# DriveSim Scenario Runner Cluster with NVIDIA DeepOps

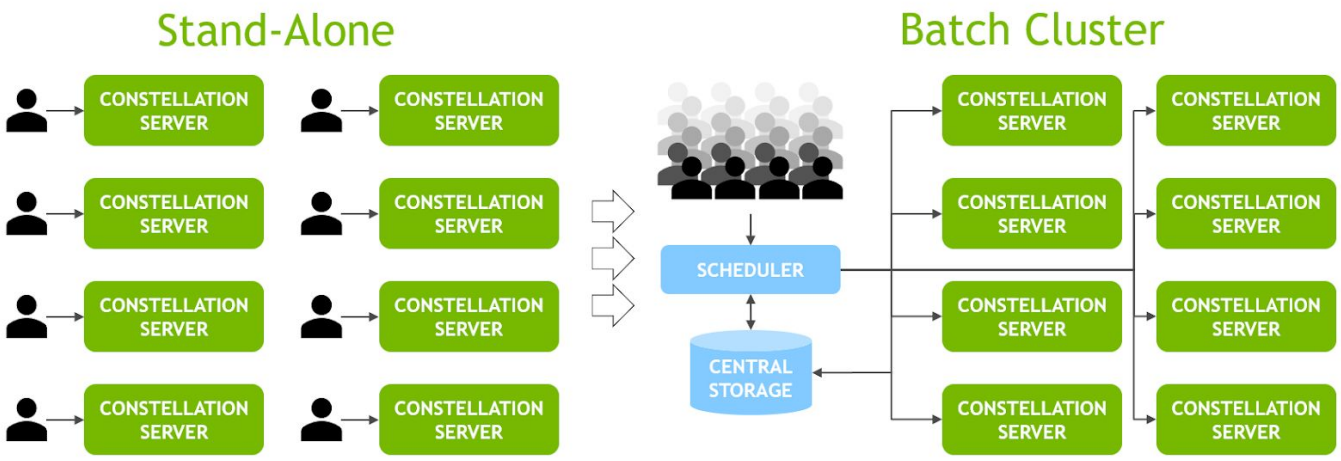
Lightweight framework for distributed batch execution of DriveSim simulation workloads across a group of servers and artifact collection

# Table of contents

# Introduction

This document describes a lightweight framework for converting a group of “stand-alone” DriveSim capable systems into an interconnected batch cluster that enables distributed execution of DriveSim jobs across the nodes.

This can be deployed on prem or in the cloud, and supports HIL/SIL on Constellation servers and SIL on workstations.




## Audience

This document is intended for an IT organization with prior experience in system administration and cluster management, who is interested in distributing DriveSim simulations on multiple servers in their datacenter or cloud.

## Scope and Assumptions

This framework intends to provide a *low-cost* solution that is *easy to implement* and *adaptable* to a wide range of environments. As such, the scope is intentionally trimmed to focus specifically on job distribution and DriveSim container orchestration.

**Scope:**

- DriveSim + AV container orchestration with **Scenario Runner (Docker Compose)**
- Job queuing and launching with **Slurm** HPC scheduler
- Deployment with **NVIDIA DeepOps Ansible** scripts from a deployment computer (laptop or VM) onto a group of pre-installed “stand-alone” Constellation servers

**Assumptions:**

- Constellation servers or DriveSim workstations are to be pre-installed and functional as “stand-alone” devices and capable of running DriveSim.
- Constellation server or DriveSim workstation platform software versions meet the minimum requirement - see “System Requirement” section for more details.

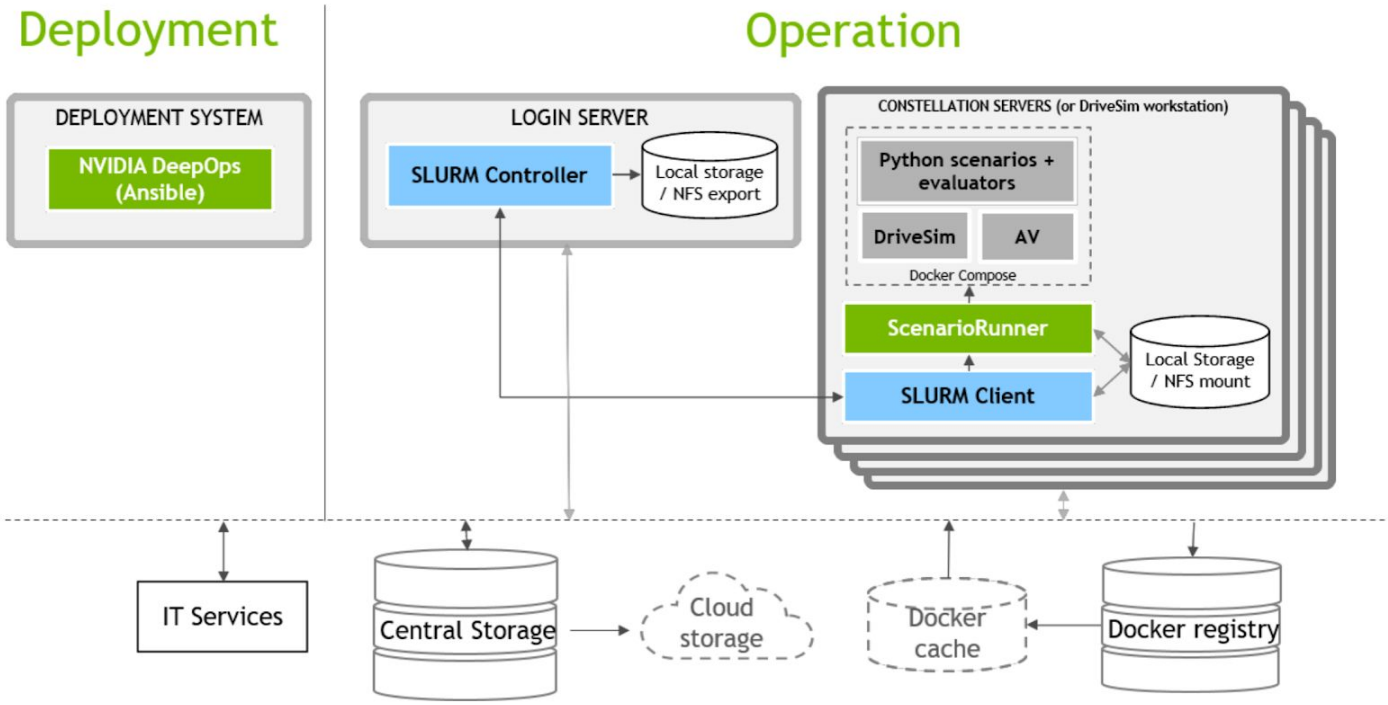
**Beyond the scope of this document:**

- How to configure IT services: user management, authentication, access control, network layout, traffic optimization, storage design, etc vary for every organization and will not be discussed here unless a specific requirement exists.
- Advanced cluster administration, advanced monitoring, and high utilization optimizations: in order to keep down the cost of entry this document will not delve into complex cluster designs.

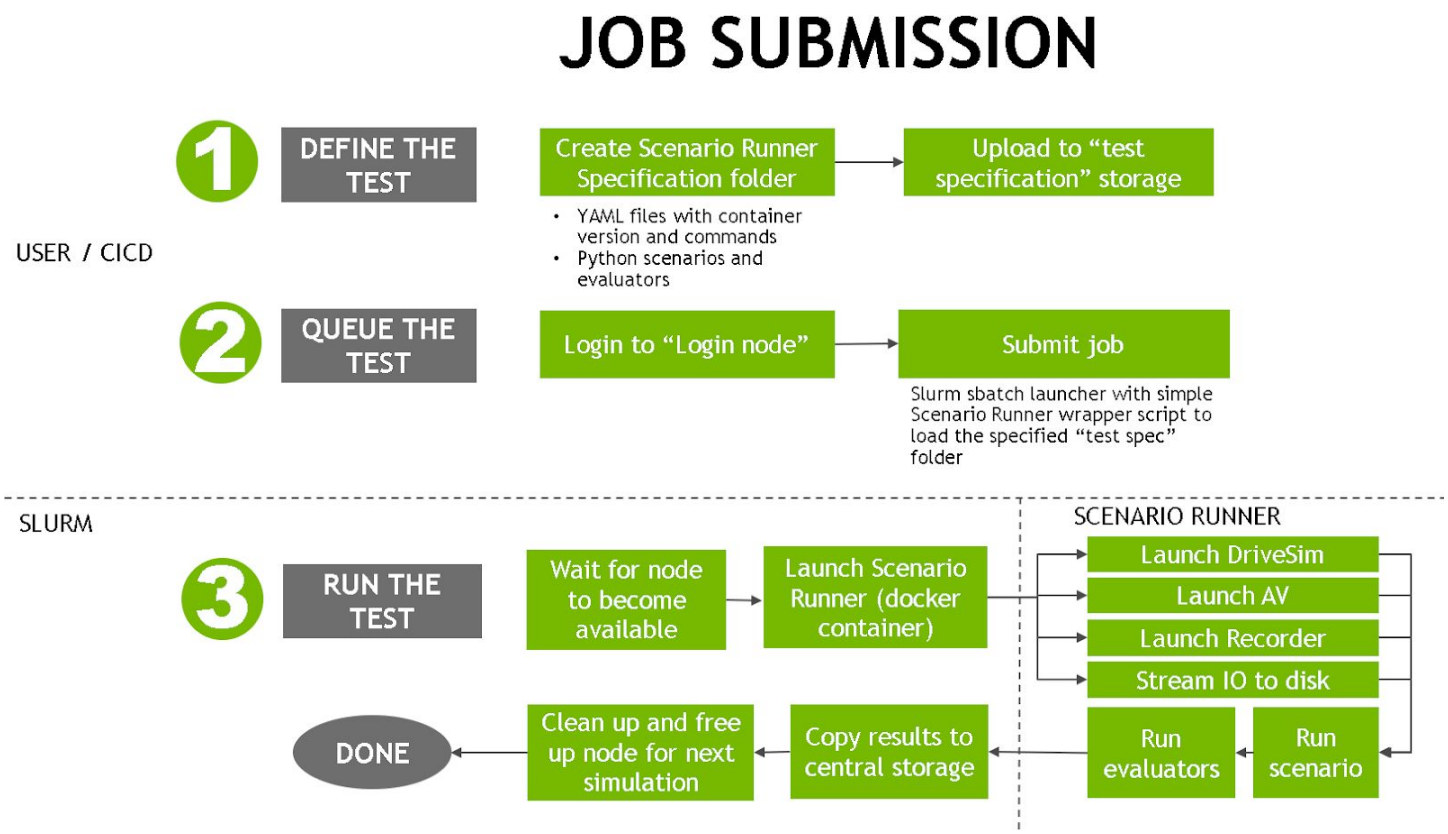
# Components

| | | |
|---|---|---|
| DRIVESIM + AV CONTAINER ORCHESTRATION | Scenario Runner | • Launch and manage containers for DriveSim, AV.<br>• Launch scenarios and evaluators.<br>• Collect logs and recordings.<br>• Standardized API for defining container names, versions and commands (YAML manifest as input). |
| JOB SCHEDULING | SLURM | • Widely adopted by HPC industry for job scheduling on distributed clusters.<br>• Easily to deploy with NVIDIA DeepOps. |
| DEPLOYMENT | NVIDIA DeepOps (Ansible) | • NVIDIA maintained Ansible toolkit for deploying GPU server clusters. |
| CENTRAL STORAGE | Local: NFS share<br>Cloud: AWS S3 | • Local NFS export for central location for scripts and configuration.<br>• Cloud storage for storing test definitions and outputs.<br>• Simple scripts to demonstrate file management, can be expanded to a more robust solution, as necessary. |

# Architecture



# Execution Flow



# Technology

## Scenario Runner

NVIDIA tool for orchestrating DriveSim and AV containers during a simulation. It is able to:

- Launch DriveSim
- Launch simulation scenario
- Launch the AV ECU
- Run the Evaluator framework
- Collect logs and results into the output folder

The primary use case is closed-loop simulation, but other use cases are supported. Scenarios can be executed on workstations in SIL mode, Constellation in HIL, or some other combination.

### Input

Test specification folder:

- ScenarioRunner manifest: YAML file specifying container name, versions, and exec commands
- Python scenario to execute
- Evaluator config files

### Output

Logs and artifacts collected by the simulation.

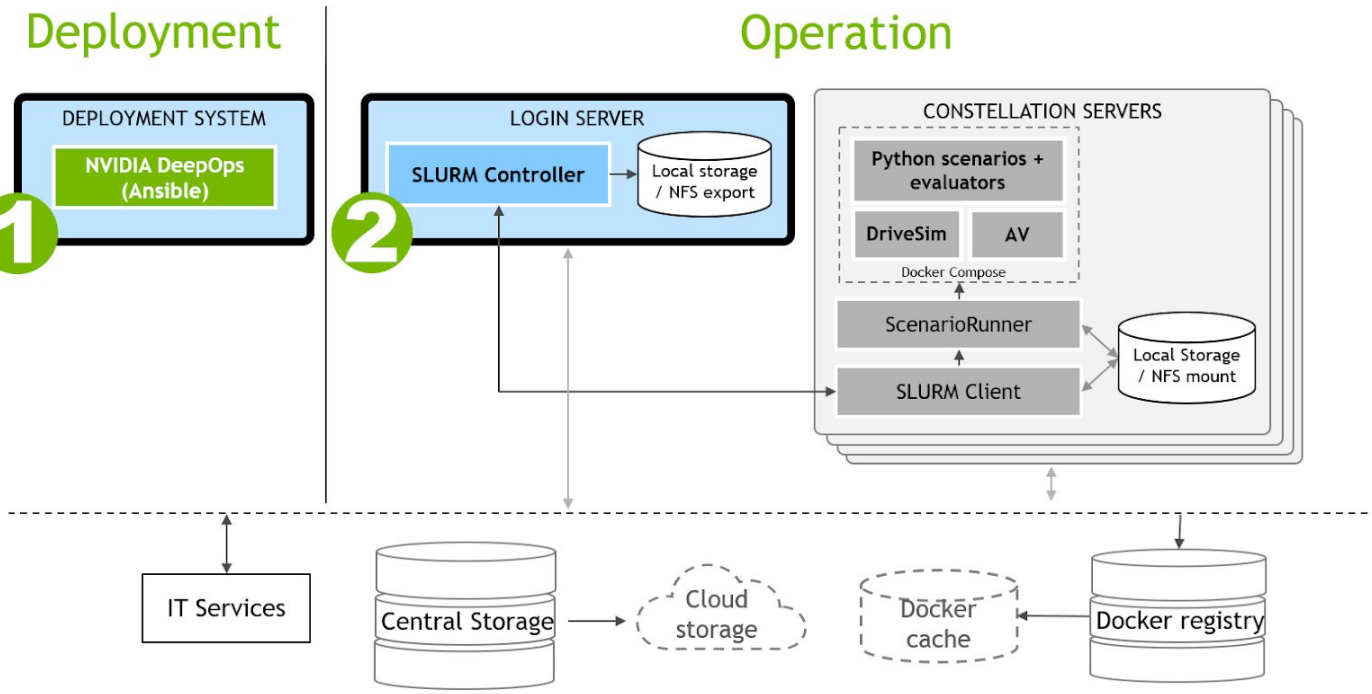
## Slurm

https://slurm.schedmd.com/overview.html
“Slurm is an open-source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters”

## NVIDIA DeepOps Ansible Toolkit

https://github.com/NVIDIA/deepops
While not required for the functionality and operation of the cluster, NVIDIA DeepOps Ansible toolkit comes with great Slurm installation roles that can be leveraged for this install. However, any deployment method of choice will work.

# System Requirements




## Cluster management servers

Two standard Linux CPU compute systems running Ubuntu 18.04 or higher. They can be metal or virtual.

1. The **deployment system** is used to run the ***Ansible*** scripts and is needed during deployment or update of the cluster software.
    - Can be a laptop or VM with ssh access to all nodes.
    - Will run the DeepOps Ansible scripts (or a deployment mechanism of your choice).
    - Minimum of 4GB of RAM, 20gb disk space.
    - The system can be repurposed once the deployment is complete and verified. The ansible files should be saved for potential maintenance tasks in the future.

2. The **login server (control plane)** will host the ***Slurm*** controller and will serve as the central job scheduling node and hold the state of the cluster. Users or CI/CD process will ssh to “Login node” to submit jobs:
    - This system must be on the same network subnet as the constellation nodes.
    - Minimum system requirements:
        i. 4GB of RAM
        ii. 60GB of disk space (for Slurm logs)
        iii. Minimum 4 core CPU

- **Assumption**: the login server will be used only for Slurm scheduling. If planning to run additional services or allow user interactivity on the login server, please be sure to adjust requirements as necessary.

### Scaling considerations for login server:

- Performance: **1 server is enough for at least 1000 nodes!** Slurm is designed to operate significantly large clusters with far more complex scheduling algorithms than what is used for this design. Since only one simulation can run on a server at a time, each node is scheduled as an atomic unit, greatly reducing the overhead necessary for scheduling jobs, and as such there are no scaling concerns by using a single server. For more information on Slurm large clusters see this link: https://slurm.schedmd.com/big_sys.html

- High Availability: Slurm offers a simple fail-over mechanism that can be configured if constant availability is a requirement. Slurm handles outages very well so unless extreme availability is desired, the cost of the complexity is not worthwhile. The reference will not configure Slurm in HA mode. For more info on Slurm HA look here: https://www.totalcae.com/learn/high-availability-slurm/ and https://slurm.schedmd.com/quickstart_admin.html#HA

## Constellation/DriveSim servers

Servers capable of running DriveSim simulations with AV stack. These can be instances in the cloud but for simplicity this document will assume they are physical servers.

**Assumption:** Constellation servers are racked, cabled, and deployed as “stand-alone” servers prior to beginning with this install**.**

Minimum software version requirement:

- Constellation BaseOS: 4.3.3
- DriveSIM: 1.16.x
- NVIDIA Driver: 440
- Scenario Runner release 0.1.11

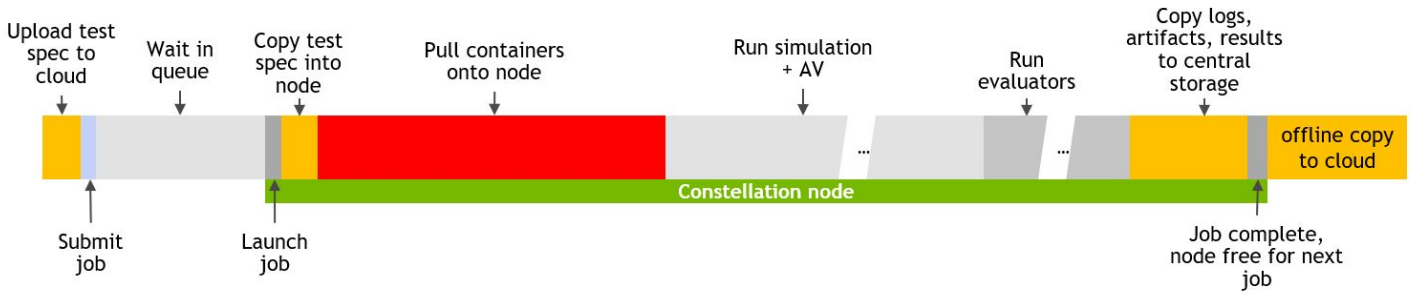
It is also recommended to have all system firmware up to date - please refer to the official product documentation for details.

# Central Storage

Central storage is needed to share execution scripts and logs, and to collect data and artifacts from the simulations.

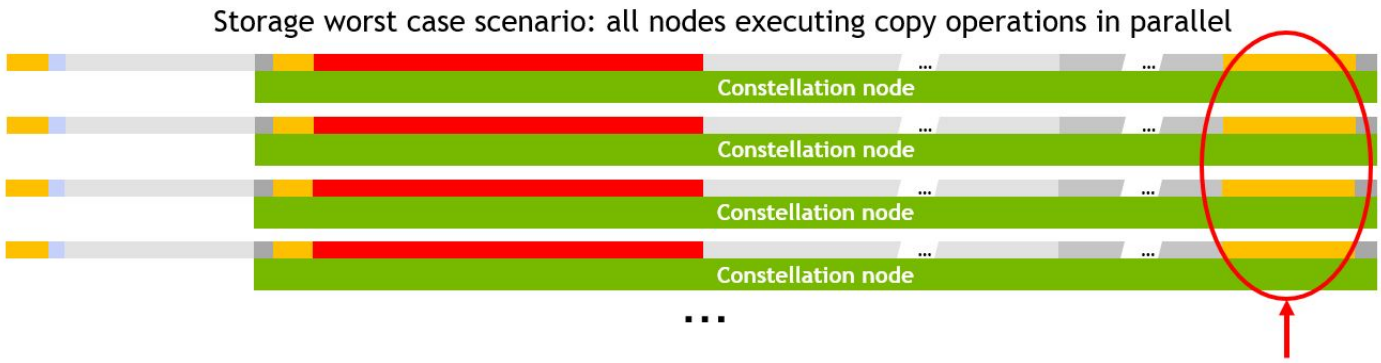
During a simulation, all IO stays local in */tmp* on the Constellation node. The YAML specification and Python scenario files are copied into the node before starting the simulation, and the generated logs and artifacts are copied out to a central location when the simulation is complete.

Here is a view of a full run, notice the yellow blocks which are storage operations.




NOTE: picture for visualization purposes, not actual scale.

When planning for storage performance consider the following **worst-case scenario** for the storage operations, where all the nodes are executing copy operations at the same time:




Three central storage locations are needed:

1. Central storage for Slurm logs and shared scripts
2. Central storage for storing test results and artifacts
3. Central storage for sharing test specification files

## 1. Slurm logs and shared scripts

Although not required by Slurm, this reference will use an NFS export mounted across all nodes. This simplifies access to the slurm logs and execution scripts by all Constellation servers. It can be hosted from the “Login node” for small clusters, or can be on a storage server nearby for additional robustness. Typically a shared $HOME directory on a central storage filer is used.

Technology requirements: NFS share
Space requirements: small log files and execution files (<100kb)
Performance requirements: consistent availability to nodes but minimal bandwidth needed. Nodes will execute small binaries from this share and slurm will stream log files.

## 2. Test results and artifacts storage

This is the location where all the (often large) files generated by a simulation get stored. The size depends significantly on the amount of content collected during the simulation and the length of the run time.

Technology requirements: no specific requirement - can be local or cloud
Space requirements: large files - simulation logs and artifacts (video recordings)
Performance requirements:

- Long running copies will take up time on the Constellation node, so a fast storage device is recommended. An additional copy can be run offline (on a CPU node or VM) to move data into the cloud without blocking the Constellation device.
- Review “Storage worst-case scenario” for your environment and make sure your storage device has sufficient performance to handle peak usage.

## 3. Test specification repository

A central location for Scenario Runner specification folder (YAML and Python) files is needed. Either of the two locations already mentioned can be leveraged, or a separate location can be configured.

Technology requirements: no specific requirement - can be local or cloud
Space requirements: small YAML and Python files
Performance requirements: no specific performance requirements

## Storage space estimation

NOTE: The values below are a generalized estimation based on offline analysis of collected data from NVIDIA’s CI/CD runs of DriveSim + DRIVE AV and will vary according to the unique usage and collection patterns of the cluster.

| Number of Constellation nodes | 1 | 8 | 100 |
|---|---|---|---|
| | | | |
| Average data rate (Mbytes/sec) | 1 | 8 | 100 |
| Roadcast data (Mbytes/sec) | 2 | 16 | 200 |
| Video data (Mbytes/sec):<br>- 8 camera 720p compressed video data<br>- 50mbits/sec * 8 cameras = 50Mbyte/node/sec | 50 | 400 | 5000 |
| | | | |
| Current Usage: single camera low resolution | | | |
| Space required for 30 day retention (TB) | 7 TB | 59 TB | 742 TB |
| Retention period for 300TB storage (days) | 1214 days | 152 days | 12 days |
| | | | |
| Future Usage: 8 cam 720p compressed video data | | | |
| Space required for 30 day retention (TB) | 131 TB | 1048 TB | 13101 TB |
| Retention period for 300TB storage (days) | 69 days | 9 days | <1 day |

## Docker Registry and Caching

Large Docker containers orchestrate the simulation and need to be loaded on the nodes prior to beginning the simulation.

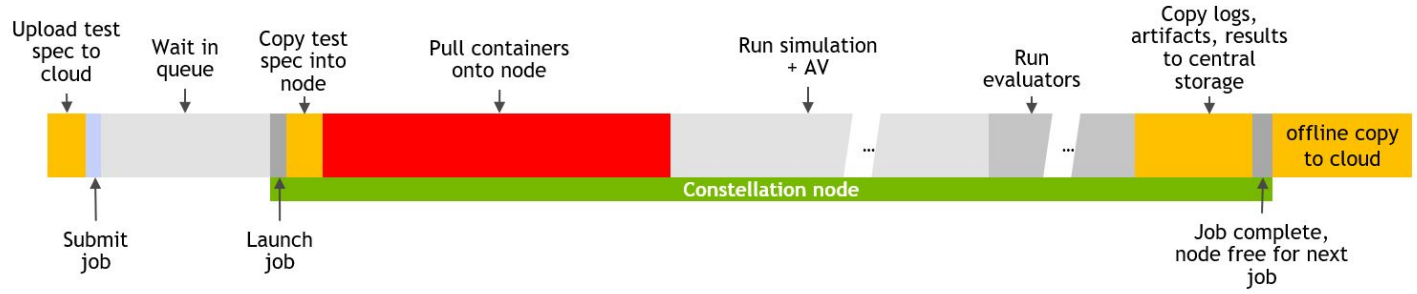
The cluster is not locked down to any single Docker registry. This is because Scenario Runner allows each individual simulation to define (via the YAML manifest) which registry and which version of each container to pull.

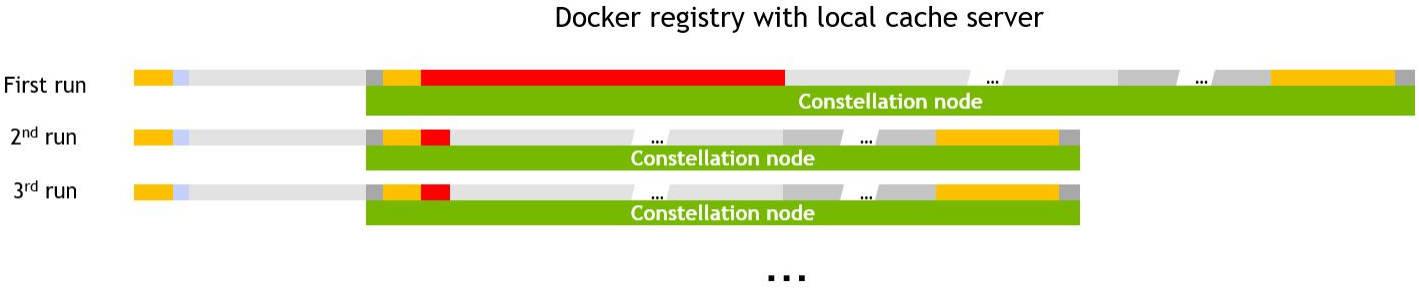
Any user that intends to run this simulation will need to have a credentials.yaml file in their $HOME/.sr directory in order to get access to the container registry.

Advice to speed up container download:

- Registry proximity: a registry server in the same network is ideal if your users are also near the cluster
- Caching proxy: if your main registry is remote (typically due to user location), you can use a cache proxy. The first download will be slow but the following downloads will be fast.

Note the red block in diagrams below where Docker container copying is occurring:




NOTE: pictures for visualization purposes, not actual scale.

# Networking

Internal network (inside the cluster): Recommend minimum 10GbE network inside the cluster for copying generated logs and artifacts from the simulation.

External network: depends on IO requirements (remote storage or remote docker registry)

# Installation

Steps:

1. Prepare for deployment
2. Download Constellation DeepOps package
3. Build inventory file
4. Edit the main playbook
5. Edit configuration
6. Deploy cluster
7. Validation

## Prepare for deployment

### Deployment Parameters

Make sure you are able to fill out the table below. This information will be used to edit the configuration files.

| | |
|---|---|
| Login Node IP and hostname | |
| Constellation nodes IPs hostnames | |
| Deployment username | |
| Validation username | |
| Plan for Slurm shared storage (stores slurm log files and launch wrappers) | *The default configuration creates nfs share hosted on login node in /shared/{{ cluster_name}} - If you customize this then further changes are required in configs* |
| Plan and access information for Central Storage (stores artifacts, logs, videos) | *The default configuration uses RCLONE tool and allows you to map to many storage devices. Make sure your device is compatible with RCLONE.* |
| Docker registry information (for Scenario Runner, DriveSim, and AV containers) | |

## Users

### Deployment user

This user will run the ansible deployment and can be used to update the cluster in the future. Only this user requires elevated permissions and not the regular users submitting jobs.

Requirements:

- **IMPORTANT**: Linux User ID and Group ID must match on ALL nodes before deployment. This is very important otherwise many things will not work.
- SSH access is setup for all the nodes using public keys for automated access
- Unrestricted sudo privilege with no password on **login node** and all **constellation nodes**

Note on user creation: usually in a data center there is a centralized user management solution already in place like . The scope for how to set that up is beyond this document. If you would like to create users that are local to the linux systems (including Constellation nodes) you are welcome to use the DeepOps user management tools, comments in the configuration files will show you how to do so.

### Validation user

This login will be used to run a validation test at the end of the installation. Future users of the system can be configured with matching permissions and setup.

Requirements:

- **IMPORTANT**: Linux User ID and Group ID must match on **login and Constellation nodes** before deployment. This is very important otherwise many things will not work.
- SSH access to **login node**

### Docker Group ID

In order to allow slurm to execute docker commands the docker gid has to match across on **login node** and **Constellation nodes**. (The Docker GID is configurable in group_vars/all.yml file and is by default set to 999)

## Deployment system

This system will be used to run the Ansible scripts and do the deployment.
Requirements:

- Laptop or linux server
- Ubuntu 18.04 or 20.04
- Deployment user with ssh and sudo access to all the nodes

## Login node

This system will be the Slurm controller where all the job scheduling is done.
Requirements:

- Laptop or any linux server installed according to HW requirements (listed earlier in document)
- OS installed Ubuntu 18.04 or 20.04
- Deployment user with ssh and sudo access to all the nodes

## Constellation nodes

The Constellation nodes should be operational as “stand-alone” units, this means latest DriveOS, drivers, and FW are installed.

Requirements:

- One or more Constellation server pre-racked and installed as “stand-alone”
- Latest BaseOS, Drivers, FW
- Deployment user with ssh and sudo access to all the nodes
- Test user for validation test

## Access

Login node and all Constellation nodes should have access to:

- Full network access to each other
- Access to central storage
- Access to the Docker registry with containers for Scenario Runner, DriveSim, and AV

## Download Constellation DeepOps package

On the **deployment node** download the NVIDIA Constellation DeepOps package.

This code is based on NVIDIA DeepOps release 20.11.

## Prepare Provisioning Environment

Run the DeepOps init script that will install the needed versions of pip, ansible, git, and all the Galaxy roles.

cd to the folder with the Constellation DeepOps package and run:

```
$ ./scripts/setup.sh
```

NOTES:

- This script will run apt commands so the command “sudo apt update” needs to be successful for this script to work.
- This script also needs access to the internet to download various Ansible galaxy roles.

## Build an inventory file

Edit the inventory: ./config/inventory

- Set the access information for each host
- Make sure they are in the slurm-master and slurm-node groups
- Add the deployment user credentials (either via key or password)

```
######
# ALL NODES
######
[all]
login-server    ansible_host=10.0.1.1
const-01        ansible_host=10.0.2.1
const-02        ansible_host=10.0.2.2

######
# SLURM
######

[slurm-master]
login-server
```

```
[slurm-node]
const-01
const-02

[slurm-cluster:children]
slurm-master
slurm-node

######
# SSH connection configuration
#####
# Using text password
[all:vars]
ansible_user=myuser
```

ansible_password=mypassword

```
# Using keys
#[all:vars]
#ansible_user=myuser
#ansible_ssh_private_key_file=/path/to/key.pem
```

To test that you are able to connect to every host in the inventory run:

```
$ ansible all -m ping
```

NOTE: Make sure to run all ansible commands from root of the DeepOps folder so that it picks ups the default ansible configuration and is able to find the inventory and configs.

## Edit the main playbook

IMPORTANT: please make sure that you understand the Ansible playbook before executing as some changes are not easy to revert. It is best that this deployment is executed by someone with Ansible expertise.

The main playbook is self-documented.  Please review this file and make necessary changes to accommodate for your installation needs.

./playbooks/nvidia-sr-cluster.yml

The file is divided into sections, each section has a tag allowing the install to happen in steps, or to rerun a section individually.

1. preinstall: The first section deals with preparing the environment. It is here to help establish connectivity across nodes and user access. If all this is already setup in your environment you may skip or replace any of these playbooks as appropriate.

2. docker: installs docker-ce on the login node, docker-compose on the constellation nodes, and validates that the docker group matches  constellation

3. nfs: Configures NFS shares to be used for slurm log files and scripts central location. The exports are configured in ./config/group_vars/all.yml in the NFS section.

4. slurm: installs Slurm. Before executing the playbook please make sure to look over the configuration section and update the slurm configuration files in ./config/group_vars.

   Note on disabled playbooks:
   - Node health check: this is disabled since it is a very simple check designed for a standard Linux server. However it is left as an example if you chose to explore and expand your design.
   - Open OnDemand: this is a web UI that allows launching of slurm commands from a web browser. This is disabled in this version since the default flow is designed to work with a standard GPU accelerated HPC environment.

5. rclone: Install RCLONE and set up the Central Storage location where all the logs will be copied to. Make sure to review the rclone configuration in ./config/group_vars/all.yml. By default it is configured to use an S3 bucket from AWS, but rclone can be configured to use various storage sources (https://rclone.org/)

6. wrapper-scripts: Copy the wrapper scripts that will be used to launch Scenario Runner and prepares any additional needs for SR execution.

## Edit group_vars configuration files

Because all the nodes are in the slurm-cluster group, a variable definition can exist in any file, however for simplicity the slurm specific variables are in slurm-cluster.yml and the rest in all.yml

1. config/group_vars/all.yml: contains variables for all the playbooks. Most is self explanatory with comments indicating the playbook they belong to.

   It's important to review this file and adjust according to your environment.

2. config/group_vars/slurm-cluster.yml:
   IMPORTANT: please copy the file from the nvidia-sr-cluster role before running the playbook:

   ```
   $ cp roles/nvidia-sr-cluster/files/slurm-cluster-for-sr-cluster-role.yml
   config/group_vars/slurm-cluster.yml
   ```

   Mostly nothing needs to change here except the slurm passwords.

   If you would like to modify the slurm configuration then make sure to edit both files:
   1. config/group_vars/slurm-cluster.yml (after copying from role)
   2. roles/nvidia-sr-cluster/templates/slurm.conf

## About Slurm Configuration

Slurm is configured to manage a node as a single unit instead of a bundle of resources. The file slurm.conf is modified from it's original (roles/slurm/templates/etc/slurm/slurm.conf) to:

```
# Disable cgroups and resource affinity
TaskPlugin=task/none

# Disable backfill scheduling
SchedulerType=sched/builtin

# Use FIFO linear scheduling alorithm
SelectType=select/linear

# Treat each system as a unit
SelectTypeParameters=CR_ONE_TASK_PER_CORE

# Avoid using cgroups
JobAcctGatherType=jobacct_gather/linux
```

## About scripts folder

A few simple wrapper scripts are provided in roles/nvidia-sr-cluster/scripts. The goal of these scripts is to provide a thin wrapper on top of launching Scenario Runner and to do the collection of the logs. They are intentionally trim and simple so that they can be easy to understand and expand on as needed.

More information on how to use these scripts in the “Using the cluster” section.

## Deploy cluster

**IMPORTANT:** Make sure to review the pre-install requirements, the main playbook, and all the configs in configs/group_vars before continuing.

Logged into the deployment system as the deployment user run the command below to launch ansible:

```
$ ansible-playbook -l slurm-cluster playbooks/nvidia-constellation-cluster.yml
```

# Validation (SIL MODE)

NOTE: For this release, only SIL mode is supported.

## Prepare user environment for test

First we need to prepare the validation user environment with all the credentials. Many commands can be run from the deployment system using ansible ad-hock command line for simplicity.

**IMPORTANT NOTE:** If this user does not have a shared $HOME directory across the nodes then some of these steps (like Docker login, and SR credentials) will have to be done individually for every node. That is because Docker and SR credentials are stored in $HOME folder.

1. Fist add the user to the docker group on all systems (including the login node) (https://docs.docker.com/engine/install/linux-postinstall/)

   # helper ansible command (replace VALIDATION_USER for actual username)
   `$ ansible all -m shell -a "usermod -aG docker <validation_user>" -vvv --become`

2. Test slurm back of simple docker command. Here is an example:

```
$ ssh <validation_user>@<login_node>
$ cd $SR_CLUSTER_ROOT_DIR
$ sbatch scripts/test_docker.sh
Submitted batch job <JOB_ID>
$ cat slurm-<JOB_ID>.out
Hello from Docker!
This message shows that your installation appears to be working
correctly.
...
```

3. Log user into Docker registries on Constellation nodes

   The user needs access to the Docker registries where Scenario Runner, DriveSim, and AV are residing (docker credentials are saved in $HOME/.docker, usually the $HOME directory is shared and credentials only need to be created once, however if your $HOME directories are not shared you will need to login to Docker registries on each node)

   To view which docker registries are needed by the validation test look inside the manifest file here: config/scripts/hil_verified_0.1.11/manifest.yaml

4. Create Scenario Runner credentials file

   From Scenario Runner documentation:

   *In your home directory create directory **.sr** and place **credentials.yaml** with the entries for different docker registries. Credentials file should look like below. Please enter corresponding registry, username and password token.*

   *Please contact your NVIDIA representative for credentials.*

   ```
   CREDENTIALS:
     - registry: 'nvcr.io/<repository>'
       username: '$oauthtoken'
       password: '<token>'
   ```

5. Copy ScenarioRunner test definition into the Central Storage. This will allow the scripts to pull the specification from the **central storage** location for testing.

   ```
   # Enter the login node
   $ ssh <validation_user>@<login_node>
   $ cd $SR_CLUSTER_ROOT_DIR

   # Load cluster environment varilables
   $ source constellation_cluster.env

   # Copy the test directory into central storage location
   $ rclone copy scripts/sil_verified_0.1.11
$SR_CLUSTER_TEST_SPEC_PATH/sil_verified_0.1.11

   # Verify the folder was copied by listing the directories
   $ rclone lsd $SR_CLUSTER_TEST_SPEC_PATH
   ```

   Note: the test specification folder can also be generated with Scenario Runner. Please refer to Scenario Runner documentation for more information.

6. Start x11vnc on constellation nodes

   You can do this your own way or use the helper script provided here: $SR_CLUSTER_ROOT_DIR/scripts/start_x11vnc.sh which you can launch with ansible **from the deployment system**:

```
# The start_x11.sh script sets DISPLAY to 0.0 runs “xhost +” and launches x11vnc storing log to ~/vnc.log
$  ansible slurm-node -m shell -a "\$SR_CLUSTER_ROOT_DIR/scripts/start_x11vnc.sh" --become --become-user <validation_user> -vvvv
```

7. Initiate vcan interface: this is required since we are running ScenarioRunner in SIL mode.

   On each Constellation node run:

```
$ sudo modprobe vcan
$ sudo ip link add dev vcan0 type vcan
$ sudo ip link set up vcan0
```

## Run validation test

```
# Enter login node
$ ssh validation_user@login_node
$ cd $SR_CLUSTER_ROOT_DIR

# Check that the nodes are showing up in the cluster (and idle)
$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
batch*     up   infinite    2   idle cc-node[1-2]

# Launch scenario runner on all nodes
# To select all nodes we will use sbatch --array parameter
# The --array parameter takes a range starting with 0:
#    for example “--array 0-1” will launch 2 jobs
#    and “--array 0-5” will launch 6 jobs
# Launch as many jobs as there are nodes for the validation
$ sbatch --array 0-1 scripts/run_sr.sh sil_verified_0.1.11
Submitted batch job <JOBID>

# To watch log unfolding
$ tail -f slurm-<JOBID>.out
…

# The results will be copied out to the cloud folder
$SR_CLUSTER_TEST_OUTPUT_PATH with the name of the job ID and the test folder name

$ rclone lsd SR_CLUSTER_TEST_OUTPUT_PATH
```

# Using slurm cluster

## Slurm commands

To use the cluster login to the “Login node” and use SLURM commands to manage cluster:

- sinfo: view information about Slurm nodes and partitions
- squeue: view information about jobs located in the Slurm scheduling queue
- sbatch: Submit a script for Slurm to launch on an available node

## Provided utility scripts

The scripts in $SR_CLUSTER_ROOT_DIR/scripts are simple wrapper scripts that are executed on the Constellation node by Slurm. Not to be run on the login node directly but to be queued with sbatch from the login node. These scripts are simple by design so they can be expanded or replaced. They read environment variables from $SR_CLUSTER_ROOT_DIR/constellation_cluster.env

Here is an example of that file:

```
export SR_CLUSTER_NAME=cc-batch-cluster-name
export SR_CLUSTER_TEST_SPEC_PATH=cc-batch-cluster-name:cc-batch-cluster-name/test_spec
export SR_CLUSTER_TEST_OUTPUT_PATH=cc-batch-cluster-name:cc-batch-cluster-name/test_output
export SR_TMP_DIR=/tmp/sr
export SR_CLUSTER_SHARE_PATH=/shared/cc-batch-cluster-name
export SR_DOCKER_PATH=nvcr.io/drive-apollo-t/drive-sim/drive_sr
export SR_DEFAULT_VERSION=0.1.11
```

- get_sr.sh [SR_version]
  users SR_DOCKER_PATH to pull Scenario Runner container and then extracts the sr binary.

- run_sr.sh <test_folder_name> [SR_version]
  This script is the main wrapper that manages the SR run. Here is a list of operations:
  1. Run get_sr.sh to make sure we have sr binary
  2. Copy test definition from the central storage location (`SR_CLUSTER_TEST_SPEC_PATH`) into /tmp
  3. Set DISPLAY to 0.0
  4. Launches Scenario Runner pointing to the test folder and capture the output
  5. When Scenario Runner is done copy the contents out to central storage (`SR_CLUSTER_TEST_OUTPUT_PATH`)

## Examples

### Copy test folder to central storage using rclone

```
# Enter login node and load cluster environment variables
$ ssh validation_user@login_node
$ cd $SR_CLUSTER_ROOT_DIR
$ source constellation_cluster.env

# Copy the test directory into central storage location
$ rclone copy path_to_test_folder $SR_CLUSTER_TEST_SPEC_PATH/test_name

# Verify the folder was copied by listing the directories
$ rclone lsd $SR_CLUSTER_TEST_SPEC_PATH
```

### Running a Scenario Runner job

```
$ ssh validation_user@login_node
$ cd $SR_CLUSTER_ROOT_DIR
$ sbatch scripts/run_sr.sh test_folder_name
Submitted batch job 11

# To watch Slurm log output stream (change ID to match job ID)
$ tail -f slurm-11.out
...
```