



AWS Ground Station 代理用户指南

# AWS Ground Station



# AWS Ground Station: AWS Ground Station 代理用户指南

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、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

概述 .....	1
AWS Ground Station 代理是什么？ .....	1
AWS Ground Station 代理的特点 .....	2
代理要求 .....	3
VPC图表 .....	4
支持的操作系统 .....	5
通过 AWS Ground Station 代理传输数据 .....	6
多个数据流，单个接收器 .....	6
多个数据流，多个接收器 .....	7
Amazon EC2 实例选择和CPU规划 .....	9
支持的亚马逊EC2实例类型 .....	9
CPU核心规划 .....	10
收集架构信息 .....	11
CPU赋值示例 .....	12
附录：c5.24xlarge 的lscpu -p输出（完整） .....	13
安装座席 .....	16
使用 AWS CloudFormation 模板 .....	16
步骤 1：创建 AWS 资源 .....	16
步骤 2：检查代理状态 .....	16
手动安装 EC2 .....	16
步骤 1：创建AWS资源 .....	16
步骤 2：创建EC2实例 .....	17
步骤 3：下载并安装座席 .....	17
步骤 4：配置座席 .....	18
步骤 5：应用性能优化 .....	18
步骤 6：管理座席 .....	18
管理座席 .....	19
AWS Ground Station 代理配置 .....	19
AWS Ground Station 代理启动 .....	19
AWS Ground Station 代理停止 .....	20
AWS Ground Station 代理升级 .....	20
AWS Ground Station 代理降级 .....	21
AWS Ground Station 代理卸载 .....	22
AWS Ground Station 代理状态 .....	22

AWS Ground Station 代理RPM信息 .....	23
配置代理 .....	24
代理配置文件 .....	24
示例 .....	24
现场细分 .....	24
EC2实例性能调整 .....	28
调整硬件中断和接收队列——影响CPU和网络 .....	28
Tune Rx 中断合并-影响网络 .....	29
调整 Rx 环形缓冲区-影响网络 .....	29
调整 CPU C-State-影响 CPU .....	30
保留入口端口-影响网络 .....	30
Reboot .....	30
附录：中断/调RPS谐的推荐参数 .....	31
准备开始联络 DigIF .....	33
最佳实践 .....	34
亚马逊EC2最佳实践 .....	34
Linux 调度器 .....	34
AWS Ground Station 托管前缀列表 .....	34
单次联络限制 .....	34
与 AWS Ground Station 代理一起运行服务和进程 .....	34
举个使用c5.24xlarge实例的例子 .....	35
关联服务 (systemd) .....	35
关联进程 (脚本) .....	36
故障排除 .....	38
座席启动失败 .....	38
故障排除 .....	38
AWS Ground Station 代理日志 .....	39
没有可用的联系人 .....	39
获取支持 .....	40
代理版本说明 .....	41
最新代理版本 .....	41
版本 1.0.3555.0 .....	41
已弃用的代理版本 .....	41
版本 1.0.2942.0 .....	41
版本 1.0.2716.0 .....	42
版本 1.0.2677.0 .....	42

---

RPM安装验证 .....	44
最新代理版本 .....	41
版本 1.0.3555.0 .....	41
验证 RPM .....	44
文档历史记录 .....	46
.....	xlvii

# 概述

## AWS Ground Station 代理是什么？

Ground Station Agent 可用RPM作，使您能够在AWS地面站接触期间AWS接收（下行链路）同步宽带数字中频（digiF）数据流。您可以选择两个数据传输选项：

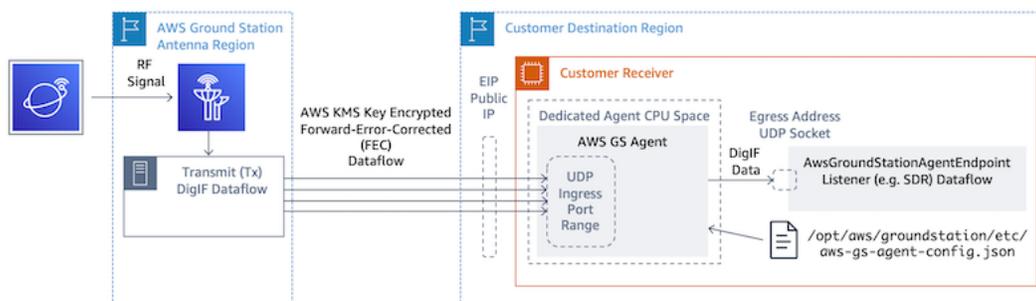
1. 向EC2实例传输数据-将数据传输到您拥有的EC2实例。由您管理代 AWS Ground Station 理。如果您需要近乎实时的数据处理，则选择此选项。有关[数据传输的信息，请参阅《向 Amazon 弹性计算云传输EC2数据》指南](#)。
2. 向 S3 存储桶传输数据-通过 Ground Stati AWS on 托管服务将数据传输到您拥有的 S3 存储桶。有关 S3 数据传输的信息，请参阅[入门 AWS Ground Station](#)指南。

两种数据传输模式都需要您创建一组AWS资源。强烈建议使用 CloudFormation 来创建AWS资源，以确保可靠性、准确性和可支持性。每个联系人只能向EC2或 S3 传送数据，但不能同时向两者传送数据。

### Note

由于 S3 数据传输是一项 Ground Station 托管服务，因此本指南重点介绍向您的EC2实例传输数据。

下图显示了使用软件定义电台 (SDR) 或类似监听器从 AWS Ground Station 天线区域到您的EC2实例的 digiF 数据流。



## AWS Ground Station 代理的特点

AWS Ground Station 代理接收数字中频 (digiF) 下行链路数据并输出解密后的数据，从而实现以下功能：

- digiF 下行链路能力从 40 MHz 到 400 MHz 的带宽不等。
- 向网络上的AWS任何公共 IPAWS ( 弹性 IP ) 传输高速率、低抖动 digiF 数据。
- 使用前向纠错 (FEC) 实现可靠的数据传输。
- 使用客户托管 AWS KMS 密钥进行加密来保护数据传输。

# 代理要求

## Note

本 AWS Ground Station 代理指南假设您已使用[AWS Ground Station 入门指南](#)登录 Ground Station。

AWS Ground Station 代理接收器EC2实例需要一组依赖AWS资源才能安全可靠地将 DigiF 数据传送到您的端点。

1. VPC在其中启动EC2接收器。
2. 用于数据加密/解AWSKMS密的密钥。
3. 为[SSM会话管理器](#)配置的SSH密钥或EC2实例配置文件。
4. 网络/安全组规则允许：
  1. UDP来自 AWS Ground Station 您的数据流终端节点组中指定的端口的流量。座席保留一系列连续端口，用于向入口数据流端点传送数据。
  2. SSH访问您的实例（注意：您也可以使用AWS会话管理器访问您的EC2实例）。
  3. 读取可公开访问的 S3 存储桶的访问权限以进行座席管理。
  4. SSL端口 443 上的流量允许代理与 AWS Ground Station 服务通信。
  5. 来自 AWS Ground Station 托管前缀列表的流量 `com.amazonaws.global.groundstation`。

此外，还需要包括公有子网的VPC配置。有关子网配置的背景信息，请参阅《[VPC用户指南](#)》。

兼容的配置：

1. 在公有子网中与您的EC2实例关联的弹性 IP。
2. 与公有子网ENI中的关联的弹性 IP，连接到您的EC2实例（与公有子网位于同一可用区中的任何子网中）。

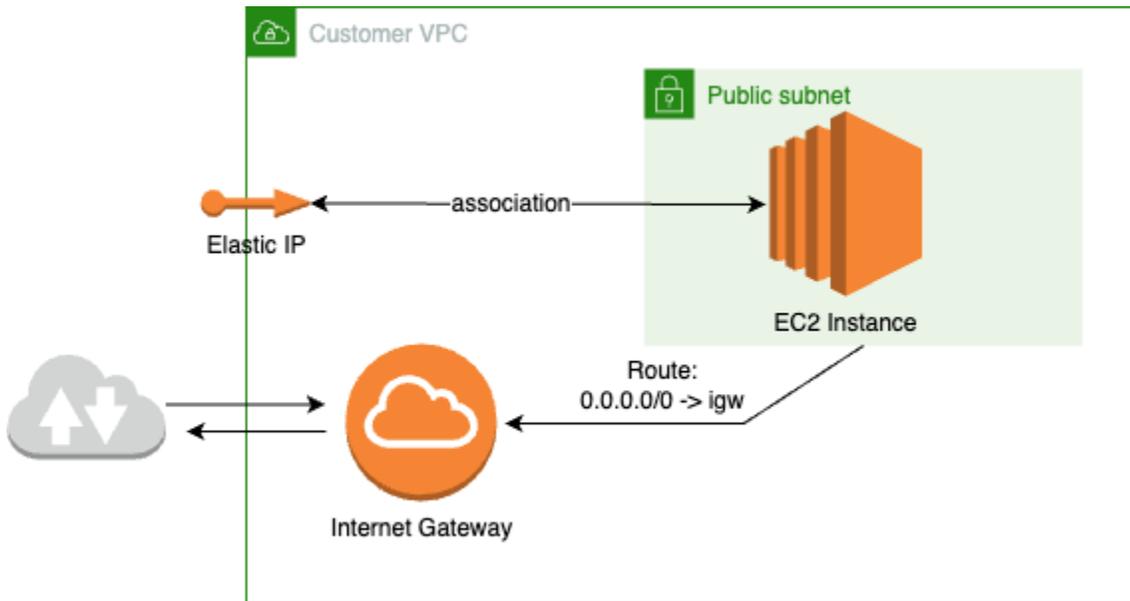
您可以使用与您的EC2实例相同的安全组，也可以指定至少包含以下最低限度规则集的安全组：

- UDP来自 AWS Ground Station 您的数据流终端节点组中指定的端口的流量。

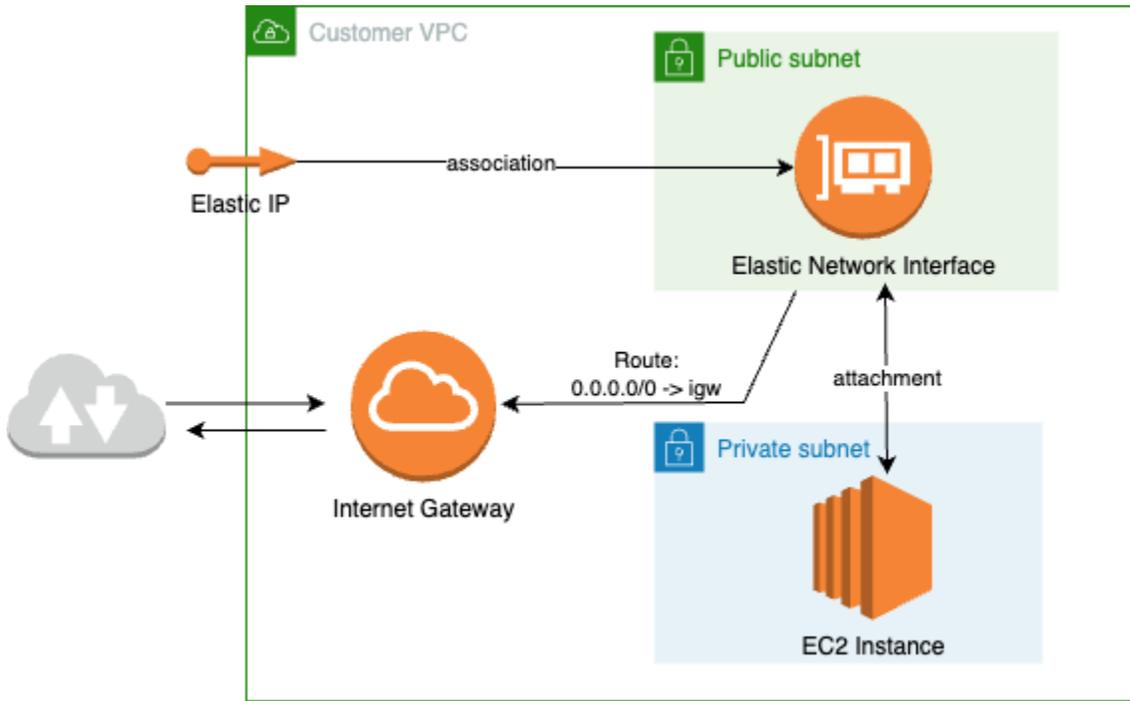
有关预配置了这些资源 AWS CloudFormation EC2的数据传送模板的示例，请参阅[使用 AWS Ground Station 代理 \( 宽带 \) 的公共广播卫星](#)。

## VPC图表

图：公有子网中与您的EC2实例关联的弹性 IP



图：与公有子网ENI中的关联的弹性 IP，连接到私有子网中的EC2实例



## 支持的操作系统

内核版本为 5.10+ 的 Amazon Linux 2

支持的实例类型列在 [Amazon EC2 实例选择和CPU规划](#)

# 通过 AWS Ground Station 代理传输数据

下图概述了宽带数字中频 (digiF) 接触 AWS Ground Station 期间数据的流经情况。

AWS Ground Station 代理将负责协调联系人的数据平面组件。在安排联系之前，必须正确配置和启动代理，并且必须使用注册代理（代理启动时自动注册）AWS Ground Station。此外，数据接收软件（例如软件定义的无线电）必须运行并配置为在接收数据 [AwsGroundStationAgentEndpoint.ingressAddress](#)。

在幕后，AWS Ground Station 代理将接收来自传输中应用的 AWS KMS 加密的任务 AWS Ground Station 并撤消在传输过程中应用的加密，然后将其转发 `egressAddress` 到您的软件定义无线电 (SDR) 正在监听的目标端点。AWS Ground Station 代理及其底层组件将遵守配置文件中设置的 CPU 边界，以确保它不会影响到在实例上运行的其他应用程序的性能。

您必须在联系中涉及的接收者实例上运行 AWS Ground Station 代理。如果您希望在单个接收器实例上接收所有数据流，则单个 AWS Ground Station 代理可以编排多个数据流，如下所示。

## 多个数据流，单个接收器

示例方案：

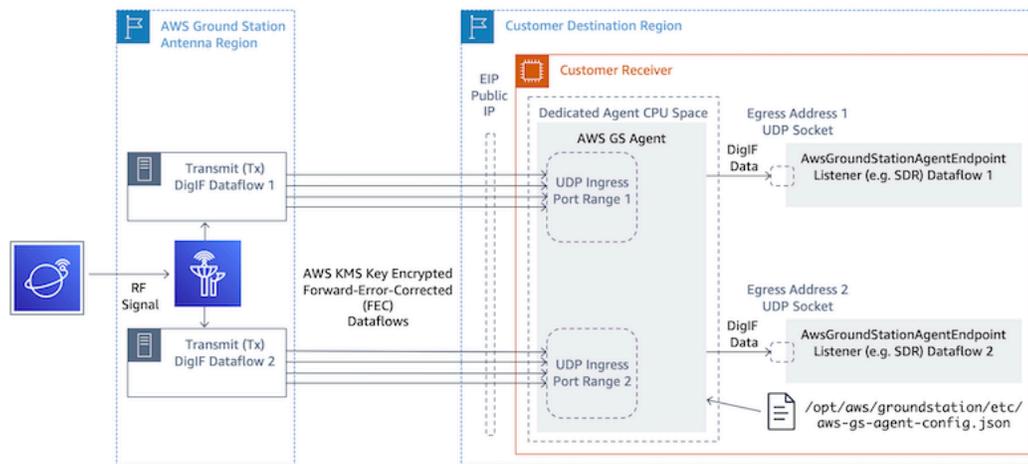
当 digif 数据流向同一个接收器实例时，您希望接收两个天线下行链路。EC2 这两个下行链路将是 200 MHz 和 100 MHz。

`AwsGroundStationAgentEndpoints`:

将有两个 `AwsGroundStationAgentEndpoint` 资源，每个资源对应一个数据流。两个端点将具有相同的公共 IP 地址 (`ingressAddress.socketAddress.name`)。入口 `portRange` 不应重叠，因为数据流是在同一个实例上接收的。EC2 二者的 `egressAddress.socketAddress.port` 必须唯一。

CPU 规划：

- 1 个内核 (2 vCPU)，用于在实例上运行单个 AWS Ground Station 代理。
- 6 个内核 (12 vCPU) 用于接收 digif Dataflow 1 (在表中 MHz 查找 200 个)。 [CPU 核心规划](#)
- 4 个内核 (8 vCPU) 用于接收 digif Dataflow 2 (在表中 MHz 查找 100 个)。 [CPU 核心规划](#)
- 总专用代理 CPU 空间 = 同一插槽上的 11 个内核 (22 vCPU)。



## 多个数据流，多个接收器

示例方案：

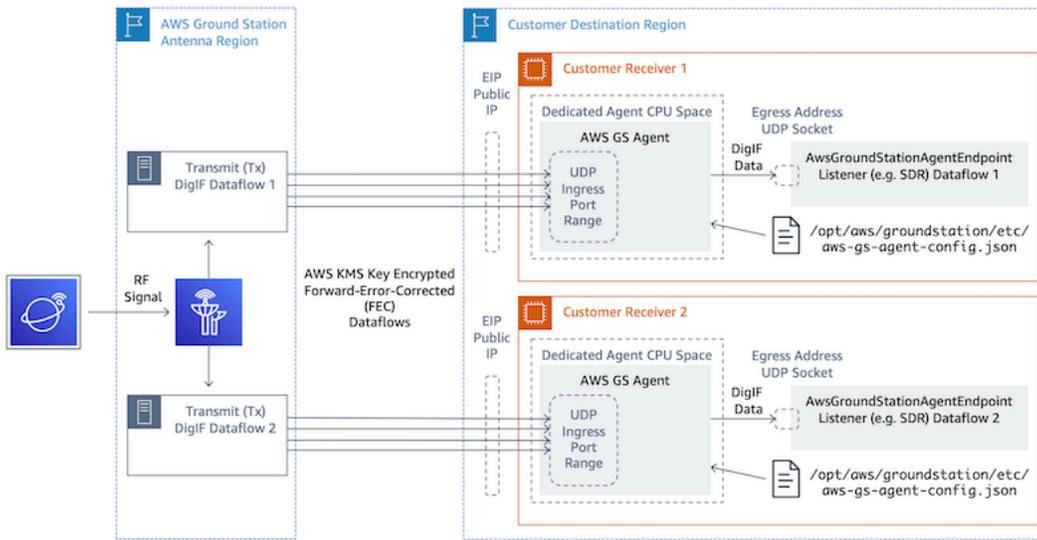
当 digif 数据流向不同的接收机实例时，您希望接收两个天线下行链路。EC2 两个下行链路都将是 400 MHz。

AwsGroundStationAgentEndpoints:

将有两个 AwsGroundStationAgentEndpoint 资源，每个资源对应一个数据流。端点将具有不同的公共 IP 地址 (ingressAddress.socketAddress.name)。ingressAddress 或 egressAddress 都对端口值没有限制，因为数据流是在单独的基础架构上接收的，并且不会相互冲突。

CPU 规划：

- 接收器实例 1
  - 1 个内核 (2 vCPU)，用于在实例上运行单个 AWS Ground Station 代理。
  - 9 个内核 (18 vCPU) 用于接收 digif Dataflow 1 (在表中 MHz 查找 400)。 [CPU 核心规划](#)
  - 总专用代理 CPU 空间 = 同一插槽上的 10 个内核 (20 vCPU)。
- 接收器实例 2
  - 1 个内核 (2 vCPU)，用于在实例上运行单个 AWS Ground Station 代理。
  - 9 个内核 (18 vCPU) 用于接收 digif Dataflow 2 (在表中 MHz 查找 400)。 [CPU 核心规划](#)
  - 总专用代理 CPU 空间 = 同一插槽上的 10 个内核 (20 vCPU)。



# Amazon EC2 实例选择和CPU规划

## 支持的亚马逊EC2实例类型

由于计算密集型的数据传输工作流程，AWS Ground Station 代理需要专用CPU内核才能运行。我们支持以下实例类型。请参阅 [CPU核心规划](#)，确定最适用您的用例的实例类型。

实例类型	默认 vCPUs	默认CPU内核
c5.12xlarge	48	24
c5.18xlarge	72	36
c5.24xlarge	96	48
c5n.18xlarge	72	36
c5n.metal	72	36
c6i.32xlarge	128	64
g4dn.12xlarge	48	24
g4dn.16xlarge	64	32
g4dn.metal	96	48
m4.16xlarge	64	32
m5.12xlarge	48	24
m5.24xlarge	96	48
m6i.32xlarge	128	64
p3dn.24xlarge	96	48
p4d.24xlarge	96	48
r5.24xlarge	96	48

实例类型	默认 vCPUs	默认CPU内核
r5.metal	96	48
r5n.24xlarge	96	48
r5n.metal	96	48
r6i.32xlarge	128	64

## CPU核心规划

AWS Ground Station 代理需要专用的处理器内核，这些内核共享每个数据流的三级缓存。该代理旨在利用超线程 (HT) CPU 对，并要求保留 HT 对以供其使用。超线程对是包含在单个内核中的一对虚拟 CPUs (vCPU)。下表提供了数据流数据速率与为代理为单个数据流保留的所需内核数量的映射。此表假设 Cascade Lake 或更高版本，CPUs 并且适用于任何支持的实例类型。如果您的带宽在表中的条目之间，请选择第二高的带宽。

代理需要一个额外的预留内核用于管理和协调，因此所需的内核总数将是每个数据流所需的内核（如下图所示）加上一个额外内核 (2 vCPUs) 的总和。

AntennaDownlink 带宽 (MHz)	预期 VITA -49.2 digiF 数据速率 (MB/s)	内核数 (HT CPU 对)	总计 v CPU
50	1000	3	6
100	2000	4	8
150	3000	5	10
200	4000	6	12
250	5000	6	12
300	6000	7	14
350	7000	8	16

AntennaDownlink 带宽 (MHz)	预期 VITA -49.2 digiF 数据速率 (MB/s)	内核数 ( HT CPU 对 )	总计 v CPU
400	8000	9	18

## 收集架构信息

lscpu提供了有关您的系统架构的信息。基本输出显示哪个 vCPUs ( 标记为 CPU "" ) 属于哪些NUMA节点 ( 并且每个NUMA节点共享一个 L3 缓存 )。下面我们将检查一个c5.24xlarge实例，以收集配置 AWS Ground Station 代理所需的信息。这包括有用的信息，例如数量vCPUs、内核和 v CPU 到节点的关联。

```
> lscpu
Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
CPU(s): 96
On-line CPU(s) list: 0-95
Thread(s) per core: 2          <-----
Core(s) per socket: 24
Socket(s): 2
NUMA node(s): 2
Vendor ID: GenuineIntel
CPU family: 6
Model: 85
Model name: Intel(R) Xeon(R) Platinum 8275CL CPU @ 3.00GHz
Stepping: 7
CPU MHz: 3601.704
BogoMIPS: 6000.01
Hypervisor vendor: KVM
Virtualization type: full
L1d cache: 32K
L1i cache: 32K
L2 cache: 1024K
L3 cache: 36608K
NUMA node0 CPU(s): 0-23,48-71   <-----
NUMA node1 CPU(s): 24-47,72-95  <-----
```

专用于 AWS Ground Station 代理的内核应包括每个分配 vCPUs 的内核的两个内核。数据流的所有内核都必须存在于同 NUMA 一个节点上。该 `lscpu` 命令的 `-p` 选项为我们提供了配置代理所需的核 CPU 关联。相关字段是 CPU ( 我们所说的 vCPU )、Core 和 L3 ( 表示该内核共享哪个 L3 缓存 )。请注意，在大多数英特尔处理器上，NUMA 节点等于三级缓存。

考虑以下 `lscpu -p` 输出子集 `ac5.24xlarge` ( 为清楚起见，进行了缩写和格式化 )。

```

CPU,Core,Socket,Node,,L1d,L1i,L2,L3
0  0  0  0  0  0  0  0
1  1  0  0  1  1  1  0
2  2  0  0  2  2  2  0
3  3  0  0  3  3  3  0
...
16 0  0  0  0  0  0  0
17 1  0  0  1  1  1  0
18 2  0  0  2  2  2  0
19 3  0  0  3  3  3  0

```

从输出中我们可以看出，酷睿 0 包括 vCPUs 0 和 16，酷睿 1 包括 vCPUs 1 和 17，酷睿 2 包括 vCPUs 2 和 18。换句话说，超线程对是：0 和 16、1 和 17、2 和 18。

## CPU 赋值示例

例如，我们将在 350 处使用双极性宽带下行链路的 `c5.24xlarge` 实例。MHz 从下表中 [CPU 核心规划](#) 我们知道，350 的 MHz 下行链路需要 8 个内核 (16 vCPUs) 才能实现单个数据流。这意味着使用两个数据流的这种双极性设置总共需要 16 个内核 (32 vCPUs) 外加一个内核 (2 vCPUs) 作为代理。

我们知道 `c5.24xlarge` 包含 NUMA node0 CPU(s): 0-23, 48-71 和的 `lscpu` 输出 NUMA node1 CPU(s): 24-47, 72-95。由于 NUMA node0 的数量超出了我们的需求，因此我们只会从内核中分配：0-23 和 48-71。

首先，我们将为共享三级缓存或节点的每个数据流选择 8 个内核。NUMA 然后，我们将在的 `lscpu -p` 输出中查找相应的 vCPUs ( 标记为 “CPU” ) [附录：c5.24xlarge 的 lscpu -p 输出 \(完整\)](#)。核心选择过程的示例可能如下所示：

- 为操作系统保留内核 0-1。
- 流程 1：选择映射到 2-9 和 50-57 的内核 vCPUs 2-9。
- 流程 2：选择映射到 10-17 和 58-65 的内核 vCPUs 10-17。

- 代理核心：选择对应到 18 和 66 的核心 vCPUs 18。

结果是 vCPUs 2-18 和 50-66，因此要提供给代理的列表是。[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66]您应确保自己的进程未按 CPUs 中所述在这些服务器上运行[与 AWS Ground Station 代理一起运行服务和进程](#)。

请注意，在此示例中选择的特定内核有些是任意的。只要满足所有内核共享每个数据流的三级缓存的要求，它们就可以正常工作。

## 附录：c5.24xlarge 的 `lscpu -p` 输出 (完整)

```
> lscpu -p
# The following is the parsable format, which can be fed to other
# programs. Each different item in every column has an unique ID
# starting from zero.
# CPU,Core,Socket,Node,,L1d,L1i,L2,L3
0,0,0,0,,0,0,0,0
1,1,0,0,,1,1,1,0
2,2,0,0,,2,2,2,0
3,3,0,0,,3,3,3,0
4,4,0,0,,4,4,4,0
5,5,0,0,,5,5,5,0
6,6,0,0,,6,6,6,0
7,7,0,0,,7,7,7,0
8,8,0,0,,8,8,8,0
9,9,0,0,,9,9,9,0
10,10,0,0,,10,10,10,0
11,11,0,0,,11,11,11,0
12,12,0,0,,12,12,12,0
13,13,0,0,,13,13,13,0
14,14,0,0,,14,14,14,0
15,15,0,0,,15,15,15,0
16,16,0,0,,16,16,16,0
17,17,0,0,,17,17,17,0
18,18,0,0,,18,18,18,0
19,19,0,0,,19,19,19,0
20,20,0,0,,20,20,20,0
21,21,0,0,,21,21,21,0
22,22,0,0,,22,22,22,0
```

```
23,23,0,0,,23,23,23,0
24,24,1,1,,24,24,24,1
25,25,1,1,,25,25,25,1
26,26,1,1,,26,26,26,1
27,27,1,1,,27,27,27,1
28,28,1,1,,28,28,28,1
29,29,1,1,,29,29,29,1
30,30,1,1,,30,30,30,1
31,31,1,1,,31,31,31,1
32,32,1,1,,32,32,32,1
33,33,1,1,,33,33,33,1
34,34,1,1,,34,34,34,1
35,35,1,1,,35,35,35,1
36,36,1,1,,36,36,36,1
37,37,1,1,,37,37,37,1
38,38,1,1,,38,38,38,1
39,39,1,1,,39,39,39,1
40,40,1,1,,40,40,40,1
41,41,1,1,,41,41,41,1
42,42,1,1,,42,42,42,1
43,43,1,1,,43,43,43,1
44,44,1,1,,44,44,44,1
45,45,1,1,,45,45,45,1
46,46,1,1,,46,46,46,1
47,47,1,1,,47,47,47,1
48,0,0,0,,0,0,0,0
49,1,0,0,,1,1,1,0
50,2,0,0,,2,2,2,0
51,3,0,0,,3,3,3,0
52,4,0,0,,4,4,4,0
53,5,0,0,,5,5,5,0
54,6,0,0,,6,6,6,0
55,7,0,0,,7,7,7,0
56,8,0,0,,8,8,8,0
57,9,0,0,,9,9,9,0
58,10,0,0,,10,10,10,0
59,11,0,0,,11,11,11,0
60,12,0,0,,12,12,12,0
61,13,0,0,,13,13,13,0
62,14,0,0,,14,14,14,0
63,15,0,0,,15,15,15,0
64,16,0,0,,16,16,16,0
65,17,0,0,,17,17,17,0
66,18,0,0,,18,18,18,0
```

```
67,19,0,0,,19,19,19,0
68,20,0,0,,20,20,20,0
69,21,0,0,,21,21,21,0
70,22,0,0,,22,22,22,0
71,23,0,0,,23,23,23,0
72,24,1,1,,24,24,24,1
73,25,1,1,,25,25,25,1
74,26,1,1,,26,26,26,1
75,27,1,1,,27,27,27,1
76,28,1,1,,28,28,28,1
77,29,1,1,,29,29,29,1
78,30,1,1,,30,30,30,1
79,31,1,1,,31,31,31,1
80,32,1,1,,32,32,32,1
81,33,1,1,,33,33,33,1
82,34,1,1,,34,34,34,1
83,35,1,1,,35,35,35,1
84,36,1,1,,36,36,36,1
85,37,1,1,,37,37,37,1
86,38,1,1,,38,38,38,1
87,39,1,1,,39,39,39,1
88,40,1,1,,40,40,40,1
89,41,1,1,,41,41,41,1
90,42,1,1,,42,42,42,1
91,43,1,1,,43,43,43,1
92,44,1,1,,44,44,44,1
93,45,1,1,,45,45,45,1
94,46,1,1,,46,46,46,1
95,47,1,1,,47,47,47,1
```

# 安装座席

可以通过以下方式安装 AWS Ground Station 代理：

1. AWS CloudFormation 模板 ( 推荐 )。
2. 在亚马逊上手动安装EC2。

## 使用 AWS CloudFormation 模板

EC2数据传输 AWS CloudFormation 模板创建向您的EC2实例传送数据所需的AWS资源。此 AWS CloudFormation 模板使用预安装了 AWS Ground Station 代理AMI的 AWS Ground Station 托管模板。然后，创建的EC2实例的启动脚本会填充代理配置文件并应用必要的性能调整 ( [EC2实例性能调整](#) )。

### 步骤 1：创建 AWS 资源

使用 [G AWSround Station Agent \( 宽带 \) 使用模板公共广播卫星](#) 创建AWS资源堆栈。

### 步骤 2：检查代理状态

默认情况下，代理已配置并处于活动状态 ( 已启动 )。要检查代理状态，您可以连接到EC2实例 ( SSH 或SSM会话管理器 ) 并查看 [AWS Ground Station 代理状态](#)。

## 手动安装 EC2

虽然 Ground Station 建议使用 CloudFormation 模板来配置AWS资源，但在某些用例中，标准模板可能还不够。对于此类情况，我们建议您自定义模板以满足您的需求。如果仍然不能满足您的要求，则可以手动创建AWS资源并安装代理。

### 步骤 1：创建AWS资源

有关手动设置联系人所需AWS资源的说明，请参阅 [任务配置文件配置示例](#)。

该AwsGroundStationAgentEndpoint资源定义了通过 AWS Ground Station 代理接收 digiF 数据流的端点，对于成功进行联系至关重要。虽然API文档位于 [API参考](#) 资料中，但本节将简要讨论与 AWS Ground Station 代理相关的概念。

端点ingressAddress是 AWS Ground Station 代理接收来自天线的 AWS KMS 加密UDP流量的地方。socketAddressname是EC2实例的公有 IP ( 来自附件EIP )。portRange 应至少是 300 个连续

的端口，并且该端口范围已从其他任何用途中预留。有关说明，请参阅[保留入口端口-影响网络](#)：必须将这些端口配置为允许接收器实例运行VPC所在的安全组上的UDP入口流量。

终端egressAddress是代理将digiF数据流移交给您的地方。你应该有一个应用程序（例如SDR）通过这个位置的UDP套接字接收数据。

## 步骤 2：创建EC2实例

支持以下AMIs：

1. AWS Ground Station AMI-groundstation-al2-gs-agent-ami-\* 其中 \* AMI 是构建日期-自带已安装代理（推荐）。
2. amzn2-ami-kernel-5.10-hvm-x86\_64-gp2.

## 步骤 3：下载并安装座席

### Note

如果您在上一步中未选择 AWS Ground Station 代理，则必须完成本节AMI中的步骤。

## 下载座席

AWS Ground Station 代理可从特定区域的 S3 存储桶中获得，也可以使用AWS命令行 (CLI) 下载到支持EC2实例，`s3://groundstation-wb-digif-software-${AWS::Region}/aws-groundstation-agent/latest/amazon_linux_2_x86_64/aws-groundstation-agent.rpm`其中 `${AWS::Region}` 指的是支持的 [AWS Ground Station 控制台和数据传输](#) 区域之一。

示例：将最新 rpm 版本从AWS区域 us-east-2 本地下载到 /tmp 文件夹。

```
aws s3 --region us-east-2 cp s3://groundstation-wb-digif-software-us-east-2/aws-groundstation-agent/latest/amazon_linux_2_x86_64/aws-groundstation-agent.rpm /tmp
```

如果您需要下载 AWS Ground Station 代理的特定版本，可以从 S3 存储桶中的特定版本文件夹中下载该代理。

示例：将 rpm 的 1.0.2716.0 版本从本地的 AWS us-east-2 区域下载到 /tmp 文件夹。

```
aws s3 --region us-east-2 cp s3://groundstation-wb-digif-software-us-east-2/aws-groundstation-agent/1.0.2716.0/amazon_linux_2_x86_64/aws-groundstation-agent.rpm /tmp
```

### Note

如果您想确认RPM您下载的内容已被出售 AWS Ground Station，请按照中的说明进行操作。[RPM安装验证](#)

## 安装座席

```
sudo yum install ${MY_RPM_FILE_PATH}
```

Example: Assumes agent is in the "/tmp" directory  
sudo yum install /tmp/aws-groundstation-agent.rpm

## 步骤 4：配置座席

安装代理后，必须更新代理配置文件。请参阅 [配置代理](#)。

## 步骤 5：应用性能优化

AWS Ground Station 代理 AMI：如果您AMI在上一步中选择了 AWS Ground Station 代理，请应用以下性能调整。

- [调整硬件中断和接收队列——影响CPU和网络](#)
- [保留入口端口-影响网络](#)
- [Reboot](#)

其他 AMIs：如果您AMI在上一步中选择了任何其他选项，请应用下面列出的所有调整[EC2实例性能调整](#)并重启实例。

## 步骤 6：管理座席

要启动、停止和检查座席状态，请参阅 [管理座席](#)。

## 管理座席

AWS Ground Station 代理提供以下功能，用于使用内置的 Linux 命令工具配置、启动、停止、升级、降级和卸载代理。

### 主题

- [AWS Ground Station 代理配置](#)
- [AWS Ground Station 代理启动](#)
- [AWS Ground Station 代理停止](#)
- [AWS Ground Station 代理升级](#)
- [AWS Ground Station 代理降级](#)
- [AWS Ground Station 代理卸载](#)
- [AWS Ground Station 代理状态](#)
- [AWS Ground Station 代理RPM信息](#)

## AWS Ground Station 代理配置

导航到 `/opt/aws/groundstation/etc`，其中应包含一个名为 `aws-gs-agent-config.json` 的文件。请参阅 [代理配置文件](#)。

## AWS Ground Station 代理启动

```
#start
sudo systemctl start aws-groundstation-agent

#check status
systemctl status aws-groundstation-agent
```

应生成显示座席处于活动状态的输出。

```
aws-groundstation-agent.service - aws-groundstation-agent
```

```
Loaded: loaded (/usr/lib/systemd/system/aws-groundstation-agent.service; enabled;
vendor preset: disabled)
Active: active (running) since Tue 2023-03-14 00:39:08 UTC; 1 day 13h ago
Docs: https://aws.amazon.com/ground-station/
Main PID: 8811 (aws-gs-agent)
CGroup: /system.slice/aws-groundstation-agent.service
##8811 /opt/aws/groundstation/bin/aws-gs-agent production
```

## AWS Ground Station 代理停止

```
#stop
sudo systemctl stop aws-groundstation-agent

#check status
systemctl status aws-groundstation-agent
```

应生成显示座席处于非活动状态 ( 已停止 ) 的输出。

```
aws-groundstation-agent.service - aws-groundstation-agent
Loaded: loaded (/usr/lib/systemd/system/aws-groundstation-agent.service; enabled;
vendor preset: disabled)
Active: inactive (dead) since Thu 2023-03-09 15:35:08 UTC; 6min ago
Docs: https://aws.amazon.com/ground-station/
Process: 84182 ExecStart=/opt/aws/groundstation/bin/launch-aws-gs-agent (code=exited,
status=0/SUCCESS)
Main PID: 84182 (code=exited, status=0/SUCCESS)
```

## AWS Ground Station 代理升级

1. 下载最新版本座席。请参阅 [下载座席](#)。
2. 停止座席。

```
#stop
sudo systemctl stop aws-groundstation-agent
```

```
#confirm inactive (stopped) state
systemctl status aws-groundstation-agent
```

### 3. 更新座席。

```
sudo yum update ${MY_RPM_FILE_PATH}

# check the new version has been installed correctly by comparing the agent version
with the starting agent version
yum info aws-groundstation-agent

# reload the systemd configuration
sudo systemctl daemon-reload

# restart the agent
sudo systemctl restart aws-groundstation-agent

# check agent status
systemctl status aws-groundstation-agent
```

## AWS Ground Station 代理降级

1. 下载您需要的座席版本。请参阅 [下载座席](#)。
2. 降级座席

```
# get the starting agent version
yum info aws-groundstation-agent

# stop the agent service
sudo systemctl stop aws-groundstation-agent

# downgrade the rpm
sudo yum downgrade ${MY_RPM_FILE_PATH}

# check the new version has been installed correctly by comparing the agent version
with the starting agent version
```

```
yum info aws-groundstation-agent

# reload the systemd configuration
sudo systemctl daemon-reload

# restart the agent
sudo systemctl restart aws-groundstation-agent

# check agent status
systemctl status aws-groundstation-agent
```

## AWS Ground Station 代理卸载

卸载代理会将 `/opt/aws/groundstation/etc/.json` 重命名为 `/opt/aws/groundstation/etc/.j aws-gs-agent-config son.rpm.save`。aws-gs-agent-config再次在同一个实例上安装代理会为 `aws-gs-agent-config.json` 写入默认值，并且需要使用与您的AWS资源对应的正确值进行更新。请参阅 [代理配置文件](#)。

```
sudo yum remove aws-groundstation-agent
```

## AWS Ground Station 代理状态

座席状态为活动（座席正在运行）或非活动（座席已停止）。

```
systemctl status aws-groundstation-agent
```

示例输出显示座席处于已安装状态、非活动状态（已停止）和已启用（开机时启动服务）。

```
aws-groundstation-agent.service - aws-groundstation-agent
Loaded: loaded (/usr/lib/systemd/system/aws-groundstation-agent.service; enabled;
       vendor preset: disabled)
Active: inactive (dead) since Thu 2023-03-09 15:35:08 UTC; 6min ago
Docs: https://aws.amazon.com/ground-station/
```

```
Process: 84182 ExecStart=/opt/aws/groundstation/bin/launch-aws-gs-agent (code=exited,
status=0/SUCCESS)
Main PID: 84182 (code=exited, status=0/SUCCESS)
```

## AWS Ground Station 代理RPM信息

```
yum info aws-groundstation-agent
```

您可以在一个 (扩展) 代码行中执行所有这些操作：

### Note

“版本”可能会有所不同，具体取决于座席发布的最新版本。

```
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
```

```
Installed Packages
```

```
Name      : aws-groundstation-agent
```

```
Arch      : x86_64
```

```
Version   : 1.0.2677.0
```

```
Release   : 1
```

```
Size      : 51 M
```

```
Repo      : installed
```

```
Summary   : Client software for AWS Ground Station
```

```
URL       : https://aws.amazon.com/ground-station/
```

```
License   : Proprietary
```

```
Description : This package provides client applications for use with AWS Ground Station
```

# 配置代理

安装座席后，必须更新 `/opt/aws/groundstation/etc/aws-gs-agent-config.json` 中的座席配置文件。

## 代理配置文件

### 示例

```
{
  "capabilities": [
    "arn:aws:groundstation:eu-central-1:123456789012:dataflow-endpoint-group/
bb6c19ea-1517-47d3-99fa-3760f078f100"
  ],
  "device": {
    "privateIps": [
      "127.0.0.1"
    ],
    "publicIps": [
      "1.2.3.4"
    ],
    "agentCpuCores":
    [ 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81
  ]
}
```

## 现场细分

### 能力

功能指定为数据流端点组 Amazon 资源名称。

必需：True

格式：字符串数组

- 值：能力 ARNs → 字符串

示例：

```
"capabilities": [  
  "arn:aws:groundstation:${AWS::Region}:${AWS::AccountId}:dataflow-endpoint-group/  
  ${DataflowEndpointGroupId}"  
]
```

## device

此字段包含枚举当前 EC2 “设备” 所需的其他字段。

必需：True

格式：对象

成员：

- privateIps
- publicIps
- agentCpuCores
- networkAdapters

## privateIps

该字段目前未使用，但包含在将来的用例中。如果未包含任何值，则默认为 ["127.0.0.1"]

必需：False

格式：字符串数组

- 值：IP 地址 → 字符串

例如：

```
"privateIps": [  
  "127.0.0.1"  
],
```

## publicIps

每个数据流终端节点组的弹性 IP (EIP)。

必需：True

格式：字符串数组

- 值：IP 地址 → 字符串

例如：

```
"publicIps": [  
  "9.8.7.6"  
],
```

## agentCPUCores

这指定了为该 aws-gs-agent 进程保留哪些虚拟内核。有关适当设置此值的要求，请参阅 [CPU核心规划](#)。

必需：True

格式：整数数组

- 值：核心数字 → 整数

例如：

```
"agentCpuCores": [  
  
  24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 8  
  ]
```

## networkAdapters

这与将接收数据的以太网适配器或连接的接口相对应。ENIs

必填项：False

格式：字符串数组

- 值：以太网适配器名称 ( 可以通过运行 `ifconfig` 查找 )

例如：

```
"networkAdapters": [  
  "eth0"  
]
```

# EC2实例性能调整

## Note

如果您使用 CloudFormation 模板配置AWS资源，则会自动应用这些调整。如果您使用AMI或手动创建了您的EC2实例，则必须应用这些性能调整才能实现最可靠的性能。

请记住，应用任何优化后重启您的实例。

## 主题

- [调整硬件中断和接收队列——影响CPU和网络](#)
- [Tune Rx 中断合并-影响网络](#)
- [调整 Rx 环形缓冲区-影响网络](#)
- [调整 CPU C-State-影响 CPU](#)
- [保留入口端口-影响网络](#)
- [Reboot](#)

## 调整硬件中断和接收队列——影响CPU和网络

本节配置 systemd、SMPIRQs、接收数据包导向 (RPS) 和接收流指导 () 的CPU核心用法。RFS有关基于您正在使用的实例类型的一组推荐设置，请参阅 [附录：中断/调RPS谐的推荐参数](#)。

1. 将 systemd 进程固定在远离代理CPU内核的地方。
2. 将硬件中断请求路由到远离代理CPU内核。
3. 配置RPS以防止单个网络接口卡的硬件队列成为网络流量的瓶颈。
4. 配置RFS为提高CPU缓存命中率，从而减少网络延迟。

提供的set\_irq\_affinity.sh脚本为您RPM配置了以上所有内容。添加到 crontab，这样它就会应用于每次启动：

```
echo "@reboot sudo /opt/aws/groundstation/bin/set_irq_affinity.sh  
'${interrupt_core_list}' '${rps_core_mask}' >> /var/log/user-data.log 2>&1" >>/var/  
spool/cron/root
```

- `interrupt_core_list` 替换为为内核和操作系统保留的内核，通常是第一和第二个，以及超线程内核。这不应与上面选择的内核重叠。（例如：“0,1,48,49”表示超线程，96个实例）。CPU
- `rps_core_mask` 是一个十六进制位掩码，用于指定CPU应处理传入的数据包，每个数字表示4。CPU此外，还必须从右边开始每8个字符用逗号分隔。建议全部允许CPU，让缓存来处理平衡。
  - 要查看每种实例类型的推荐参数列表，请参阅 [附录：中断/调RPS谐的推荐参数](#)。
- 96实例的示CPU例：

```
echo "@reboot sudo /opt/aws/groundstation/bin/set_irq_affinity.sh '0,1,48,49'  
'ffffffff,ffffffff,ffffffff' >> /var/log/user-data.log 2>&1" >>/var/spool/cron/root
```

## Tune Rx 中断合并-影响网络

中断合并有助于防止主机系统中出现过多的中断，并帮助提高网络吞吐量。使用此配置，可以收集数据包，并且每128微秒生成一个中断。添加到 crontab，这样它就会应用于每次启动：

```
echo "@reboot sudo ethtool -C ${interface} rx-usecs 128 tx-usecs 128 >>/var/log/user-  
data.log 2>&1" >>/var/spool/cron/root
```

- 替换 `interface` 为配置为接收数据的网络接口（以太网适配器）。通常，这是 `eth0` 因为这是为 EC2 实例分配的默认网络接口。

## 调整 Rx 环形缓冲区-影响网络

增加 Rx 环形缓冲区的环形条目数，以防止在突发连接期间丢包或溢出。添加到 crontab，这样每次启动时都会正确设置它：

```
echo "@reboot sudo ethtool -G ${interface} rx 16384 >>/var/log/user-data.log 2>&1" >>/  
var/spool/cron/root
```

- 替换 `interface` 为配置为接收数据的网络接口（以太网适配器）。通常，这是 `eth0` 因为这是为 EC2 实例分配的默认网络接口。
- 如果设置 `c6i.32xlarge` 实例，则需要修改命令以将环形缓冲区设置为 8192，而不是 16384。

## 调整 CPU C-State-影响 CPU

设置 CPU C 状态以防止空转，这可能会导致在联系开始时丢失数据包。要求实例重启。

```
echo "GRUB_CMDLINE_LINUX_DEFAULT=\"console=tty0 console=ttyS0,115200n8
net.ifnames=0 biosdevname=0 nvme_core.io_timeout=4294967295 intel_idle.max_cstate=1
processor.max_cstate=1 max_cstate=1\" >/etc/default/grub
echo "GRUB_TIMEOUT=0" >>/etc/default/grub
grub2-mkconfig -o /boot/grub2/grub.cfg
```

## 保留入口端口-影响网络

预留您的 `AwsGroundStationAgentEndpoint` 入口地址端口范围内的所有端口，以防止与内核使用发生冲突。端口使用冲突将导致联络和数据传输失败。

```
echo "net.ipv4.ip_local_reserved_ports=${port_range_min}-${port_range_max}" >> /etc/
sysctl.conf
```

- 示例：`echo "net.ipv4.ip_local_reserved_ports=42000-43500" >> /etc/
sysctl.conf。`

## Reboot

成功应用所有优化后，重启实例才能使优化生效。

```
sudo reboot
```

## 附录：中断/调RPS谐的推荐参数

本节确定了在“调整硬件中断”和“接收队列-影响CPU和网络”部分中使用的推荐参数值。

系列	实例类型	$\{\text{interru}$ $\text{pt\_core\_list}\}$	$\{\text{rps\_cor}$ $\text{e\_mask}\}$
c6i	<ul style="list-style-type: none"> <li>c6i.32xlarge</li> </ul>	<ul style="list-style-type: none"> <li>0,1,64,65</li> </ul>	<ul style="list-style-type: none"> <li>ffffff,</li> <li>ffffff,</li> <li>ffffff,</li> <li>ffffff</li> </ul>
c5	<ul style="list-style-type: none"> <li>c5.24xlarge</li> <li>c5.18xlarge</li> <li>c5.12xlarge</li> </ul>	<ul style="list-style-type: none"> <li>0,1,48,49</li> <li>0,1,36,37</li> <li>0,1,24,25</li> </ul>	<ul style="list-style-type: none"> <li>ffffff,</li> <li>ffffff,</li> <li>ffffff</li> <li>ff,ffff</li> <li>ff,ffffff</li> <li>ffff,ffffff</li> </ul>
c5n	<ul style="list-style-type: none"> <li>c5n.metal</li> <li>c5n.18xlarge</li> </ul>	<ul style="list-style-type: none"> <li>0,1,36,37</li> <li>0,1,36,37</li> </ul>	<ul style="list-style-type: none"> <li>ff,ffff</li> <li>ff,ffffff</li> <li>ff,ffff</li> <li>ff,ffffff</li> </ul>
m5	<ul style="list-style-type: none"> <li>m5.24xlarge</li> <li>m5.12xlarge</li> </ul>	<ul style="list-style-type: none"> <li>0,1,48,49</li> <li>0,1,24,25</li> </ul>	<ul style="list-style-type: none"> <li>ffffff,</li> <li>ffffff,</li> <li>ffffff</li> <li>ffff,ffffff</li> </ul>
r5	<ul style="list-style-type: none"> <li>r5.metal</li> <li>r5.24xlarge</li> </ul>	<ul style="list-style-type: none"> <li>0,1,48,49</li> <li>0,1,48,49</li> </ul>	<ul style="list-style-type: none"> <li>ffffff,</li> <li>ffffff,</li> <li>ffffff</li> </ul>

系列	实例类型	$\{\text{interrupt\_core\_list}\}$	$\{\text{rps\_core\_mask}\}$
			<ul style="list-style-type: none"> <li>• ffffffff, ffffffff, ffffffff</li> </ul>
r5n	<ul style="list-style-type: none"> <li>• r5n.metal</li> <li>• r5n.24xlarge</li> </ul>	<ul style="list-style-type: none"> <li>• 0,1,48,49</li> <li>• 0,1,48,49</li> </ul>	<ul style="list-style-type: none"> <li>• ffffffff, ffffffff, ffffffff</li> <li>• ffffffff, ffffffff, ffffffff</li> </ul>
g4dn	<ul style="list-style-type: none"> <li>• g4dn.metal</li> <li>• g4dn.16xlarge</li> <li>• g4dn.12xlarge</li> </ul>	<ul style="list-style-type: none"> <li>• 0,1,48,49</li> <li>• 0,1,32,33</li> <li>• 0,1,24,25</li> </ul>	<ul style="list-style-type: none"> <li>• ffffffff, ffffffff, ffffffff</li> <li>• ffffffff, ffffffff</li> <li>• ffff, ffffffff</li> </ul>
p4d	<ul style="list-style-type: none"> <li>• p4d.24xlarge</li> </ul>	<ul style="list-style-type: none"> <li>• 0,1,48,49</li> </ul>	<ul style="list-style-type: none"> <li>• ffffffff, ffffffff, ffffffff</li> </ul>
p3dn	<ul style="list-style-type: none"> <li>• p3dn.24xlarge</li> </ul>	<ul style="list-style-type: none"> <li>• 0,1,48,49</li> </ul>	<ul style="list-style-type: none"> <li>• ffffffff, ffffffff, ffffffff</li> </ul>

## 准备开始联络 DigIF

1. 查看CPU核心规划以了解所需的数据流，并提供代理可以使用的核心列表。请参阅 [CPU核心规划](#)。
2. 查看 AWS Ground Station 代理配置文件。请参阅 [AWS Ground Station 代理配置](#)。
3. 确认已应用必要的性能优化。请参阅 [EC2实例性能调整](#)。
4. 确认您遵循了所有已提及的最佳实践。请参阅 [最佳实践](#)。
5. 通过以下方式确认 AWS Ground Station 代理已在预定的联系开始时间之前启动：

```
systemctl status aws-groundstation-agent
```

6. 通过以下方式确认 AWS Ground Station 代理在预定的联系开始时间之前运行正常：

```
aws groundstation get-dataflow-endpoint-group --dataflow-endpoint-group-id  
${DATAFLOW-ENDPOINT-GROUP-ID} --region ${REGION}
```

验证你agentStatus的 awsGroundStationAgentEndpoint is ACTIVE 和 i auditResults 是HEALTHY。

# 最佳实践

## 亚马逊EC2最佳实践

遵循当前EC2的最佳实践，确保足够的数据存储可用性。

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-best-practices.html>

## Linux 调度器

如果相应的进程未固定到特定的内核，Linux 调度器可以重新排序UDP套接字上的数据包。在数据传输期间，任何发送或接收UDP数据的线程都应将自己固定在特定的内核上。

## AWS Ground Station 托管前缀列表

在指定允许来自天线的通信的网络规则时，建议使用 `com.amazonaws.global.groundstation` AWS-managed 前缀列表。有关[AWS托管前缀列表的更多信息](#)，请[参阅使用AWS托管前缀列表](#)。

## 单次联络限制

G AWS round Station Agent 支持每个联系人有多个直播，但一次只能支持一个接触。为防止出现计划问题，请勿在多个数据流端点组之间共享实例。如果单个代理配置与多个不同的代理配置相关联 DFEGARNs，则注册失败。

## 与 AWS Ground Station 代理一起运行服务和进程

在与代理相同的EC2实例上启动服务和进程时，务必将其绑定到 AWS Ground Station AWS Ground Station 代理和 Linux 内核 vCPUs 未使用的状态，因为这可能会导致瓶颈，甚至在联系期间丢失数据。这种与特定绑定的概念被 vCPUs 称为亲和力。

要避免使用的内核：

- agentCpuCores来自 [代理配置文件](#)
- 来自 [调整硬件中断和接收队列——影响CPU和网络](#) 的 `interrupt_core_list`。
  - 默认值可从中找到 [附录：中断/调RPS谐的推荐参数](#)

## 举个使用c5.24xlarge实例的例子

如果你指定了

```
"agentCpuCores": [24,25,26,27,72,73,74,75]"
```

然后跑了

```
echo "@reboot sudo /opt/aws/groundstation/bin/set_irq_affinity.sh  
'0,1,48,49' 'ffffffff,ffffffff,ffffffff' >> /var/log/user-data.log 2>&1"  
>>/var/spool/cron/root
```

然后避免使用以下内核：

```
0,1,24,25,26,27,48,49,72,73,74,75
```

## 关联服务 (systemd)

新推出的服务将自动关联到interrupt\_core\_list前面提到的服务。如果您启动的服务的用例需要额外的内核，或者需要更少拥挤的内核，请按照本节进行操作。

使用以下命令检查您的服务当前配置的关联度：

```
systemctl show --property CPUAffinity <service name>
```

如果你看到一个空值CPUAffinity=，比如，这意味着它很可能会使用上面命令中的默认内核  
...bin/set\_irq\_affinity.sh <using the cores here> ...

要覆盖并设置特定的关联性，请运行以下命令来查找服务文件的位置：

```
systemctl show -p FragmentPath <service name>
```

打开并修改文件（使用vinano、等），然后将其放在如下[Service]部分CPUAffinity=<core list>中：

```
[Unit]
...

[Service]
...
CPUAffinity=2,3

[Install]
...
```

保存文件并重新启动服务以应用关联性：

```
systemctl daemon-reload
systemctl restart <service name>

# Additionally confirm by re-running
systemctl show --property CPUAffinity <service name>
```

欲了解更多信息，请访问：[红帽企业 Linux 8-管理、监控和更新内核-第 27 章。使用 systemd 配置 CPU 关联和 NUMA 策略。](#)

## 关联进程（脚本）

强烈建议手动关联新启动的脚本和进程，因为默认的 Linux 行为允许它们使用计算机上的任何内核。

为避免任何正在运行的进程（例如 python、bash 脚本等）发生核心冲突，请使用以下命令启动该进程：

```
taskset -c <core list> <command>
# Example: taskset -c 8 ./bashScript.sh
```

如果该进程已在运行，请使用诸如 pidof 或 ps 之类的命令来查找特定进程的进程 ID (PID)。有了这个，PID 你可以看到当前的亲和力：

```
taskset -p <pid>
```

并且可以用以下方式对其进行修改：

```
taskset -p <core mask> <pid>  
# Example: taskset -p c 32392 (which sets it to cores 0xc -> 0b1100 -> cores 2,3)
```

有关任务集的更多信息，请参阅[任务集-Linux](#) 手册页

# 故障排除

## 座席启动失败

AWS Ground Station 代理可能由于多种原因而无法启动，但最常见的情况可能是代理配置文件配置错误。启动座席后（请参阅 [AWS Ground Station 代理启动](#)），您可能会获得如下状态：

```
#agent is automatically retrying a restart
aws-groundstation-agent.service - aws-groundstation-agent
Loaded: loaded (/usr/lib/systemd/system/aws-groundstation-agent.service; enabled;
        vendor preset: disabled)
Active: activating (auto-restart) (Result: exit-code) since Fri 2023-03-10 01:48:14
        UTC; 23s ago
Docs: https://aws.amazon.com/ground-station/
Process: 43038 ExecStart=/opt/aws/groundstation/bin/launch-aws-gs-agent (code=exited,
        status=101)
Main PID: 43038 (code=exited, status=101)

#agent has failed to start
aws-groundstation-agent.service - aws-groundstation-agent
Loaded: loaded (/usr/lib/systemd/system/aws-groundstation-agent.service; enabled;
        vendor preset: disabled)
Active: failed (Result: start-limit) since Fri 2023-03-10 01:50:15 UTC; 13s ago
Docs: https://aws.amazon.com/ground-station/
Process: 43095 ExecStart=/opt/aws/groundstation/bin/launch-aws-gs-agent (code=exited,
        status=101)
Main PID: 43095 (code=exited, status=101)
```

## 故障排除

```
sudo journalctl -u aws-groundstation-agent | grep -i -B 3 -A 3 'Loading Config' | tail
-6
```

可能会导致以下输出：

```
launch-aws-gs-agent[43095]: Running with options Production(ProductionOptions
  { endpoint: None, region: None })
launch-aws-gs-agent[43095]: Loading Config
launch-aws-gs-agent[43095]: System has 96 logical cores
systemd[1]: aws-groundstation-agent.service: main process exited, code=exited,
  status=101/n/a
systemd[1]: Unit aws-groundstation-agent.service entered failed state.
```

在“加载配置”后仍未能启动座席表示座席配置存在问题。要验证您的座席配置，请参阅 [代理配置文件](#)。

## AWS Ground Station 代理日志

AWS Ground Station 代理将有关联系执行、错误和运行状况的信息写入运行代理的实例上的日志文件。您可以通过手动连接到实例来查看日志文件。

您可以在以下位置中查看座席日志。

```
/var/log/aws/groundstation
```

## 没有可用的联系人

安排联系人需要一个健康的 AWS Ground Station 代理。请 AWS Ground Station API 通过以下方式查询，确认您的 AWS Ground Station 代理已启动并且运行正常 `get-dataflow-endpoint-group`：

```
aws groundstation get-dataflow-endpoint-group --dataflow-endpoint-group-id ${DATAFLOW-
ENDPOINT-GROUP-ID} --region ${REGION}
```

验证你 `agentStatus` 的 `awsGroundStationAgentEndpoint` is `ACTIVE` 和 `auditResults` 是 `HEALTHY`。

## 获取支持

通过 Support 联系 Ground Station AWS 团队。

1. 为任何受影响的联络提供 `contact_id`。如果没有这些信息，AWS Ground Station 团队就无法调查特定的联系人。
2. 提供有关已采取的所有故障排除步骤的详细信息。
3. 请提供在运行故障排除指南中的命令时发现的任何错误消息。

# 代理版本说明

## 最新代理版本

### 版本 1.0.3555.0

发布日期：2024 年 3 月 27 日

Support 终止日期：2024 年 8 月 31 日

RPM校验和：

- SHA256: 108f3aceb00e5af549839cd766c56149397e448a6e1e1429c89a9eebb6bc0fc1
- MD5: 65b72fa507fb0af32651adbb18d2e30f

更改：

- 在任务启动期间，为选定的可执行文件版本添加代理指标。
- 添加配置文件支持，以便在有其他版本可用时避免使用特定的可执行版本。
- 添加网络和路由诊断。
- 其他安全功能。
- 修复了某些指标报告错误被写入 stdout/journal 而不是日志文件的问题。
- 优雅地处理网络无法访问的套接字错误。
- 测量源代理和目标代理之间的数据包丢失和延迟。
- 发布 2.0 aws-gs-datapipe 版本以支持新的协议功能以及透明地将联系人升级到新协议的功能。

## 已弃用的代理版本

### 版本 1.0.2942.0

发布日期：2023 年 6 月 26 日

Support 终止日期：2024 年 5 月 31 日

RPM校验和：

- SHA256: 7d94b642577504308a58bab28f938507f2591d4e1b2c7ea170b77bea97b5a9b6
- MD5: 661ff2b8f11aba5d657a6586b56e0d8f

更改：

- 添加了错误日志，说明何时在磁盘上更新代理并且需要重新启动代理才能使更改生效。RPM
- 添加了网络调整验证，以确保代理用户指南的调整步骤得到正确遵循和应用。
- 修复了导致代理日志中出现有关日志存档的错误警告的错误。
- 改进了丢包检测。
- 更新了代理安装以防止安装或升级（RPM如果代理已在运行）。

## 版本 1.0.2716.0

发布日期：2023 年 3 月 15 日

Support 终止日期：2024 年 5 月 31 日

RPM校验和：

- SHA256: cb05b6a77dfcd5c66d81c0072ac550affbcefefc372cc5562ee52fb220844929
- MD5: 65266490c4013b433ec39ee50008116c

更改：

- 当代理在任务执行过程中遇到故障时，启用上传日志。
- 修复提供的网络调整脚本中的 linux 兼容性错误。

## 版本 1.0.2677.0

发布日期：2023 年 2 月 15 日

Support 终止日期：2024 年 5 月 31 日

RPM校验和：

- SHA256: 77cfe94acb00af7ca637264b17c9b21bd7afdc85b99dfdd627aec9e99397489
- MD5: b8533be7644bb4d12ab84de21341adac

更改：

- 第一个公开发行的 Agent 版本。

# RPM安装验证

下面显示了最新RPM版本、经过MD5哈希验证的RPM哈SHA256希值以及使用 sha256sum 的哈希值。这些值组合在一起可用于验证用于地面站代理的RPM版本。

## 最新代理版本

### 版本 1.0.3555.0

发布日期：2024 年 3 月 27 日

Support 终止日期：2024 年 8 月 31 日

RPM校验和：

- SHA256: 108f3aceb00e5af549839cd766c56149397e448a6e1e1429c89a9eebb6bc0fc1
- MD5: 65b72fa507fb0af32651adbb18d2e30f

更改：

- 在任务启动期间，为选定的可执行文件版本添加代理指标。
- 添加配置文件支持，以便在有其他版本可用时避免使用特定的可执行版本。
- 添加网络和路由诊断。
- 其他安全功能。
- 修复了某些指标报告错误被写入 stdout/journal 而不是日志文件的问题。
- 优雅地处理网络无法访问的套接字错误。
- 测量源代理和目标代理之间的数据包丢失和延迟。
- 发布 2.0 aws-gs-datapipe 版本以支持新的协议功能以及透明地将联系人升级到新协议的功能。

## 验证 RPM

验证此RPM安装所需的工具有：

- [sha256sum](#)
- [rpm](#)

默认情况下，这两个工具都在 Amazon Linux 2 上提供。这些工具将有助于验证RPM您使用的版本是否正确。首先RPM从 S3 存储桶下载最新版本（有关下载[下载座席](#)的说明，请参阅RPM）。下载此文件后，需要检查以下几点：

- 计算文件的 sha256sum。RPM在您正在使用的计算实例的命令行中执行以下操作：

```
sha256sum aws-groundstation-agent.rpm
```

取此值并将其与上表进行比较。这表明下载RPM的文件是 G AWS round Station 已出售给客户的有效文件，可供使用。如果哈希值不匹配，请不要安装RPM，并将其从计算实例中删除。

- 还要检查文件的MD5哈希值，以确保RPM没有遭到破坏。为此，请通过运行以下RPM命令来使用命令行工具：

```
rpm -Kv ./aws-groundstation-agent.rpm
```

验证此处列出的MD5哈希值是否与上表中版本的MD5哈希值相同。根据AWS文档中列出的此表对这两个哈希值进行验证后，可以确保客户下载和安装的RPM哈希值是安全且不折不扣的版本。RPM

## 《AWS Ground Station 代理用户指南》的文档历史记录

下表描述了每个版本的《AWS Ground Station 代理用户指南》中的重要更改。

变更	说明	日期
<a href="#">文档更新</a>	在 <a href="#">代理要求</a> 中添加了有关将子网和 Amazon EC2 实例保持在同一可用区域内的评论。	2024 年 7 月 18 日
<a href="#">文档更新</a>	将 AWS Ground Station 代理拆分成自己的用户指南。 有关之前的更改，请参阅： <a href="#">G AWSround Station 用户指南的文档历史记录</a> 。	2024 年 7 月 18 日

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