必需：是

## Type

会话类型。有关会话类型的更多信息，请参阅《Amazon DCV 管理员指南》中的 [Amazon DCV 会话简介](#)。

有效值：CONSOLE | VIRTUAL

类型：字符串

必需：是

## InitFile

Linux Amazon DCV 服务器上的虚拟会话支持。Windows 和 Linux Amazon DCV 服务器上的控制台会话不支持该参数。这是 Amazon DCV 服务器上的自定义脚本的路径，用于在创建会话时运行以初始化会话。文件路径相对于为 `agent.init_folder` Agent 配置参数指定的 `init` 目录。如果文件位于指定的 `init` 目录中，请仅指定文件名。如果文件没有位于指定的 `init` 目录中，请指定相对路径。有关更多信息，请参阅《Amazon DCV Session Manager 管理员指南》中的 [Agent 配置文件](#)。

类型：字符串

必需：否

## MaxConcurrents

并发 Amazon DCV 客户端的最大数量。

类型：整数

必需：否

## DcvG1Enabled

指示虚拟会话是否配置为使用基于硬件的 OpenGL。仅虚拟会话支持。Windows Amazon DCV 服务器不支持该参数。

有效值：true | false

类型：布尔值

必需：否

## PermissionsFile

Base64 编码的权限文件内容。如果省略，则默认为服务器默认值。有关更多信息，请参阅《Amazon DCV 管理员指南》中的[配置 Amazon DCV 授权](#)。

类型：字符串

必需：否

## EnqueueRequest

指示在无法立即完成请求时是否将其排入队列。

类型：布尔值

默认：false

必需：否

## AutorunFile

Windows Amazon DCV 服务器上的控制台会话和 Linux Amazon DCV 服务器上的虚拟会话支持。Linux Amazon DCV 服务器上的控制台会话不支持该参数。

位于主机服务器上并在会话中运行的文件的路径。文件路径相对于为 `agent.autorun_folder` Agent 配置参数指定的 `autorun` 目录。如果文件位于指定的 `autorun` 目录中，请仅指定文件名。如果文件没有位于指定的 `autorun` 目录中，请指定相对路径。有关更多信息，请参阅《Amazon DCV Session Manager 管理员指南》中的[Agent 配置文件](#)。

该文件是代表指定所有者运行的。指定的所有者必须有权在服务器上运行该文件。在 Windows Amazon DCV 服务器上，该文件是在所有者登录到会话时运行的。在 Linux Amazon DCV 服务器上，该文件是在创建会话时运行的。

类型：字符串

必需：否

## AutorunFileArguments

Linux Amazon DCV 服务器上的虚拟会话支持。Windows 和 Linux Amazon DCV 服务器上的控制台会话不支持该参数。在会话中执行 `AutorunFile` 时为其传递的命令行参数。参数是按照它们在给定数组中的出现顺序传递的。可以配置最大允许参数数量以及每个参数的最大允许长度。有关更多信息，请参阅《Amazon DCV Session Manager 管理员指南》中的[Broker 配置文件](#)。

类型：字符串数组

必需：否

## DisableRetryOnFailure

指示创建会话请求在 Amazon DCV 主机上由于任何原因失败后是否不重试。有关创建会话重试机制的更多信息，请参阅《Amazon DCV Session Manager 管理员指南》中的 [Broker 配置文件](#)。

类型：布尔值

默认：false

必需：否

## Requirements

服务器必须满足才能放置会话的要求。这些要求可以包括服务器标签和/或服务器属性，服务器标签和服务器属性都是通过调用 DescribeServers API 检索的。

要求条件表达式：

- $a \neq b$  如果  $a$  不等于  $b$ ，则为 true
- $a = b$  如果  $a$  等于  $b$ ，则为 true
- $a > b$  如果  $a$  大于  $b$ ，则为 true
- $a \geq b$  如果  $a$  大于或等于  $b$ ，则为 true
- $a < b$  如果  $a$  小于  $b$ ，则为 true
- $a \leq b$  如果  $a$  小于或等于  $b$ ，则为 true
- $a = b$  如果  $a$  包含字符串  $b$ ，则为 true

要求布尔运算符：

- $a$  and  $b$  如果  $a$  和  $b$  为 true，则为 true
- $a$  or  $b$  如果  $a$  或  $b$  为 true，则为 true
- not  $a$  如果  $a$  为 false，则为 true

标签键必须以 tag: 为前缀，服务器属性必须以 server: 为前缀。要求表达式支持圆括号 ( )。

要求示例：

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

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

支持的服务器属性包括：

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

类型：字符串

必需：否

## StorageRoot

指定用于会话存储的文件夹的路径。有关 Amazon DCV 会话存储的更多信息，请参阅《Amazon DCV 管理员指南》中的[启用会话存储](#)。

类型：字符串

必需：否

## 响应参数

### Id

会话的唯一 ID。

### Name

会话名称。

### Owner

会话所有者。

### Type

会话的类型。

### State

会话的状态。如果请求成功完成，会话将进入 CREATING 状态。

### Substate

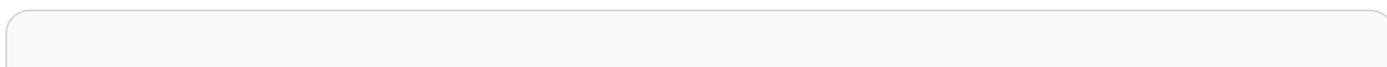
会话的子状态。如果请求成功完成，会话将进入 SESSION\_PLACING 子状态。

## 示例

### Python

请求

以下示例创建三个会话。



```

from swagger_client.models.create_session_request_data import
    CreateSessionRequestData

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

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

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

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

```

## 响应

以下是示例输出。

```
{
  "RequestId": "e32d0b83-25f7-41e7-8c8b-e89326ecc87f",
  "SuccessfulList": [
    {
      "Id": "78b45deb-1163-46b1-879b-7d8fcbe9d9d6",
      "Name": "session1",
      "Owner": "user1",
      "Type": "CONSOLE",
      "State": "CREATING"
    },
    {
      "Id": " a0c743c4-9ff7-43ce-b13f-0c4d55a268dd",
      "Name": "session2",
      "Owner": "user1",
      "Type": "VIRTUAL",
      "State": "CREATING"
    },
    {
      "Id": " 10311636-df90-4cd1-bcf7-474e9675b7cd",
      "Name": "session3",
      "Owner": "user1",
      "Type": "VIRTUAL",
      "State": "CREATING"
    }
  ],
  "UnsuccessfulList": [
  ]
}
```

## DescribeServers

描述一个或多个 Amazon DCV 服务器。

### 主题

- [请求参数](#)
- [响应参数](#)
- [示例](#)

## 请求参数

### ServerIds

要描述的 Amazon DCV 服务器的 ID。如果未指定 ID，则在分页输出中返回所有服务器。

类型：字符串数组

必需：否

### NextToken

用于检索下一页结果的标记。

类型：字符串

必需：否

### MaxResults

请求在分页输出中返回的最大结果数。在使用该参数时，请求仅在单个页面中返回指定数量的结果以及 NextToken 响应元素。可以使用返回的 NextToken 值发送另一个请求，以查看初始请求的其余结果。

有效范围：1 - 1000

默认值：1000

类型：整数

必需：否

## 响应参数

### RequestId

请求的唯一 ID。

### Servers

有关 Amazon DCV 服务器的信息。该数据结构包括以下嵌套的响应参数：

#### Id

Amazon DCV 服务器的唯一 ID。

**Ip**

Amazon DCV 服务器的 IP 地址。

**Hostname**

Amazon DCV 服务器的主机名。

**Endpoints**

有关 Amazon DCV 服务器端点的信息。该数据结构包括以下嵌套的响应参数：

**IpAddress**

服务器终端节点的 IP 地址。

**Port**

服务器终端节点的端口。

**Protocol**

服务器终端节点使用的协议。可能的值包括：

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

**WebUrlPath**

服务器终端节点的 Web URL 路径。仅适用于 HTTP 协议。

**Version**

Amazon DCV 服务器的版本。

**SessionManagerAgentVersion**

在 Amazon DCV 服务器上运行的 Session Manager Agent 版本。

**Availability**

Amazon DCV 服务器的可用性。可能的值包括：

- AVAILABLE - 服务器可用并准备好放置会话。
- UNAVAILABLE - 服务器不可用，并且无法接受放置会话。

**UnavailabilityReason**

Amazon DCV 服务器不可用的原因。可能的值包括：

- SERVER\_FULL - Amazon DCV 服务器已达到它可以运行的最大并发会话数。

- **SERVER\_CLOSED** - 已使用 CloseServer API 使 Amazon DCV 服务器变得不可用。
- **UNREACHABLE\_AGENT** - Session Manager Broker 无法与 Amazon DCV 服务器上的 Session Manager Agent 进行通信。
- **UNHEALTHY\_DCV\_SERVER** - Session Manager Agent 无法与 Amazon DCV 服务器进行通信。
- **EXISTING\_LOGGED\_IN\_USER** - ( 仅限 Windows Amazon DCV 服务器 ) 用户当前使用 RDP 登录到 Amazon DCV 服务器。
- **UNKNOWN** - Session Manager Broker 无法确定原因。

### **ConsoleSessionCount**

Amazon DCV 服务器上的控制台会话数。

### **VirtualSessionCount**

Amazon DCV 服务器上的虚拟会话数。

### **Host**

有关运行 Amazon DCV 服务器的主机服务器的信息。该数据结构包括以下嵌套的响应参数：

#### **Os**

有关主机服务器的操作系统的信息。该数据结构包括以下嵌套的响应参数：

#### **Family**

操作系统系列。可能的值包括：

- **windows** - 主机服务器运行 Windows 操作系统。
- **linux** - 主机服务器运行 Linux 操作系统。

#### **Name**

操作系统的名称。

#### **Version**

操作系统的版本。

#### **KernelVersion**

( 仅限 Linux ) 操作系统的内核版本。

#### **BuildNumber**

( 仅限 Windows ) 操作系统的内部版本号。

## Memory

有关主机服务器的内存的信息。该数据结构包括以下嵌套的响应参数：

### TotalBytes

主机服务器上的总内存（以字节为单位）。

### UsedBytes

主机服务器上使用的内存（以字节为单位）。

## Swap

有关主机服务器的交换文件的信息。该数据结构包括以下嵌套的响应参数：

### TotalBytes

主机服务器上的总交换文件大小（以字节为单位）。

### UsedBytes

主机服务器上使用的交换文件大小（以字节为单位）。

## Aws

仅适用于在 Amazon EC2 实例上运行的 Amazon DCV 服务器。AWS 特定的信息。该数据结构包括以下嵌套的响应参数：

### Region

Amazon EC2 实例的 AWS 区域。

### Ec2InstanceType

Amazon EC2 实例的类型。

### Ec2InstanceId

Amazon EC2 实例的 ID。

### Ec2ImageId

Amazon EC2 映像的 ID。

## CpuInfo

有关主机服务器的 CPU 的信息。该数据结构包括以下嵌套的响应参数：

**Vendor**

主机服务器的 CPU 的供应商。

**ModelName**

主机服务器的 CPU 的型号名称。

**Architecture**

主机服务器的 CPU 的架构。

**NumberOfCpus**

主机服务器上的 CPU 数量。

**PhysicalCorePerCpu**

每个 CPU 的 CPU 核心数。

**CpuLoadAverage**

有关主机服务器的 CPU 负载的信息。该数据结构包括以下嵌套的响应参数：

**OneMinute**

过去 1 分钟内的平均 CPU 负载。

**FiveMinutes**

过去 5 分钟内的平均 CPU 负载。

**FifteenMinutes**

过去 15 分钟内的平均 CPU 负载。

**Gpus**

有关主机服务器的 GPU 的信息。该数据结构包括以下嵌套的响应参数：

**Vendor**

主机服务器的 GPU 的供应商。

**ModelName**

主机服务器的 GPU 的型号名称。

**LoggedInUsers**

当前登录到主机服务器的用户。该数据结构包括以下嵌套的响应参数：

## Username

登录用户的用户名。

## Tags

分配给服务器的标签。该数据结构包括以下嵌套的响应参数：

### Key

标签键。

### Value

标签值。

## 示例

### Python

请求

以下示例描述了所有可用的 Amazon DCV 服务器。结果进行分页以每页显示两个结果。

```
from swagger_client.models.describe_servers_request_data import
    DescribeServersRequestData

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

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

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

响应

以下是示例输出。

```
{
  "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

描述一个或多个 Amazon DCV 会话。

主题

- [请求参数](#)
- [响应参数](#)
- [示例](#)

## 请求参数

### SessionIds

要描述的会话的 ID。

类型：字符串

必需：否

### NextToken

用于检索下一页结果的标记。

类型：字符串

必需：否

### Filters

应用于请求的额外筛选条件。支持的筛选条件包括：

- tag:key - 分配给会话的标签。
- owner - 会话所有者。

类型：字符串

必需：否

## 响应参数

### Id

会话的唯一 ID。

### Name

会话的名称。

### Owner

会话的所有者。

### Server

有关运行会话的服务器的信息。该数据结构包括以下嵌套的响应参数：

#### Ip

Amazon DCV 服务器主机的 IP 地址。

#### Hostname

Amazon DCV 服务器主机的主机名。

#### Port

Amazon DCV 服务器与 Amazon DCV 客户端通信时使用的端口。

### Endpoints

有关 Amazon DCV 服务器端点的信息。该数据结构包括以下嵌套的响应参数：

#### IpAddress

服务器终端节点的 IP 地址。

#### Port

服务器终端节点的端口。

#### Protocol

服务器终端节点使用的协议。可能的值包括：

- HTTP - 终端节点使用 WebSocket ( TCP ) 协议。

- QUIC - 终端节点使用 QUIC ( UDP ) 协议。

### **WebUrlPath**

服务器终端节点的 Web URL 路径。仅适用于 HTTP 协议。

### **Tags**

分配给服务器的标签。该数据结构包括以下嵌套的响应参数：

#### **Key**

标签键。

#### **Value**

标签值。

### **Type**

会话的类型。

### **State**

会话的当前状态。可能的值有：

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

### **Substate**

会话的当前子状态。可能的值有：

- SESSION\_PLACING - 会话正在等待放置在可用的 DCV 服务器上。
- PENDING\_PREPARATION - 会话已创建但不可用；已链接到 DCV 服务器。

### **CreationTime**

创建会话的日期和时间。

## LastDisconnectionTime

上次客户端断开连接的日期和时间。

## NumOfConnections

活动客户端连接数。

## StorageRoot

指定用于会话存储的文件夹的路径。有关 Amazon DCV 会话存储的更多信息，请参阅《Amazon DCV 管理员指南》中的[启用会话存储](#)。

类型：字符串

必需：否

## 示例

### Python

请求

以下示例描述了由 user1 拥有并具有 os=windows 标签的会话。

```
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'}])
```

## 响应

以下是示例输出。

```
{
  "Sessions": [
    {
      "Id": "SessionId1897",
      "Name": "a session name",
      "Owner": "an owner 1890",
      "Server": {
        "Ip": "1.1.1.123",
        "Hostname": "server hostname",
        "Port": "1222",
        "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

删除指定的 Amazon DCV 会话，并将其从 Broker 的缓存中删除。

### 主题

- [请求参数](#)
- [响应参数](#)
- [示例](#)

## 请求参数

### SessionId

要删除的会话的 ID。

类型：字符串

必需：是

### Owner

要删除的会话的所有者。

类型：字符串

必需：是

### Force

从 Broker 的缓存中删除会话，并尝试从 Amazon DCV 服务器中删除会话。这对于从 Broker 缓存中删除过时的会话非常有用。例如，如果 Amazon DCV 服务器已停止，但仍在 Broker 上注册会话，请使用该标记从 Broker 的缓存中清除会话。

请记住，如果会话仍处于活动状态，Broker 将重新缓存会话。

有效值：true | false

类型：布尔值

必需：否

## 响应参数

### SessionId

会话 ID

### State

仅在成功删除会话时返回。指示会话的当前状态。如果请求成功完成，会话将转变为 DELETING 状态。删除会话可能需要几分钟的时间。在已删除会话后，状态将从 DELETING 转变为 DELETED。

### FailureReason

仅在无法删除某些会话时返回。指示无法删除会话的原因。

## 示例

### Python

#### 请求

以下示例删除两个会话 - 一个会话由 user1 拥有并具有 ID SessionId123，另一个会话由 user99 拥有并具有 ID SessionIdabc。

```
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',
'owner99')])
```

#### 响应

以下是示例输出。已成功删除 SessionId123，但无法删除 SessionIdabc。

```
{
  "RequestId": "10311636-df90-4cd1-bcf7-474e9675b7cd",
  "SuccessfulList": [
    {
```

```
        "SessionId": "SessionId123",
        "State": "DELETING"
    }
],
"UnsuccessfulList": [
    {
        "SessionId": "SessionIdabc",
        "FailureReason": "The requested dcvSession does not exist"
    }
]
}
```

## GetSessionConnectionData

获取特定用户到特定 Amazon DCV 会话的连接的连接信息。

主题

- [请求参数](#)
- [响应参数](#)
- [其他信息](#)
- [示例](#)

### 请求参数

#### SessionId

要查看连接信息的会话的 ID。

类型：字符串

必需：是

#### User

要查看连接信息的用户的名称。

类型：字符串

必需：是

## 响应参数

### Id

会话的唯一 ID。

### Name

会话的名称。

### Owner

会话的所有者。

### Server

有关运行会话的服务器的信息。该数据结构包括以下嵌套的响应参数：

#### Ip

Amazon DCV 服务器主机的 IP 地址。

#### Hostname

Amazon DCV 服务器主机的主机名。

#### Port

Amazon DCV 服务器与 Amazon DCV 客户端通信时使用的端口。

### Endpoints

有关 Amazon DCV 服务器端点的信息。该数据结构包括以下嵌套的响应参数：

#### IpAddress

服务器终端节点的 IP 地址。

#### Port

服务器终端节点的端口。

#### Protocol

服务器终端节点使用的协议。可能的值包括：

- HTTP - 终端节点使用 WebSocket ( TCP ) 协议。

- QUIC - 终端节点使用 QUIC ( UDP ) 协议。

**WebUrlPath**

服务器终端节点的 Web URL 路径。仅适用于 HTTP 协议。

**WebUrlPath**

Amazon DCV 服务器配置文件的路径。

**Tags**

分配给服务器的标签。该数据结构包括以下嵌套的响应参数：

**Key**

标签键。

**Value**

标签值。

**Type**

会话的类型。

**State**

会话的当前状态。可能的值有：

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

**CreationTime**

创建会话的日期和时间。

**LastDisconnectionTime**

上次客户端断开连接的日期和时间。

## NumOfConnections

用户到会话的并发连接数。

## ConnectionToken

用于连接到会话的身份验证令牌。

## 其他信息

可以将从该 API 中获取的信息传送到 Amazon DCV 客户端，以便连接到 Amazon DCV 会话。

对于 Amazon DCV Web 客户端，您可以构建一个可在浏览器中打开的 URL。该 URL 采用以下格式：

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

对于 Amazon DCV 本机客户端，您可以使用 `dcv://` 模式构建一个 URL。在安装 Amazon DCV 本机客户端后，它在系统中将自身注册为 `dcv://` URL 的处理程序。该 URL 采用以下格式：

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

### Note

如果您使用 Amazon EC2，则 IP 地址应该是公有地址。如果您的配置在网关后面具有 Amazon DCV 主机，请指定网关地址，而不是 SessionConnectionData API 返回的地址。

## 示例

### Python

请求

以下示例获取具有用户名 `user1` 的用户和具有 ID `sessionId12345` 的会话的连接信息。

```
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')
```

## 响应

以下是示例输出。

```
{
  "Session": {
    "Id": "sessionId12345",
    "Name": "a session name",
    "Owner": "an owner 1890",
    "Server": {
      "Ip": "1.1.1.123",
      "Hostname": "server hostname",
      "Port": "1222",
      "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":
"EXAMPLEi0iJm0WM1YTRhZi1jZmU0LTQ0ZjEtYjZlOC04ZjY0YjM4ZTE2ZDkiLCJ0eXAiOiJKV1QiLCJhbGciOiJSUz
tngiKXEvUxhhJm3BPJYRs9NPE4GCJRTc13EXAMPLEIxNEPPH5IMcVmR0fU1WKPnry4ypPTp3rsZ7YWjCTSfs1GoN3R
Kqtpd5GH0D-E8FwsedV-
Q2bRQ4y9y1q0MgFU4QjaSMypUuYR0YjkCaoainjmEZew4A33fG40wATrBvoivBiNwdNpytHX2CD0uk_k0k_DWeZjMvv9
h_GaMgHmltqBIA4jdPD7i0CmC2e7413KFy-
EQ4Ej1cM7RjLwhFuWpKWAVJxogJjYpfoKkaPo4KxvJjJIPYhkscklINQpe2W5rnlxCq7sC7ptcGw17DUobP7egRv9H37
hK1G4G8erHv19HIrTR9_c884fNrTCC8DvC062e4KYdLkAhhJmboN9CAGIGFyd2c1AY_CzzvDL0EXAMPLE"
}

```

## GetSessionScreenshots

获取一个或多个 Amazon DCV 会话的屏幕截图。

屏幕截图的图像文件类型和分辨率取决于 Session Manager Broker 配置。要修改图像文件类型，请配置 `session-screenshot-format` 参数。要修改分辨率，请配置 `session-screenshot-max-width` 和 `session-screenshot-max-height` 参数。有关更多信息，请参阅《Amazon DCV Session Manager 管理员指南》中的 [Broker 配置文件](#)。

## 主题

- [请求参数](#)
- [响应参数](#)
- [示例](#)

## 请求参数

### **SessionId**

要从中获取屏幕截图的 Amazon DCV 会话的 ID。

类型：字符串

必需：是

## 响应参数

### **RequestId**

请求的唯一 ID。

### **SuccessfulList**

有关成功屏幕截图的信息。该数据结构包括以下嵌套的响应参数：

#### **SessionScreenshot**

有关屏幕截图的信息。该数据结构包括以下嵌套的响应参数：

##### **SessionId**

从中获取屏幕截图的 Amazon DCV 会话的 ID。

##### **Images**

有关图像的信息。该数据结构包括以下嵌套的响应参数：

**Format**

图像的格式。可能的值包括：jpeg 和 png。

**Data**

屏幕截图 Base64 编码格式。

**CreationTime**

获取屏幕截图的日期和时间。

**Primary**

指示屏幕截图是否为 Amazon DCV 会话的主显示器。

**UnsuccessfulList**

有关失败屏幕截图的信息。该数据结构包括以下嵌套的响应参数：

**GetSessionScreenshotRequestData**

失败的原始请求。

**SessionId**

从中获取屏幕截图的 Amazon DCV 会话的 ID。

**FailureReason**

失败的原因。

## 示例

### Python

请求

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

```
from swagger_client.models.describe_servers_request_data import
    DescribeServersRequestData

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

```
    return api_instance

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

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

## 响应

以下是示例输出。

```
{
  "RequestId": "542735ef-f6ab-47d8-90e5-23df31d8d166",
  "SuccessfulList": [
    {
      "SessionScreenshot": {
        "SessionId": "sessionId1",
        "Images": [
          {
            "Format": "png",
            "Data": "iVBORw0KGgoAAAANSUhEUgAAEXAMPLE",
            "CreationTime": "2021-03-30T15:47:06.822Z",
            "Primary": true
          }
        ]
      }
    }
  ],
  "UnsuccessfulList": [
    {
      "GetSessionScreenshotRequestData": {
        "SessionId": "sessionId2"
      },
      "FailureReason": "Dcv session not found."
    }
  ]
}
```

# OpenServers

打开一个或多个 Amazon DCV 服务器。您必须先将服务器的状态变为打开，然后才能在 Amazon DCV 服务器上创建 Amazon DCV 会话。在 Amazon DCV 服务器状态为打开后，您可以在服务器上创建 Amazon DCV 会话。

## 主题

- [请求参数](#)
- [响应参数](#)
- [示例](#)

## 请求参数

### ServerId

要打开的服务器的 ID。

类型：字符串

必需：是

## 响应参数

### RequestId

请求的唯一 ID。

### SuccessfulList

有关成功打开的 Amazon DCV 服务器的信息。该数据结构包括以下嵌套的响应参数：

#### ServerId

成功打开的服务器的 ID。

### UnsuccessfulList

有关无法打开的 Amazon DCV 服务器的信息。该数据结构包括以下嵌套的响应参数：

#### OpenServerRequestData

有关失败的原始请求的信息。该数据结构包括以下嵌套的响应参数：

**ServerId**

无法打开的 Amazon DCV 服务器的 ID。

**FailureCode**

失败代码。

**FailureReason**

失败的原因。

## 示例

### Python

请求

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

```
from swagger_client.models import OpenServerRequestData

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

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

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

响应

以下是示例输出。

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

## UpdateSessionPermissions

更新特定 Amazon DCV 会话的用户权限。

主题

- [请求参数](#)
- [响应参数](#)
- [示例](#)

### 请求参数

#### SessionId

要更改权限的会话的 ID。

类型：字符串

必需：是

#### Owner

要更改权限的会话的所有者。

类型：字符串

必需：是

## PermissionFile

要使用的权限文件的 Base64 编码内容。有关更多信息，请参阅《Amazon DCV 管理员指南》中的[配置 Amazon DCV 授权](#)。

类型：字符串

必需：是

## 响应参数

### SessionId

会话 ID。

## 示例

### Python

请求

以下示例为具有会话 ID SessionId1897 的会话设置新权限。

```
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_instance

def 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')])
```

## 响应

以下是示例输出。

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

# Amazon DCV Session Manager 发布说明和文档历史记录

该页面提供 Amazon DCV Session Manager 的发布说明和文档历史记录。

## 主题

- [Amazon DCV Session Manager 发布说明](#)
- [文档历史记录](#)

## Amazon DCV Session Manager 发布说明

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

## 主题

- [2024.0-457 - 2024 年 10 月 1 日](#)
- [2023.1-17652 - 2024 年 8 月 1 日](#)
- [2023.1-16388 - 2024 年 6 月 26 日](#)
- [2023.1 - 2023 年 11 月 9 日](#)
- [2023.0-15065 - 2023 年 5 月 4 日](#)
- [2023.0-14852 - 2023 年 3 月 28 日](#)
- [2022.2-13907 - 2022 年 11 月 11 日](#)
- [2022.1-13067 - 2022 年 6 月 29 日](#)
- [2022.0-11952 - 2022 年 2 月 23 日](#)
- [2021.3-11591 - 2021 年 12 月 20 日](#)
- [2021.2-11445 - 2021 年 11 月 18 日](#)
- [2021.2-11190 - 2021 年 10 月 11 日](#)
- [2021.2-11042 - 2021 年 9 月 1 日](#)
- [2021.1-10557 - 2021 年 5 月 31 日](#)
- [2021.0-10242 - 2021 年 4 月 12 日](#)
- [2020.2-9662 - 2020 年 12 月 4 日](#)
- [2020.2-9508 - 2020 年 11 月 11 日](#)

## 2024.0-457 - 2024 年 10 月 1 日

内部版本号	更改和错误修复
<ul style="list-style-type: none"><li>• Broker : 457</li><li>• Agent : 748</li><li>• CLI : 140</li></ul>	<ul style="list-style-type: none"><li>• 将 NICE DCV 更名为 Amazon DCV。</li><li>• 增加了对 Ubuntu 24.04 的支持。</li></ul>

## 2023.1-17652 - 2024 年 8 月 1 日

内部版本号	更改和错误修复
<ul style="list-style-type: none"><li>• Broker : 426</li><li>• Agent : 748</li><li>• CLI : 140</li></ul>	<ul style="list-style-type: none"><li>• 缺陷修复和性能改进。</li></ul>

## 2023.1-16388 - 2024 年 6 月 26 日

内部版本号	更改和错误修复
<ul style="list-style-type: none"><li>• Broker : 417</li><li>• Agent : 748</li><li>• CLI : 140</li></ul>	<ul style="list-style-type: none"><li>• 修复了内存不正确显示为 TB 而不是 GB 的错误。</li><li>• 缺陷修复和性能改进。</li></ul>

## 2023.1 - 2023 年 11 月 9 日

内部版本号	更改和错误修复
<ul style="list-style-type: none"><li>• Broker : 410</li><li>• Agent : 732</li><li>• CLI : 140</li></ul>	<ul style="list-style-type: none"><li>• 错误修复和性能改进</li></ul>

## 2023.0-15065 - 2023 年 5 月 4 日

内部版本号	更改和错误修复
<ul style="list-style-type: none"><li>• Broker : 392</li><li>• Agent : 675</li><li>• CLI : 132</li></ul>	<ul style="list-style-type: none"><li>• 添加了对 ARM 平台上的 Red Hat Enterprise Linux 9、Rocky Linux 9 和 CentOS Stream 9 的支持。</li></ul>

## 2023.0-14852 - 2023 年 3 月 28 日

内部版本号	更改和错误修复
<ul style="list-style-type: none"><li>• Broker : 392</li><li>• Agent : 642</li><li>• CLI : 132</li></ul>	<ul style="list-style-type: none"><li>• 添加了对 Red Hat Enterprise Linux 9、Rocky Linux 9 和 CentOS Stream 9 的支持。</li></ul>

## 2022.2-13907 - 2022 年 11 月 11 日

内部版本号	更改和错误修复
<ul style="list-style-type: none"><li>• Broker : 382</li><li>• Agent : 612</li><li>• CLI : 123</li></ul>	<ul style="list-style-type: none"><li>• 添加了一个 Substate 字段以作为 DescribeSessions 响应。</li><li>• 修复了可能导致 CLI 无法连接到 Broker 的问题 ( 具体取决于使用的 URL ) 。</li></ul>

## 2022.1-13067 - 2022 年 6 月 29 日

内部版本号	更改和错误修复
<ul style="list-style-type: none"><li>• Broker : 355</li><li>• Agent : 592</li><li>• CLI : 114</li></ul>	<ul style="list-style-type: none"><li>• 添加了在 AWS Graviton 实例上运行 Broker 的支持。</li><li>• 为 Ubuntu 22.04 添加了 Agent 和 Broker 支持。</li></ul>

## 2022.0-11952 - 2022 年 2 月 23 日

内部版本号	更改和错误修复
<ul style="list-style-type: none"><li>• Broker : 341</li><li>• Agent : 520</li><li>• CLI : 112</li></ul>	<ul style="list-style-type: none"><li>• 为 Agent 添加了日志轮换功能。</li><li>• 添加了配置参数以在 Broker 中设置 Java 主目录。</li><li>• 在 Broker 中改进了从缓存到磁盘的数据刷新。</li><li>• 修复了 CLI 中的 URL 验证。</li></ul>

## 2021.3-11591 - 2021 年 12 月 20 日

内部版本号	新功能
<ul style="list-style-type: none"><li>• Broker : 307</li><li>• Agent : 453</li><li>• CLI : 92</li></ul>	<ul style="list-style-type: none"><li>• 为与 Amazon DCV Connection Gateway 集成添加了支持。</li><li>• 为 Ubuntu 18.04 和 Ubuntu 20.04 添加了 Broker 支持。</li></ul>

## 2021.2-11445 - 2021 年 11 月 18 日

内部版本号	更改和错误修复
<ul style="list-style-type: none"><li>• Broker : 288</li><li>• Agent : 413</li><li>• CLI : 54</li></ul>	<ul style="list-style-type: none"><li>• 修复了包含 Windows 域的登录名的验证问题。</li></ul>

## 2021.2-11190 - 2021 年 10 月 11 日

内部版本号	更改和错误修复
<ul style="list-style-type: none"><li>• Broker : 254</li><li>• Agent : 413</li><li>• CLI : 54</li></ul>	<ul style="list-style-type: none"><li>• 修复了命令行界面中的一个问题，该问题导致无法启动 Windows 会话。</li></ul>

## 2021.2-11042 - 2021 年 9 月 1 日

内部版本号	新功能	更改和错误修复
<ul style="list-style-type: none"> <li>Broker : 254</li> <li>Agent : 413</li> <li>CLI : 37</li> </ul>	<ul style="list-style-type: none"> <li>Amazon DCV Session Manager 现在提供命令行界面 ( CLI ) 支持。您可以在 CLI 中创建和管理 Amazon DCV 会话，而不是调用 API。</li> <li>Amazon DCV Session Manager 引入了 Broker 数据持久性。为了获得更高的可用性，Broker 可以将服务器状态信息持久保留在外部数据存储上，并在启动时恢复数据。</li> </ul>	<ul style="list-style-type: none"> <li>在注册外部授权服务器时，您现在可以指定授权服务器用于对 JSON 格式的 Web 令牌进行签名的算法。通过该更改，您可以将 Azure AD 作为外部授权服务器。</li> </ul>

## 2021.1-10557 - 2021 年 5 月 31 日

内部版本号	新功能	更改和错误修复
<ul style="list-style-type: none"> <li>Broker : 214</li> <li>Agent : 365</li> </ul>	<ul style="list-style-type: none"> <li>Amazon DCV Session Manager 添加了对传递到 Linux 上的自动运行文件的输入参数的支持。</li> <li>现在可以将服务器属性作为要求传递给 <a href="#">CreateSessions</a> API。</li> </ul>	<ul style="list-style-type: none"> <li>我们修复了 Windows 上的自动运行文件存在的一个问题。</li> </ul>

## 2021.0-10242 - 2021 年 4 月 12 日

内部版本号	更改和错误修复
<ul style="list-style-type: none"> <li>Broker : 183</li> <li>Agent : 318</li> </ul>	<ul style="list-style-type: none"> <li>Amazon DCV Session Manager 引入了以下新的 API : <ul style="list-style-type: none"> <li><a href="#">OpenServers</a></li> <li><a href="#">CloseServers</a></li> <li><a href="#">DescribeServers</a></li> <li><a href="#">GetSessionScreenshots</a></li> </ul> </li> </ul>

内部版本号	更改和错误修复
	<ul style="list-style-type: none"> <li>它还引入了以下新的配置参数： <ul style="list-style-type: none"> <li><a href="#">Broker 参数</a>：<code>session-screenshot-max-width</code>、<code>session-screenshot-max-height</code>、<code>session-screenshot-format</code>、<code>create-sessions-queue-max-size</code> 和 <code>create-sessions-queue-max-time-seconds</code>。</li> <li><a href="#">Agent 参数</a>：<code>agent.autorun_folder</code>、<code>max_virtual_sessions</code> 和 <code>max_concurrent_sessions_per_user</code>。</li> </ul> </li> </ul> <p><a href="#">Agent 参数</a>：<code>agent.autorun_folder</code>、<code>max_virtual_sessions</code> 和 <code>max_concurrent_sessions_per_user</code>。</p> <p><a href="#">Agent 参数</a>：<code>agent.autorun_folder</code>、<code>max_virtual_sessions</code> 和 <code>max_concurrent_sessions_per_user</code>。</p>

## 2020.2-9662 - 2020 年 12 月 4 日

内部版本号	更改和错误修复
<ul style="list-style-type: none"> <li>Broker：114</li> <li>Agent：211</li> </ul>	<ul style="list-style-type: none"> <li>我们修复了自动生成的 TLS 证书存在的一个问题，该问题导致 Broker 无法启动。</li> </ul>

## 2020.2-9508 - 2020 年 11 月 11 日

内部版本号	更改和错误修复
<ul style="list-style-type: none"> <li>Broker：78</li> <li>Agent：183</li> </ul>	<ul style="list-style-type: none"> <li>Amazon DCV Session Manager 初始版本。</li> </ul>

## 文档历史记录

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

更改	描述	日期
Amazon DCV 版本 2024.0-457	已针对 Amazon DCV 2024.0-457 更新了 Amazon DCV Session Manager。有关更多信息，请参阅 <a href="#">2024.0-457 - 2024 年 10 月 1 日</a> 。	2024 年 9 月 30 日
Amazon DCV 版本 2023.1-17652	已针对 Amazon DCV 2023.1-17652 更新了 Amazon DCV Session Manager。有关更多信息，请参阅 <a href="#">2023.1-17652 - 2024 年 8 月 1 日</a> 。	2024 年 8 月 1 日
Amazon DCV 版本 2023.1-16388	已针对 Amazon DCV 2023.1-16388 更新了 Amazon DCV Session Manager。有关更多信息，请参阅 <a href="#">2023.1-16388 - 2024 年 6 月 26 日</a> 。	2024 年 6 月 26 日
Amazon DCV 版本 2023.1	已针对 Amazon DCV 2023.1 更新了 Amazon DCV Session Manager。有关更多信息，请参阅 <a href="#">2023.1 - 2023 年 11 月 9 日</a> 。	2023 年 11 月 9 日
Amazon DCV 版本 2023.0	已针对 Amazon DCV 2023.0 更新了 Amazon DCV Session Manager。有关更多信息，请参阅 <a href="#">2023.0-14852 - 2023 年 3 月 28 日</a> 。	2023 年 3 月 28 日
Amazon DCV 版本 2022.2	已针对 Amazon DCV 2022.2 更新了 Amazon DCV Session Manager。有关更多信息，请参阅 <a href="#">2022.2-13907 - 2022 年 11 月 11 日</a> 。	2022 年 11 月 11 日

更改	描述	日期
Amazon DCV 版本 2022.1	已针对 Amazon DCV 2022.1 更新了 Amazon DCV Session Manager。有关更多信息，请参阅 <a href="#">2022.1-13067 - 2022 年 6 月 29 日</a> 。	2022 年 6 月 29 日
Amazon DCV 版本 2022.0	已针对 Amazon DCV 2022.0 更新了 Amazon DCV Session Manager。有关更多信息，请参阅 <a href="#">2022.0-11952 - 2022 年 2 月 23 日</a> 。	2022 年 2 月 23 日
Amazon DCV 版本 2021.3	已针对 Amazon DCV 2021.3 更新了 Amazon DCV Session Manager。有关更多信息，请参阅 <a href="#">2021.3-11591 - 2021 年 12 月 20 日</a> 。	2021 年 12 月 20 日
Amazon DCV 版本 2021.2	已针对 Amazon DCV 2021.2 更新了 Amazon DCV Session Manager。有关更多信息，请参阅 <a href="#">2021.2-11042 - 2021 年 9 月 1 日</a> 。	2021 年 9 月 1 日
Amazon DCV 版本 2021.1	已针对 Amazon DCV 2021.1 更新了 Amazon DCV Session Manager。有关更多信息，请参阅 <a href="#">2021.1-10557 - 2021 年 5 月 31 日</a> 。	2021 年 5 月 31 日
Amazon DCV 版本 2021.0	已针对 Amazon DCV 2021.0 更新了 Amazon DCV Session Manager。有关更多信息，请参阅 <a href="#">2021.0-10242 - 2021 年 4 月 12 日</a> 。	2021 年 4 月 12 日
Amazon DCV Session Manager 初始版本	该内容的第一版。	2020 年 11 月 11 日