NGC on AWS Virtual Machines

Release Notes and Setup Guide

DU-08786-001 _v20 | January 2024

Table of Contents

Chapter 1. Using NGC on AWS Virtual Machines

NVIDIA makes available on the Amazon Web Services (AWS) platform three different VMIs, known within the AWS ecosystem as an Amazon Machine Image (AMI). These are GPU-optimized AMIs for AWS instances with NVIDIA V100 (EC2 P3 instances), NVIDIA T4 GPUs (EC2 G4 instances), NVIDIA A100 GPUs (EC2 P4d). Additionally, the NVIDIA AMI also supports ARM64 (EC2 G5g) instances.

For those familiar with the AWS platform, the process of launching the instance is as simple as logging in, selecting the NVIDIA GPU-optimized image of choice, configuring settings as needed, then launching the VM. After launching the VM, you can SSH into it and start building a host of AI applications in deep learning, machine learning and data science by leveraging the wide range of GPU-accelerated containers, pre-trained models and resources available from the NGC Catalog.

This document provides step-by-step instructions for accomplishing this, including how to use the AWS CLI.

Prerequisites

These instructions assume the following:

- You have an AWS account - https://aws.amazon.com
- Browse the NGC website and identified an available NGC container and tag to run on the virtual machine instance (VMI).
- Windows Users: The CLI code snippets are for bash on Linux or Mac OS X. If you are using Windows and want to use the snippets as-is, you can use the Windows Subsystem for Linux and use the bash shell (you will be in Ubuntu Linux).

1.1. Security Best Practices

Cloud security starts with the security policies of your CSP account. Refer to the following link for how to configure your security policies for your CSP:

- Security in Amazon Virtual Private Cloud

Users must follow the security guidelines and best practices of their CSP to secure their VM and account.

1.2. Before You Get Started

Perform these preliminary setup tasks to simplify the process of launching the NVIDIA Deep Learning AMI.

1.2.1. Setting Up Your AWS Key Pair

If you do not already have Key Pairs defined, then you will need to setup your AWS Key Pair and have it on the machine on which you will use the AWS CLI, or from which you will SSH to the instance. In the examples, the key pair is named "my-key-pair".

Once you have your key pair downloaded, make sure they are only readable by you, and (Linux or OSX) move them to your `~/.ssh/` directory.

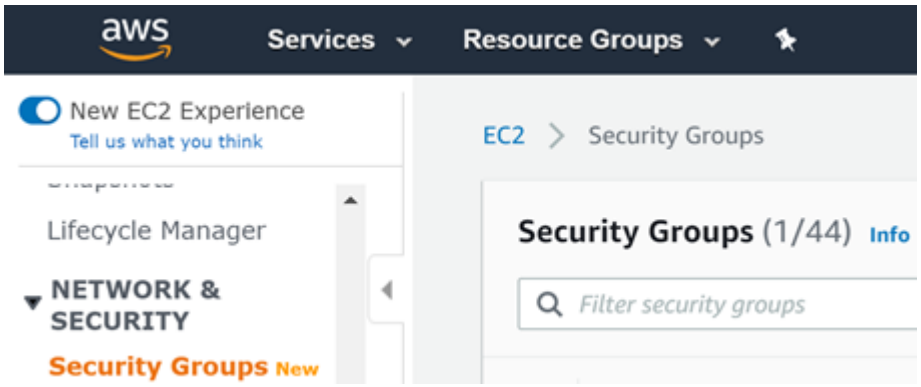
```
chmod 400 my-key-pair*
```

```
mv my-key-pair* ~/.ssh/
```

1.2.2. Set Up Security Groups for the EC2 Instance

Security groups define the network connection restrictions you place on your virtual machine instance. In order to connect to your running instances you will need a Security Group allowing (at minimum) SSH access.

1. Log into the AWS Console (https://aws.amazon.com), then click EC2 under the Console section located within the All Services drop-down menu..
2. Enter the Security Groups screen, located on the left under "Network & Security", "Security Groups".

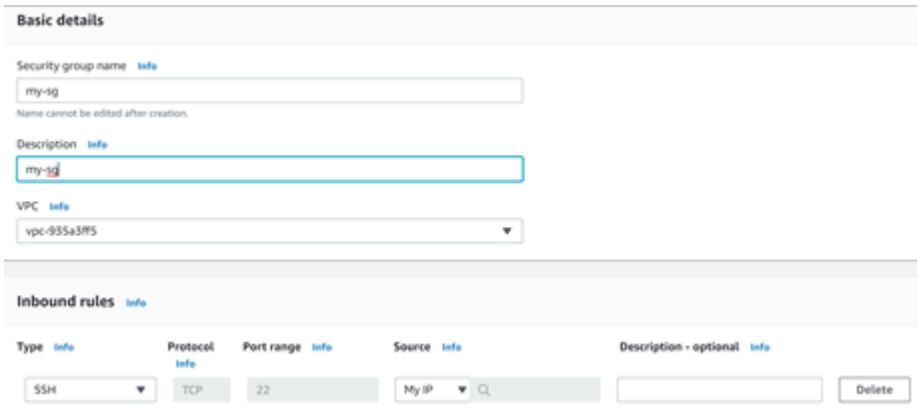


3. Click Create Security Group.

4. Give the Security Group a name (for example, "my-sg"), description, and then click Add Rule
5. Under the "Inbound" section, click Add a rule with the following parameters to enable SSH:
   - Type: SSH
   - Protocol: TCP
   - Port Range: 22
   - Source: My IP

   You may need to widen the resulting IP filter if you're not on a fixed IP address, or want to access the instance from multiple locations such as work and home.

   The following shows the filled-out Create Security Group form using the example naming.



6. (Optional) Add additional rules.

   You may need to add additional rules for HTTP, HTTPS, or other Custom TCP ports depending on the deep learning frameworks you use.

   Continue adding additional rules by clicking Add Rule, then create rules as needed.

   Examples:

   - For DIGITS4
     - Type: Custom TCP Rule
     - Protocol: TCP
     - Port Range: 3448
     - Source: My IP
   - For HTTPS secure web frameworks
     - Type: HTTPS
     - Protocol: TCP
     - Port Range: 443
     - Source: My IP

Security Warning

It is important to use proper precautions and security safeguards prior to granting access, or sharing your AMI over the internet. By default, internet connectivity to the AMI instance is blocked. You are solely responsible for enabling and securing access to your AMI. Please refer to AWS guides for managing security groups.

7. Click Create Security Group to complete creation of the Security Group on the bottom right corner.

   Once created, the Group ID is listed in the Security Group table.

1.3. Creating an NGC Certified Virtual Machine using AWS Console

1.3.1. Log In and Select the AWS Region

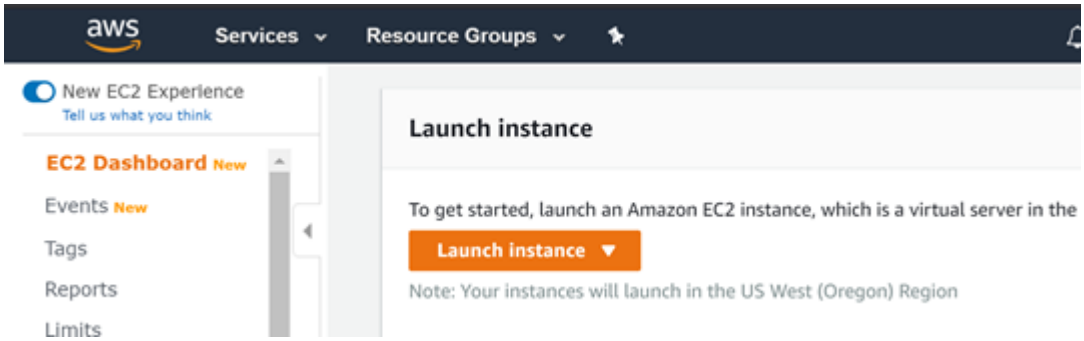
1. Log into the AWS Console (https://aws.amazon.com), then under the Compute section, click EC2 .
2. Select the AWS Zone from the upper right of the top menu.

   In order to use NVIDIA Volta and Turing GPUs in AWS, you must select a region that has Amazon EC2 P3 or G4 instances available. The examples in this guide use instances in US West (Oregon) - us-west-2. Check with AWS for Amazom EC2 P3 or G4 instance availability in other regions.

1.3.2. Create a VM and Choose an NVIDIA GPU-Optimized AMI

NVIDIA publishes and maintains multiple flavors of a GPU-optimized AMI with all the software needed to pull and run content from NGC. These AMIs should be used to launch your GPU instances.

1. Click Launch Instance.

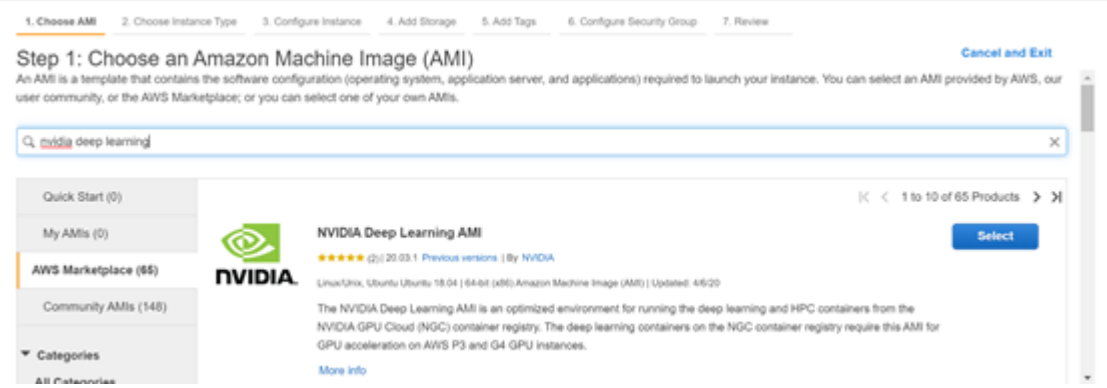


2. Select the NVIDIA Deep Learning AMI.

a). Select AWS Marketplace on the Step 1 of the process.

b). Search for and select the NVIDIA GPU-optimized AMIs that best suits your purpose by simply typing in “nvidia” into the search bar.

c). Click Continue on the details page.



1.3.3. Select an Instance Type with GPUs and Configure Instance Settings

1. Select one of the Amazon EC2 P3 or G4 instance types according to your GPU, CPU, and memory requirements.
2. Click Review and Launch to review the default configuration settings, or continue with the instructions in the next section to configure each setting step-by-step
3. After choosing an instance type, click Next: Configure Instance Details.

   There are no instance details that need to be configured, so you can proceed to the next step.
4. Add storage.

   Click Next: Add Storage.

   While the default 32 GiB for the root volume can be changed, users should not use the root volume for storing datasets since the root volume is destroyed when the instance is terminated.
5. Add tags.

   Naming your instances helps to keep multiple instances organized.

   a). Click Next: Add Tag.

   b). Click Add Tag and then fill in the following information:

   Key: "Name"

   Value: <instance name, such as "My GPU">
6. Configure a Security Group

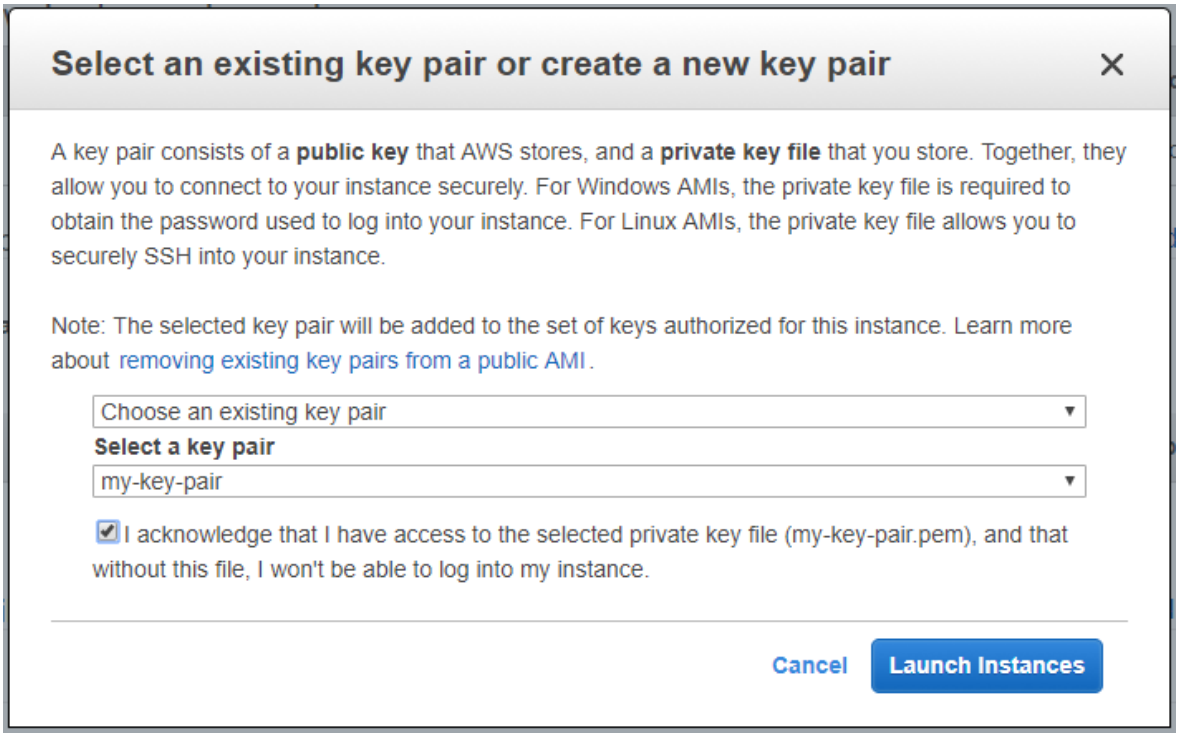
   a). Click Next: Configure Security Group.

   b). Click Select an existing security group and select the Security Group you created during Before You Get Started.

1.3.4. Launching Your VM Instance

1. Click Review and Launch.

   A window pops up and asks which key pair to use.



2. Select Choose an existing key pair, select your key pair, then check the acknowledgement checkbox.
3. Click Launch Instances.

1.3.5. Connect to Your VM Instance

1. After launching your instance, click View Instances, locate your instance from the list, then wait until it is in the 'running' state.
2. When it is in the running state, select it from the list and then click Connect.
3. Follow the instructions in the pop-up window to establish an SSH connection to the instance.

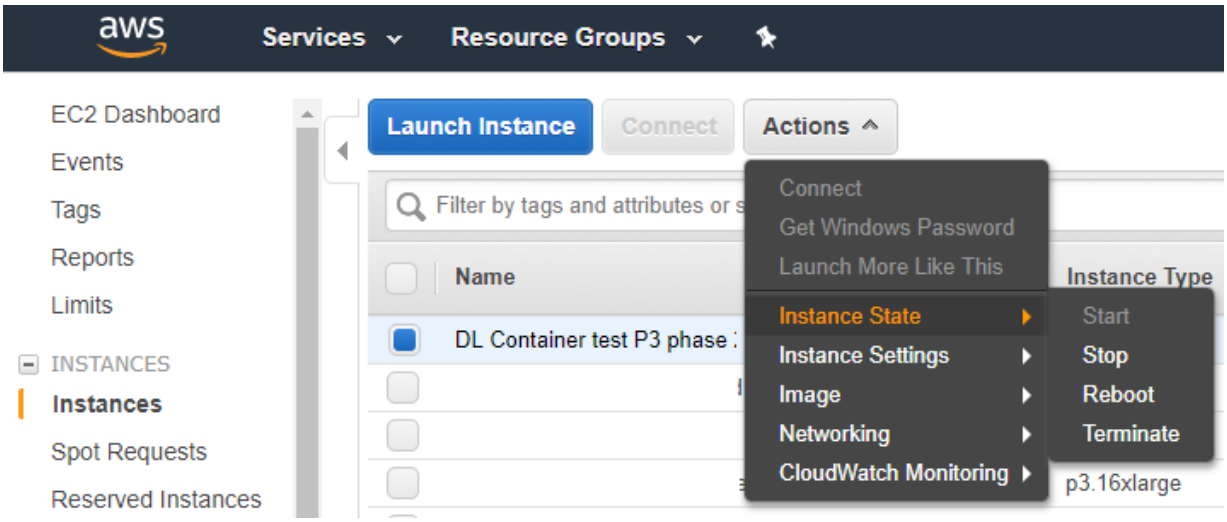
   Be sure to use 'ubuntu' for the username.

   If the instructions for SSH login do not work, see the AWS Connect to Your Linux Instance documentation for additional information.

1.3.6. Start/Stop/Terminate Your VM Instance

Once you are done with your instance you can stop (to be started again later) or terminate (delete) it. Refer to the Instance Lifecycle in the AWS documentation for more information.

Instances can be controlled from the Instances page, using the "Actions”->"Instance State" menu to stop, start, or terminate Instances.



1.4. Creating an NGC Certified Virtual Machine Through the AWS CLI

If you plan to use AWS CLI, then the CLI must be installed (Windows Users: inside the Windows Subsystem for Linux), updated to the latest version, and configured.

Some of the AWS CLI snippets in these instructions make use of `jq`, which should be installed on the machine from which you'll run the AWS CLI. You may paste these snippets into your own bash scripts or type them at the command line.

1.4.1. Set Up Environment Variables

Set up the following environment variables which can be used in the commands for launching the VM instance:

Security Group

The Security Group ID is used as part of the instance creation process. Once created the Group ID can be looked up in the AWS Console, or retrieved by name with the following snippet, and stored in the `$NVAWS_SG_ID` environment variable.

```
NVAWS_SG_NAME='my-sg'
```

```
NVAWS_SG_ID=$(aws ec2 describe-security-groups --group-name "$NVAWS_SG_NAME" |
 jq .SecurityGroups[0].GroupId | sed 's/\"//g') && echo NVAWS_SG_ID=$NVAWS_SG_ID
```

Image ID

The following snippet will list the current "NVIDIA Deep Learning AMI" Image ID, and stored in the `$NVAWS_IMAGE_ID` environment variable.

```
NVAWS_IMAGE_NAME='NVIDIA Deep Learning AMI'
```

```
NVAWS_IMAGE_ID=$(aws ec2 describe-images --filters "Name=name,Values=$NVAWS_IMAGE_NAME" |
 jq .Images[0].ImageId | sed 's/\"//g') && echo NVAWS_IMAGE_ID=$NVAWS_IMAGE_ID
```

Other Environment Variables

Set up other env variables as follows, using your information:

```
NVAWS_KEYNAME=my-key-pair
```

```
NVAWS_KEYPATH=~/.ssh/
```

```
NVAWS_REGION=us-west-2
```

```
NVAWS_INSTANCE_TYPE=p3.2xlarge
```

```
NVAWS_EBS_GB=32
```

```
NVAWS_NAME_TAG='My GPU'
```

Be sure to set a unique `NVAWS_NAME_TAG` for each instance you launch.

1.4.2. Launch Your VM Instance

Launch the instance and capture the resulting JSON:

```
NVAWS_LAUNCH_JSON=$(aws ec2 run-instances --image-id $NVAWS_IMAGE_ID \
  --instance-type $NVAWS_INSTANCE_TYPE \
  --region $NVAWS_REGION \
  --key-name $NVAWS_KEYNAME \
  --security-group-ids $NVAWS_SG_ID \
  --block-device-mapping
 "[{\"DeviceName\":\"/dev/sda1\",\"Ebs\":{\"VolumeSize\":$NVAWS_EBS_GB}}]" \
  --tag-specifications
 "ResourceType=instance,Tags=[{Key=Name,Value=$NVAWS_NAME_TAG}]")
NVAWS_INSTANCE_ID=$(echo $NVAWS_LAUNCH_JSON | jq .Instances[0].InstanceId | sed 's/
\"//g') && echo NVAWS_INSTANCE_ID=$NVAWS_INSTANCE_ID
```

The resulting Instance ID is stored in the `NVAWS_INSTANCE_ID` environment variable.

The launch process can take several minutes once a machine is available, and can be watched in the AWS Console Instances page or with the CLI using:

```
aws ec2 describe-instance-status --instance-id $NVAWS_INSTANCE_ID
 | jq '.InstanceStatuses[0].InstanceState.Name + " "
 + .InstanceStatuses[0].SystemStatus.Status'
```

Once the instance is "running initializing", you will be able to get the Public DNS name with:

```
NVAWS_DNS=$(aws ec2 describe-instances --instance-id $NVAWS_INSTANCE_ID | jq
 '.Reservations[0].Instances[0].PublicDnsName' | sed 's/\"//g') && \  echo
 NVAWS_DNS=$NVAWS_DNS
```

1.4.3. Connect To Your VM Instance

SSH should work shortly after the instance reaches "running ok".

If started with CLI snippets and environment variables above, the command to SSH to your instance is:

```
ssh -i $NVAWS_KEYPATH/$NVAWS_KEYNAME.pem ubuntu@$NVAWS_DNS
```

Otherwise use your .pem key filename and the Public DNS name from the AWS Console to connect:

```
ssh -i my-key-pair.pem ubuntu@public-dns-name
```

If these instructions for SSH login do not work, see the AWS Connect to Your Linux Instance documentation for additional information.

1.4.4. Start/Stop/Terminate Your VM Instance

Once you are done with your instance you can stop (to be started again later) or terminate (delete) it. Refer to the Instance Lifecycle in the AWS documentation for more information.

Stop:

```
aws ec2 stop-instances --instance-ids $NVAWS_INSTANCE_ID
```

Start:

```
aws ec2 start-instances --instance-ids $NVAWS_INSTANCE_ID
```

Terminate:

```
aws ec2 terminate-instances --instance-ids $NVAWS_INSTANCE_ID
```

1.5. Persistent Data Storage for AWS Virtual Machines

You can create elastic block storage (EBS) from the AWS Console. EBS is used for persistent data storage, however, EBS cannot be shared across multiple VMs. To share persistent data storage, you need to use EFS.

The instructions set up a general purpose SSD volume type. However, you can specify a provisioned IOPS SSD for higher throughput, or set up software RAID, using mdadm, to create a volume with multiple EBS volumes.

See the Amazon documentation RAID Configuration on Linux for instructions on how to set up software RAID on local disks.

EBS is available in most regions with Amazon EC2 P3 or G4 instances.

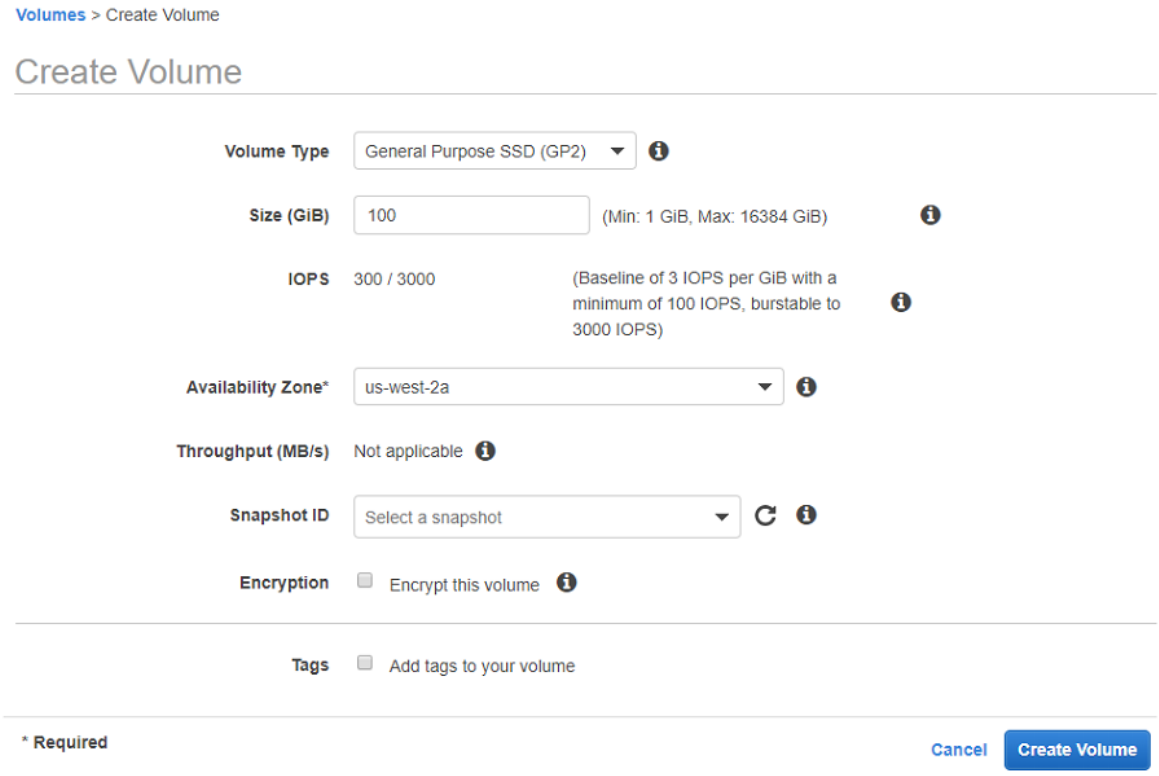
1.5.1. Create an EBS

1. Open the EBS Volumes Console.

Go to the main AWS console, click EC2, then expand Elastic Block Store from the side menu, if necessary, and click Volumes.

2. Click Create Volume.
3. Make selections at the Create Volume page.
   - Select General Purpose SSD (GP2) for the Volume Type.

     If higher throughput is needed, select Provisioned IOPS SSD (IO1).
   - Specify the volume size and Availability Zone.
   - (Optional) Add Tags.
   - Encryption is not needed if you are working with public datasets.
   - Snapshot ID is not needed.



4. Review the options and then click Create Volume.

1.5.2. Attach an EBS Volume to an EC2 Instance

1. Once you have created the EBS volume, select the volume and then select Actions->Attach Volume.
2. Specify your EC2-instance ID as well as a drive letter for the device name (for example, sdf), then click Attach.

   This creates a `/dev/xvdf` (or the driver letter that you picked) virtual disk on your EC2 instance.

You can view the volume by running the `lsblk` command.

```
~$ lsblk

NAME     MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
xvda     202:0    0  128G 0 disk
└─xvda1 202:1    0  128G 0 part /
xvdf     202:16   0  250G 0 disk
```

3. Create a filesystem on the EBS volume.

```
~# mkfs.ext4 /dev/xvdf

mke2fs 1.42.13 (17-May-2015)
Creating filesystem with 65536000 4k blocks and 16384000 inodes
Filesystem UUID: b0e3dee3-bf86-4e69-9488-cf4d4b57b367
Superblock backups stored on blocks:
 32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
 4096000, 7962624, 11239424, 20480000, 23887872
Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
```

4. Mount the volume to a mount directory.

```
~# mount /dev/xvdf /data
```

To mount the volume automatically every time the instance is stopped and restarted, add an entry to `/etc/fstab`. Refer to Amazon Documentation Making a Volume Available for Use

1.5.3. Delete an EBS Volume

Be aware that once you delete an EBS, you cannot undelete it.

1. Open the EBS Volumes Console.

   Go to the main AWS console, click EC2, then expand Elastic Block Store from the side menu, if necessary, and click Volumes.
2. Select your EBS.
3. Detach the volume from the EC2 instance.

   Select Actions->Detach Volume, then click Yes, Detach from the confirmation dialog.
4. Delect the storage volume.

   Select Actions->Delete Volume and then click Yes, Delete from the confirmation dialog.

1.5.4. Add a Dataset to an EBS Volume

Once you have created the EBS volume, you can upload datasets to the volume.

1.5.4.1. Upload a Dataset to the EBS Volume

1. Mount the EBS volume to `/data`.

   Issue the following to perform the one-time mount.

```
sudo mkdir /data
```

```
sudo mount /dev/xvdf /data
sudo chmod 777 /data
```

2. Copy the dataset onto the EBS volume in `/data`.

```
scp -i <.pem> -r local_dataset_dir/ ubuntu@<ec2-instance>:/data
```

1.5.4.2. Copy an Existing Dataset from EFS

1. Mount the EFS storage to `/data`, using the EFS storage DNS name.

   Issue the following to perform the one-time mount.

```
sudo mkdir /efs
sudo mount -t nfs4 -o \
  nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2 \
  EFS-DNS-NAME:/ /efs
sudo chmod 777 /efs
sudo cp -r /efs/<dataset> to /data
```

2. Copy the dataset from the EFS to the EBS volume..

```
sudo cp -r /efs/<dataset> to /data
```

1.5.5. Manage an EBS Volume Using AWS CLI

It is recommended that you use the AWS Console for EBS management. If you need to manage EBS file systems with the CLI, NVIDIA has created scripts available on GitHub at https://github.com/nvidia/ngc-examples.

These scripts will let you perform basic EBS management and can serve as the basis for further automation.

Chapter 2. NVIDIA Virtual Machine Images on AWS

NVIDIA makes available on the Amazon Web Service (AWS) platform a customized Amazon Machine Instance (AMI) optimized for the latest generations of NVIDIA GPUs - NVIDIA Volta™ GPUs and NVIDIA Turing GPUs. Running NVIDIA® GPU Cloud containers on AWS instances with NVIDIA Volta or NVIDIA Turing GPUs provides optimum performance of NGC containers for deep learning, machine learning, and HPC workloads.

See the NGC AWS Setup Guide for instructions on setting up and using the AMI, including instructions on using the following features:

- Automated login to the NGC container registry.
- Elastic Block Storage (EBS) mounting.

2.1. NVIDIA AI Enterprise AMI

2.1.1. Information

NVIDIA AI Enterprise is a secure, end-to-end, cloud-native suite of AI software enabling organizations to solve new challenges while increasing operational efficiency. It accelerates the data science pipeline and streamlines the development, deployment, and management of predictive AI models to automate essential processes and gain rapid insights from data. With an extensive library of full-stack software, including AI solution workflows, frameworks, pre-trained models, and infrastructure optimization. Global enterprise support and regular security reviews ensure business continuity and AI projects are successful and stay on track.

With NVIDIA AI Enterprise, customers get support and access to the following:

- NVIDIA AI Workflows, prepackaged reference applications that include Helm Charts, Jupyter Notebooks, and documentation to enable fast time to production for contact center intelligent virtual assistants, audio transcription, and cybersecurity digital fingerprinting to detect anomalies. Only available with an NVIDIA AI Enterprise subscription.

- Unencrypted pre-trained models for AI explainability, understanding model weights and biases, and faster debugging and customization. Only available with an NVIDIA AI Enterprise subscription.
- Frameworks and tools to accelerate AI development (PyTorch, TensorFlow, NVIDIA RAPIDS, TAO Toolkit, TensorRT, and Triton Inference Server).
- Healthcare-specific frameworks and applications including NVIDIA Clara MONAI and NVIDIA Clara Parabricks.
- NVIDIA AI Enterprise includes support for all NVIDIA AI software published on the NGC public catalog labeled "NVIDIA AI Enterprise Supported." Over 50 pre-trained models, frameworks, and development tools.
- The NVIDIA AI Enterprise marketplace offer also includes an AMI, which provides a standard, optimized run time for easy access to the NVIDIA AI Enterprise software mentioned above and ensures development compatibility between clouds and on-premises infrastructure. Develop once, run anywhere.

Contact NVIDIA to learn more about NVIDIA AI Enterprise on AWS and for private pricing by filling out the form here.

To get started, refer to the NVIDIA AI Enterprise on AWS Marketplace Quick Start Guide.

2.1.2. Release Notes

Version 23.11-NVAIE-4.1

- Ubuntu Server 22.04 (x86)
- NVIDIA vGPU Driver 535.129.03
- Docker-CE 24.0.7
- NVIDIA Container Toolkit 1.14.3-1
- Latest AWS CLI
- Miniconda
- JupyterLab latest and other Jupyter core packages
- NGC CLI 3.22.0
- Git, Python3-PIP

Version 23.09.2-NVAIE-4.0

- Ubuntu Server 22.04
- NVIDIA AI Enterprise Catalog access script
- NVIDIA vGPU Driver 535.54.03 (v16.0)
- Docker-CE 24.0.5
- NVIDIA Container Toolkit 1.13.5
- NGC CLI 3.22.0

- Miniconda
- JupyterLab (within miniconda)

Version 23.04.0-NVAIE-3.1

- Ubuntu Server 20.04
- NVIDIA AI Enterprise Catalog access script
- NVIDIA vGPU Driver 525.105.17
- Docker-CE 23.0.3
- NVIDIA Container Toolkit 1.13.0-1
- AWS CLI 2.11.14
- NGC CLI 3.20.0
- Miniconda 23.3.1
- JupyterLab (within conda base env) 3.5.3
- Git 2.25.1

2.2. NVIDIA Riva AMI

2.2.1. Information

NVIDIA® Riva is a GPU-accelerated software development kit (SDK) for building and deploying fully customizable, real-time multilingual speech and translation AI pipelines —including automatic speech recognition (ASR), neural machine translation (NMT), and text-to-speech (TTS). Riva delivers world-class, out-of-the-box, accurate transcriptions and translations and engaging professional voices. It can be deployed on-premises, in any cloud environment, in embedded systems, and at the edge.

With NVIDIA Riva, customers can customize for different languages, accents, domains, vocabulary, and contexts to achieve the best possible accuracy for their use cases, as well as the desired brand voice and intonation. It provides consistent experiences for hundreds of thousands of input streams with higher inference performance compared to existing technologies. To accelerate building speech-AI-based AI solutions, Riva offers pre-packaged AI workflows for audio transcription and intelligent virtual assistants that include pretrained models and resources such as Helm Charts, Jupyter Notebooks, and documentation.

Riva, a premium edition of the NVIDIA AI Enterprise software platform, is available for $60 per GPU hour for high-performance GPUs such as NVIDIA A100. Contact NVIDIA for special pricing for alternative GPU options or private offers.

2.2.2. Release Notes

Version 2023.06.2-riva

- Ubuntu Server 22.04
- NVIDIA vGPU Driver 525.60.13
- Docker-CE 24.0.2
- NVIDIA Container Toolkit 1.11.0-1
- AWS CLI
- NGC CLI 3.22.0
- Miniconda 23.5.0
- JupyterLab (within conda base env) 34.0.2
- Git 2.34.1

2.3. NVIDIA GPU-Optimized AMI

2.3.1. Information

The NVIDIA GPU-Optimized AMI is a virtual machine image for accelerating your Machine Learning, Deep Learning, Data Science and HPC workloads. Using this AMI, you can spin up a GPU-accelerated EC2 VM instance in minutes with a pre-installed Ubuntu OS, GPU driver, Docker and NVIDIA container toolkit.

Moreover, this AMI provides easy access to NVIDIA's NGC Catalog, a hub for GPU-optimized software, for pulling & running performance-tuned, tested, and NVIDIA certified docker containers. NGC provides free access to containerized AI, Data Science, and HPC applications, pre-trained models, AI SDKs and other resources to enable data scientists, developers, and researchers to focus on building solutions, gathering insights, and delivering business value.

This GPU-optimized AMI is provided free of charge for developers with an enterprise support option. For more information on enterprise support, please visit NVIDIA AI Enterprise.

2.3.2. Release Notes

Version 23.09.1

- Ubuntu Server 22.04 (x86)
- NVIDIA TRD Driver 535.54.03
- Docker-ce 24.0.6
- NVIDIA Container Toolkit 1.13.5

- Latest AWS CLI
- Miniconda
- JupyterLab latest and other Jupyter core packages
- NGC CLI 3.22.0
- Git, Python3-PIP

Version 23.03.0

- Ubuntu Server 20.04
- NVIDIA Driver 525.85.12
- Docker-ce 23.0.1
- NVIDIA Container Toolkit 1.21.1-1
- AWS Command Line Interface (CLI), NGC-CLI 3.16.0
- Miniconda 23.1.0
- JupyterLab and other Jupyter core packages
- Git, Python3-PIP

Key Changes

- Updated NVIDIA Driver to 525.85.12
- Updated Docker-ce to 23.0.1
- Updated Nvidia Container Toolkit to Version 1.12.1-1
- Updated Miniconda, JupyterLab, NGC-CLI, Git, Python3-PIP to latest

Version 22.06.0

- Ubuntu Server 20.04
- NVIDIA Driver 515.48.07
- Docker-ce 20.10.17
- NVIDIA Container Toolkit 1.10.0-1
- NVIDIA Container Runtime 3.10.0-1
- AWS Command Line Interface (CLI)
- Miniconda 4.13.0
- JupyterLab 3.4.3 and other Jupyter core packages
- NGC-CLI 3.0.0
- Git, Python3-PIP

Key Changes

- Updated NVIDIA Driver to 515.48.07
- Updated Docker-ce to 20.10.17

- Updated Nvidia Container Toolkit to Version 1.10.0-1
- Updated Nvidia Container Runtime to Version 3.10.0-1
- Packaged additional tools: Miniconda, JupyterLab, NGC-CLI, Git, Python3-PIP

2.4. NVIDIA HPC SDK GPU-Optimized AMI

2.4.1. Information

The NVIDIA HPC SDK is a comprehensive suite of compilers, libraries and tools essential to maximizing developer productivity and the performance and portability of HPC applications.

The NVIDIA HPC SDK C, C++, and Fortran compilers support GPU acceleration of HPC modeling and simulation applications with standard C++ and Fortran, OpenACC directives, and CUDA. GPU-accelerated math libraries maximize performance on common HPC algorithms, and optimized communications libraries enable standards-based multi-GPU and scalable systems programming. Performance profiling and debugging tools simplify porting and optimization of HPC applications, and containerization tools enable easy deployment on-premises or in the cloud.

Key features of the NVIDIA HPC SDK for Linux include:

- Support for NVIDIA Ampere Architecture GPUs with FP16, TF32 and FP64 tensor cores
- NVC++ ISO C++17 compiler with Parallel Algorithms acceleration on GPUs, OpenACC and OpenMP
- NVFORTRAN ISO Fortran 2003 compiler with array intrinsics acceleration on GPUs, CUDA Fortran, OpenACC and OpenMP
- NVC ISO C11 compiler with OpenACC and OpenMP
- NVCC NVIDIA CUDA C++ compiler
- NVIDIA Math Libraries including cuBLAS, cuSOLVER, cuSPARSE, cuFFT, cuTENSOR and cuRAND
- Thrust, CUB, and libcu++ GPU-accelerated libraries of C++ parallel algorithms and data structures
- NCCL, NVSHMEM and Open MPI libraries for fast multi-GPU/multi-node communications
- NVIDIA Nsight Systems/Compute for interactive HPC applications performance profiler

2.4.2. Release Notes

Version 23.11

- Ubuntu Server 22.04 (x86)
- NVIDIA Driver 535.129.03
- Docker-ce 24.0.7
- NVIDIA Container Toolkit Version: 1.14.3-1
- Latest AWS CLI
- Miniconda latest
- JupyterLab latest and other Jupyter core packages
- NGC-CLI 3.35.0
- Git
- Updated HPC SDK 23.11

Version 23.03.0

- Ubuntu Server 20.04
- NVIDIA Driver 525.85.12
- Docker-ce 23.0.1
- NVIDIA Container Toolkit Version: 1.12.1-1
- AWS CLI, NGC-CLI 3.16.0
- Miniconda latest
- JupyterLab and other Jupyter core packages
- Git, Python3-PIP
- HPC SDK 23.1
- NVIDIA Peer Memory: 1.3
- MOFED: 5.8-1.0.1.1

Key Changes

- Updated NVIDIA Driver to 525.85.12
- Updated Docker-ce to 23.0.1
- Updated Nvidia Container Toolkit to Version 1.12.1-1
- Updated Nvidia Container Runtime to Version 3.12.0-1
- Updated NGC-CLI to 3.16.0
- Updated HPC SDK to 23.1

Version 22.08.0

- Ubuntu Server 20.04
- NVIDIA Driver 515.65.01
- Docker-ce 20.10.17
- NVIDIA Container Toolkit Version: 1.10.1-1
- NVIDIA Container Runtime Version: 3.10.0-1
- AWS Command Line Interface (CLI)

Key Changes

- Updated NVIDIA Driver to 515.48.07
- Updated Docker-ce to 20.10.17
- Updated NVIDIA Container Toolkit to Version 1.10.0-1
- Updated NVIDIA Container Runtime to Version 3.10.0-1

Known Issues

- The version of Nsight Systems bundled with the HPC SDK 22.7 fails with the error 'Agent launcher failed' on some instance types. The issue is fixed in Nsight Systems version 2022.3.4 and later, which can be installed separately from the Nsight Systems downloads page. For more information, refer to the Nsight Systems documentation.

Version 22.01.0

- Ubuntu Server 20.04
- NVIDIA Driver 470.103.01
- Docker-ce 20.10.11
- NVIDIA Container Toolkit 1.7.0-1
- NVIDIA Container Runtime 3.7.0-1
- MOFED Version: 5.4-1.0.3.0
- NVIDIA Peer Memory Version: 1.2
- NVIDIA HPC SDK Version: 22.1

2.5. NVIDIA GPU-Optimized AMI (ARM64)

2.5.1. Information

The NVIDIA GPU-Optimized AMI (ARM64) is an optimized environment for running the Deep Learning, Data Science, and HPC containers available from NVIDIA's NGC Catalog. The Docker containers available on the NGC Catalog are tuned, tested, and certified by NVIDIA to take full advantage of NVIDIA GPUs with ARM CPU instances. Deep Learning, Data Science, and HPC containers from the NGC Catalog require this AMI for the best GPU acceleration on AWS ARM64 GPU instances.

2.5.2. Release Notes

Version 22.06.0

- Ubuntu Server 20.04
- NVIDIA Driver 515.48.07
- Docker-ce 20.10.17
- NVIDIA Container Toolkit 1.10.0-1
- NVIDIA Container Runtime 3.10.0-1
- AWS Command Line Interface (CLI)
- Miniconda 4.13.0
- JupyterLab 3.4.3 and other Jupyter Core packages
- NGC CLI 3.0.0
- Git, Python3-PIP

Key Changes

- Updated NVIDIA Driver to 515.48.07
- Updated Docker-ce to 20.10.17
- Updated NVIDIA Container Toolkit to Version 1.10.0-1
- Updated NVIDIA Container Runtime to Version 3.10.0-1
- Packaged additional tools: Miniconda, JupyterLab, NGC CLI, Git, Python3-PIP

Version 21.11.0

- Ubuntu Server 20.04
- NVIDIA Driver 460.73.01
- Docker-ce 20.10.6
- NVIDIA Container Toolkit 1.5.0-1
- NVIDIA Container Runtime 3.5.0-1

2.6. NVIDIA Omniverse GPU-Optimized AMI

2.6.1. Information

NVIDIA Omniverse is a platform for building custom 3D pipelines and operating large-scale virtual worlds.

The NVIDIA Omniverse GPU-Optimized AMI is a virtual machine image optimized to run NVIDIA Omniverse 3D graphics and simulation workloads, including:

- Omniverse Farm, for scaling 3D and simulation compute across cloud instances
- Omniverse Replicator, for 3D synthetic data generation
- NVIDIA Isaac Sim, for training and simulating autonomous machines

Using this AMI, you can spin up a GPU-accelerated EC2 instance in minutes with a pre-installed Ubuntu OS, GPU driver, X11, Docker, and NVIDIA container toolkit.

This AMI is optimized for pulling and running performance-tuned, tested, and Omniverse docker containers. The NGC catalog provides free access to containerized AI, data science, simulation and HPC applications, pre-trained models, AI SDKs and other resources to enable data scientists, developers, roboticists, and researchers to focus on building and deploying solutions.

2.6.2. Release Notes

Version 1.3.4

- Ubuntu Server OS: 20.04.5 LTS
- NVIDIA Driver: 470.141.03
- Docker CE: 20.10.18
- NVIDIA Container Toolkit: 2.11.0-1
- X11: 1:7.7+19ubuntu14

2.7. NVIDIA Cloud Native Stack AMI

2.7.1. Information

NVIDIA Cloud Native Stack AMI is a GPU-accelerated AMI that is pre-installed with Cloud Native Stack, which is a reference architecture that includes upstream Kubernetes and the NVIDIA GPU and Network Operator. NVIDIA Cloud Native Stack AMI allows developers to build, test and run GPU-accelerated containerized applications that are orchestrated by Kubernetes.

2.7.2. Release Notes

Version 6.2

- Ubuntu Server 20.04
- Containerd 1.6.5
- Kubernetes 1.23.8
- Helm 3.8.2
- GPU Operator 1.11.0
- NVIDIA Driver 515.65.01

2.8. NVIDIA cuQuantum Appliance AMI

2.8.1. Information

The NVIDIA cuQuantum Appliance is a highly performant multi-GPU multi-node solution for quantum circuit simulation. It contains NVIDIA cuStateVec and cuTensorNet libraries which optimize state vector and tensor network simulation, respectively. The cuTensorNet library functionality is accessible through Python for Tensor Network operations. NVIDIA provides the following simulators with the cuStateVec libraries:

- IBM Qiskit Aer frontend via cusvaer, NVIDIA distributed state vector backend solver.
- Multi-GPU-optimized Google Cirq frontend via qsim, Google state vector simulator.

2.8.2. Release Notes

Version 23.03

- Ubuntu Server 22.04
- NVIDIA Driver 525.105.17
- NVIDIA cuQuantum Appliance Docker Container 23.03
- Docker-ce 24.0.1
- NVIDIA Container Toolkit 1.13.0-1
- AWS CLI, NGC CLI

- Miniconda, JupyterLab (within conda base env), Git: latest

Version 22.11

- Ubuntu Server 20.04
- NVIDIA Driver 525.85.12
- NVIDIA cuQuantum Appliance Docker Container 22.11
- Docker-ce 23.0.1
- NVIDIA Container Toolkit 1.12.0-1
- AWS CLI, NGC CLI
- Miniconda, JupyterLab (within conda base env), Git: latest

Known Issues

- Some instances on Azure, (specifically ND40rs v2) may emit an "unsupported instance type" warning at log-in. This is a known issue and will be addressed in the next release.

Chapter 3. Known Security Vulnerabilities

The NVIDIA GPU-Optimized AMI includes conda by default in order to use jupyter-lab notebooks. The internal Python dependencies may be patched in newer Python versions, but conda must use the specific versions in the AMI. These vulnerabilities are not directly exploitable unless there is a vulnerability in conda itself. An attacker would need to obtain access to a VM running conda, so it is important that VM access must be protected. See the security best practices section.

The following releases are affected by the vulnerabilities:

- NVIDIA GPU-Optimized AMI 22.06
- NVIDIA GPU-Optimized AMI (ARM64) 22.06

The list of vulnerabilities are:

- GHSA-3gh2-xw74-jmcw: High; Django 2.1; SQL injection
- GHSA-6r97-cj55-9hrq: Critical; Django 2.1; SQL injection
- GHSA-c4qh-4vgv-qc6g: High; Django 2.1; Uncontrolled resource consumption
- GHSA-h5jv-4p7w-64jg: High; Django 2.1; Uncontrolled resource consumption
- GHSA-hmr4-m2h5-33qx: Critical; Django 2.1; SQL injection
- GHSA-v6rh-hp5x-86rv: High; Django 2.1; Access control bypass
- GHSA-v9qg-3j8p-r63v: High; Django 2.1; Uncontrolled recursion
- GHSA-vfq6-hq5r-27r6: Critical; Django 2.1; Account hijack via password reset form
- GHSA-wh4h-v3f2-r2pp: High; Django 2.1; Uncontrolled memory consumption
- GHSA-32gv-6cf3-wcmq: Critical; Twisted 18.7.0; HTTP/2 DoS attack
- GHSA-65rm-h285-5cc5: High; Twisted 18.7.0; Improper certificate validation
- GHSA-92x2-jw7w-xvvx: High; Twisted 18.7.0; Cookie and header exposure
- GHSA-c2jg-hw38-jrqq: High; Twisted 18.7.0; HTTP request smuggling
- GHSA-h96w-mmrf-2h6v: Critical; Twisted 18.7.0; Improper input validation
- GHSA-p5xh-vx83-mxcj: Critical; Twisted 18.7.0; HTTP request smuggling
- GHSA-5545-2q6w-2gh6: High; numpy 1.15.1; NULL pointer dereference

- CVE-2019-6446: Critical; numpy 1.15.1; Deserialization of untrusted data
- GHSA-h4m5-qpfp-3mpv: High; Babel 2.6.0; Arbitrary code execution
- GHSA-ffqj-6fqr-9h24: High; PyJWT 1.6.4; Key confusion through non-blocklisted public key formats
- GHSA-h7wm-ph43-c39p: High; Scrapy 1.5.1; Uncontrolled memory consumption
- CVE-2022-39286: High; jupyter_core 4.11.2; Arbitrary code execution
- GHSA-55x5-fj6c-h6m8: High; lxml 4.2.4; Crafted code allowed through lxml HTML cleaner
- GHSA-wrxv-2j5q-m38w: High; lxml 4.2.4; NULL pointer dereference
- GHSA-gpvv-69j7-gwj8: High; pip 8.1.2; Path traversal
- GHSA-hj5v-574p-mj7c: High; py 1.6.0; Regular expression DoS
- GHSA-x84v-xcm2-53pg: High; requests 2.19.1; Insufficiently protected credentials
- GHSA-mh33-7rrq-662w: High; urllib3 1.23; Improper certificate validation
- CVE-2021-33503: High; urllib3 1.23; Denial of service attack
- GHSA-2m34-jcjv-45xf: Medium; Django 2.1; XSS in Django
- GHSA-337x-4q8g-prc5: Medium; Django 2.1; Improper input validation
- GHSA-68w8-qjq3-2gfm: Medium; Django 2.1; Path traversal
- GHSA-6c7v-2f49-8h26: Medium; Django 2.1; Cleartext transmission of sensitive information
- GHSA-6mx3-3vqg-hpp2: Medium; Django 2.1; Django allows unprivileged users can read the password hashes of arbitrary accounts
- GHSA-7rp2-fm2h-wchj: Medium; Django 2.1; XSS in Django
- GHSA-hvmf-r92r-27hr: Medium; Django 2.1; Django allows unintended model editing
- GHSA-wpjr-j57x-wxfw: Medium; Django 2.1; Data leakage via cache key collision in Django
- GHSA-9x8m-2xpf-crp3: Medium; Scrapy 1.5.1; Credentials leakage when using HTTP proxy
- GHSA-cjvr-mfj7-j4j8: Medium; Scrapy 1.5.1; Incorrect authorization and information exposure
- GHSA-jwqp-28gf-p498: Medium; Scrapy 1.5.1; Credential leakage
- GHSA-mfjm-vh54-3f96: Medium; Scrapy 1.5.1; Cookie-setting not restricted
- GHSA-6cc5-2vg4-cc7m: Medium; Twisted 18.7.0; Injection of invalid characters in URI/method
- GHSA-8r99-h8j2-rw64: Medium; Twisted 18.7.0; HTTP Request Smuggling
- GHSA-vg46-2rrj-3647: Medium; Twisted 18.7.0; NameVirtualHost Host header injection
- GHSA-39hc-v87j-747x: Medium; cryptography 37.0.2; Vulnerable OpenSSL included in cryptography wheels

- GHSA-hggm-jpg3-v476: Medium; cryptography 2.3.1; RSA decryption vulnerable to Bleichenbacher timing vulnerability
- GHSA-jq4v-f5q6-mjqq: Medium; lxml 4.2.4; XSS
- GHSA-pgww-xf46-h92r: Medium; lxml 4.2.4; XSS
- GHSA-xp26-p53h-6h2p: Medium; lxml 4.2.4; Improper Neutralization of Input During Web Page Generation in LXML
- GHSA-6p56-wp2h-9hxr: Medium; numpy 1.15.1; NumPy Buffer Overflow, very unlikely to be exploited by an unprivileged user
- GHSA-f7c7-j99h-c22f: Medium; numpy 1.15.1; Buffer Copy without Checking Size of Input in NumPy
- GHSA-fpfv-jqm9-f5jm: Medium; numpy 1.15.1; Incorrect Comparison in NumPy
- GHSA-5xp3-jfq3-5q8x: Medium; pip 8.1.2; Improper Input Validation in pip
- GHSA-w596-4wvx-j9j6: Medium; py 1.6.0; ReDoS in py library when used with subversion
- GHSA-hwfp-hg2m-9vr2: Medium; pywin32 223; Integer overflow in pywin32
- GHSA-r64q-w8jr-g9qp: Medium; urllib3 1.23; Improper Neutralization of CRLF Sequences
- GHSA-wqvq-5m8c-6g24: Medium; urllib3 1.23; CRLF injection

Notice

THE INFORMATION IN THIS GUIDE AND ALL OTHER INFORMATION CONTAINED IN NVIDIA DOCUMENTATION REFERENCED IN THIS GUIDE IS PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE INFORMATION FOR THE PRODUCT, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the product described in this guide shall be limited in accordance with the NVIDIA terms and conditions of sale for the product.

THE NVIDIA PRODUCT DESCRIBED IN THIS GUIDE IS NOT FAULT TOLERANT AND IS NOT DESIGNED, MANUFACTURED OR INTENDED FOR USE IN CONNECTION WITH THE DESIGN, CONSTRUCTION, MAINTENANCE, AND/OR OPERATION OF ANY SYSTEM WHERE THE USE OR A FAILURE OF SUCH SYSTEM COULD RESULT IN A SITUATION THAT THREATENS THE SAFETY OF HUMAN LIFE OR SEVERE PHYSICAL HARM OR PROPERTY DAMAGE (INCLUDING, FOR EXAMPLE, USE IN CONNECTION WITH ANY NUCLEAR, AVIONICS, LIFE SUPPORT OR OTHER LIFE CRITICAL APPLICATION). NVIDIA EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR SUCH HIGH RISK USES. NVIDIA SHALL NOT BE LIABLE TO CUSTOMER OR ANY THIRD PARTY, IN WHOLE OR IN PART, FOR ANY CLAIMS OR DAMAGES ARISING FROM SUCH HIGH RISK USES.

NVIDIA makes no representation or warranty that the product described in this guide will be suitable for any specified use without further testing or modification. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to ensure the product is suitable and fit for the application planned by customer and to do the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this guide. NVIDIA does not accept any liability related to any default, damage, costs or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this guide, or (ii) customer product designs.

Other than the right for customer to use the information in this guide with the product, no other license, either expressed or implied, is hereby granted by NVIDIA under this guide. Reproduction of information in this guide is permissible only if reproduction is approved by NVIDIA in writing, is reproduced without alteration, and is accompanied by all associated conditions, limitations, and notices.

Trademarks

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2024 NVIDIA CORPORATION & AFFILIATES. All rights reserved.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
www.nvidia.com