Hewlett Packard Enterprise

HPE Swarm Learning User Guide

Part Number: 10-191040-Q224
Published: February 2024
Edition: 5

Revision history

Part number	Publication date	Edition	Summary of changes
10-191040-Q222	April 2022	1	New manual
10-191040-Q322	July 2022	2	Content related to Swarm Learning 1.1.0 is updated: • **How to Swarm enable an ML algorithm** section updated for default values for `useAdaptiveSync` and `mlPlatform`. • **Swarm Learning component interactions** section updated. • **Swarm Learning Command Interface** section updated for `WAIT FOR TASKRUNNER` and `SLEEP`. • **SWCI APIs** section updated for `sleep ()` and an example snippet is added at the end of the section for `Swci()`. • **Examples** section updated for all four examples. • **Frequently asked questions** section updated with two new FAQ's.

Table Continued

Part number	Publication date	Edition	Summary of changes
10-191040-Q223	April 2023	3	Content related to Swarm Learning 2.0.0 is updated: • Added the following sections: ◦ **Swarm Learning Component Interactions using Reverse Proxy** ◦ **SWOP profile schema example using reverse proxy** ◦ **Examples using reverse proxy** ◦ **MNIST using Reverse Proxy** ◦ **CIFAR-10 using Reverse Proxy** ◦ **Swarm Learning Log Collector** ◦ **Swarm Learning Examples** ◦ **System setup for the examples** ◦ **Taskrunner Framework** • **Working of a Swarm Learning node** section updated for Leader Failure Detection and Recovery feature. • **Swarm Learning Command Interface** section updated for `CREATE CONTEXT`, `ASSIGN TASK` and `LIST NODES`. • **SWCI API methods** section updated for `createContext()`. • **SWOP profile schema** section updated. • **Launch Swarm Learning using SWOP** section updated for DNS. • **Troubleshooting** section updated for `Unable to contact API-Server`. • **Frequently asked questions** section updated for AMD GPUs, Reverse Proxy, Guidance on `minPeers` vs `WITH peersNeeded` and Unable to contact API server.

Table Continued

Part number	Publication date	Edition	Summary of changes
10-191040-Q323	October 2023	4	Content related to Swarm Learning 2.1.0 is updated: • **Swarm Learning architecture** section updated for Persist data in the Blockchain and Swarm Learning Management UI (SLM-UI). • **Frequently asked questions** section updated for Persist data in the Blockchain, `mergeMethod`, License related issues and SLM-UI. • **Working of a Swarm Learning node** section updated for `mergeMethod`. • **How to Swarm enable an ML algorithm** section updated for `adsValData`, `adsValBatchSize` and `mergeMethod`. • **Troubleshooting** section updated for `mergeMethod`, `API server is down` and `Failed to start thread "GC Thread#0" - pthread_create failed`. • **Running the MNIST example**, **CPU based MNIST-PYT**, **AMD GPU based MNIST-PYT**, **NVIDIA GPU based MNIST-PYT**, **Running the credit card fraud detection example** sections updated with new `docker cp` command. • **How to Swarm enable an ML algorithm** section updated with new parameters. • **CPU based MNIST-PYT**, **AMD GPU based MNIST-PYT**, and **NVIDIA GPU based MNIST-PYT** sections updated with new folder path. • Added the following section: ◦ **Monitoring Swarm Learning training using SLM-UI** ◦ **Running MNIST example using SLM-UI** • **Swarm Learning Command Interface** updated with `PERFDATA <SLContractName:string>`. • **SWCI APIs** updated with `getTrainingContractPerformanceData`. • **MNIST using Reverse Proxy** and **CIFAR-10 using Reverse Proxy** sections updated with new run commands.
10-191040-Q224	February 2024	5	Content related to Swarm Learning 2.2.0 is updated: • Added the following sections:

Part number	Publication date	Edition	Summary of changes
			◦ **Experiment tracking** ◦ **CIFAR-10 using SPIRE** ◦ **National Institutes of Health Chest X-Ray Dataset (NIH)** • **Swarm Learning Command Interface** section updated for `PERFDATA <SLContractName:string>`, `ASSIGN TASK <taskName : string> TO <trName : string> WITH ALL PEERS`, and `ASSIGN TASK <taskName> TO <taskRunnerName> TARGET <SWOP-UID>`. • **SWCI API** updated with example snippet of API. • **Launch Swarm Learning using SWOP** section updated with new note. • **Examples** and **Swarm Learning Examples** sections updated with SPIRE feature. • **Examples using reverse proxy** section updated with new content. • **Troubleshooting** section updated with `sudo groupadd docker`, `Emulate Docker CLI using podman` and `rootlessport cannot expose privileged port 80`. • **Frequently asked questions** section updated with new quiz.

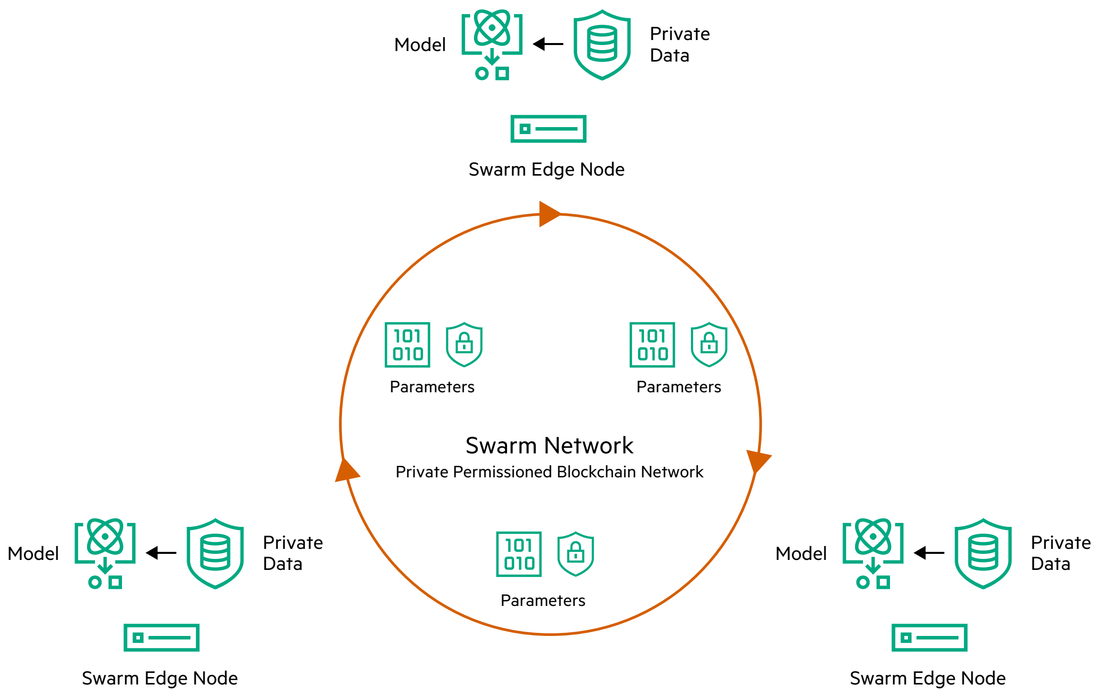
Contents

Introduction to HPE Swarm Learning

HPE Swarm Learning extends the concept of federated learning to decentralized learning by adding functionality that obviates the need for a central leader. It combines the use of AI, edge computing, and blockchain.

HPE Swarm Learning is a decentralized, privacy-preserving Machine Learning (ML) framework. Swarm Learning framework uses the computing power at, or near, the distributed data sources to run the ML algorithms that train the models. It uses the security of a blockchain platform to share learnings with peers safely and securely. In Swarm Learning, training of the model occurs at the edge, where data is most recent, and where prompt, data-driven decisions are mostly necessary. In this decentralized architecture, only the insights learned are shared with the collaborating ML peers, not the raw data. This tremendously enhances data security and privacy.

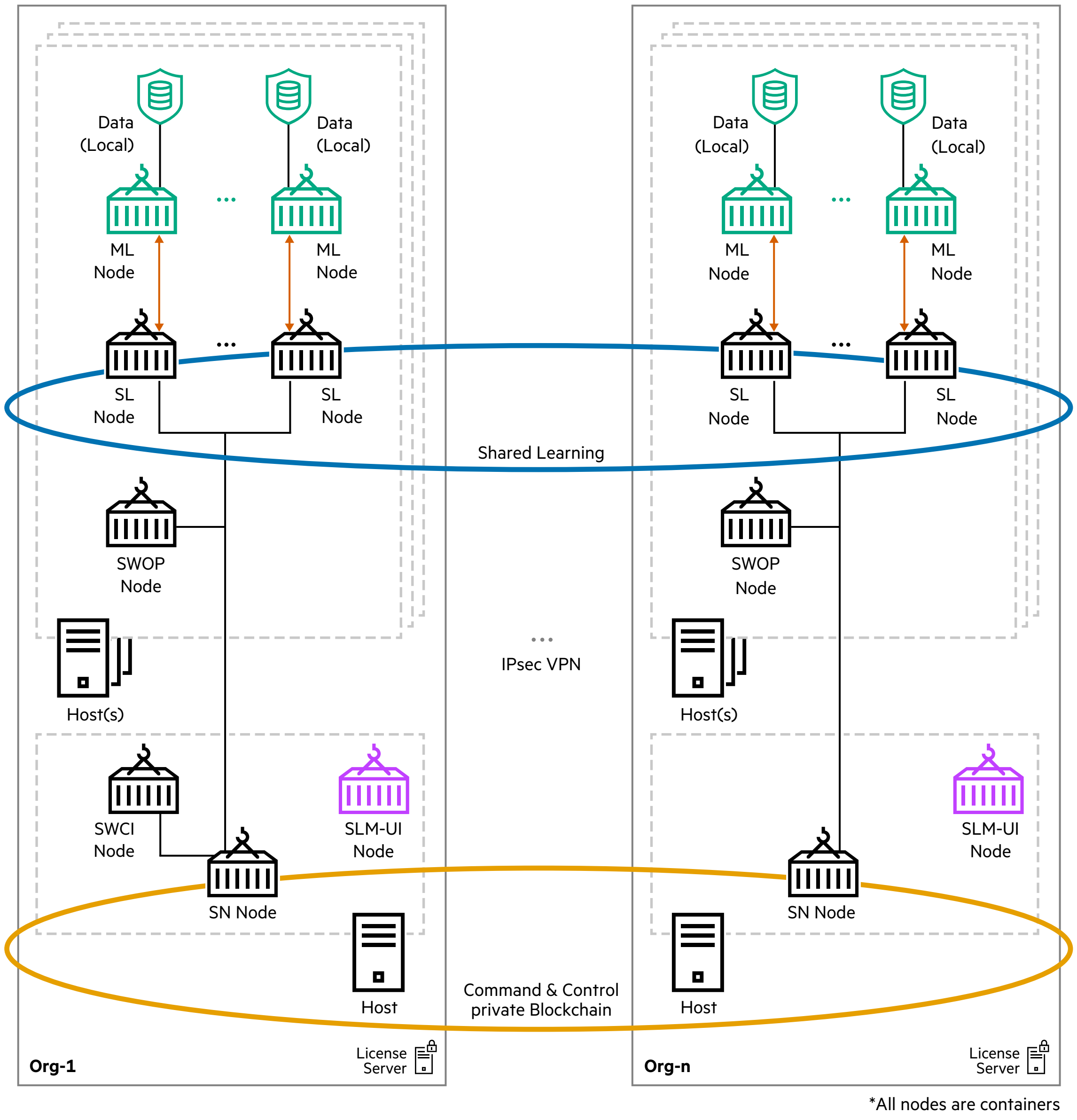
The following image provides an overview of the Swarm Learning framework. Only the model parameters (learnings) are exchanged between the various edge nodes, and not the raw data. This ensures that the privacy of data is preserved.



Swarm Learning architecture

Swarm Learning is made up of various components known as nodes, such as Swarm Learning (SL) nodes, Swarm Network (SN) nodes, Swarm Learning Command Interface (SWCI) nodes, and Swarm Operator (SWOP) nodes. Each node of SL is modularized and runs in a separate container.

The following illustration is the architecture of Swarm Learning components:



The nodes in this illustration represent different Swarm Learning functionality rather than physical server nodes.

For more information on how these components interact, see **Swarm Learning component interactions**.

- SL nodes run the core of Swarm Learning. An SL node works in collaboration with all the other SL nodes in the network. It regularly shares its learnings with the other nodes and incorporates their insights. SL nodes act as an interface between the user model application and other Swarm Learning components. SL nodes take care of distributing and merging model weights in a secured way.

- SN nodes form the blockchain network. The current version of Swarm Learning uses an open-source version of Ethereum as the underlying blockchain platform. The SN nodes interact with each other using this blockchain platform to maintain and track progress. The SN nodes use this state and progress information to co-ordinate the working of the other swarm learning components. Blockchain can be persisted across SN restart to preserve the past training history. User can lookup blockchain and see all history of operations. Users have the flexibility to stop Swarm after training is completed. Once user restarts the SN network, the existing history can be accessed using the `get` or `list` command of SWCI management interface.

  Sentinel Node is a special SN node. The Sentinel node is responsible for initializing the blockchain network. This is the first node to start.

  NOTE: Only metadata is written to the blockchain. The model itself is not stored in the blockchain.

- SWCI node is the command interface tool to the Swarm Learning framework. It is used to monitor the Swarm Learning framework. SWCI nodes can connect to any of the SN nodes in a given Swarm Learning framework to manage the framework. For more information on SWCI, see **Swarm Learning Command Interface**.

- SWOP node is an agent that can manage Swarm Learning operations. SWOP is responsible to execute tasks that are assigned to it. A SWOP node can execute only one task at a time. SWOP helps in executing tasks such as starting and stopping Swarm runs, building and upgrading ML containers, and sharing models for training. For more information about SWOP, see **Swarm Operator node (SWOP)**.

- Swarm Learning security and digital identity aspects are handled by X.509 certificates. Communication among Swarm Learning components are secured using X.509 certificates. User can either generate their own certificates or directly use certificates generated by any Standard Security software such as SPIRE. For more information on SPIRE, see **https://thebottomturtle.io/Solving-the-bottom-turtle-SPIFFE-SPIRE-Book.pdf** and **https://spiffe.io/**.

  NOTE: Swarm Learning framework does not initialize if certificates are not provided.

- Swarm Learning components communicate with each other using a set of TCP/IP ports.

  NOTE: The participating nodes must be able to access each other's ports.

  For more information on port details that must be opened, see *Exposed Ports* in *HPE Swarm Learning Installation and Configuration Guide*.

- License Server installs and manages the license that is required to run the Swarm Learning framework. The licenses are managed by the AutoPass License Server (APLS) that runs on a separate node. For more information on APLS, see *HPE Swarm Learning Installation and Configuration Guide*.

User ML components

User can transform any Keras or PyTorch based ML program that is written using Python3 into a Swarm Learning ML program by making a few simple changes to the model training code by including the `SwarmCallback` API. For more information, see the examples included with the Swarm Learning package for a sample code.

The transformed user Machine Learning (user ML node) program can be run on the host or user can build it as a Docker container.

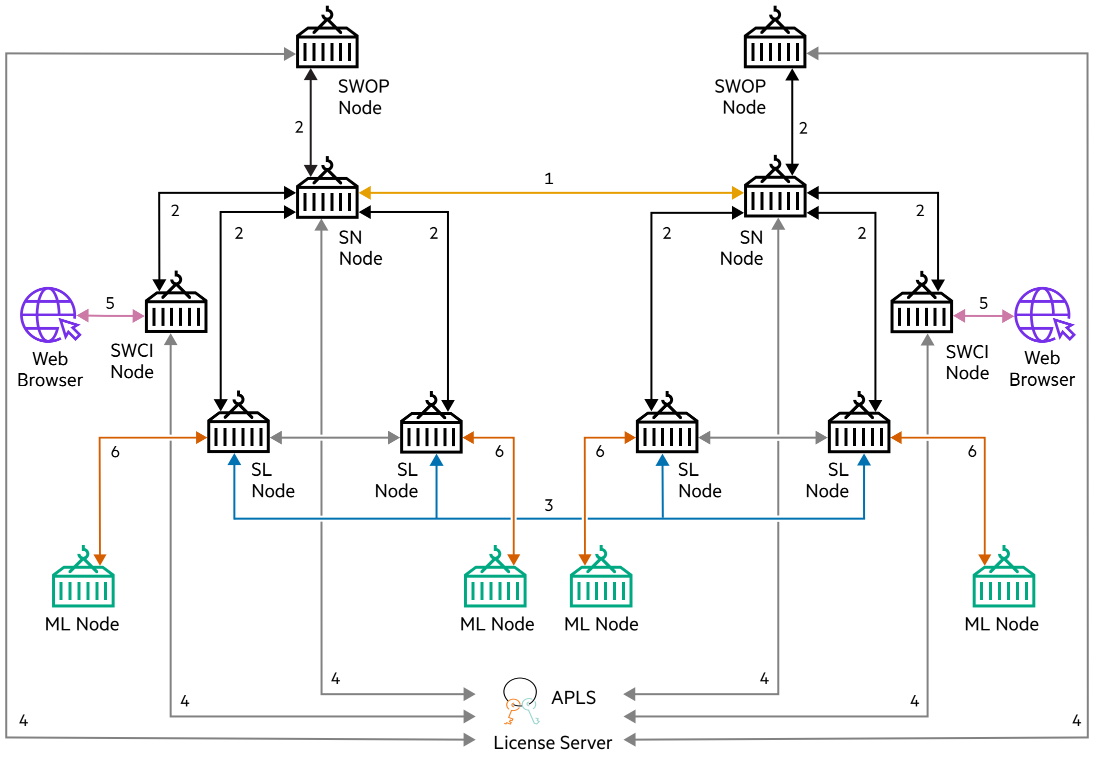
NOTE: HPE recommends users to build an ML Docker container.

The ML node is responsible to train and iteratively update the model. For each ML node, there is a corresponding SL node in the Swarm Learning framework, which performs the Swarm training. Each pair of ML and SL nodes must run on the same host. This process continues until the SL nodes train the model to the desired state.

NOTE: All the ML nodes must use the same ML platform either Keras (based on TensorFlow 2 backend) or PyTorch. Using Keras for some of the nodes and PyTorch for the other nodes is not supported.

Swarm Learning component interactions

The Swarm nodes interact with each other in many ways using network ports that are dedicated for each purpose:



Callout	Description
1 **(SN Peer-to-Peer Port)**	This port is used by each SN node to share blockchain internal state information with the other SN nodes. The default value of this port is 30303.
2 **(SN API Port)**	This API server is used by the SL nodes to send and receive state information from the SN node that they are registered with. It is also used by SWCI and SWOP nodes to manage and view the status of the Swarm Learning framework. The default value of this port is 30304.
3 **(SL File Server Port)**	This port is used by each SL node to run a file server. This file server is used to share insights learned from training the model with the other SL nodes in the network. The default value of this port is 30305.
4 **(License Server API Port)**	This port is used by the License Server node to run a REST-based API server and a management interface. The API server is used by the SN, SL, SWOP, and SWCI nodes to connect to the License Server and acquire licenses. The management interface is used by Swarm Learning platform administrators to connect to the License Server from browsers and administer licenses. The default value of this port is 5814.

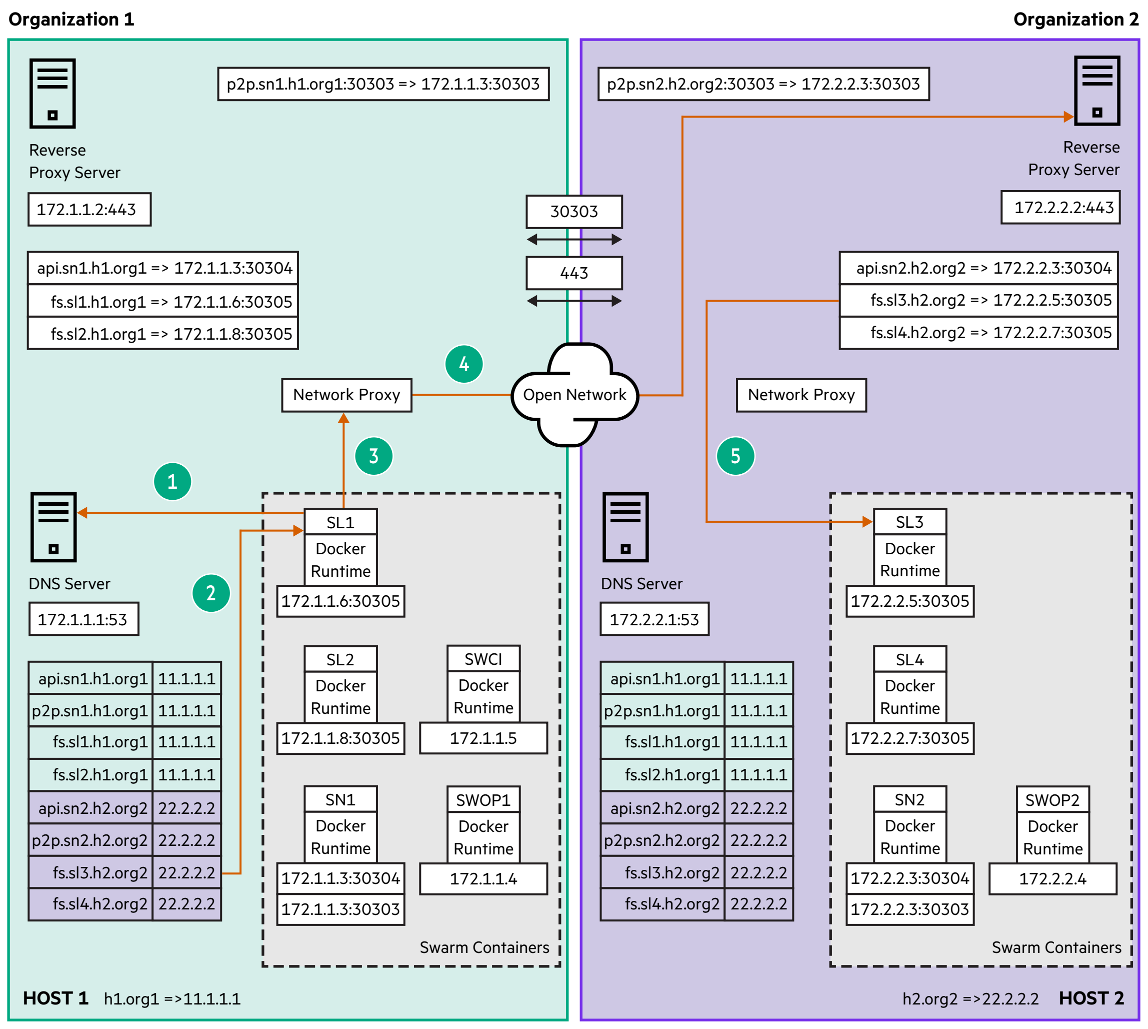
Table Continued

Callout	Description
5 (**SWCI API server port**)	This port is used by the SWCI node to optionally run a REST-based API service. This SWCI API service can be used to control and manage the Swarm Learning framework from a program by using the library provided in the wheels package. The default value of this port is 30306.
6 (**SL_REQUEST_CHANNEL and SL_RESPONSE_CHANNEL**)	These named pipes (FIFO) are used between each pair of ML and SL nodes for the communication.

Swarm Learning Component Interactions using Reverse Proxy

A Reverse Proxy Server is a type of proxy server that presents behind the firewall in a private network and directs client requests to appropriate backend or origin servers. Reverse Proxy for Swarm Learning is used to minimize the number of network ports that must be opened for communication between various host machines. An Open port refers to a TCP or UDP port number that is configured to accept packets.

The following illustration is the multi-organization setup using reverse proxy:



Organization 1
Organization 2
p2p.sn1.h1.org1:30303 => 172.1.1.3:30303
p2p.sn2.h2.org2:30303 => 172.2.2.3:30303
Reverse Proxy Server
172.1.1.2:443
Reverse Proxy Server
172.2.2.2:443
30303
443
api.sn1.h1.org1 => 172.1.1.3:30304
fs.sl1.h1.org1 => 172.1.1.6:30305
fs.sl2.h1.org1 => 172.1.1.8:30305
api.sn2.h2.org2 => 172.2.2.3:30304
fs.sl3.h2.org2 => 172.2.2.5:30305
fs.sl4.h2.org2 => 172.2.2.7:30305
4
Network Proxy
Open Network
Network Proxy
3
5
1
2
DNS Server
172.1.1.1:53
DNS Server
172.2.2.1:53
api.sn1.h1.org1 11.1.1.1
p2p.sn1.h1.org1 11.1.1.1
fs.sl1.h1.org1 11.1.1.1
fs.sl2.h1.org1 11.1.1.1
api.sn2.h2.org2 22.2.2.2
p2p.sn2.h2.org2 22.2.2.2
fs.sl3.h2.org2 22.2.2.2
fs.sl4.h2.org2 22.2.2.2
SL1 Docker Runtime 172.1.1.6:30305
SL2 Docker Runtime 172.1.1.8:30305
SWCI Docker Runtime 172.1.1.5
SN1 Docker Runtime 172.1.1.3:30304 172.1.1.3:30303
SWOP1 Docker Runtime 172.1.1.4
SL3 Docker Runtime 172.2.2.5:30305
SL4 Docker Runtime 172.2.2.7:30305
SN2 Docker Runtime 172.2.2.3:30304 172.2.2.3:30303
SWOP2 Docker Runtime 172.2.2.4
Swarm Containers
Swarm Containers
HOST 1 h1.org1 =>11.1.1.1
h2.org2 =>22.2.2.2 HOST 2

Orange line path shows the SL-FS communication between SL1 to SL3.

1. SL1 does a DNS look up for fs.sl3.h2.org2.
2. DNS resolves to 22.2.2.2.
3. SL1 sends request to 22.2.2.2:443 via network proxy.
4. SL1 forwards request to reverse proxy of org2.
5. Request reaches to SL3 using 172.2.2.3:30305.

Swarm Learning currently communicates between host machines by enabling (aka opening) a different set of pre-defined ports on each machine. For example, SN-API-PORT, SL-FS-PORT, and SN-P2P-PORT. Enabling all of these ports between all host machines may be a time-consuming process. Moreover, opening many ports may make the system more vulnerable to security attacks.

SN serves as the hub for all major communications among the Swarm nodes. Using a Reverse Proxy will tunnel all port-based communications through a single default HTTPS port of 443. The SN P2P service needs port 30303 to be opened for each SN node. All other communication between the swarm components flows via the reverse proxy, thus rather than externalizing ports, we create services and manage them with unique Fully Qualified Domain Names (FQDN). The routing mechanism is managed through a reverse proxy (for example, NGINX). The reverse proxy approach will avoid the need of opening up multiple SN-API-PORT & SN-P2P-PORTs.

NOTE: Typically Reverse proxy runs on the default HTTPS port of 443 which would be opened up in most organizations. If its configured to run on any other port, then that port must be opened up.

Table 1:

SN API Service	This API service (aka Fully Qualified Domain Name (FQDN)) is used by the SL nodes to send and receive state information from the SN node that they are registered with. It is also used by SWCI and SWOP nodes to manage and view the status of the Swarm Learning Framework. This parameter is used to tunnel the requests via reverse proxy to the API service using an HTTPS port instead of an SN API port. Either IP along with the SN API port or this SN API service is required. (Default port used along with this FQDN is 443).
SN P2P Service	This P2P service (aka FQDN) is used by the each SN node to share blockchain internal state information with the other SN nodes. This FQDN parameter is used as an alias for the SN P2P port based ethereum communications via reverse proxy. Either SN node's IP & P2P port or this SN P2P Service is required. (Default port used along with this FQDN is 30303).
SL FS Service	This File Server (FS) service (aka FQDN) is used by each SL node to run a file server. This FS service is used to share insights learned from training the model with the other SL nodes in the network. This parameter is used to tunnel the requests via reverse proxy to the File server using HTTPS port instead of SL FS Port. Either SL node's IP and FS port or this SL FS Service is required. (Default port used along with this FQDN is 443).

NOTE: SN P2P Service still uses 30303 port.

SN P2P service uses 30303 port and needs to be opened for each SN node.

Swarm client interface - wheels package

The Swarm Learning client interface is provided as a python wheels package. It consists of Swarm Callback API and SWCI API.

- Swarm Callback API - Python3 APIs for the following ML platforms - PyTorch and Keras (based on TensorFlow 2 backend). Swarm Callback is a custom callback class that consists of a set of functions that can be applied at various stages of the ML training process. For more information on Swarm Callback API usage, see **How to Swarm enable an ML algorithm**.
- SWCI API - A Python3 SWCI API to perform the SWCI operations programmatically. For more information, see **SWCI APIs**.

Swarm Learning concepts

This section provides information about key Swarm Learning concepts. The subsequent section describes working of a Swarm Learning node, how to Swarm enable an ML algorithm, and interactions with SL, contexts, training contract, Taskrunner contract, and task.

SWCI is the command interface tool to the Swarm Learning framework. It is used to view the status, control, and manage the Swarm Learning framework. SWCI manages the Swarm Learning framework using contexts and contracts. For more information about SWCI, see **Swarm Learning Command Interface**.

An SWCI context is a string identifier. It identifies an SWCI command environment and has artifacts (API server IP, port, environment variables, and versions) that are used to execute SWCI commands. SWCI can have only one active context at any given time even if multiple contexts are created. For more information related to context commands, see **SWCI commands related to context**.

Swarm Learning training contracts are used to control the swarm learning training process. It is an instance of Ethereum smart contract. It is deployed into the blockchain and registered into Swarm Learning Network using the `CREATE CONTRACT` command. For more information related to contract commands, see **SWCI commands related to contract**.

NOTE: When a contract is created, it is permanent and cannot be deleted.

SWOP is the central component of the Taskrunner framework. It is packaged as a SWOP container. Taskrunner framework is a decentralized task management framework. For more information about SWOP, see **Swarm Operator node (SWOP)**.

A Task is a well-defined unit of work, that can be assigned to a Taskrunner. Tasks are instantiated according to the schema specified in the Task Schema YAML file. For more information related to Task commands, see **SWCI commands related to Task**.

A Taskrunner is an instance of Ethereum smart contract used to coordinate execution of task by SWOP nodes. For more information related to Taskrunner commands, see **SWCI commands related to Taskrunner**.

Working of a Swarm Learning node

1. SL node starts by acquiring a digital identity, which can be either user provided CERTS or SPIRE certificates.
2. SL node acquires a license to run.
3. SL node registers itself with an SN node.
4. SL node starts a file server and announces to the SN node that it is ready to run the training program.
5. Waits for the User ML component to start the ML model training.

SL node works in collaboration with all the other SL nodes in the network. It regularly shares its learnings with the other nodes and incorporates their insights. Users can control the periodicity of this sharing by defining a **Sync**hronization **Frequency**(from now on referred to as, *sync frequency*.) This frequency specifies the number of training batches after which the nodes share their learnings.

NOTE: Specifying a large value reduces the rate of synchronization and a small value increases it. Frequent synchronization slows down the training process while infrequent ones may reduce the accuracy of the final model. Therefore, the *sync frequency* must be treated as a *hyperparameter* and chosen with some caution.

Swarm Learning can automatically control the *sync frequency*, this feature is called **Adaptive Sync**hronization **Frequency**. This feature judges the training progress by monitoring the mean loss. A reduction in the mean loss infers that the training is progressing well. As a response, it increases the *sync frequency* and enables more batches to run before sharing the learnings. This makes the training run faster. In contrast, when the loss does not improve, **Adaptive Sync** frequency reduces the *sync frequency* and synchronizes the models more frequently.

At the end of every *sync frequency*, when it is time to share the learning from the individual model, one of the SL nodes is designated as "leader". This leader node collects the individual models from each peer node and merges them into a single model by combining parameters of all the individuals.

Merge Methods are one of the core operations in Swarm Learning that ensures learning is shared across SL nodes. Swarm Learning provides three different merge options via Swarm callback. The merge options are 'mean', 'coordmedian' and 'geomedian'. User can pass one of these options via `mergeMethod` parameter in the Swarm callback. This is an optional parameter for the callback. These options will consider the weightages of individual SL nodes as well. All nodes that have participated in the sync round will receive aggregated model parameters for the next sync round.

The 'mean' option aggregates intermediate model parameters using a weighted mean method (also known as weighted federative averaging). This method sums up all the intermediate model parameters in an iterative way and at the end of the merge method, it divides with sum of weightages. This is the default merge method in Swarm.

The 'coordmedian' option finds the weighted coordinate median of the intermediate model parameters. These parameters from all the nodes that are participating in the given merge round will get transformed and transposed before calculating the coordinate median. This merge method finds the 50th percentile and tries to converge along with the model parameters over the sync cycles.

The 'geomedian' option finds the weighted geometric median of the intermediate model parameters. It uses Weiszfeld's algorithm which is an iterative method to calculate the geometric median. It starts with model parameters average as an initial estimate and iterate to find a better estimate. This process is repeated until the difference between new estimate and old estimate reaches to a threshold value.

Leader Failure Detection and Recovery (LFDR) feature enables SL nodes to continue Swarm training during merging process when an SL leader node fails. A new SL leader node is selected to continue the merging process. If the failed SL leader node comes back after the new SL leader node is in action, the failed SL leader node is treated as a normal SL node and contributes its learning to the swarm global model.

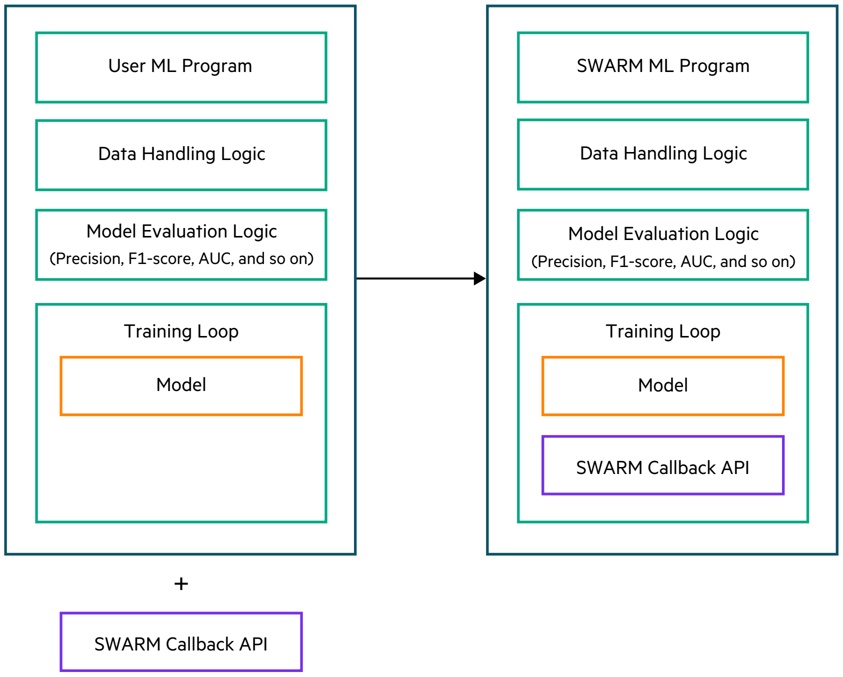
Each peer SL node then uses this merged model to start the next training batch. This process is coordinated by the SN network. The models are exchanged using the Swarm Learning file server.

A Swarm Learning ML program can specify a **Minimum Number of Peers** that are required to perform the synchronization. If the actual number of peers is less than this threshold value, the platform blocks the synchronization process until the required number of peers becomes available and reaches the synchronization point.

Adapting an ML program for Swarm Learning

You can transform any Keras (with TensorFlow 2 backend) or PyTorch based ML program that has been written using Python3 into a Swarm Learning ML program by making a few simple changes to the model training code by including the `SwarmCallback` API object. See the examples included with the Swarm Learning package for a sample code.

Your ML program algorithm can be any parametrized supervised learning model. It can be fully trainable model or a partially trainable model when using transfer learning.



How to Swarm enable an ML algorithm

To convert an ML program into a Swarm Learning ML program:

1. Import `SwarmCallback` class from the Swarm library.

   ```
   from swarmlearning.tf import SwarmCallback
   ```

   `SwarmCallback` is a custom callback class that consists of a set of functions that can be applied at various stages of the training process. During training, the `SwarmCallback` and the set of functions provide a view of internal states and statistics of the model to the Swarm Learning framework. `SwarmCallback` executes Swarm Learning specific operations during training like sharing parameters with all the network peers at the end of a sync interval.

   For TensorFlow-based Keras platforms, `SwarmCallback` is based on the Keras `tf.keras.callbacks.Callback` class. The methods of this class are called automatically by Keras at appropriate stages of the training.

   Unlike Keras, PyTorch does not have an in-built Callback class. Therefore, on PyTorch-based platforms, user must call the methods of this class. These methods are described subsequently in this section. An example of their usage is shown in the MNIST sample program.

2. Instantiate an object of the `SwarmCallback` class:

   ```
   from swarmlearning.tf import SwarmCallback #for TensorFlow
   from swarmlearning.pyt import SwarmCallback #for PyTorch

   # Create Swarm callback
   swarmCallback = SwarmCallback(syncFrequency=128,
   minPeers=min_peers,
   useAdaptiveSync=True,
   adsValData=(x_test, y_test),
   adsValBatchSize=8,
   swarmCallback.logger.setLevel(logging.DEBUG)
   ```

NOTE: Some of the parameters have default values as mentioned in the following table. User must provide values as applicable to their use case.

Parameter	Description
`syncFrequency`	Specifies the number of batches of local training to be performed between two swarm sync rounds. If adaptive sync enabled, this is the frequency to be used at the start. This is a mandatory parameter.
`minPeers`	Specifies the minimum number of SL peers required during each synchronization round for Swarm Learning to proceed further. This is a mandatory parameter.
`useAdaptiveSync`	Modulate the next syncFrequency value post each synchronization round based on performance on validation data. The default value is false. This is an optional parameter.
`adsValData`	Specifies the dataset for generating metrics for adaptive sync logic. It can be either an (x_val, y_val) tuple or a generator. This is an optional parameter for Swarm training. **NOTE:** This parameter must be provided to enable Swarm training monitoring (for example, loss and metrics) through SLM-UI.
`adsValBatchSize`	Specifies the batch size for `adsValData` if batch processing is required. This is used when `useAdaptiveSync` is turned ON. This is an optional parameter for Swarm training.

Table Continued

Parameter	Description
`checkinModelOnTrainEnd`	Specifies the merge behavior of a SL node after it has achieved stopping criterion (completed its local training) and it is waiting for all other peers to complete their training. During this period this SL node does not train the model with local data. This parameter decides the nature of the weights that this SL node contributes to the merge process. This is an optional parameter. Allowed values are: `inactive`: Node does not contribute its weights in the merge process but participates as non-contributing peer in the merge process. Specifying inactive for all the nodes is not a valid configuration. `snapshot`: Node always contributes the weights that it had when it reached the stopping criterion, it does not accept merged weights. `active`: Node behaves as if it is in active training (does not train model locally) and it contributes back the current merged weights that it got from Swarm merge. With this setting, the final model can be biased towards nodes which complete their training at the end. `snapshot` is the default value.
`trainingContract`	Training contract associated with this learning. It is a user-defined string. This is an optional parameter. Default value is `defaultbb.cqdb.sml.hpe`. **NOTE:** This parameter enables a user to run **concurrent** swarm learning trainings, within the same swarm network. User must create this training contract using SWCI and then use it as the parameter value.
`nodeWeightage`	A number between 1–100 to indicate the relative importance of this node compared with others during the parameter merge process. This is an optional parameter. By default, all nodes are equal and have the same weight-age of one.
`mlPlatform`	Specifies ML platform. Allowed values are either TF, KERAS or PYTORCH. This is an optional parameter. If TF platform is used, the default value is KERAS. If PYTORCH platform is used, the default value is PYTORCH.

Table Continued

Parameter	Description
`mergeMethod`	Specifies what kind of merge method to be used in Swarm merge process. When SL node becomes leader, it reads this parameter and performs merge of intermediate trainable parameters (weights and biases). This is an optional parameter. Allowed values are as follows: • mean: Merges the model's trainable parameters using the weighted mean method. This is the default merge method. • coordmedian: Merges the model's trainable parameters using the weighted coordinate wise median method. • geomedian: Merges the model's trainable parameters using the weighted geometric median method.

`logger`	Provides information about Python logger. This is an optional parameter. `SwarmCallback` class invokes info, debug, and error methods of this logger for logging. If no logger is passed, then `SwarmCallback` class creates its own logger from basic python logger. If required, user can get hold of this logger instance to change the log level as follows:
	
	`import logging`
	`from swarmlearning.tf import SwarmCallback`
	`swCallback = SwarmCallback(syncFrequency=128, minPeers=3)`
	`swCallback.logger.setLevel(logging.DEBUG)`

The following parameters are introduced to enable Swarm training monitoring through SLM-UI or SWCI command. Swarm training monitoring includes model accuracy, loss, progress of epochs, or ETA of training completion. These are not mandatory parameters for Swarm Learning. But, without these parameters, training progress may have limitations.

Parameter	Description
`totalEpochs`	Specifies the total number of epochs used in local training. It also logs the warning message in the User or ML container log when `totalEpochs` is not provided in Swarm Callback. For more information, see **CPU based MNIST-PYT** and **MNIST** examples.
`lossFunction`	Specifies the name of the loss function to be used for loss computation. Loss computed is used for display and adaptive sync frequency. This parameter is applicable only for PyTorch-based platforms. If this parameter is provided for Keras platforms, then it will be ignored. For more information, see **CPU based MNIST-PYT** example. **NOTE:** User is responsible to pass the same loss function used for local training. If the loss function is not matching, it can result in differences in loss displayed by Swarm and actual loss used in the local model training. `lossFunction` string must match with loss function class defined in PyTorch. For more information, see **https://pytorch.org/docs/stable/nn.html#loss-functions**.
`lossFunctionArgs`	Specifies dictionary of parameters (as keys) to be used in `lossFunction`. This parameter is applicable only for PyTorch-based platforms. If this parameter is provided for Keras platforms, then it will be ignored. For more information, see **CPU based MNIST-PYT** example. `lossFunctionArgs` is used to pass any mandatory parameters or overwrite any default parameters of `lossFunction`.

Table Continued

Parameter	Description
`metricFunction`	Specifies the name of metric function to be used for metrics computation. This parameter is applicable only for PyTorch-based platforms. If this parameter is provided for Keras platforms, then it will be ignored. For more information, see **CPU based MNIST-PYT** example. User is responsible to pass the same metric function used for local training. If the metric function is not matching, it can result in differences in metrics displayed by Swarm and actual metrics used in the local model training. `metricFunction` string must match with metric function class defined in torchmetrics. For more information, see **https://torchmetrics.readthedocs.io/en/stable/all-metrics.html**.
`metricFunctionArgs`	Specifies dictionary of parameters (as keys) to be used in `metricFunction`. This parameter is applicable only for PyTorch-based platforms. If this parameter is provided for Keras platforms, then it will be ignored. For more information, see **CPU based MNIST-PYT** example. `metricFunctionArgs` is used to pass any mandatory parameters or overwrite any default parameters of `metricFunction`.

3. Use the `SwarmCallback` object for training the model.

   - For Keras platforms:
     - Pass the object to the list of callbacks in Keras training code. The class methods are invoked automatically.

       ```
       model.fit(..., callbacks = [swarm_callback])
       ```

     - `SwarmCallback` can be included along with other callbacks also:

       ```
       es_callback = EarlyStopping(...)
       model.fit(..., callbacks = [es_callback, swarm_callback])
       ```

   - For PyTorch platforms, you must invoke the class methods:
     - Call `on_train_begin()` before starting the model training:

       ```
       swarmCallback.on_train_begin()
       ```

     - Call `on_batch_end()` after the end of each batch training:

       ```
       swarmCallback.on_batch_end()
       ```

    - Call `on_epoch_end()` after the end of each epoch training:

      ```
      swarmCallback.on_epoch_end(epoch)
      ```

    - Call `on_train_end()` after the end of the model training:

      ```
      swarmCallback.on_train_end()
      ```

4. Run Swarm Learning using the supplied scripts. For more information, see Running Swarm Learning in the *HPE Swarm Learning Installation and Configuration Guide* with the data and model directories as input parameters.

Swarm Learning Command Interface

Swarm Learning Command Line Interface (SWCI) is the command interface tool to the Swarm Learning framework. It is used to view the status, control, and manage the Swarm Learning framework. SWCI manages the Swarm Learning framework using contexts and contracts. For more information on how to start the SWCI tool, see *HPE Swarm Learning Installation and Configuration Guide*.

The user can enter any command from the predefined set of commands in the SWCI prompt. The entered command is parsed, then executed and the output is shown in the command line.

The subsequent sections mention about SWCI-related commands, and these commands are also part of the help text.

SWCI commands related to context

Users are required to explicitly create a context (using the CREATE context command) and activate it as follows:

```
+----------------------------------------------------------------+
| SWCI:13 > CREATE CONTEXT **testContext** WITH IP sn.test.sw.net|
| API Server is UP!                                              |
| CONTEXT CREATED : **testContext**                              |
| SWCI:14 > SWITCH CONTEXT testContext                           |
| DEFAULT CONTEXT SET TO : testContext                           |
| SWCI:15 >                                                      |
+----------------------------------------------------------------+
```

Command	Description and parameters
`CREATE CONTEXT <contextName : string> WITH IP <ip : string> [port : string]`	Creates a SWCI context. • `contextname`: A user given identifier for this context. • `ip`: IP address or FQDN of the API server (Swarm Network node) serving Swarm Learning APIs. • `port`(optional): The port number on which the API server is listening.
`CREATE CONTEXT <contextName : string> WITH SERVICE <service : string>`	Creates a SWCI context. This is used when using reverse proxy. • `contextname`: A user given identifier for this context. • `service`: FQDN for the API Service of the associated Swarm Network node. If the reverse proxy is configured to run on any non-default port, it has to be passed along with the service string parameter with ':' separator. This is an optional port.
`GET CONTEXT INFO [contextName : string]`	Prints the context information of the current Context when `contextName` is not specified. If `contextName` is specified prints information for the specified context.

Table Continued

Command	Description and parameters
`GET CONTEXT ENV <contextName:string> <env:string>`	Obtains the value of the specified context-specific environment variable. If the `contextName` is not specified, the default context is used. • `contextName`: Specifies the name of the target context. • `env`: Specifies the target environment variable within the context.
`GET CONTEXT VERSION`	Returns the API Server's version information.
`LIST CONTEXTS`	Prints the list of contexts related to the current SWCI session.
`LIST CONTEXT ENV <contextName : string>:`	Displays the list of context-specific environment variables. If the `contextName` is not specified, the default context is used. `contextName`: Specifies the context from which these environment variables are listed.
`SET CONTEXT ENV <contextName : string> <env : string> <envValue : string>`	Sets the value of the specified context-specific environment variable. If the `contextName` is not specified, the default context is used. • `contextName`: Specifies the name of the target context. • `env`: Specifies the target environment variable within the context. • `envValue`: The value for the variable being set.
`SWITCH CONTEXT <contextName : string>`	Switches current SWCI context to a specified context.

SWCI commands related to contract

Command	Description and parameters
`CREATE CONTRACT <SLContractName:string>`	Registers the specified Swarm Learning contract into the Swarm Learning network.
`GET CONTRACT INFO <SLContractName:string>`	Displays the static information about the specified Swarm Learning contract.
`GET CONTRACT STATUS <SLContractName:string>`	Reports the current dynamic status of the specified Swarm Learning contract.
`LIST CONTRACTS`	Displays the list of Swarm Learning contracts currently registered into the Swarm Learning network.

Table Continued

Command	Description and parameters
`PERFDATA <SLContractName:string>`	Displays performance data about the training under the Swarm Learning contract. It provides training performance data like UID, SL ADMIN status, model loss, model metric, total number of epochs and total number of completed epochs, for each SL-ML pair.
`RESET CONTRACT <SLContractName:string>`	Resets the state of the contract to uninitialized state. **WARNING:** This action cannot be undone, reset only completed Swarm Learning contracts. Resetting the active contracts can result in unexpected behavior. A typical scenario would be when a user wants to reuse a completed training contract and start a new training session.

SWCI commands related to Task

Command	Description and parameters
`APPEND TASK BODY <taskName : string> <idx : int> <contentLine : string>`	Add or overwrite a specified line in the task body. **NOTE:** Finalized task cannot be modified. • `taskName`: Specifies the task that must be modified. • `idx`: Specifies a nonzero line that must be added or modified. • `contentLine`: Actual text that is less the 80 characters.
`CREATE TASK <taskType : string> <taskName : string> <author : string> <prereq : string> <outcome : string>`	Creates a task and registers into Swarm network. • `taskType`: Specifies the type of task to be created. • `taskName`: Specifies the unique name of task. • `author`: Specifies the author of the task. • `prereq`: Specifies the prerequisite required to create and execute the task. • `outcome`: Specifies the name of the artifact that is produced as a result of this task.
`CREATE TASK FROM <taskDefFile:string>:`	`taskDefFile` is a relative path to task definition file.
`DELETE TASK <taskName : string>`	Delete the specified task. This command is used for removing tasks that have errors while being created. **NOTE:** Finalized task cannot be deleted. `taskName`: Specifies the task that must be deleted.

Table Continued

Command	Description and parameters
`FINALIZE TASK <taskName : string>`	Finalizes the specified task. Once finalized, task body cannot be modified further. `taskName`: Specifies the task that must be finalized.
`GET TASK BODY <taskName : string>`	Prints the consolidated task body for the specified Task ID.
`GET TASK INFO <taskName : string>`	Prints task information of the specified Task ID.
`LIST TASKS`	Displays the list of tasks that are registered into the Swarm Learning network.

SWCI commands related to Taskrunner

Command	Description and parameters
`ASSIGN TASK <taskName : string> TO <trName : string> WITH <peersNeeded> PEERS`	Assigns a task to a taskrunner instance and specify the minimum peer count for declaring the result. This triggers SWOPs to execute this task • `taskName`: Specifies the unique name of task. • `trName`: Specifies the unique name of Taskrunner. SWOP's listening on this Taskrunner participates and executes this task. • `WITH <peersNeeded> PEERS`: `peersNeeded` is a nonzero positive integer specifying the minimum number of peers required to complete this task. Meaningful value of minimum peers is dependent on the task type. For RUN_SWARM task, it is the number of SL and ML node pairs. For all other TASK types, it is the number of SWOP nodes. For RUN_SWARM task type, the actual numbers of SL/ML peers started could be equal or greater than `peersNeeded`, depending on the number of the SL nodes defined in the SWOP profiles. For other TASK types, the number of SWOPs participating would be equal to `peersNeeded`. **NOTE:** This command can be executed to start or initiate the task runner. This command cannot be used to add new SWOP targets on an already running task runner. If this command is executed on the already running task runner, it will fail.
`ASSIGN TASK <taskName : string> TO <trName : string> WITH ALL PEERS`	Assigns a task to all SWOP peers listening on the specified taskrunner. This triggers all SWOPs to execute this task • `taskName`: Specifies the unique name of task. • `trName`: Specifies the unique name of Taskrunner. All SWOP's listening on this Taskrunner participates and executes this task. Here, the minimum number of peers required to complete this task is considered based on all active SWOPs listening on the specifed taskrunner at the time when this command is issued. Meaningful value of minimum peers is dependent on the task type. For RUN_SWARM task, it is the number of SL and ML node pairs. For all other TASK types, it is the number of SWOP nodes. For RUN_SWARM task type, the actual numbers of SL/ML peers started equal to the total number of the SL nodes defined in all the combined SWOP profiles. For other TASK types, the number of SWOPs listening to the specified taskrunner. **NOTE:** This command can be executed to start or initiate the task runner. This command cannot be used to add new SWOP targets on an already running task runner. If this command is executed on the already running task runner, it will fail.

Table Continued

Command	Description and parameters
`ASSIGN TASK <taskName> TO <taskRunnerName> TARGET <SWOP-UID>`	Assigns a task to a taskrunner instance to target only specified SWOP peer. This triggers targeted SWOP to execute this task • `taskName`: Specifies the unique name of task. • `trName`: Specifies the unique name of Taskrunner. • `TARGET <SWOP-UID>`: Specifies UID of the SWOP peer that will execute this task. User can get the UID of the SWOP from the list nodes output. Here, the minimum number of peers required to complete this task is considered based on targeted SWOP if listening on the specified taskrunner. Meaningful value of minimum peers is dependent on the task type. For RUN_SWARM task, it is the number of SL and ML node pairs. For all other TASK types, it is the number of SWOP node. For RUN_SWARM task type, the actual numbers of SL/ML peers started equal to the total number of the SL nodes defined in the targeted SWOP profile. For other TASK types, it is 1. **NOTE:** This command can be executed to start or initiate the task and also to target a specific SWOP to join an already running task runner. This command helps in retrying a failed task on a particular SWOP.
`CREATE TASKRUNNER <trName : string>`	Creates and registers a Taskrunner contract into Swarm network. `trName`: Specifies the unique name of Taskrunner.
`GET TASKRUNNER INFO <taskRunnerName : string>`	Displays the current status of the specified Taskrunner ID.

Table Continued

Command	Description and parameters
`GET TASKRUNNER PEER STATUS <taskRunnerName : string> <swopIndex| swopUid>:`	Displays current status for the specified PEER in the Taskrunner's context. • `taskRunnerName`: Specifies the unique name of Taskrunner. • `swopIndex`: The index number of SWOP Node, starts with 0 up to `ENROLLEDSWOP`. • `swopUid`: The UID String of the SWOP Node. You can specify either `swopIndex` or `swopUid`, not both. The status of the PEER differs based on the type of the current TASK that has been assigned to TASKRUNNER. For RUN_SWARM task, the status summary reports SWOP node UID, Number of SL PEERs this SWOP has spawned, and list of all SL node information (UID, Status, Description). For all other types of tasks, the status summary reports SWOP node status (UID, Status, Description). **NOTE:** Node UID can be used to identify the container name/id from 'LIST NODES' command. With container name/id, user can debug the error with docker logs.
`GET TASKRUNNER STATUS <trName : string>`	Displays the current status of the specified Taskrunner. `trName`: Specifies the unique name of Taskrunner. Provides below information like, • TASK NAME – Current running task or 'Empty' if no task is assigned. • PEER TYPE – SL for RUN_SWARM tasks, SWOP for all other tasks. • TASK STATE – Current task state. • ACTIVE, COMPLETED and FAILED PEERS information. • TIME STAMPs of various events on the TASKRUNNER contract.
`LIST TASKRUNNERS`	Displays the list of Taskrunners that are registered into the Swarm Learning network.
`LIST TASKRUNNER PEERS <taskRunnerName:string>`	Displays list of enrolled peers for the specified Taskrunner ID.

Table Continued

Command	Description and parameters
`RESET TASKRUNNER <trName : string>`	Resets the state of the taskrunner contract to an uninitialized state. ⚠ **WARNING:** This action cannot be undone, reset only completed Taskrunner contracts. Resetting the active taskrunner contract can result in unexpected behavior. `trName`: Specifies the unique name of Taskrunner.
`WAIT FOR TASKRUNNER <trName : string>`	Waits for the Taskrunner to complete its current task. `trName`: Specifies the unique name of the Taskrunner. `WAITING FOR TASKRUNNER TO COMPLETE - Maximum wait time is :` *`<SWCI_TASK_MAX_WAIT_TIME>`*. If `SWCI_TASK_MAX_WAIT_TIME` is not set, the default value is 120 mins. Prints # (a progress indicator) periodically until the task completes. When the task completes, the final state of the task is printed. If the maximum wait time is reached, the following warning message is displayed: `WARNING - Maximum configured SWCI wait time is over`

Miscellaneous commands in SWCI

Command	Description and parameters
`cd <dirPath>`	This method changes the current working directory of SWCI container.
`EXIT`	This command exits the SWCI session unconditionally.
`EXIT ON FAILURE [ON/OFF]`	This command instructs SWCI to exit the current session when any of the subsequent commands fail. The default value is OFF.
`LIST NODES`	This command displays the list of Swarm nodes that have registered and are currently active. For each Swarm node, it displays the Node type, Host IP, Port, Container name, UUID, parent UUID and the last received 'i-am-alive' Timestamp. SWCI and ML nodes are not displayed. For a reverse proxy scenario, it displays the service name of the associated node, instead of Host IP. Users can look at their NGINX configuration file to know the IP addresses of the respective service names. For Sentinel and Non-Sentinel nodes, it displays its respective API service names. For SL nodes, it displays the FS service names. For SWOP node, it displays the API service name of the associated SN node.

Table Continued

Command	Description and parameters
`ls()`	This method displays the directory contents of the SWCI container.
`pwd()`	This method displays the present working directory of the SWCI container.
`SLEEP`	This command sleeps for a specified time before executing the subsequent commands. For example, in between a `WAIT FOR TASKRUNNER` and `RESET TASKRUNNER`, one can use a `SLEEP 10`, to give a grace time of 10 secs, before the `RESET` command cleans up the SL and user container. This would be required to allow the user ML code to save the model or do any inference of the model, after the Swarm training is over. For more information, see the example SWCI scripts in the `swarm-learning/examples/` directory.
`WAIT FOR IP <ip:string> [port:string] [retries:string]`	This command waits for the specified API server to accept connections. • `ip`: The IP address or FQDN of the API server (SN node) serving Swarm Learning APIs. • `port`: (Optional) The string representation of the port number on which the API Server is listening. • `retries`: (Optional). Default "retries" is 360 times (30 mins). This is the maximum number of times SWCI reattempts to connect after waiting for a 5 seconds timeout period.
`WAIT FOR SERVICE <service : string> [retries:string]`	This command waits for the specified API service to accept connections. • `service`: FQDN for the API Service of the associated Swarm Network node. If the Port number exists, it has to be passed along with the service string parameter with ':' separator. This is an optional port. • `retries`: (Optional). Default "retries" is 360 times (30 mins). This is the maximum number of times SWCI reattempts to connect after waiting for a 5 seconds timeout period.

SWCI APIs

The following API methods help to invoke SWCI operations programmatically. It is used to view the status, control, and manage the Swarm Learning framework.

NOTE:

- Arguments that do not have default values are mandatory.
- This is an experimental API methods for developers.

The python3 program needs to import the SWCI class, and then use the below APIs.

```
from swarmlearning.swci import swci
```

SWCI API methods	Description	Arguments

`Swci()`	This method invokes the constructor of the SWCI class and creates a SWCI object instance.	`swciIP,port = int (30306), clientCert = None, clientPKey = None, clientCABundle = None, logger = None, logLv = WARNING, enableCaching = True`
		
		`swciIP`: The IP address or Name of the SWCI container that the user wants to connect to.
		
		`clientCert/ clientPKey/ ClientCABundle`: If you need secure mTLS connection to the SWCI container, then specify the certificates.
		
		`logLv`: Log level - One of `{ CRITICAL, ERROR, WARNING, INFO, DEBUG }`
		
		`enableCaching`: If True, it enables caching of command output of some commands whose output does not change frequently. This is done to avoid network round trips.

`assignTask()`	This method assigns a task to a Taskrunner instance and specify the minimum peer count for declaring the result.	This triggers SWOPs to execute this task. `taskName`: Specifies the unique name of a task. `trName`: Specifies the unique name of a Taskrunner. SWOP's listening on this Taskrunner participates and executes this task. `WITH <peersNeeded> PEERS: peersNeeded` is a nonzero positive integer specifying the minimum number of peers required to complete this task. Meaningful value of minimum peers is dependent on the task type. For `RUN_SWARM` task, it is the number of SL and ML node pairs. For all other task types, it is the number of SWOP nodes. • For RUN_SWARM task type, the actual numbers of SL/ML peers started could be equal or greater

Table Continued

SWCI API methods	Description	Arguments
		than `peersNeeded`, depending on the number of the SL nodes defined in the SWOP profiles. • For other TASK types, the number of SWOPs participating would be equal to `peersNeeded`.
`cd()`	This method changes the current working directory of SWCI container.	`dirPath`
`clearCache()`	This method clears any cached values in the SWCI instance.	
`createContext()`	This method creates an SWCI context. **NOTE:** User can create SWCI context with (ip and port) or with service [for example:api.sn.swarm:30304]. For example, **1.** `createContext(test, ip=172.1.1.1)` - This variation assumes SN API running on default 30304 port. **2.** `createContext(test, ip=172.1.1.1, port=16000)` – This variation is used when SN API running on non-default port. **3.** `createContext(test,service=api.sn.swarm)` – In reverse proxy scenario, this variation means SN API service running on default 443 port. **4.** `createContext(test,service=api.sn.swarm:17000)` – In reverse proxy scenario, this variation means SN API service running on non-default port.	`ctxName, ip=None, port=30304, service=None`
`createTaskFrom()`	This method creates a new task from the YAML file, the YAML definition file must be exported to the SWCI container using `uploadTaskDefintion` method.	`yamlFileName`
`createTrainingContract()`	This method registers the specified Swarm Learning training contract into the Swarm Learning network.	`ctName`

Table Continued

SWCI API methods	Description	Arguments
`deleteTask()`	This method deletes the specified task which is not finalized.	`taskName`
`executeTask()`	This method executes a task. It assigns, monitors, and resets Taskrunner at the end of the task execution.	`taskName, tr='defaulttaskbb.taskdb.sml.hpe', peers=1, pollWaitInSec=120, resetTROnSuccess=True` `pollWaitInSec`: Wait time before polling for status. `resetTROnSuccess`: If the Taskrunner contract has to be reset on success.
`finalizeTask()`	This method finalizes the specified task. Once finalized, the task body cannot be modified further.	`taskName`
`getContextInfo()`	This method prints the context information of the current context when `contextName` is not specified. If `contextName` is specified, it prints information for the specified context.	`ctxName`
`getErrors()`	This method prints the error if any from the previously executed SWCI method.	
`getTaskBody()`	This method prints the consolidated task body for the specified Task ID.	`taskName`
`getTaskInfo()`	This method prints the task information of the specified Task ID.	`taskName`
`getTaskRunnerInfo()`	This method prints the current status of the specified Taskrunner ID.	`trName`
`getTaskRunnerPeerStatus()`	This method prints the current status for the specified PEER (SWOP node) in the Taskrunner's context.	`trName, idx`

Table Continued

SWCI API methods	Description	Arguments
`getTaskrunnerStatus() <trName:string>`	This method displays the current status of the specified Taskrunner.	`trName: Specifies the unique name of Taskrunner.` Provides below information like, • TASK NAME – Current running task or ‘Empty’ if no task is assigned. • PEER TYPE – SL for RUN_SWARM tasks, SWOP for all other tasks. • TASK STATE – Current task state. • ACTIVE, COMPLETED and FAILED PEERS information. • TIME STAMPs of various events on the TASKRUNNER contract.
`getTrainingContractInfo()`	This method prints static information about a training contract.	`ctName`
`getTrainingContractPerformanceData`	This method prints Performance Data for a training contract. It provides training performance data like UID, SL ADMIN status, model loss, model metric, total number of epochs and total number of completed epochs, for each SL-ML pair.	`ctName='defaultbb.cqdb.sml.hpe'`
`getTrainingContractStatus()`	This method prints the current dynamic status of a training contract.	`ctName`
`isTaskDone()`	This method displays true if a Taskrunner has completed the current task, else false.	`trName`
`listContexts()`	This method displays the list of contexts related to the current SWCI session.	
`listNodes()`	This method displays the list of Swarm nodes that have registered and are currently active. For each Swarm node, it displays the Node type, Host IP, Port, Container name, UUID, parent UUID and the last received ‘i-amalive’ Timestamp. SWCI and ML nodes are not displayed.	
`listTaskRunners()`	This method displays the list of Taskrunners that are registered into the Swarm Learning network.	

Table Continued

SWCI API methods	Description	Arguments
`listTasks()`	This method displays the list of tasks that are registered into the Swarm Learning network.	
`listTrainingContracts()`	This method displays the list of training contracts registered with Swarm Learning network.	
`ls()`	This method displays the directory contents of the SWCI container.	`optStr=''`
`plotTopology()`	This method displays the PNG object showing the current topology of the Swarm Learning network. By default, it will display the Swarm Node type and Host IP on which the node is running. **NOTE:** Color codes are hexadecimal triplets representing the colors red, green, and blue (#RRGGBB).	User can pass additional attributes which they want to see on the plot. These additional attributes are: 'Port', 'ContainerName', 'UUID', 'parentUUID' and 'i-am-alive'. `SNColour="#ADD8E6", SWOPColor="#33FF33", SLColor="#FFCCCB", attrs=['ContainerName']`
`pwd()`	This method displays the present working directory of the SWCI container.	
`registerTask()`	This method registers a task into the SN network and finalizes it, if the task is valid.	`yamlFileName, finalize=True`
`resetTaskRunner()`	This method resets the state of the Taskrunner contract to an uninitialized state. **WARNING:** This action cannot be undone, reset only completed Taskrunner contracts. Resetting the active Taskrunner contract can result in unexpected behavior.	`trName='defaulttaskbb.taskdb.sml.hpe'`
`resetTrainingContract()`	This method resets the state of the training contract to an uninitialized state. **WARNING:** This action cannot be undone, reset only completed Swarm Learning contracts. Resetting the active contracts can result in unexpected behavior.	`ctName='defaultbb.cqdb.sml.hpe'`

Table Continued

SWCI API methods	Description	Arguments
`sleep ()`	This method sleeps for a specified time before executing the subsequent commands. For example, in between a `WAIT FOR TASKRUNNER` and `RESET TASKRUNNER`, one can use a `SLEEP 10`, to give a grace time of 10 secs, before the `RESET` command cleans up the SL and user container. This would be required to allow the user ML code to save the model or do any inference of the model, after the Swarm training is over. For more information, see the example SWCI scripts in the `swarm-learning/examples/` directory.	time in seconds

`setLogLevel()`	This method sets the logging level for the SWCI container.	`logLv`
		
		One of {`CRITICAL, ERROR, WARNING, INFO, DEBUG`}

`uploadTaskDefintion()`	This method uploads the local task definition file to the SWCI container.	`taskFilePath`

Example snippet of an API

```
################################################################
# This code snippet shows how an user can use SWCI API's
#
# We assume the following things before running this script:
# 1. Swarm Learning Infrastructure is setup and ready.
# 2. SWCI container is running in WEB mode (-e SWCI_MODE='WEB')
# 3. There should be explicit port forwarding for SWCI_WEB_PORT
#    while running the SWCI container (ex: -p 30306:30306)
# 4. Swarm learning wheel package should be installed in
#    the environment where we run this file.
################################################################

# Import swci from the swarmlearning whl package
import swarmlearning.swci as sw

swciServerName = 'SWCI Server Name or IP'
snServerName = 'SN Server Name or IP'

# Provide certificates for secured connection to the SWCI container.
clCert='<Full path of SWCI public certificate>'
clPKey='<Full path of SWCI private certificate path>'
clCABundle='<Full path of SWCI CA certificate directory>'

# Secured connection to the SWCI via SWCI_WEB_PORT
s = sw.Swci(swciSrvName,port=30306,clientCert=clCert,clientPKey=clPKey,clientCABundle=clCABundle)
#30306 is the default port
# Connect to SN and create context
print(s.createContext('testContext', ip=snServerName))
# Switches the context to testContext
print(s.switchContext('testContext'))
# Creates a training contract
```

```
print(s.createTrainingContract('testContract'))
# Lists all the created Contexts
print(s.listContexts())
# Lists all the tasks that includes root task
print(s.listTasks())
```

Swarm Operator node (SWOP)

SWOP is responsible to execute tasks that are assigned to it. A SWOP node can execute only one task at a time. Other than executing tasks, SWOP provides multiple functionalities such as:

- Managing Swarm Learning operations—SWOP eases the management of tasks because of its ability to perform control operations across the Swarm Learning network. It also uses a simple interface to manage operations from anywhere.
- Scalability—SWOP can manage hundreds of Swarm Learning nodes with the use of one command.
- Automation—SWOP eases automation of Swarm operations by providing APIs for scripting.

Taskrunner Framework

Taskrunner framework is a decentralized task management framework.

It differs significantly from centralized task and workflow management frameworks (for example, Apache Airflow) in many aspects.

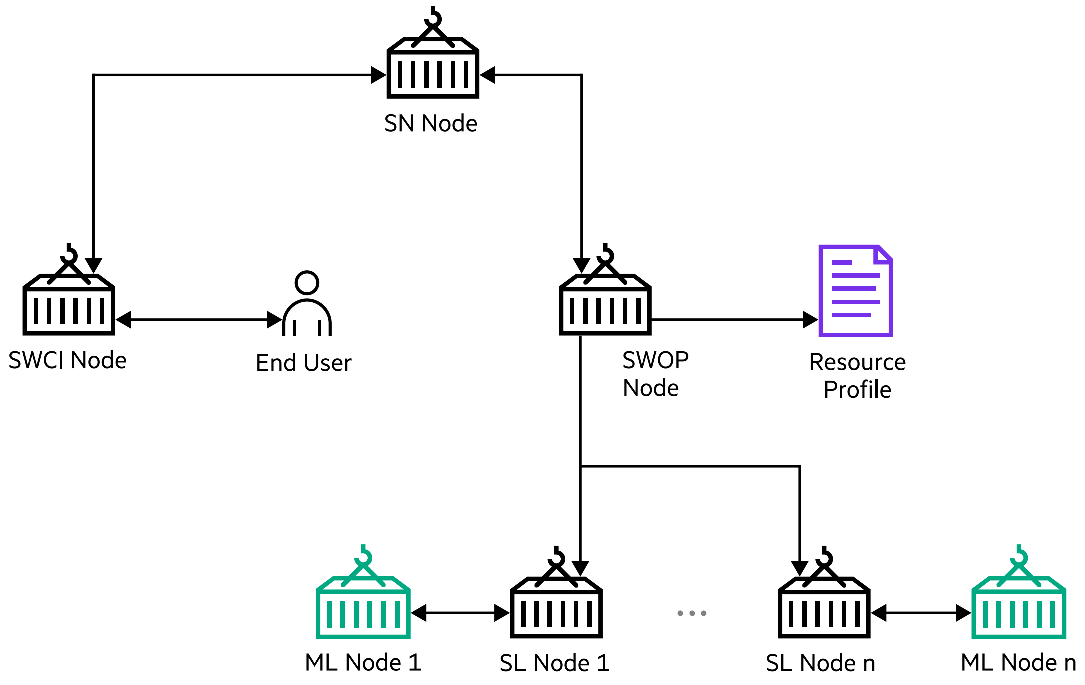
Here, we point out a few key differences.

- The total size of the participating entities is not always known. In other words, we know only the quorum that is needed to take a decision and never know the full set of entities. The total size of the participating entities can change dynamically.
- Success or Failure of a scheduled task depends on whether the predetermined quorum of nodes completed the task successfully or not.
- It is very computationally complex to ensure all participating entities are in the same state at any given instant. Only eventual consistency is feasible. This implies that all participating entities will tend towards a common stable state eventually.

Considering the above aspects, rolling back a failed task is not possible, as we have only partial knowledge of the full state of the system at any given time. Instead, we can always take corrective action by re-running a task or creating new tasks that supersede earlier failed tasks.

SWOP architecture overview

The following is the architectural overview of the SWOP (**SWarm OPerator**) component:



SWOP container builds or pulls the user container to match the workload configuration requirements. It builds volumes that the user containers use as specified in the workloads configuration. Initiates the Swarm Learning training sessions by launching user ML and SL container pairs.

NOTE: HPE recommends users to start their user ML and SL container pairs automatically through SWOP.

Launch Swarm Learning using SWOP

SWOP components obtain information about other components from the SWOP profile file supplied by the user. This profile file is used at start up to initialize the SWOP component, create containers, and build new containers. The SWOP profile file must conform to the schema specified in the `profile-schema.yaml` file. The SWOP profile file specifies the following component information:

- Name of the group.
- The number of SL nodes in the group.
- Taskrunner contract of the group.
- Infrastructure information such as APLS, SPIRE, SN, and DNS.
- Environmental variables (optional)

The SWOP container must be started by the user by providing the YAML profile file and `run-swop` script along with other parameters.

NOTE: To effectively utilize the new feature of **Targeted Execute** on a particular SWOP, HPE recommends to give unique container names for each SWOP container.

SWOP profile schema

NOTE: The source code of the profile schema (`SWOP-profile-schema.yaml`) is present in the same directory as the User Guide.

```
######################################################################
## (C)Copyright 2021-2023 Hewlett Packard Enterprise Development LP
######################################################################
"$schema"   : "https://json-schema.org/draft/2020-12/schema"
title       : "SWOP Profile definition"
description : "Describes the structure of the profile file"
type        : object
properties  :
    groupname  :
        type        : string
        description : "name of the profile group"
    taskrunner :
        type        : string
        description : "id of the taskrunner contract to use"
    policyuri  :
        description : "None (no policy accept all tasks) or uri to the
policy engine"
        oneOf :
            - const : ~
            - type  : string
    resourcemgr     :
        type         : object
        description  : "Specification of the underlying resourcemgr.
                        Currently only DOCKER is supported"
        properties   :
            mgrtype :
                description  : "Identifies the resource manager type"
                type : string
                enum :
                    - DOCKER
            accessinfo :
                oneOf  :
                    - type  : object
                      description : "Specifies how to instantiate docker
client
                                     FROMENV - Return a client configured
from environment variables.
                                     The environment variables used are the
same as those
                                     used by the Docker command-line client.
Currently FROMENV only supported.
                                     It can be extended to instantiate
client that will allow you to communicate
                                     with a Docker daemon through UNIX
socket or SSL"
                      properties  :
                          accesstype:
```

```
                        type : string
                        enum :
                            - FROMENV
                required    :
                    - accesstype
                additionalProperties : false
        required     :
            - mgrtype
            - accessinfo
        additionalProperties : false
    network    :
        type        : string
        description : "Nework to attach the SL containers"
    apls       :
        type        : object
        description : "APLS object definition"
        properties  :
            locator :
                type        : "object"
                properties  :
                    host  :
                        type : string
                    port :
                        oneOf :
                            - type : integer
                              minimum: 0
                            - const : ~
                required :
                    - host
                additionalProperties : false
        required    :
            - locator
        additionalProperties : false
    apisrv     :
        type         : object
        description  : "API SERVER object definition"
        properties   :
            locator  :
                type       : object
                properties :
                    host  :
                        oneOf:
                            - type: string
                              maxLength: 253
                              minLength: 1
                            - const : ~
                    port :
                        oneOf :
                            - type : integer
                              minimum: 0
                            - const : ~
                    service:
                        oneOf:
                            - type: string
                              maxLength: 253
                              minLength: 1
                            - const : ~
                required :
                    - host
                    - service
                additionalProperties : false
    envvars    :
        oneOf       :
          - const   : ~
          - type    : array
            description : "List of global environmental variables as K-V
pair common to all SL nodes"
            items       :
                type        : object
                uniqueItems : true

    # Allow service endpoints to be specified as domain names, rather than
IPs.
    # Traffic can be routed through a reverse proxy or a web gateway by
making
    # these name servers resolve the domain names into the IP of the reverse
    # proxy and then, configuring the reverse proxy to forward to the
appropriate
    # service. This is at a global level for now - primarily because we
cannot
    # think of a use case for two containers on a production box wanting to
use
    # different name servers. If and when someone demonstrates the scenario,
we
    # can move this field down to the nodes section. To limit the current
scope
    # of work, we do not support any of the other related parameters, such as
    # dns-opt, dns-search and domain-name. Further, we feel we should explore
    # ways to create an open specification that can accept all valid
parameters
    # and not just a chosen handful.
    dns:
        oneOf:
          - const: ~
          - type: array
            description: "List of custom DNS servers"
            items:
                type: string
                maxLength: 253
                minLength: 1
                uniqueItems: true

    nodes      :
        type        : array
        description : "List of Node specifications managed by this SWOP
entity"
        items       :
            - slnodedef :
                  type       : object
                  properties :
                      idx         :
                          type    : integer
                          description : "Identifier for the SL node in this
SWOP profile"
```

```
                          # this index is provided by user it has to start
with 0
                          # and be positive integer only
                          minimum : 0
                      identity    :
                        type        : array
                        description : "Information about identity of this
node"
                        items       :
                            - attribute  :
                                type        : object
                                description : "Specification of the
certificates or spiffe endpoint"
                                properties  :
                                    aType       :
                                        type        :
string
                                        description : "Identity
specification attribute type"
                                        enum        :
                                            - KEY
                                            - CERT
                                            - CACERT
                                            - CAPATH
                                            - SPIFFE_SOCKETPATH
                                    mType       :
                                        type        : string
                                        description : "Type of mount to be
used to mount the user data"
                                        enum        :
                                            - BIND
                                            - VOLUME
                                    src          :
                                        type        : string
                                        description : "Host directory or
fullpath or Volume name to be used for Mount"
                                        maxLength   : 240
                                        minLength   : 2
                                    tgt          :
                                        type        : string
                                        description : "Target directory or
fullpath inside container"
                                        maxLength   : 240
                                        minLength   : 2
                                    subPath         :
                                        oneOf :
                                            - const    : ~
                                            - type         : string
                                              description : "Optional sub-
path to the resource realtive to tgt"
                                              maxLength   : 240
                                              minLength   : 2
                                required    :
                                    - aType
                                    - mType
                                    - src
                                    - tgt
                                    - subpath
                                additionalProperties : false
                      slhostip    :
                          oneOf:
                              - const: ~
                              - type    : string
                                description : "Externally visible IPv4
Address or FQDN of the SL Container"
                                maxLength   : 240
                                minLength   : 2
                      slport      :
                          oneOf :
                              - const   : ~
                              - type    : integer
                                description : "FS Server port exposed by
this SL NODE (default 30305)"
                                minimum : 0
                      slfsservice:
                          oneOf:
                              - const: ~
                              - type: string
                                description: "FQDN and optional port of the
FS service for the SL Container"
                                # https://stackoverflow.com/a/32294443.
                                maxLength: 253
                                minLength: 2
                      slhostname :
                          oneOf :
                              - const   : ~
                              - type    : string
                                description : "Docker host name for the SL
container"
                                maxLength   : 240
                                minLength   :
2
                      privatedata :
                          oneOf :
                              - const   : ~
                              - type    : object
                                description : "Private data store this node
exposes as bind or volume mount"
                                properties:
                                    src :
                                        type        : string
                                        description : "Source Volume or
directory"

                                        maxLength   : 240
                                        minLength   : 2
                                    # destination will come from the RUN-
SWARM task definition
                                    # the program will decide where in the
Filesystem this dir
                                    # will be mounted to.
                                    mType :
                                        type : string
                                        description : "Type of mount to be
used to mount the user data"
```

```
                                            enum :
                                                - BIND
                                                -
VOLUME
                                    required:
                                        - src
                                        - mType
                                    additionalProperties: false
                        slenvvars   :
                            oneOf       :
                                - const   : ~
                                - type    : array
                                  description : "List of local environmental
variables as K-V pair specific to SL node.
                                                 This will take precedence
over global environmental variables."
                                  items       :
                                      type        : object
                                      uniqueItems : true

                        # In our practice, we are using labels only for the
SPIFFE selector not envvars.
                        sllabels:
                            oneOf:
                              - const: ~
                              - type: object
                                description: "Dictionary of labels for the
Swarm Learning container"
                                items:
                                    uniqueItems: true
                              - type: array
                                description: "List of labels with empty values
for the Swarm Learning container"
                                items:
                                    type: string
                                    uniqueItems: true
                        # This attribute helps in starting SL container with
pre-defined python docker network SDK option called "ipv4_address"
                        # Added as part of reverse proxy multi-host scenario
where instead of using hostIP with in nginx configuration we can
                        # now use continer IP's for easier and simpler IP and
port mapping.
                        slnetworkopts   :
                            oneOf       :
                                - const   : ~
                                - type    : array
                                  description : "List of K-V pairs passed
directly to connect SL docker container to specific network.
                                                This option is needed to start
the SL container with a static IP. The IP address is
                                                passed using 'ip' key inside
slnetworkopts. This will get converted to ipv4_address
                                                argument of the python docker
sdk network connect method. This is same as passing '--ip'
                                                argument to the docker cli run
command.
                                                "
                                  items :
                                      type : object
                                      uniqueItems : true

                        usrhostname :
                            oneOf :
                                - const    : ~
                                - type     : string
                                  description : "Externally visible IPv4
Address or FQDN of the USR Container"
                                  maxLength   : 240
                                  minLength   : 2
                        usrenvvars   :
                            oneOf       :
                                - const    : ~
                                - type     : array
                                  description : "List of local environmental
variables as K-V pair specific to USR Container.
                                                 This will take precedence
over global environmental variables specified in
                                                 run task definition."
                                  items       :
                                      type        : object
                                      uniqueItems : true

                        usrlabels:
                            oneOf:
                              - const: ~
                              - type: object
                                description: "Dictionary of labels for the
user ML container"
                                items:
                                    uniqueItems: true
                              - type: array
                                description: "List of labels with empty values
for the user ML container"
                                items:
                                    type: string
                                    uniqueItems: true
                        # This attribute helps in starting ML container with
pre-defined python docker network SDK options called "ipv4_address"
                        # Added as part of reverse proxy multi-host scenario
where instead of using hostIP with in nginx configuration we can
                        # now use continer IP's for easier and simpler IP and
port mapping.
                        usrnetworkopts   :
                            oneOf       :
                                - const    : ~
                                - type     : array
                                  description : "List of K-V pairs passed
directly to connect user docker container to specific network.
                                                This option is needed to start
the user container with a static IP. The IP address is
                                                passed using 'ip' key inside
usrnetworkopts. This will get converted to ipv4_address
                                                argument of the python docker
sdk network connect method. This is same as passing '--ip'
```

```
                                              argument to the docker cli run
command.
                                              "
                              items :
                                  type : object
                                  uniqueItems : true
                    usrcontaineropts   :
                        oneOf       :
                            - const   : ~
                            - type    : array
                              description : |
                                               "List of K-V pairs passed
directly to docker while starting USR container.
                                               These options are needed to
use GPUs for user ML local training.
                                               These options are specific to
GPU vendors. Refer options as applicable to your GPU vendor."

                                               1. Nvidia GPUs -
                                               "Only 'gpus' key is
supported. Refer below link to know how to specify the values.
                                               https://docs.docker.com/
config/containers/resource_constraints/#gpu "

                                               2. AMD GPUs -
                                               "Keys supported are 'device',
'ipc', 'shm-size', 'group-add',
                                               'cap-add', 'security-opt',
'privileged'.

                                               Refer below link to know the
options for AMD GPU access.
                                               https://developer.amd.com/
resources/rocm-learning-center/deep-learning/

                                               Required options varies for
tensorflow and pytorch.
                                               Tensorflow ====>> sudo docker
run -it --network=host --device=/dev/kfd --device=/dev/dri
                                               --ipc=host --shm-size 16G --
group-add video --cap-add=SYS_PTRACE
                                               --security-opt
seccomp=unconfined -v $HOME/dockerx:/dockerx rocm/tensorflow:latest
                                               PyTorch ====>> sudo docker
run -it -v $HOME:/data --privileged --rm --device=/dev/kfd
                                               --device=/dev/dri --group-add
video rocm/pytorch:rocm3.5_ubuntu16.04_py3.6_pytorch"


                                               Example for specifing values
-
                                               - device : ["/dev/kfd", "/dev/
dri"] # (list of str) -> Expose host devices to the container,

        as a list of strings
                                               - ipc :
"host"                    # (str) -> Set the IPC mode for the container
```

```
                                                  - shm-size :
"16G"                  # (str) -> Size of /dev/shm (e.g. 1G)
                                                  - group-add :
["video"]             # (list of str) -> List of additional group names

        and/or IDs that the container process will run as.
                                                  - cap-add :
["SYS_PTRACE"]          # (list of str) -> Add kernel capabilities.
                                                  - security-opt :
["seccomp=unconfined"] # (list of str) -> A list of string values to
customize labels

            for MLS systems, such as SELinux.
                                                  - privileged :
True                  # (bool) -> Give extended privileges to this container.

                                items :
                                    type : object
                                    uniqueItems : true
                  required   :
                      - idx
                      - identity
                      - slhostname
                      - privatedata
                      - slenvvars
                      - usrhostname
                      - usrenvvars
                  additionalProperties: false
        uniqueItems : true

required    :
    - groupname
    - taskrunner
    - policyuri
    - resourcemgr
    - network
    - apls
    - apisrv
    - envvars
    - nodes
additionalProperties: false
```

SWOP profile schema example

```
######################################################################
## (C)Copyright 2021 Hewlett Packard Enterprise Development LP
######################################################################
groupname  : demo
taskrunner : defaulttaskbb.taskdb.sml.hpe
policyuri  : ~
resourcemgr :
    mgrtype    : DOCKER
    accessinfo :
        accesstype : FROMENV
network    : host-1-net
apls :
    locator :
        host : 172.1.1.1
        port : ~
apisrv :
    locator :
        host : 172.1.1.1
        port : ~
# ENVVARs passed to the SL Container at start up
envvars :
    - SL_LOG_LEVEL : INFO
    - http_proxy : "http://web-proxy.x.y.z:8080"
    - https_proxy : "http://web-proxy.x.y.z:8080"
    - no_proxy : ~
    - HTTP_PROXY : ~
    - HTTPS_PROXY : ~
    - NO_PROXY : ~
# Docker image name and tag for the SL containers
# will come from tasks pre-reqs
# Docker user image name and tag for the USR containers
# will come from tasks pre-reqs

# Detail on the number and specification of each SL NODE
# this profile will create.
nodes :
    - slnodedef :
        idx : 0
        identity :
            - attribute :
                aType : KEY
                mType : BIND
                src : "<CURRENT-PATH>/workspace/mnist-pyt/cert/sl-1-key.pem"
                tgt : "/swarm-cert/sl-1-key.pem"
                subPath : null
            - attribute :
                aType : CERT
                mType : BIND
                src : "<CURRENT-PATH>/workspace/mnist-pyt/cert/sl-1-cert.pem"
                tgt : "/swarm-cert/sl-1-cert.pem"
                subPath : null
            - attribute :
                aType : CAPATH
                mType : BIND
                src : "<CURRENT-PATH>/workspace/mnist-pyt/cert/ca/capath"
                tgt : "/swarm-cert/capath"
                subPath : null
        slhostname : sl1
        slhostip   : 172.1.1.1
        slport : 16000
        usrhostname : user1
        privatedata :
            src: "<CURRENT-PATH>/workspace/mnist-pyt/user1/data-and-scratch"
            mType : BIND
        slenvvars : null
        usrenvvars : null
        usrcontaineropts :
            - gpus : "all" #To run on all available GPUs
    - slnodedef :
        idx : 1
        identity :
            - attribute :
                aType : KEY
                mType : BIND
                src : "<CURRENT-PATH>/workspace/mnist-pyt/cert/sl-1-key.pem"
                tgt : "/swarm-cert/sl-2-key.pem"
                subPath : null
            - attribute :
                aType : CERT
                mType : BIND
                src : "<CURRENT-PATH>/workspace/mnist-pyt/cert/sl-1-cert.pem"
                tgt : "/swarm-cert/sl-2-cert.pem"
                subPath : null
            - attribute :
                aType : CAPATH
                mType : BIND
                src : "<CURRENT-PATH>/workspace/mnist-pyt/cert/ca/capath"
                tgt : "/swarm-cert/capath"
                subPath : null
        slhostname : sl2
        slhostip   : 172.1.1.1
        slport : 17000
        usrhostname : user2
        privatedata :
            src: "<CURRENT-PATH>/workspace/mnist-pyt/user2/data-and-scratch"
            mType : BIND
        slenvvars : null
        usrenvvars : null
        usrcontaineropts :
            - gpus : "device=3,5" #To run on GPUs 3 and 5
```

SWOP profile schema example using reverse proxy parameters

```
######################################################################
## (C)Copyright 2022,2023 Hewlett Packard Enterprise Development LP
######################################################################

groupname: demo
taskrunner: defaulttaskbb.taskdb.sml.hpe
network: <NETWORK-NAME>
policyuri: ~
resourcemgr:
    mgrtype: DOCKER
    accessinfo:
        accesstype: FROMENV
apls:
    locator:
        host: <APLS-IP>
        port: ~
apisrv:
    locator:
        host: ~
        port: ~
        service: ~
envvars:
    - SL_LOG_LEVEL : INFO
    - http_proxy : ~
    - https_proxy : ~
    - no_proxy : ~
    - HTTP_PROXY : ~
    - HTTPS_PROXY : ~
    - NO_PROXY : ~
dns:
    - <Bind9-IP>
    - <Host-DNS-IP>
nodes:
    - slnodedef:
        idx: 0
        identity:
            - attribute:
                aType: KEY
                mType: BIND
                src: "<CURRENT-PATH>/workspace/reverse-proxy/cifar10/cert/sl-1-key.pem"
                tgt: "/swarm-cert/swarm-key.pem"
                subPath: null
            - attribute:
                aType: CERT
                mType: BIND
                src: "<CURRENT-PATH>/workspace/reverse-proxy/cifar10/cert/sl-1-cert.pem"
                tgt: "/swarm-cert/swarm-cert.pem"
                subPath: null
            - attribute:
                aType: CAPATH
                mType: BIND
                src: "<CURRENT-PATH>/workspace/reverse-proxy/cifar10/cert/ca/capath"
                tgt: "/swarm-cert/capath"
                subPath: null
        slhostname: sl-1.swarm
        slhostip: ~
        slport: ~
        slfsservice: fs.sl-1.swarm
        slenvvars: null
        slnetworkopts:
            - ip: <SL-IP-1>
        usrhostname: ml-1
        usrenvvars: null
        usrnetworkopts:
            - ip: <ML-IP-1>
        privatedata : ~
    - slnodedef:
        idx: 1
        identity:
            - attribute:
                aType: KEY
                mType: BIND
                src: "<CURRENT-PATH>/workspace/reverse-proxy/cifar10/cert/sl-1-key.pem"
                tgt: "/swarm-cert/swarm-key.pem"
                subPath: null
            - attribute:
                aType: CERT
                mType: BIND
                src: "<CURRENT-PATH>/workspace/reverse-proxy/cifar10/cert/sl-1-cert.pem"
                tgt: "/swarm-cert/swarm-cert.pem"
                subPath: null
            - attribute:
                aType: CAPATH
                mType: BIND
                src: "<CURRENT-PATH>/workspace/reverse-proxy/cifar10/cert/ca/capath"
                tgt: "/swarm-cert/capath"
                subPath: null
        slhostname: sl-2.swarm
        slhostip: ~
        slport: ~
        slfsservice: fs.sl-2.swarm
        slenvvars: null
        slnetworkopts:
            - ip: <SL-IP-2>
        usrhostname: ml-2
        usrenvvars: null
        usrnetworkopts:
            - ip: <ML-IP-2>
        privatedata : ~
```

SWOP task

- A Task is a reusable unit of work relevant for SWARM LEARNING Operations. For example, **swarm_mnist_task**, runs the MNIST demo using SL and ML nodes.
- A Task is defined by the user.
- A Task has a prerequisite, a body, and an outcome. Tasks can be used to build on "outcomes" created by other tasks. For example, a build task becomes a prerequisite for a `run-swarm` task that runs a ML program.
- Tasks can be chained together to create linear workflows.
- Tasks are instantiated according to schema specified in the **TASK SCHEMA YAML** file.
- Currently supported Task types are:
    - RUN_SWARM—A task to run an ML program in the Swarm Learning Network.
    - MAKE_SWARM_USER_CONTENT—A task to create a Docker volume that contains artifacts to execute the ML program including the ML program itself.
    - MAKE_USER_CONTAINER—A task with the instructions to build a user Docker container in place.
    - PULL_IMAGE—A task to pull a prebuilt Docker image.

SWOP task schema

NOTE: The source code of the task schema (`SWOP-task-schema.yaml`) is present in the same directory as the User Guide.

```
########################################################################
## (C)Copyright 2021-2022 Hewlett Packard Enterprise Development LP
########################################################################
"$schema"   : "https://json-schema.org/draft/2020-12/schema"
title       : "SWOP TASK definition"
description : "Describes a task structure"
type        : object
# ALL FREE STRINGS are MAX of 160 chars long
# we use utf-8 encoding followed by base64 encoding
# to make strings safe for transmission and storage.
# All string are stored in 256 byte containers in
# the blockchain. Base64 encoding bloats string by
# ~33%. Hence 160 char long utf-8 sting will become
# 213 ,8-bit chars. 160 is chosen so that we have
# some space for internal appends ~32 more chars.
properties  :
    Name     :
        description : "Name of the Task being deployed"
        type        : string
```

```
         maxLength   : 160
         minLength   : 1
     TaskType :
        description  : "Type of the task one the above enum values"
        type         : "string"
        enum         :
            - RUN_SWARM
            - MAKE_SWARM_USER_CONTENT
            - MAKE_USER_CONTAINER
            - PULL_IMAGE
     Author   :
         description : "Author of this task"
         type        : string
         maxLength   : 160
         minLength   : 1
     Prereq   :
         description : "Name of the prerequisite task for this task"
         type        : string
         maxLength   : 160
         minLength   : 1
     Outcome  :
         description : "Identifier of the outcome of this task"
         type        : string
         maxLength   : 160
         minLength   : 1
     Body     :
         description : "Specification of the task, that matches the type of
the task"
         oneOf :
             - type     : object
               description : "FORMAT FOR MAKE_USER_CONTAINER"
               properties  :
                 BuildContext :
                     oneOf:
                         - const  : ~
                         - type   : string
                           description : "Name of the volume containing the
context for this build"
                           # NOTE: contents of this volume will be copied
into the build context
                           # temp dir before that actual build process begins
                           maxLength : 160
                           minLength :
3
                 BuildSteps :
                     type     : array
                     description : "BUILD STEPS FOR MAKE_USER_CONTAINER"
                     items    :
                         type      : string
                         maxLength : 160
                         minLength : 0
                     minItems : 1
                     maxItems : 80
                 BuildType  :
                     type : string
                     description : "Specifies the nature of BuildSteps, IF
INLINE BuildSteps is Contents of docker File, IF REMOTE its an URL pointing
```

```
to the Docker file (currently not implemented)"
                    enum :
                        - INLINE
                        - REMOTE
              required    :
                - BuildContext
                - BuildSteps
                - BuildType
              additionalProperties : false
            - type     : object
              description : "FORMAT FOR RUN_SWARM"
              properties  :
                Command         :
                    type        : string
                    description : "Command string to execute Ex: mnist.py --
minpeers=3"
                    maxLength   : 160
                    minLength   : 1
                Entrypoint      :
                    type        : string
                    description : "Entrypoint used to execute Ex: python3,
bash"
                    maxLength   : 160
                    minLength   : 1
                WorkingDir      :
                    type        : string
                    description : "Absolute path with in the container to
set as Working directory"
                    maxLength   : 160
                    minLength   : 2
                Envvars         :
                    oneOf:
                        - const : ~
                        - type : array
                          description : "List of environmental values as K-V
pair"
                          items :
                            type : object
                            uniqueItems : true
                PrivateContent  :
                    oneOf:
                        - const       : ~
                        - type        : string
                          description : "The mount point the USR container
where the private data will be mounted"
                          maxLength   : 160
                          minLength   : 2
                SharedContent   :
                    oneOf:
                        - const       : ~
                        - type        : array
                          description : "list of user content volume to be
mounted on the user image"
                          items :
                                - type : object
                                  description : "Mount information"
                                  properties  :
                                        Src :
                                            type        : string
                                            description : "Source Volume or
directory"
                                            maxLength   : 160
                                            minLength   : 2
                                        Tgt :
                                            type        : string
                                            description : "destination
directory within container"
                                            maxLength   : 160
                                            minLength   : 2
                                        MType :
                                            type : string
                                            description : "Type of mount to
be used to mount the user data"
                                            enum :
                                                - BIND
                                                -
VOLUME
                                  required :
                                    - Src
                                    - Tgt
                                    - MType
                                  additionalProperties : false
                          maxItems    : 10
                          minItems    : 1
              required     :
                - Command
                - Entrypoint
                - SharedContent
                - WorkingDir
                - Envvars
                - PrivateContent
              additionalProperties : false
            - type      : object
              description : "FORMAT FOR PULL_IMAGE"
              properties  :
                Tag             :
                    type        : string
                    description : "image tag , Ex: latest, 0.3.0, 1.0.0, GA-
release"
                    maxLength   : 160
                    minLength   : 1
                RepoName        :
                    type        : string
                    description : "Name of the repository in which the tag
is defined"
                    maxLength   : 160
                    minLength   : 1
                OrgAndReg       :
                    type        : string
                    description : "Org and Registry under which the
repository is hosted"
                    maxLength   : 160
                    minLength   : 1
                Auth :
                    oneOf:
                        - const       : ~
                        - type        : string
                          description : "Path to custom config.json"
                          maxLength   : 160
                          minLength   : 1
                        - type        : object
                          description : "USERNAME and PASSWORD"
                          properties  :
                            Username :
                                type : string
                                maxLength : 160
                                minLength : 1
                            Password :
                                type : string
                                maxLength : 160
                                minLength : 0
                          required    :
                                - Username
                                - Password
                          additionalProperties : false
              required    :
                - Tag
                - RepoName
                - OrgAndReg
                - Auth
              additionalProperties : false
            - type     : object
              description : "FORMAT FOR MAKE_SWARM_USER_CONTENT"
              properties :
                  ContentType :
                      type : string
                      description : "Purpose of the volume being created"
                      # Note: if contenttype is 'BUILDCONTENT,
                      # only one copy of the volume is made.
                      # if contenttype is 'SWARMCONTENT',
                      # many copies are made depending on SWOP profile.
                      enum :
                        - BUILDCONTENT
                        - SWARMCONTENT
                  OpsList     :
                      description : "Content to populate in model dir or
docker build content dir, etc"
                      type        : array
                      minItems    : 1
                      maxItems    : 20
                      items       :
                          - type       : object
                            properties :
                                Operation :
                                    description  : "Type of the sub-task"
                                    type         : "string"
                                    enum         :
                                        - DOWNLOAD
                                        - EXTRACT
                                Target    :
                                    type        : string
                                    description : "URI to perform the
actions"
                                    maxLength   : 160
                                    minLength   : 1
                                Options  :
                                    type        : array
                                    description : "options that is passed to
OPERATION : format key:value"
                                    # Currently we support "Out" option for
both the tasks
                                    items       :
                                        type  : object
                                    uniqueItems : true
                                    minItems    : 0
                            required    :
                                - Target
                                - Options
                                - Operation
                            additionalProperties : false
              required    :
                - ContentType
                - OpsList
              additionalProperties : false
required    :
    - Name
    - TaskType
    - Author
    - Prereq
    - Outcome
    - Body
additionalProperties : false
```

SWOP task schema examples

Task for running ML nodes

```
Name: swarm_mnist_task
TaskType: RUN_SWARM
Author: HPE-TEST
Prereq: user_env_tf_build_task
Outcome: swarm_mnist_task
Body:
    Command: model/mnist_tf.py
    Entrypoint: python3
    WorkingDir: /tmp/test
    Envvars: ["DATA_DIR": app-data, "MODEL_DIR": model, "MAX_EPOCHS": 2, "MIN_PEERS": 2]
    PrivateContent: /tmp/test/
    SharedContent:
    -   Src: <CURRENT-PATH>/workspace/mnist/model
        Tgt: /tmp/test/model
        MType: BIND
```

Task for building the user container

```
Name: user_env_tf_build_task
TaskType: MAKE_USER_CONTAINER
Author: HPE-TEST
Prereq: ROOTTASK
```

```
Outcome: user-env-tf2.7.0-swop
Body:
    BuildContext: sl-cli-lib    #A docker volume that is created by User.
                                #It should have the Swarm Learning wheel
                                #file copied in to it.
    BuildType: INLINE
    BuildSteps:
    - FROM tensorflow/tensorflow:2.7.0
    - ' '
    - RUN pip3 install --upgrade pip && pip3 install \
    - '   keras matplotlib opencv-python pandas protobuf==3.15.6 '
    - ' '
    - RUN mkdir -p /tmp/hpe-swarmcli-pkg
    - COPY swarmlearning-client-py3-none-manylinux_2_24_x86_64.whl /tmp/hpe-
swarmcli-pkg/swarmlearning-client-py3-none-manylinux_2_24_x86_64.whl
    - RUN pip3 install /tmp/hpe-swarmcli-pkg/swarmlearning-client-py3-none-
manylinux_2_24_x86_64.whl
```

Task for creating a Docker volume of model program and data

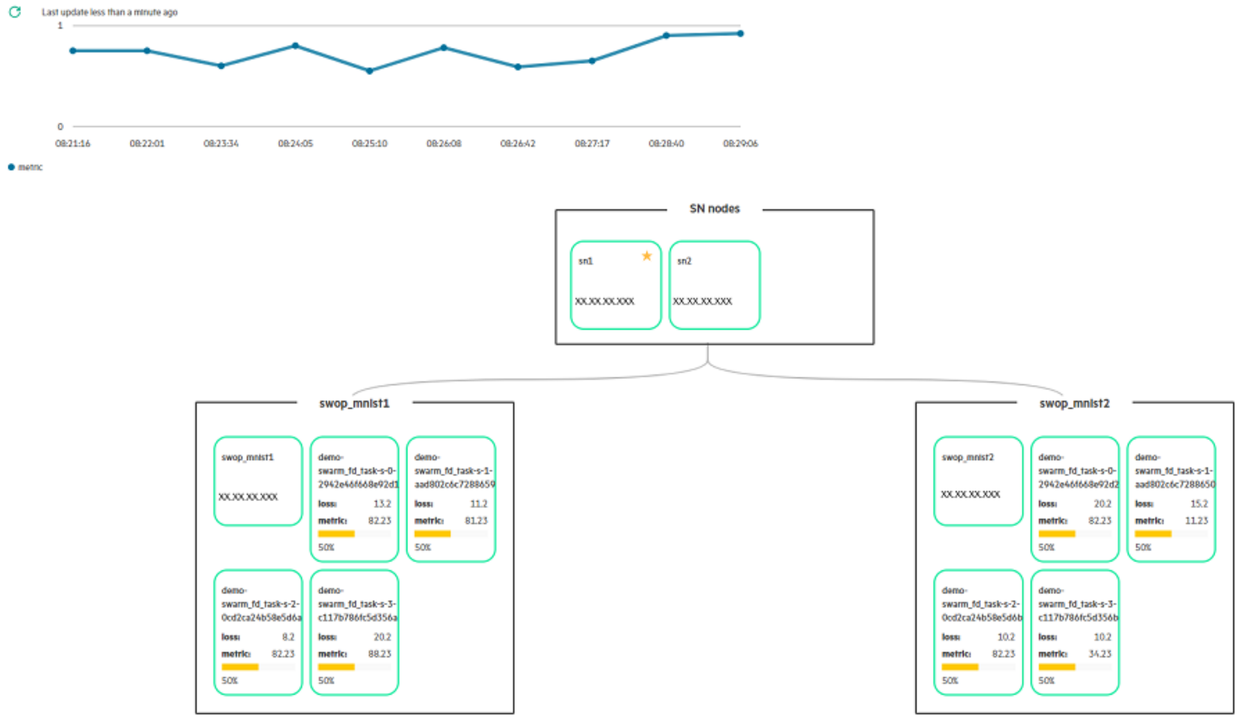
```
Name: download-task-mnist-local
TaskType : MAKE_SWARM_USER_CONTENT
Author   : HPE-TEST
Prereq   : ROOTTASK
Outcome  : mnist-msn
Body     :
    ContentType : SWARMCONTENT
    OpsList :
        - Operation : DOWNLOAD
#          Target    : "https://raw.githubusercontent.com/HewlettPackard/
swarm-learning/master/examples/mnist-keras/model/mnist_tf.py"
          Target    : "http://172.2.2.2:9292/mnist/model/mnist_tf.py"
          Options   :
            - Out   : "model/mnist_tf.py"
        - Operation : DOWNLOAD
#          Target    : "https://github.com/HewlettPackard/swarm-learning/raw/
master/examples/mnist-keras/app-data/mnist.npz"
          Target    : "http://172.2.2.2:9292/mnist/data/mnist.npz"
          Options   :
            - Out   : "data/mnist.npz"
```

Task for pulling a prebuilt Docker image

```
Name: pull-task-tf2
TaskType : PULL_IMAGE
Author   : HPE-TEST
Prereq   : ROOTTASK
Outcome  : "hub.docker.hpecorp.net/hub/tensorflow-tensorflow:2.7.0"
Body     :
    Tag         : "2.7.0"
    RepoName    : "tensorflow-tensorflow"
    OrgAndReg   : "hub.docker.hpecorp.net/hub/"
    Auth        : ~
```

Monitoring Swarm Learning training using SLM-UI

Once the Swarm training has started, user can monitor it as shown in the following image.

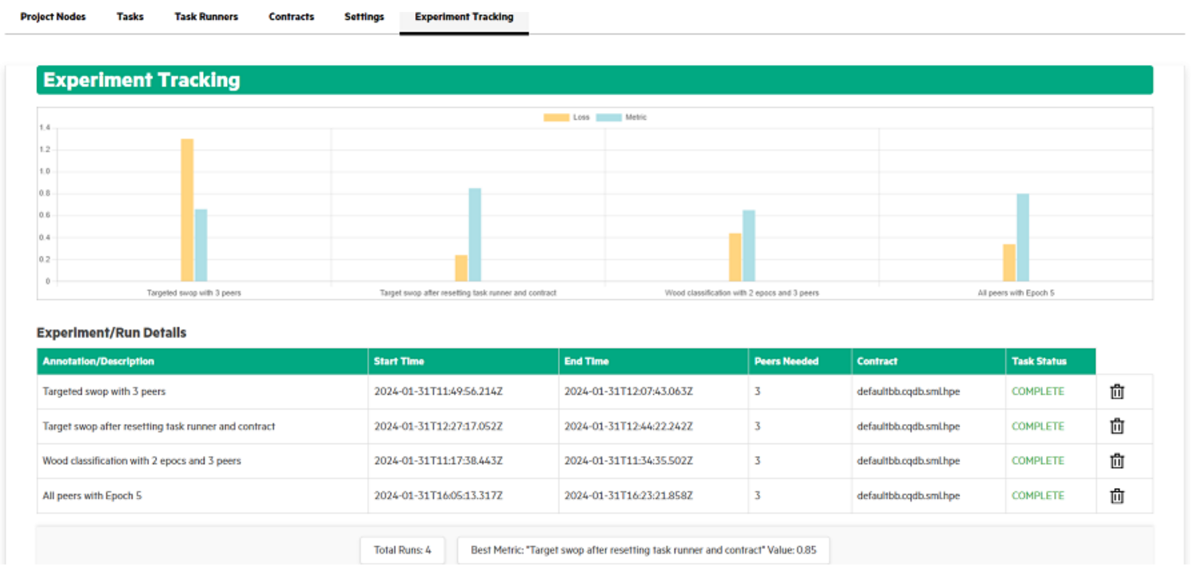


This image shows all running swarm nodes associated with the project. It also shows loss, model metric (for example, accuracy) and overall training progress for each SL-ML pair node. On hovering over the mouse on progress bar, user can view the total number of epochs and the total number of completed epochs.

NOTE: Post training completion, the completed nodes are still displayed for approximately 7 to 10 minutes prior to being cleaned up in the SLM-UI.

Experiment tracking

In 2.2.0 release, experiment tracking feature is introduced to compare different model training runs. This would be useful for doing hyper-parameter tuning and choosing the best run.



Experiment/Run Details

Annotation/Description	Start Time	End Time	Peers Needed	Contract	Task Status	
Targeted swop with 3 peers	2024-01-31T11:49:56.214Z	2024-01-31T12:07:43.063Z	3	defaultbb.cqdb.sml.hpe	COMPLETE	🗑
Target swop after resetting task runner and contract	2024-01-31T12:27:17.052Z	2024-01-31T12:44:22.242Z	3	defaultbb.cqdb.sml.hpe	COMPLETE	🗑
Wood classification with 2 epocs and 3 peers	2024-01-31T11:17:38.443Z	2024-01-31T11:34:35.502Z	3	defaultbb.cqdb.sml.hpe	COMPLETE	🗑
All peers with Epoch 5	2024-01-31T16:05:13.317Z	2024-01-31T16:23:21.858Z	3	defaultbb.cqdb.sml.hpe	COMPLETE	🗑

Total Runs: 4

Best Metric: "Target swop after resetting task runner and contract" Value: 0.85

To know the hyper parameters used and compare the training metrics (like accuracy) and loss of different training runs of an SLM-UI project, user needs to provide the details of a particular experiment in the **Project** > **Tasks** > **Execute task** of the SLM-UI using the following fields.

- **Annotation/Description of the experiment** – This field allows user to enter the description of the experiment. User can provide verbose details and annotate a given experimental run with details of the hyper parameters used in this particular run.
- **Select to save the experiment** – This field allows users to select the checkbox to save the experiment.

Examples

Prerequisites

1. Start license server and install valid license before running any of the examples. For more information, see *HPE Swarm Learning Installation* section in *HPE Swarm Learning Installation and Configuration Guide*.
2. Install the Swarm Learning product using the Web UI installer.

For more information on starting license server and installing the Swarm Learning, see *HPE Swarm Learning Installation and Configuration Guide*.

In this section, examples use different models, data, ML platforms, and Swarm cluster configurations. All examples require valid X.509 certificates to be used by different Swarm Learning components.

NOTE: SPIRE based example is automated and does not need any certificates to be generated. SPIRE agent and server containers manage the certificates internally.

A certificate generation utility (`gen-cert`) is provided with each example to enable users to run the examples quickly, using self-signed certificates.

NOTE: HPE recommends that users must use their own certificates in actual production environment.

Swarm Learning Examples

The following four examples are provided with the package for using Swarm Learning:

1. MNIST (`swarm-learning/examples/mnist`)
2. MNIST-PYT (`swarm-learning/examples/mnist-pyt`)
3. CIFAR-10 (`swarm-learning/examples/cifar10`)
4. Credit card fraud detection (`swarm-learning/examples/fraud-detection`)
5. Reverse Proxy based examples (`swarm-learning/examples/reverse-proxy`)
6. SPIRE based example (`swarm-learning/examples/spire/cifar10`)

Each example uses different models, data, platforms, and Swarm cluster configurations. It requires valid certificates to be used by different Swarm Learning components. A certificate generation utility (`gen-cert`) is provided under the `swarm-learning/examples/utils` folder so that users can quickly generate certificates and run the examples.

```
./swarm-learning/examples/utils/gen-cert -h
Usage: gen-cert -e EXAMPLE-NAME -i HOST-INDEX
        -e Name of the example e.g. mnist, mnist-pyt, cifar10 etc.
        -i Host index like 1,2 etc.
        -h Show help.
```

NOTE: HPE strongly recommends that users must use their own certificates in an actual production environment.

System setup for the examples

1. The instructions in these examples assume that Swarm Learning runs on 1 to 2 host systems.

- These systems have IP addresses 172.1.1.1 and 172.2.2.2.
- License server is installed on 172.1.1.1 and running on default port 5814.
- 172.1.1.1 runs the Sentinel SN node.
- MNIST example has one more SN node. 172.2.2.2 runs the other SN node. A SL and ML node pair are spawned across these 2 host systems - 172.1.1.1 and 172.2.2.2.
- Spawning of SL and ML nodes can be automatic or manual.
- SWOP node, which runs on each host system, is automatically spawned using the task framework. MNIST example shows how Swarm training can be automatically launched using SWCI and SWOP nodes. SWCI runs on 172.1.1.1 host, while Swarm Operator runs on both 172.1.1.1 and 172.2.2.2 hosts.
- CIFAR-10 example shows how Swarm training can be manually launched using `run-sl` script on each host.
- Model training starts once minimum number (`minPeers`) of specified ML nodes are launched, either automatically or manually.
- After training, the final Swarm model is saved by each ML node.

2. The files created under the workspace directory that include certs, models, data, etc., are expected to have the minimum file permission as 664. Once the files are copied to the workspace directory, check the permissions of the files inside it. If desired permissions are not met, user might observe `file permission denied` in the respective swarm component's Docker logs. To overcome such cases, user can change the permissions of the files by using the `chmod` command.
3. Finally, these instructions assume that `swarm-learning` is the current working directory on all the systems: `cd swarm-learning`

MNIST

This example runs MNIST[1] on the Swarm Learning platform. It uses TensorFlow as the backend. The code for this example is taken from[2] and modified to run on a Swarm Learning platform.

This example uses one training batch and one test batch. The files for both these batches are in an archive file, called `mnist.npz`.

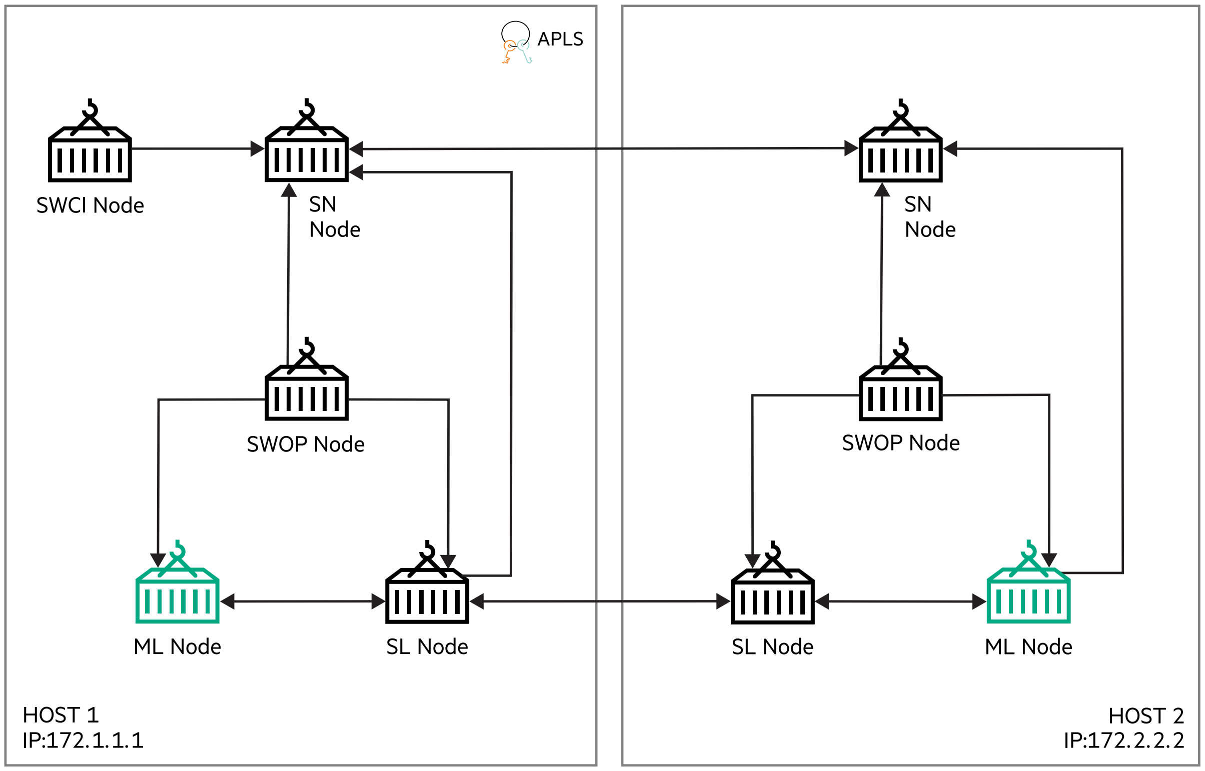
NOTE: See data license associated with this dataset, `examples/mnist/app-data/mnist-npz.md`.

The Machine Learning program, after conversion to Swarm Learning for the TensorFlow-based Keras platform, is in `examples/mnist/model` and the file name is `mnist_tf.py`.

This example shows the Swarm training of MNIST model using two ML nodes. ML nodes are automatically spawned by SWOP nodes running on two different hosts. Swarm training is initiated by SWCI node and orchestrated by two SN nodes running in different hosts. This example also shows how private data and shared model can be mounted to ML nodes for Swarm training.

1 **https://yann.lecun.com/exdb/mnist/**
2 **https://www.tensorflow.org/tutorials/quickstart/beginner**

The following image illustrates a cluster setup for the MNIST example:



- This example uses two SN nodes. The names of the Docker containers representing these two nodes are SN1 and SN2. SN1 is the Sentinel Node. SN1 runs on host 172.1.1.1 and SN2 runs on host 172.2.2.2.
- SL and ML nodes are automatically spawned by SWOP nodes during training and removed after training. This example uses two SWOP nodes – one connects to each SN node. The names of the docker containers representing these two SWOP nodes are SWOP1 and SWOP2. SWOP1 runs on host 172.1.1.1 and SWOP2 runs on host 172.2.2.2.
- Training is initiated by SWCI node that runs on host 172.1.1.1.
- This example assumes that License Server already runs on host 172.1.1.1. All Swarm nodes connect to the License Server, on its default port 5814.

Running the MNIST example

1. On both host-1 and host-2, navigate to swarm-learning folder (that is, parent to `examples` directory).

   ```
   cd swarm-learning
   ```

2. On both host-1 and host-2, create a temporary `workspace` directory, copy `mnist` example, and `gen-cert` utility.

   ```
   mkdir workspace
   cp -r examples/mnist workspace/
   cp -r examples/utils/gen-cert workspace/mnist/
   ```

3. On both host-1 and host-2, run the gen-cert utility to generate certificates for each Swarm component using the command (`gen-cert -e <EXAMPLE-NAME> -i <HOST-INDEX>`):

   On host-1:

   ```
   ./workspace/mnist/gen-cert -e mnist -i 1
   ```

On host-2:

```
./workspace/mnist/gen-cert -e mnist -i 2
```

4. On both host-1 and host-2, share the CA certificates between the hosts as follows:

   On host-1:

```
scp host-2:<PATH>workspace/mnist/cert/ca/capath/ca-2-cert.pem workspace/mnist/cert/ca/
capath
```

   On host-2:

```
scp host-1:<PATH>workspace/mnist/cert/ca/capath/ca-1-cert.pem workspace/mnist/cert/ca/
capath
```

5. On both host-1 and host-2, create a Docker network for SN, SWOP, SWCI, SL, and user containers running in a host.

   On host-1:

```
docker network create host-1-net
```

   On host-2:

```
docker network create host-2-net
```

6. On both host-1 and host-2, declare and assign values to the variables such as `APLS_IP`, `SN_IP`, `HOST_IP`, and `SN_API_PORT`. For example:

```
APLS_IP=172.1.1.1
SN_1_IP=172.1.1.1
SN_2_IP=172.2.2.2
HOST_1_IP=172.1.1.1
HOST_2_IP=172.2.2.2
SN_API_PORT=30304
SN_P2P_PORT=30303
```

   NOTE: You must use the appropriate values to these variables as per your Swarm network.

7. On both host-1 and host-2, search and replace all occurrences of placeholders and replace them with appropriate values.

```
sed -i "s+<PROJECT-MODEL>+$(pwd)/workspace/mnist/model+g" workspace/mnist/swci/taskdefs/
swarm_mnist_task.yaml
sed -i "s+<SWARM-NETWORK>+host-1-net+g" workspace/mnist/swop/swop1_profile.yaml
sed -i "s+<SWARM-NETWORK>+host-2-net+g" workspace/mnist/swop/swop2_profile.yaml
sed -i "s+<HOST_ADDRESS>+${HOST_1_IP}+g" workspace/mnist/swop/swop1_profile.yaml
sed -i "s+<HOST_ADDRESS>+${HOST_2_IP}+g" workspace/mnist/swop/swop2_profile.yaml
sed -i "s+<LICENSE-SERVER-ADDRESS>+${APLS_IP}+g" workspace/mnist/swop/swop*_profile.yaml
sed -i "s+<PROJECT>+$(pwd)/workspace/mnist+g" workspace/mnist/swop/swop*_profile.yaml
sed -i "s+<PROJECT-CERTS>+$(pwd)/workspace/mnist/cert+g" workspace/mnist/swop/swop*_profile.yaml
sed -i "s+<PROJECT-CACERTS>+$(pwd)/workspace/mnist/cert/ca/capath+g" workspace/mnist/swop/
swop*_profile.yaml
```

   NOTE: If the Swarm installation directory is different for both hosts, you must manually modify values from the `swarm_mnist_task.yaml` file to the path that is common to both hosts.

8. On both host-1 and host-2, create a Docker volume and copy Swarm Learning wheel file.

```
docker volume rm sl-cli-lib
docker volume create sl-cli-lib
docker container create --name helper -v sl-cli-lib:/data hello-world
docker cp lib/swarmlearning-client-py3-none-manylinux_2_24_x86_64.whl helper:/
data
docker rm helper
```

9. On host-1, run SN node (SN1).

   ```
   ./scripts/bin/run-sn -d --rm --name=sn1 \
   --network=host-1-net --host-ip=${HOST_1_IP} \
   --sentinel --sn-p2p-port=${SN_P2P_PORT} \
   --sn-api-port=${SN_API_PORT} \
   --key=workspace/mnist/cert/sn-1-key.pem \
   --cert=workspace/mnist/cert/sn-1-cert.pem \
   --capath=workspace/mnist/cert/ca/capath \
   --apls-ip=${APLS_IP}
   ```

   Use the Docker logs command to monitor the Sentinel SN node and wait for the node to finish initializing. The Sentinel node is ready when these messages appear in the log output:

   ```
   swarm.blCnt : INFO : Starting SWARM-API-SERVER on port: 30304
   ```

   On host-2, run SN node (SN2).

   ```
   ./scripts/bin/run-sn -d --rm --name=sn2 \
   --network=host-2-net --host-ip=${HOST_2_IP} \
   --sentinel-ip=${SN_1_IP} --sn-p2p-port=${SN_P2P_PORT} \
   --sn-api-port=${SN_API_PORT} --key=workspace/mnist/cert/sn-2-key.pem \
   --cert=workspace/mnist/cert/sn-2-cert.pem \
   --capath=workspace/mnist/cert/ca/capath \
   --apls-ip=${APLS_IP}
   ```

10. On host-1, run SWOP node (SWOP1).

    NOTE: If required, modify proxy, according to environment, either in the following command or in the SWOP profile files under `workspace/mnist/swop` folder.

    ```
    ./scripts/bin/run-swop -d --rm --name=swop1 --network=host-1-net \
    --sn-ip=${SN_1_IP} --sn-api-port=${SN_API_PORT} \
    --usr-dir=workspace/mnist/swop --profile-file-name=swop1_profile.yaml \
    --key=workspace/mnist/cert/swop-1-key.pem \
    --cert=workspace/mnist/cert/swop-1-cert.pem \
    --capath=workspace/mnist/cert/ca/capath -e http_proxy= -e https_proxy= \
    --apls-ip=${APLS_IP}
    ```

    On host-2, run SWOP node (SWOP2).

    NOTE: If required, modify proxy, according to environment, either in the following command or in the SWOP profile files under `workspace/mnist/swop` folder.

    ```
    ./scripts/bin/run-swop -d --rm --name=swop2 --network=host-2-net \
    --sn-ip=${SN_2_IP} --sn-api-port=${SN_API_PORT} \
    --usr-dir=workspace/mnist/swop --profile-file-name=swop2_profile.yaml \
    --key=workspace/mnist/cert/swop-2-key.pem \
    --cert=workspace/mnist/cert/swop-2-cert.pem \
    --capath=workspace/mnist/cert/ca/capath -e http_proxy= -e https_proxy= \
    --apls-ip=${APLS_IP}
    ```

11. On host-1, run SWCI node. It creates, finalizes, and assigns the following tasks to task-framework for sequential execution:

    - `user_env_tf_build_task` - builds TensorFlow based ML nodes with model and data.
    - `swarm_mnist_task` - run Swarm training across for two ML nodes.

NOTE: If required, according to the environment, modify SN IP in `workspace/mnist/swci/swci-init` file.

```
./scripts/bin/run-swci --rm --name=swci1 --network=host-1-net \
--usr-dir=workspace/mnist/swci --init-script-name=swci-init \
--key=workspace/mnist/cert/swci-1-key.pem \
--cert=workspace/mnist/cert/swci-1-cert.pem \
--capath=workspace/mnist/cert/ca/capath   \
-e http_proxy= -e https_proxy= --apls-ip=${APLS_IP}
```

12. On both host-1 and host-2, two nodes of Swarm trainings are automatically started when the run task (`swarm_mnist_task`) gets assigned and executed. Open a new terminal on both host-1 and host-2 and monitor the docker logs of ML nodes for Swarm training. Swarm training will end with the following log message:

    ```
    SwarmCallback : INFO : All peers and Swarm training rounds finished. Final Swarm model was
    loaded.
    ```

    Final Swarm model is saved inside `<PROJECT>` location which will likely be `workspace/mnist/model` directory on both the hosts. All the dynamically spawned SL and ML nodes exit after Swarm training. The SN and SWOP nodes continue to run.

13. On both host-1 and host-2, to clean up, run the `scripts/bin/stop-swarm` script on all the systems to stop and remove the container nodes of the previous run. If required, backup the container logs. Remove Docker networks (`host-1-net` and `host-2-net`) and Docker volume (`sl-cli-lib`), and delete the `workspace` directory.

Running MNIST example using SLM-UI

About this task

Perform the following steps to run the MNIST two node example using SLM-UI.

Procedure

1. Add a Swarm host.

   This step is optional. User can use this step to install Swarm on hosts where they want to run this project. This example needs two hosts [ip1 and ip2]. For more information on adding a host in SLM-UI, see *Installing Swarm Learning using SLM-UI* section in *HPE Swarm Learning Installation and Configuration Guide*.

2. Create a project in SLM-UI.

   This example can be downloaded from `example/mnist` folder. For more information on creating a project in SLM-UI, see *Creating a Project in SLM-UI* section in *HPE Swarm Learning Installation and Configuration Guide*.

3. Create SN (sentinel) node by selecting `sn` node in the **Add Swarm Node** screen. Make sure that the sentinel node and the SWCI are started properly from the logs.

   a. Go to **Host** tab and click on **sentinel IP**. It shows all running containers.

   b. Click on the SN container to view the logs.

   The SWCI container starts automatically. User does not have to create it manually. User can see the container while clicking on the host where they added the sentinel node. User needs to provide all information as mentioned in the *Adding Swarm Nodes* section in *HPE Swarm Learning Installation and Configuration Guide* to add [ip1].

4. Create a second SN node using the sentinel SN IP address from the dropdown.

5. In the **Create Node** section drop down, select SWOP [ip1] to create SWOP in first node.

6. In the **Create Node** section drop down, select SWOP [ip2] to create SWOP in second node.

7. Create `MAKE_USER_CONTAINER` task.

   For more information, see *Creating a task* section in *HPE Swarm Learning Installation and Configuration Guide*.

8. Create `RUN_SWARM` task.

   For more information, see *Creating a task* section in *HPE Swarm Learning Installation and Configuration Guide*.

9. Execute `MAKE_USER_CONTAINER` task with 2 peers.

   For more information, see *Executing a Task* section in *HPE Swarm Learning Installation and Configuration Guide*.

10. Execute `RUN_SWARM` task with 2 peers.

    For more information, see *Executing a Task* section in *HPE Swarm Learning Installation and Configuration Guide*.

11. Click on the project to view topology.

12. Check the task runner to view the status of the task.

MNIST-PYT

The MNIST-PYT example branches out to show case the following CPU based and GPU based (AMD, NVIDIA) local training of machine learning application:

- **CPU based MNIST-PYT**
- **AMD GPU based MNIST-PYT**
- **NVIDIA GPU based MNIST-PYT**

CPU based MNIST-PYT

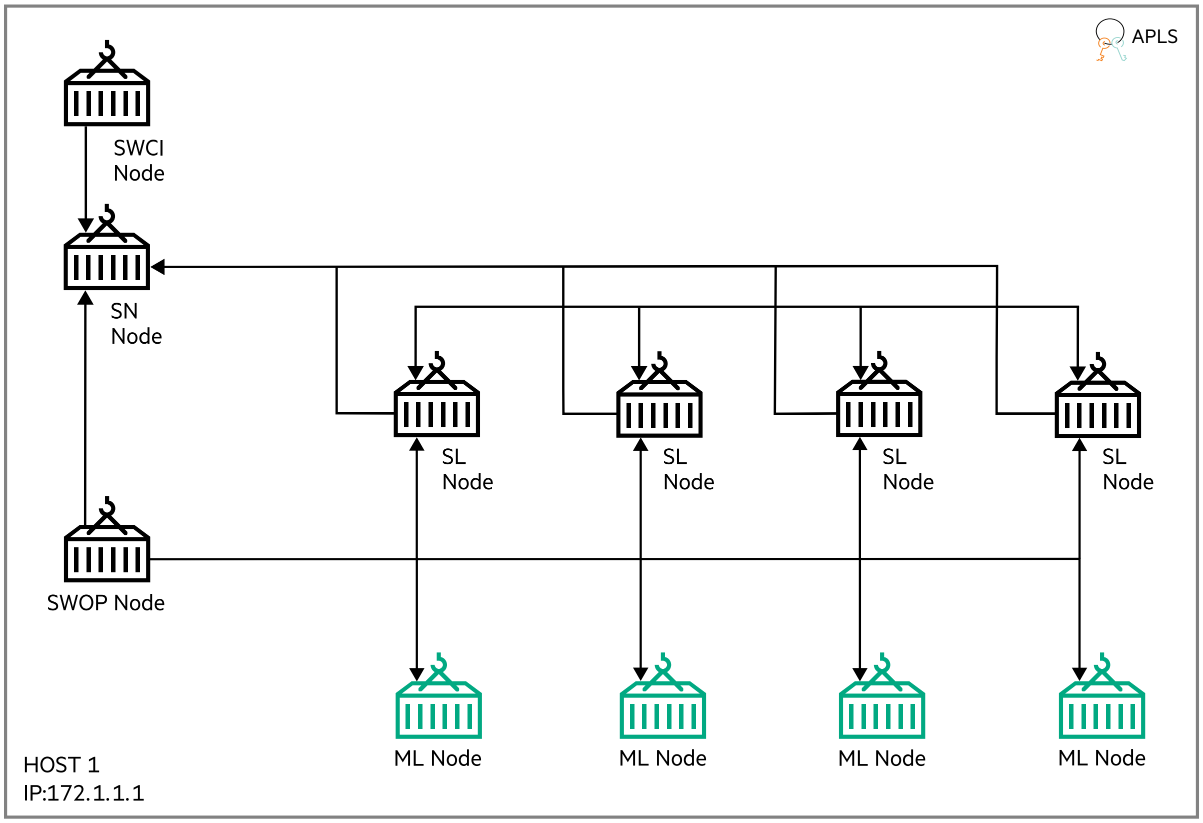
This example runs MNIST[3]application with CPU based local training on the Swarm Learning platform.

The Machine Learning program, after conversion to Swarm Learning for the PyTorch platform, is in `examples/mnist-pyt/cpu-based/model`. The PyTorch-based file is called `mnist_pyt.py`.

This example shows the Swarm training of MNIST model using four ML nodes. ML nodes along with SL nodes are automatically spawned by SWOP nodes running on two different hosts. Swarm training is initiated by SWCI node and orchestrated by two SN nodes running in different hosts. This example also shows how private data, private scratch area and shared model can be mounted to ML nodes for Swarm training.

[3] **https://yann.lecun.com/exdb/mnist/**

The following image illustrates a cluster setup for the MNIST example:



- This example uses one SN node. The name of the docker containers representing this node is SN1. SN1 is also the Sentinel Node. SN1 runs on the host 172.1.1.1.
- Four SL and ML nodes are automatically spawned by SWOP node during training and removed after the training. This example uses one SWOP node that connects to the SN node. The name of the docker container representing this SWOP node is SWOP1. SWOP1 runs on the host 172.1.1.1.
- Training is initiated by SWCI node that runs on host 172.1.1.1.
- This example assumes that License Server already runs on host 172.1.1.1. All Swarm nodes connect to the License Server, on its default port 5814.

Running the MNIST-PYT example

1. Navigate to `swarm-learning` folder (that is, parent to examples directory).

   ```
   cd swarm-learning
   ```

2. Create a temporary `workspace` directory and copy `mnist-pyt` example.

   ```
   mkdir workspace
   cp -r examples/mnist-pyt/cpu-based/ workspace/mnist-pyt/
   cp -r examples/utils/gen-cert workspace/mnist-pyt/
   ```

3. Run the gen-cert utility to generate certificates for each Swarm component using the command (`gen-cert -e <EXAMPLE-NAME> -i <HOST-INDEX>`):

   ```
   ./workspace/mnist-pyt/gen-cert -e mnist-pyt -i 1
   ```

4. Create a docker network *'host-1-net'*for SN, SWOP, SWCI, SL, and user containers running in a host. Ignore this step if the network is already created.

```
docker network create host-1-net
```

5. Declare and assign values to the variables such as `APLS_IP`, `SN_IP`, `HOST_IP`, and `SN_API_PORT`. For example:

```
APLS_IP=172.1.1.1
SN_IP=172.1.1.1
HOST_IP=172.1.1.1
SN_API_PORT=30304
SN_P2P_PORT=30303
```

NOTE: You must use appropriate values as per your Swarm network.

6. Search and replace all occurrences of placeholders and replace them with appropriate values.

```
sed -i "s+<PROJECT-MODEL>+$(pwd)/workspace/mnist-pyt/model+g" workspace/mnist-pyt/swci/taskdefs/run_mnist_pyt.yaml
sed -i "s+<SWARM-NETWORK>+host-1-net+g" workspace/mnist-pyt/swop/swop*_profile.yaml
sed -i "s+<LICENSE-SERVER-ADDRESS>+${APLS_IP}+g" workspace/mnist-pyt/swop/swop*_profile.yaml
sed -i "s+<PROJECT>+$(pwd)/workspace/mnist-pyt+g" workspace/mnist-pyt/swop/swop*_profile.yaml
sed -i "s+<PROJECT-CERTS>+$(pwd)/workspace/mnist-pyt/cert+g" workspace/mnist-pyt/swop/swop*_profile.yaml
sed -i "s+<PROJECT-CACERTS>+$(pwd)/workspace/mnist-pyt/cert/ca/capath+g" workspace/mnist-pyt/swop/
swop*_profile.yaml
```

7. Create a docker volume and copy Swarm Learning wheel file:

```
docker volume rm sl-cli-lib
docker volume create sl-cli-lib
docker container create --name helper -v sl-cli-lib:/data hello-world
docker cp lib/swarmlearning-client-py3-none-manylinux_2_24_x86_64.whl helper:/
data
docker rm helper
```

8. Run SN node (SN1 - sentinel node)

```
./scripts/bin/run-sn -d --rm --name=sn1 --network=host-1-net \
--host-ip=${HOST_IP} --sentinel --sn-p2p-port=${SN_P2P_PORT} \
--sn-api-port=${SN_API_PORT} --key=workspace/mnist-pyt/cert/sn-1-key.pem \
--cert=workspace/mnist-pyt/cert/sn-1-cert.pem \
--capath=workspace/mnist-pyt/cert/ca/capath --apls-ip=${APLS_IP}
```

Use the Docker logs command to monitor the Sentinel SN node and wait for the node to finish initializing. The Sentinel node is ready when these messages appear in the log output:

```
swarm.blCnt : INFO : Starting SWARM-API-SERVER on port: 30304
```

9. Run SWOP node (SWOP1).

NOTE: If required, according to environment, modify IP and proxy in the following command or in the SWOP profile file under `workspace/mnist-pyt/swop` folder.

```
./scripts/bin/run-swop -d --rm --name=swop1 --network=host-1-net \
--sn-ip=${SN_IP} --sn-api-port=${SN_API_PORT} \
--usr-dir=workspace/mnist-pyt/swop --profile-file-name=swop_profile.yaml \
--key=workspace/mnist-pyt/cert/swop-1-key.pem \
--cert=workspace/mnist-pyt/cert/swop-1-cert.pem \
--capath=workspace/mnist-pyt/cert/ca/capath -e SWOP_KEEP_CONTAINERS=True \
-e http_proxy= -e https_proxy= --apls-ip=${APLS_IP}
```

NOTE: –e `SWOP_KEEP_CONTAINERS=True` is an optional argument, by default, it is False.

`SWOP_KEEP_CONTAINERS` is set to True so that SWOP does not remove stopped SL and ML containers. Without this setting, if there are any internal errors in SL or ML, SWOP removes them automatically. For more information, see the *Environment Variables* section, *HPE Swarm Learning Installation and Configuration Guide*.

10. Run SWCI node and observe sequential execution of two tasks – build task (`build_pyt_user_image`) and run task (`run_mnist_pyt`).

```
./scripts/bin/run-swci -ti --rm --name=swci1 --network=host-1-net \
--usr-dir=workspace/mnist-pyt/swci --init-script-name=swci-init \
--key=workspace/mnist-pyt/cert/swci-1-key.pem \
--cert=workspace/mnist-pyt/cert/swci-1-cert.pem \
--capath=workspace/mnist-pyt/cert/ca/capath \
-e http_proxy= -e https_proxy= --apls-ip=${APLS_IP}
```

- `build_pyt_user_image` - builds pytorch based user image.
- `run_mnist_pyt` - runs Swarm training across for four ML nodes.

NOTE: If required, according to the environment, modify SN IP in `workspace/mnist-pyt/swci/swci-init` file.

11. Four nodes of Swarm trainings are automatically started when the run task (`run_mnist_pyt`) gets assigned and executed. Open a new terminal on both host-1 and host-2 and monitor the Docker logs of ML nodes for Swarm training. Swarm training ends with the following log message:

```
SwarmCallback : INFO : All peers and Swarm training rounds finished. Final Swarm model was
loaded.
```

Final Swarm model is saved inside each user's specific directory in `workspace/mnist-pyt/<userN>`. All the dynamically spawned SL and ML containers exits after Swarm training if `SWOP_KEEP_CONTAINERS` is not set, otherwise SL and ML containers must be removed manually. The SN and SWOP nodes continues to run.

12. To clean up, run the `scripts/bin/stop-swarm` script on all the systems to stop and remove the container nodes of the previous run. If required, backup the container logs. Remove Docker networks (`host-1-net`) and Docker volume (`sl-cli-lib`), and delete the `workspace` directory.

AMD GPU based MNIST-PYT

This example runs MNIST-PYT[4] application with AMD GPU based local training on the Swarm Learning platform.

When compared to CPU based `mnist-pyt` example following are the key differences in this example:

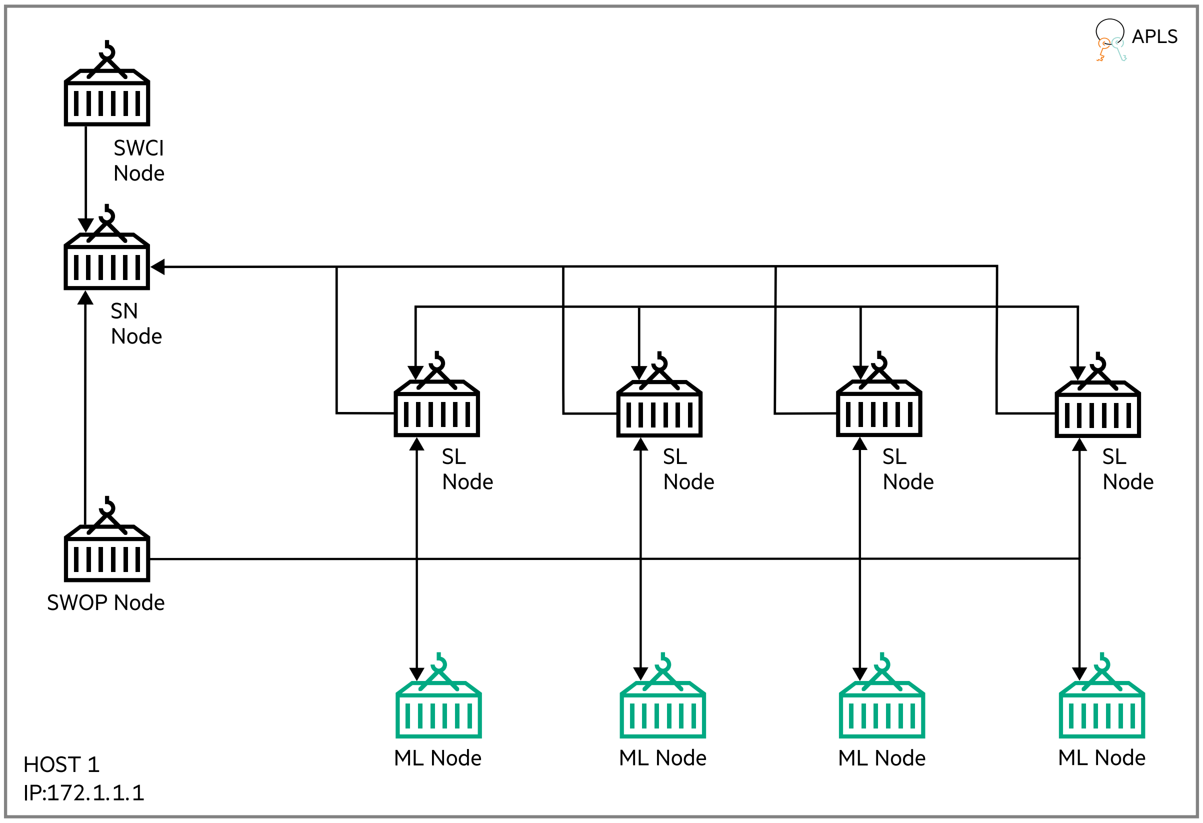
- User image build uses `rocm/pytorch` image as base image. Base image needs to be selected such that it support the host AMD setup. User image build uses `rocm/pytorch` image as base image. Base image needs to be selected such that it support the host AMD setup. For more information, see **https://hub.docker.com/r/rocm/pytorch**.
- SWOP options needs additional tags to access AMD GPUs. For more information, see **Launch Swarm Learning using SWOP**.
- Run task command needs additional argument to be passed in. See run task yaml in this example.
- User ML application code needs modifications to access AMD GPUs.

[4] **https://yann.lecun.com/exdb/mnist/**

The Machine Learning program, after conversion to Swarm Learning for the PyTorch platform, is in `examples/mnist-pyt/gpu-based/amd/model`. The PyTorch-based file is called `mnist_pyt.py`.

This example shows the Swarm training of MNIST model using four ML nodes. ML nodes along with SL nodes are automatically spawned by SWOP nodes - all running on a single host. Swarm training gets initiated by the SWCI node and orchestrated by one SN node running on the same host. This example also shows how private data, private scratch area, and shared model can be mounted to ML nodes for Swarm training.

The following image illustrates a cluster setup for the MNIST example:



- This example uses one SN node. The name of the docker containers representing this node is SN1. SN1 is also the Sentinel Node. SN1 runs on the host 172.1.1.1.
- Four SL and ML nodes are automatically spawned by SWOP node during training and removed after the training. This example uses one SWOP node that connects to the SN node. The name of the docker container representing this SWOP node is SWOP1. SWOP1 runs on the host 172.1.1.1.
- Training is initiated by SWCI node that runs on host 172.1.1.1.
- This example assumes that License Server already runs on host 172.1.1.1. All Swarm nodes connect to the License Server, on its default port 5814.

Running the MNIST-PYT example

1. Navigate to `swarm-learning` folder (that is, parent to `examples` directory).

   ```
   cd swarm-learning
   ```

2. Create a temporary `workspace` directory and copy `mnist-pyt` example.

   ```
   mkdir workspace
   cp -r examples/mnist-pyt/gpu-based/amd/ workspace/mnist-pyt/
   cp -r examples/utils/gen-cert workspace/mnist-pyt/
   ```

3. Run the gen-cert utility to generate certificates for each Swarm component using the command (`gen-cert -e <EXAMPLE-NAME> -i <HOST-INDEX>`):

   ```
   ./workspace/mnist-pyt/gen-cert -e mnist-pyt -i 1
   ```

4. Create a docker network *'host-1-net'*for SN, SWOP, SWCI, SL, and user containers running in a host. Ignore this step if the network is already created.

   ```
   docker network create host-1-net
   ```

5. Declare and assign values to the variables such as `APLS_IP`, `SN_IP`, `HOST_IP`, and `SN_API_PORT`. For example:

   ```
   APLS_IP=172.1.1.1
   SN_IP=172.1.1.1
   HOST_IP=172.1.1.1
   SN_API_PORT=30304
   SN_P2P_PORT=30303
   ```

   NOTE: You must use appropriate values as per your Swarm network.

6. Search and replace all occurrences of placeholders and replace them with appropriate values.

   ```
   sed -i "s+<PROJECT-MODEL>+$(pwd)/workspace/mnist-pyt/model+g" workspace/mnist-pyt/swci/taskdefs/run_mnist_pyt.yaml
   sed -i "s+<SWARM-NETWORK>+host-1-net+g" workspace/mnist-pyt/swop/swop*_profile.yaml
   sed -i "s+<LICENSE-SERVER-ADDRESS>+${APLS_IP}+g" workspace/mnist-pyt/swop/swop*_profile.yaml
   sed -i "s+<PROJECT>+$(pwd)/workspace/mnist-pyt+g" workspace/mnist-pyt/swop/swop*_profile.yaml
   sed -i "s+<PROJECT-CERTS>+$(pwd)/workspace/mnist-pyt/cert+g" workspace/mnist-pyt/swop/swop*_profile.yaml
   sed -i "s+<PROJECT-CACERTS>+$(pwd)/workspace/mnist-pyt/cert/ca/capath+g" workspace/mnist-pyt/swop/
   swop*_profile.yaml
   ```

7. Create a docker volume and copy Swarm Learning wheel file:

   ```
   docker volume rm sl-cli-lib
   docker volume create sl-cli-lib
   docker container create --name helper -v sl-cli-lib:/data hello-world
   docker cp lib/swarmlearning-client-py3-none-manylinux_2_24_x86_64.whl helper:/
   data
   docker rm helper
   ```

8. Run SN node (SN1 - sentinel node)

   ```
   ./scripts/bin/run-sn -d --rm --name=sn1 --network=host-1-net \
   --host-ip=${HOST_IP} --sentinel --sn-p2p-port=${SN_P2P_PORT} \
   --sn-api-port=${SN_API_PORT} --key=workspace/mnist-pyt/cert/sn-1-key.pem \
   --cert=workspace/mnist-pyt/cert/sn-1-cert.pem \
   --capath=workspace/mnist-pyt/cert/ca/capath --apls-ip=${APLS_IP}
   ```

   Use the Docker logs command to monitor the Sentinel SN node and wait for the node to finish initializing. The Sentinel node is ready when these messages appear in the log output:

   ```
   swarm.blCnt : INFO : Starting SWARM-API-SERVER on port: 30304
   ```

9. Run SWOP node (SWOP1).

NOTE: If required, according to environment, modify IP and proxy in the following command or in the SWOP profile file under `workspace/mnist-pyt/swop` folder.

```
./scripts/bin/run-swop -d --rm --name=swop1 --network=host-1-net \
--sn-ip=${SN_IP} --sn-api-port=${SN_API_PORT} \
--usr-dir=workspace/mnist-pyt/swop --profile-file-name=swop_profile.yaml \
--key=workspace/mnist-pyt/cert/swop-1-key.pem \
--cert=workspace/mnist-pyt/cert/swop-1-cert.pem \
--capath=workspace/mnist-pyt/cert/ca/capath -e SWOP_KEEP_CONTAINERS=True \
-e http_proxy= -e https_proxy= --swop-uid=0 --apls-ip=${APLS_IP}
```

NOTE: `–e SWOP_KEEP_CONTAINERS=True` is an optional argument, by default, it is False.

`SWOP_KEEP_CONTAINERS` is set to True so that SWOP does not remove stopped SL and ML containers. Without this setting, if there are any internal errors in SL or ML, SWOP removes them automatically. For more information, see the *Environment Variables* section, *HPE Swarm Learning Installation and Configuration Guide*.

10. Run SWCI node and observe sequential execution of two tasks – build task (`build_pyt_user_image`) and run task (`run_mnist_pyt`).

```
./scripts/bin/run-swci -ti --rm --name=swci1 --network=host-1-net \
--usr-dir=workspace/mnist-pyt/swci --init-script-name=swci-init \
--key=workspace/mnist-pyt/cert/swci-1-key.pem \
--cert=workspace/mnist-pyt/cert/swci-1-cert.pem \
--capath=workspace/mnist-pyt/cert/ca/capath \
-e http_proxy= -e https_proxy= --apls-ip=${APLS_IP}
```

- `build_pyt_user_image` - builds PyTorch based user image.
- `run_mnist_pyt` - runs Swarm training across for four ML nodes.

NOTE: If required, according to the environment, modify SN IP in `workspace/mnist-pyt/swci/swci-init` file.

11. Four nodes of Swarm trainings are automatically started when the run task (`run_mnist_pyt`) gets assigned and executed. Open a new terminal on both host-1 and host-2 and monitor the Docker logs of ML nodes for Swarm training. Swarm training ends with the following log message:

```
SwarmCallback : INFO : All peers and Swarm training rounds finished. Final Swarm model was
loaded.
```

Final Swarm model is saved inside each user's specific directory in `workspace/mnist-pyt/<userN>`. All the dynamically spawned SL and ML containers exits after Swarm training if `SWOP_KEEP_CONTAINERS` is not set, otherwise SL and ML containers must be removed manually. The SN and SWOP nodes continues to run.

12. To clean up, run the `scripts/bin/stop-swarm` script on all the systems to stop and remove the container nodes of the previous run. If required, backup the container logs. Remove Docker networks (`host-1-net`) and Docker volume (`sl-cli-lib`), and delete the `workspace` directory.

Running using `run-sl` command

In place of using SWOP and SWCI, execute the following `run-sl` command to start user and ML containers. For more information, see `run-sl` documentation and AMD GPU specific docker options from swop profile. User image needs to be built through docker build.

~~~
```
./scripts/bin/run-sl --name=sl1 --host-ip=${HOST_IP} \
--sn-ip=${SN_IP} --sn-api-port=${SN_API_PORT} --sl-fs-port=16000 \
--key=workspace/mnist-pyt/cert/sl-1-key.pem \
--cert=workspace/mnist-pyt/cert/sl-1-cert.pem \
--capath=workspace/mnist-pyt/cert/ca/capath \
--ml-image=user-image-amd-pyt --ml-name=ml1 \
--ml-w=/tmp/test --ml-entrypoint=python3 --ml-cmd=model/mnist_pyt.py \
--ml-v=<CURRENT-PATH>/workspace/mnist-pyt/model:/tmp/test/model \
--ml-e DATA_DIR=app-data \
--ml-e MODEL_DIR=model \
--ml-e MAX_EPOCHS=500 \
--ml-e MIN_PEERS=2 \
--ml-e HIP_VISIBLE_DEVICES=0 \
--ml-privileged \
--ml-device=/dev/kfd \
--ml-device=/dev/dri \
--ml-group-add video \
--ml-user=0:0 \
--apls-ip=${APLS_IP}
```
~~~

NVIDIA GPU based MNIST-PYT

This example runs MNIST-PYT[5] application with NVIDIA GPU based local training on the Swarm Learning platform.

When compared to CPU based `mnist-pyt` example following are the key differences in this example:

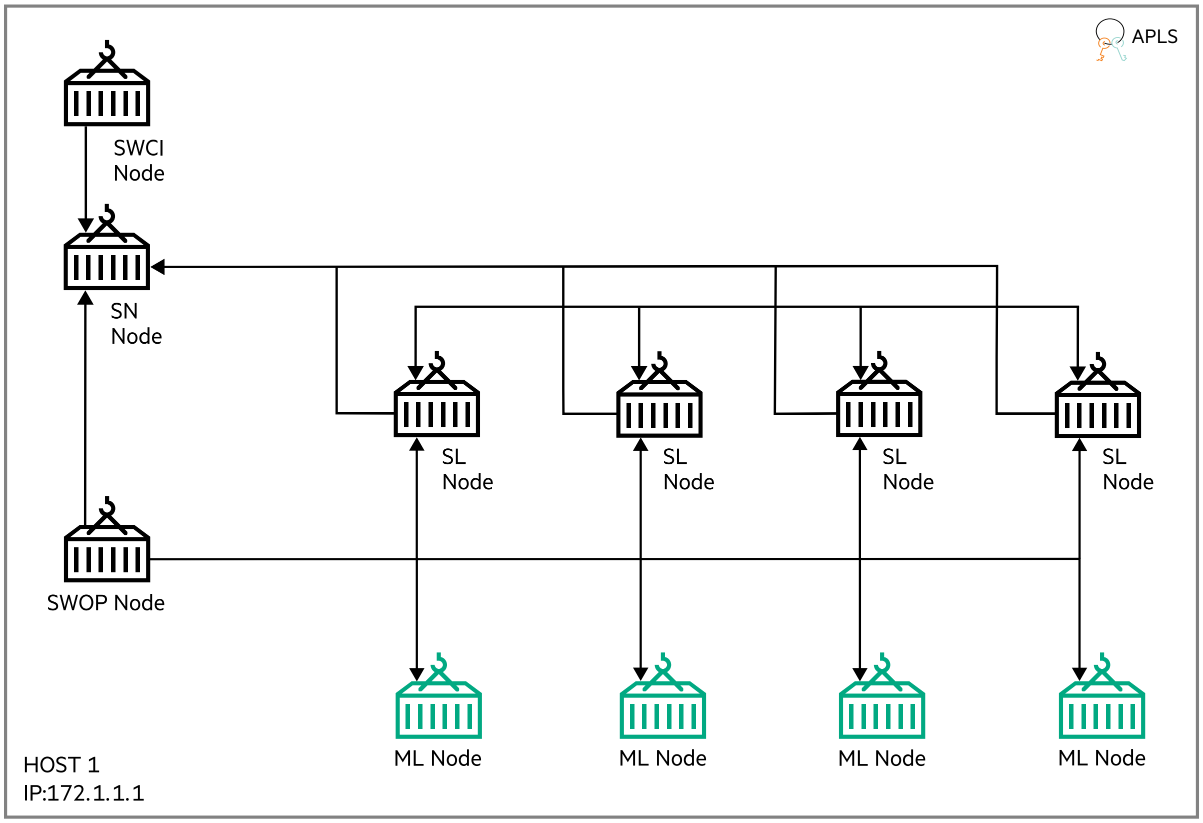
- User image build uses `rocm/pytorch` image as base image. Base image needs to be selected such that it support the host NVIDIA setup. For more information, see **https://hub.docker.com/r/pytorch/pytorch**.
- SWOP options needs additional tags to access NVIDIA GPUs. For more information, see **Launch Swarm Learning using SWOP**.
- Run task command needs additional argument to be passed in. See run task yaml in this example.
- User ML application code needs modifications to access NVIDIA GPUs.

The Machine Learning program, after conversion to Swarm Learning for the PyTorch platform, is in `examples/mnist-pyt/gpu-based/nvidia/model`. The PyTorch-based file is called `mnist_pyt.py`.

This example shows the Swarm training of MNIST model using four ML nodes. ML nodes along with SL nodes are automatically spawned by SWOP nodes - all running on a single host. Swarm training gets initiated by the SWCI node and orchestrated by one SN node running on the same host. This example also shows how private data, private scratch area, and shared model can be mounted to ML nodes for Swarm training.

The following image illustrates a cluster setup for the MNIST example:

5 **https://yann.lecun.com/exdb/mnist/**



- This example uses one SN node. The name of the docker containers representing this node is SN1. SN1 is also the Sentinel Node. SN1 runs on the host 172.1.1.1.
- Four SL and ML nodes are automatically spawned by SWOP node during training and removed after the training. This example uses one SWOP node that connects to the SN node. The name of the docker container representing this SWOP node is SWOP1. SWOP1 runs on the host 172.1.1.1.
- Training is initiated by SWCI node that runs on host 172.1.1.1.
- This example assumes that License Server already runs on host 172.1.1.1. All Swarm nodes connect to the License Server, on its default port 5814.

Running the MNIST-PYT example

1. Navigate to `swarm-learning` folder (that is, parent to `examples` directory).

   ```
   cd swarm-learning
   ```

2. Create a temporary `workspace` directory and copy `mnist-pyt` example.

   ```
   mkdir workspace
   cp -r examples/mnist-pyt/gpu-based/nvidia/ workspace/mnist-pyt/
   cp -r examples/utils/gen-cert workspace/mnist-pyt/
   ```

3. Run the gen-cert utility to generate certificates for each Swarm component using the command (`gen-cert -e <EXAMPLE-NAME> -i <HOST-INDEX>`):

   ```
   ./workspace/mnist-pyt/gen-cert -e mnist-pyt -i 1
   ```

4. Create a docker network *'host-1-net'*for SN, SWOP, SWCI, SL, and user containers running in a host. Ignore this step if the network is already created.

```
docker network create host-1-net
```

5. Declare and assign values to the variables such as `APLS_IP`, `SN_IP`, `HOST_IP`, and `SN_API_PORT`. For example:

```
APLS_IP=172.1.1.1
SN_IP=172.1.1.1
HOST_IP=172.1.1.1
SN_API_PORT=30304
SN_P2P_PORT=30303
```

NOTE: You must use appropriate values as per your Swarm network.

6. Search and replace all occurrences of placeholders and replace them with appropriate values.

```
sed -i "s+<PROJECT-MODEL>+$(pwd)/workspace/mnist-pyt/model+g" workspace/mnist-pyt/swci/taskdefs/run_mnist_pyt.yaml
sed -i "s+<SWARM-NETWORK>+host-1-net+g" workspace/mnist-pyt/swop/swop*_profile.yaml
sed -i "s+<LICENSE-SERVER-ADDRESS>+${APLS_IP}+g" workspace/mnist-pyt/swop/swop*_profile.yaml
sed -i "s+<PROJECT>+$(pwd)/workspace/mnist-pyt+g" workspace/mnist-pyt/swop/swop*_profile.yaml
sed -i "s+<PROJECT-CERTS>+$(pwd)/workspace/mnist-pyt/cert+g" workspace/mnist-pyt/swop/swop*_profile.yaml
sed -i "s+<PROJECT-CACERTS>+$(pwd)/workspace/mnist-pyt/cert/ca/capath+g" workspace/mnist-pyt/swop/
swop*_profile.yaml
```

7. Create a docker volume and copy Swarm Learning wheel file:

```
docker volume rm sl-cli-lib
docker volume create sl-cli-lib
docker container create --name helper -v sl-cli-lib:/data hello-world
docker cp lib/swarmlearning-client-py3-none-manylinux_2_24_x86_64.whl helper:/
data
docker rm helper
```

8. Run SN node (SN1 - sentinel node)

```
./scripts/bin/run-sn -d --rm --name=sn1 --network=host-1-net \
--host-ip=${HOST_IP} --sentinel --sn-p2p-port=${SN_P2P_PORT} \
--sn-api-port=${SN_API_PORT} --key=workspace/mnist-pyt/cert/sn-1-key.pem \
--cert=workspace/mnist-pyt/cert/sn-1-cert.pem \
--capath=workspace/mnist-pyt/cert/ca/capath --apls-ip=${APLS_IP}
```

Use the Docker logs command to monitor the Sentinel SN node and wait for the node to finish initializing. The Sentinel node is ready when these messages appear in the log output:

```
swarm.blCnt : INFO : Starting SWARM-API-SERVER on port: 30304
```

9. Run SWOP node (SWOP1).

NOTE: If required, according to environment, modify IP and proxy in the following command or in the SWOP profile file under `workspace/mnist-pyt/swop` folder.

```
./scripts/bin/run-swop -d --rm --name=swop1 --network=host-1-net \
--sn-ip=${SN_IP} --sn-api-port=${SN_API_PORT} \
--usr-dir=workspace/mnist-pyt/swop --profile-file-name=swop_profile.yaml \
--key=workspace/mnist-pyt/cert/swop-1-key.pem \
--cert=workspace/mnist-pyt/cert/swop-1-cert.pem \
--capath=workspace/mnist-pyt/cert/ca/capath -e SWOP_KEEP_CONTAINERS=True \
-e http_proxy= -e https_proxy= --apls-ip=${APLS_IP}
```

NOTE: `-e SWOP_KEEP_CONTAINERS=True` is an optional argument, by default, it is False.

`SWOP_KEEP_CONTAINERS` is set to True so that SWOP does not remove stopped SL and ML containers. Without this setting, if there are any internal errors in SL or ML, SWOP removes them automatically. For more information, see the *Environment Variables* section, *HPE Swarm Learning Installation and Configuration Guide*.

10. Run SWCI node and observe sequential execution of two tasks – build task (`build_pyt_user_image`) and run task (`run_mnist_pyt`).

```
./scripts/bin/run-swci -ti --rm --name=swci1 --network=host-1-net \
--usr-dir=workspace/mnist-pyt/swci --init-script-name=swci-init \
--key=workspace/mnist-pyt/cert/swci-1-key.pem \
--cert=workspace/mnist-pyt/cert/swci-1-cert.pem \
--capath=workspace/mnist-pyt/cert/ca/capath \
-e http_proxy= -e https_proxy= --apls-ip=${APLS_IP}
```

- `build_pyt_user_image` - builds pytorch based user image.
- `run_mnist_pyt` - runs Swarm training across for four ML nodes.

NOTE: If required, according to the environment, modify SN IP in `workspace/mnist-pyt/swci/swci-init` file.

11. Four nodes of Swarm trainings are automatically started when the run task (`run_mnist_pyt`) gets assigned and executed. Open a new terminal on both host-1 and host-2 and monitor the Docker logs of ML nodes for Swarm training. Swarm training ends with the following log message:

```
SwarmCallback : INFO : All peers and Swarm training rounds finished. Final Swarm model was
loaded.
```

Final Swarm model is saved inside each user's specific directory in `workspace/mnist-pyt/<userN>`. All the dynamically spawned SL and ML containers exits after Swarm training if `SWOP_KEEP_CONTAINERS` is not set, otherwise SL and ML containers must be removed manually. The SN and SWOP nodes continues to run.

12. To clean up, run the `scripts/bin/stop-swarm` script on all the systems to stop and remove the container nodes of the previous run. If required, backup the container logs. Remove Docker networks (`host-1-net`) and Docker volume (`sl-cli-lib`), and delete the `workspace` directory.

Running using `run-sl` command

In place of using SWOP and SWCI, execute the following `run-sl` command to start user and ML containers. For more information, see `run-sl` documentation and NVIDIA GPU specific docker options from swop profile. User image needs to be built through docker build.

```
```
./scripts/bin/run-sl --name=sl1 --host-ip=${HOST_IP} \
--sn-ip=${SN_IP} --sn-api-port=${SN_API_PORT} --sl-fs-port=16000 \
--key=workspace/mnist-pyt/cert/sl-1-key.pem \
--cert=workspace/mnist-pyt/cert/sl-1-cert.pem \
--capath=workspace/mnist-pyt/cert/ca/capath \
--ml-image=user-image-nvidia-pyt --ml-name=ml1 \
--ml-w=/tmp/test --ml-entrypoint=python3 --ml-cmd=model/mnist_pyt.py \
--ml-v=<CURRENT-PATH>/workspace/mnist-pyt/model:/tmp/test/model \
--ml-e DATA_DIR=app-data \
--ml-e MODEL_DIR=model \
--ml-e MAX_EPOCHS=500 \
--ml-e MIN_PEERS=2 \
```
```

```
--ml-e HIP_VISIBLE_DEVICES=0 \
--ml-privileged \
--ml-device=/dev/kfd \
--ml-device=/dev/dri \
--ml-group-add video \
--ml-user=0:0 \
--apls-ip=${APLS_IP}
```

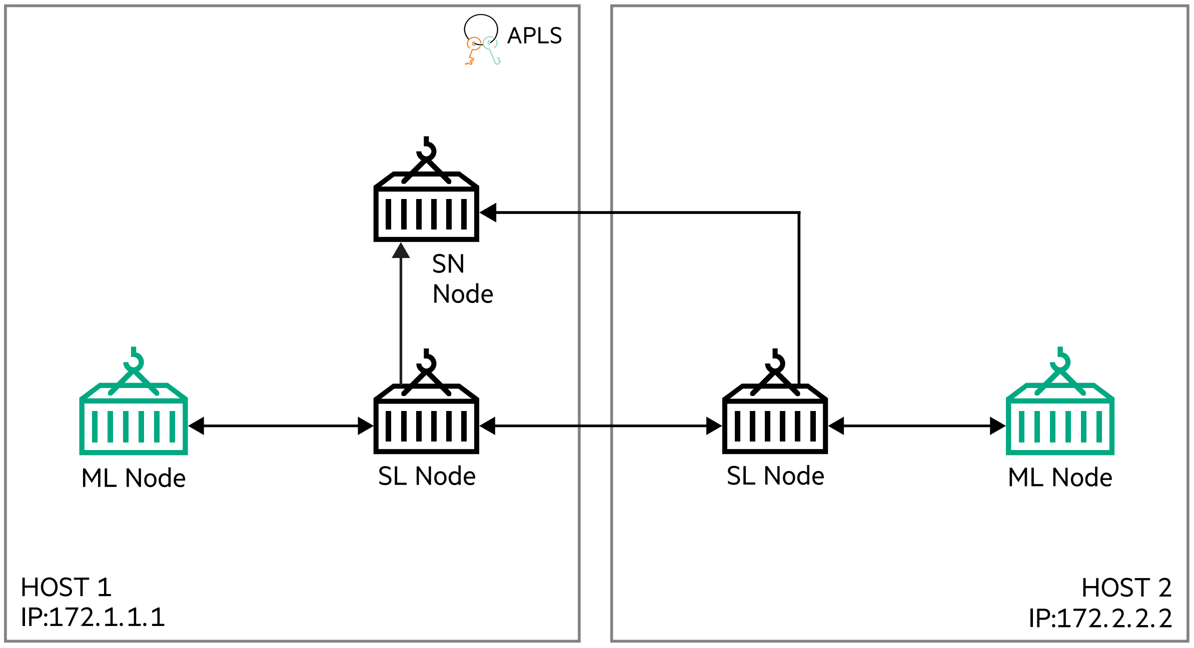
CIFAR-10

This example runs CIFAR-10[6] on the Swarm Learning platform. It uses Tensorflow as the backend.

This example uses CIFAR-10 dataset distributed along with TensorFlow package. The ML program, after conversion to Swarm Learning, is in `swarm-learning/examples/cifar10/model` and is called `cifar10.py`. It contains a tiny ML model for the purpose of showing steps of converting ML code for Swarm Learning.

This example shows the Swarm training of Keras based CIFAR-10 model using two ML nodes. It also shows how ML environment can be built manually using `run-sl` command and how Swarm training can be launched without SWCI or SWOP nodes. Here `scripts/bin/run-sl` script is used to spawn each ML node to run CIFAR-10 model as a `sidecar` container of each SL node.

The following image illustrates a cluster setup for the CIFAR-10 example which uses two hosts:



- This example uses one SN node. SN1 is the name of the Docker container that runs on host 172.1.1.1.
- Two SL and ML nodes are manually spawned by running the `run-sl` script. Swarm training gets invoked once ML nodes are started. Name of the SL nodes that runs as container are SL1 and SL2. Name of the ML nodes that runs as container are ML1 and ML2.

[6] **https://www.cs.toronto.edu/%7Ekriz/cifar.html**

- SL1 and ML1 pair runs on host 172.1.1.1, whereas SL2 and ML2 pair runs on host 172.2.2.2.
- This example assumes that License Server already runs on host 172.1.1.1. All Swarm nodes connect to the License Server, on its default port 5814.

Running the CIFAR-10 example

1. On both host-1 and host-2, navigate to `swarm-learning` folder (that is, parent to `examples` directory).

   ```
   cd swarm-learning
   ```

2. On both host-1 and host-2, create a temporary `workspace` directory, copy `cifar10`, and `gen-cert` utility.

   ```
   mkdir workspace
   cp -r examples/cifar10 workspace/
   cp -r examples/utils/gen-cert workspace/cifar10/
   ```

3. On both host-1 and host-2, run the `gen-cert` utility to generate certificates for each Swarm component using the command `gen-cert -e <EXAMPLE-NAME> -i <HOST-INDEX>`:

   On host-1:

   ```
   ./workspace/cifar10/gen-cert -e cifar10 -i 1
   ```

   On host-2:

   ```
   ./workspace/cifar10/gen-cert -e cifar10 -i 2
   ```

4. On both host-1 and host-2, share the CA certificates between the hosts as follows:

   On host-1:

   ```
   scp host-2:<PATH>workspace/cifar10/cert/ca/capath/ca-2-cert.pem workspace/cifar10/cert/ca/
   capath
   ```

   On host-2:

   ```
   scp host-1:<PATH>workspace/cifar10/cert/ca/capath/ca-1-cert.pem workspace/cifar10/cert/ca/
   capath
   ```

5. On both host-1 and host-2, copy Swarm Learning wheel file inside build context and build Docker image for ML that contains environment to run Swarm training of user models.

   ```
   cp -L lib/swarmlearning-client-py3-none-manylinux_2_24_x86_64.whl workspace/cifar10/ml-
   context/
   docker build -t user-ml-env-tf2.7.0 workspace/cifar10/ml-context
   ```

6. On both host-1 and host-2, declare and assign values to the variables like `APLS_IP`, `SN_IP`, `HOST_IP` and `ports`. For example,

   ```
   APLS_IP=172.1.1.1
   SN_IP=172.1.1.1
   HOST_1_IP=172.1.1.1
   HOST_2_IP=172.2.2.2
   SN_API_PORT=30304
   SN_P2P_PORT=30303
   SL_1_FS_PORT=16000
   SL_2_FS_PORT=17000
   ```

   NOTE: You must use the appropriate values as per your Swarm network.

7. On host-1, Run Swarm Network node (sentinel node)

   ```
   ./scripts/bin/run-sn -d --rm --name=sn1 --host-ip=${HOST_1_IP} \
   --sentinel --sn-api-port=${SN_API_PORT} --key=workspace/cifar10/cert/sn-1-
   ```

```
key.pem \
--cert=workspace/cifar10/cert/sn-1-cert.pem \
--capath=workspace/cifar10/cert/ca/capath \
--apls-ip=${APLS_IP}
```

Use the Docker logs command to monitor this Sentinel SN node and wait for the node to finish initializing. The Sentinel node is ready when these messages appear in the log output:

```
swarm.blCnt : INFO : Starting SWARM-API-SERVER on port: 30304
```

8. On host-1, run Swarm Learning node and Machine Learning node (as a side-car). Set the proxy server as appropriate, as the ML program needs to download the CIFAR dataset.

```
./scripts/bin/run-sl -d --name=sl1 --host-ip=${HOST_1_IP} \
--sn-ip=${SN_IP} --sn-api-port=${SN_API_PORT} \
--sl-fs-port=${SL_1_FS_PORT} --key=workspace/cifar10/cert/sl-1-key.pem \
--cert=workspace/cifar10/cert/sl-1-cert.pem  --capath=workspace/cifar10/cert/ca/
capath \
--ml-image=user-ml-env-tf2.7.0 --ml-name=ml1 --ml-w=/tmp/test --ml-entrypoint=python3
\
--ml-cmd=model/cifar10.py --ml-v=workspace/cifar10/model:/tmp/test/model \
--ml-e MODEL_DIR=model --ml-e MAX_EPOCHS=1 --ml-e MIN_PEERS=2 \
--ml-e https_proxy=http://<your-proxy-server-ip>:<port-number> --apls-ip=${APLS_IP}
```

9. On host-2, run Swarm Learning node and Machine Learning node (as a side-car). Set the proxy server as appropriate, as the ML program needs to download the CIFAR dataset.

```
./scripts/bin/run-sl -d --name=sl2 --host-ip=${HOST_2_IP} \
--sn-ip=${SN_IP} --sn-api-port=${SN_API_PORT} --sl-fs-port=${SL_2_FS_PORT} \
--key=workspace/cifar10/cert/sl-2-key.pem --cert=workspace/cifar10/cert/sl-2-cert.pem
\
--capath=workspace/cifar10/cert/ca/capath --ml-image=user-ml-env-tf2.7.0 --ml-
name=ml2 \
--ml-w=/tmp/test --ml-entrypoint=python3 --ml-cmd=model/cifar10.py \
--ml-v=workspace/cifar10/model:/tmp/test/model \
--ml-e MODEL_DIR=model --ml-e MAX_EPOCHS=1 --ml-e MIN_PEERS=2 \
--ml-e https_proxy=http://<your-proxy-server-ip>:<port-number> --apls-ip=${APLS_IP}
```

10. On both host-1 and host-2, Two node of Swarm training are started. User can monitor the Docker logs of ML nodes (ML1 and ML2 containers) for Swarm training on both host-1 and host-2. Training ends with the following log message:

```
SwarmCallback : INFO : Saved the trained model - model/saved_models/
cifar10.h5
```

Final Swarm model is saved inside the model directory that is `workspace/cifar10/model/saved_models` directory on both the hosts. SL and ML nodes exit but it is not removed after the Swarm training.

11. On both host-1 and host-2, To clean up, run the `scripts/bin/stop-swarm` script on all the systems to stop and remove the container nodes of the previous run. If required, backup the container logs and delete the `workspace` directory.

Credit card fraud detection

This example runs a credit card fraud detection algorithm on the Swarm Learning platform. It uses Keras and TensorFlow.

This example uses a subset of the data from[7] for each node. These subset datasets are biased with respect to the class and the volume of data.

This example uses four training batches and one test batch. These files are located in the respective `examples/fraud-detection/data-and-scratch<n>` directories.

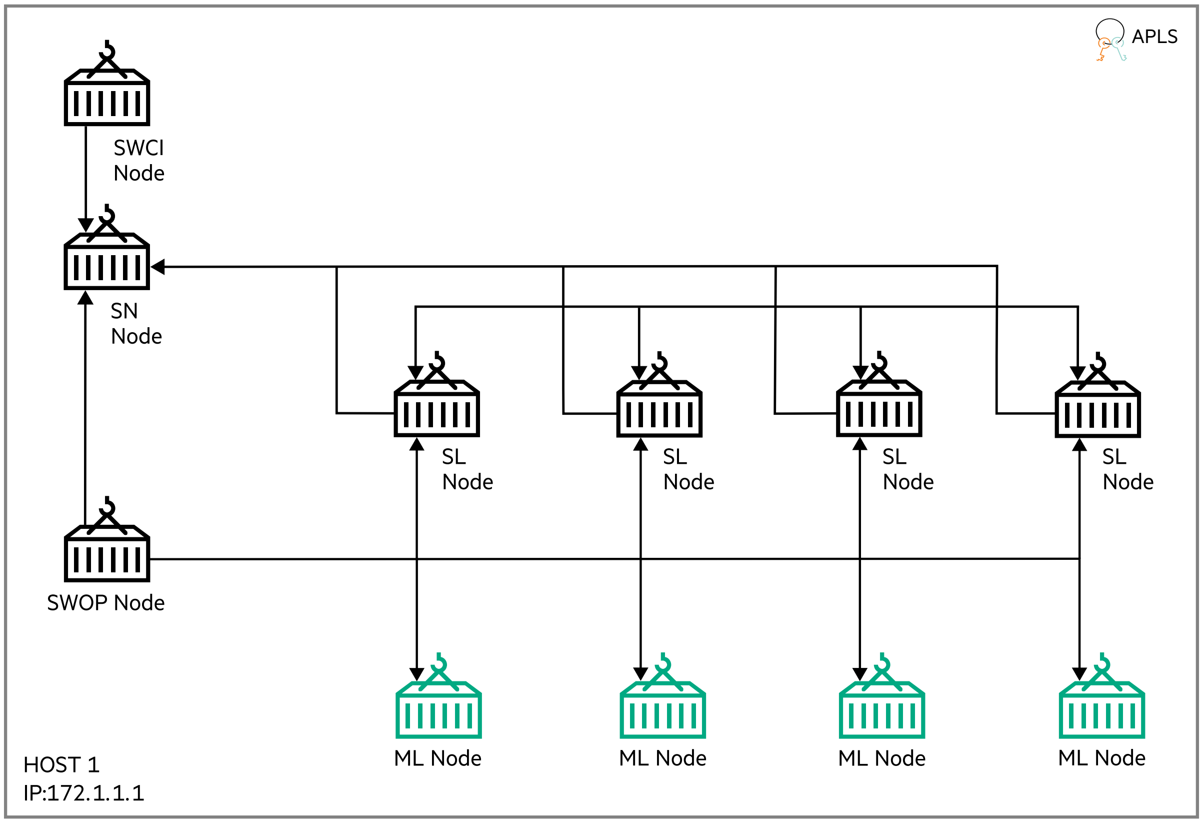
[7] **https://www.kaggle.com/mlg-ulb/creditcardfraud**

NOTE: For data license associated with this dataset, see `/examples/fraud-detection/data_license.md`.

The ML program, after conversion to Swarm Learning, is in `examples/fraud-detection/model` and is called `fraud-detection.py`.

This example shows the Swarm training of the credit card fraud detection model using four ML nodes. ML nodes along with SL nodes are automatically spawned by SWOP nodes - all running on a single host. Swarm training gets initiated by the SWCI node and orchestrated by one SN node running on the same host. This example also shows how private data, private scratch area, and shared model can be mounted to ML nodes for Swarm training. For more information, see the profile files and task definition files placed under `examples/fraud-detection/swop` and `examples/fraud-detection/swci`.

The following image illustrates a cluster setup that uses only one host:



- This example uses one SN node. The names of the docker containers representing this node is SN1. SN1 is also the Sentinel Node. SN1 runs on the host 172.1.1.1.
- Four SL and ML nodes are automatically spawned by SWOP node during training and removed after the training. This example uses one SWOP node that connects to the SN node. The names of the docker containers representing this SWOP node is SWOP1. SWOP1 runs on the host 172.1.1.1.
- Training is initiated by SWCI node (SWCI1) that runs on the host 172.1.1.1.
- This example assumes that License Server already runs on the host 172.1.1.1. All Swarm nodes connect to the License Server, on its default port 5814.

Running the credit card fraud detection example

1. Navigate to `swarm-learning` folder (that is, parent to `examples` directory).

   ```
   cd swarm-learning
   ```

2. Create a temporary `workspace` directory, `fraud-detection` example, and `gen-cert` utility.

   ```
   mkdir workspace
   cp -r examples/fraud-detection workspace/
   cp -r examples/utils/gen-cert workspace/fraud-detection/
   ```

3. Run the `gen-cert` utility to generate certificates for each Swarm component using the command, `gen-cert -e <EXAMPLE-NAME> -i <HOST-INDEX>`.

   ```
   ./workspace/fraud-detection/gen-cert -e fraud-detection -i 1
   ```

4. Create a docker network for SN, SWOP, SWCI, SL, and user containers running on the same host.

   ```
   docker network create host-1-net
   ```

5. Declare and assign values to the variables like `APLS_IP`, `SN_IP`, `HOST_IP` and `SN_API_PORT`. For example,

   ```
   APLS_IP=172.1.1.1
   HOST_IP=172.1.1.1
   SN_IP=172.1.1.1
   SN_API_PORT=30304
   ```

   NOTE: You must use the appropriate values as per your Swarm network.

6. Search and replace all occurrences of placeholders and replace them with appropriate values.

   ```
   sed -i "s+<PROJECT-MODEL>+$(pwd)/workspace/fraud-detection/model+g"
   workspace/fraud-detection/swci/taskdefs/swarm_fd_task.yaml
   sed -i "s+<SWARM-NETWORK>+host-1-net+g" workspace/fraud-detection/swop/
   swop*_profile.yaml
   sed -i "s+<CURRENT-PATH>/examples+$(pwd)/workspace+g" workspace/fraud-
   detection/swop/swop*_profile.yaml
   sed -i "s+<LICENSE-SERVER-ADDRESS>+${APLS_IP}+g" workspace/fraud-
   detection/swop/swop*_profile.yaml
   sed -i "s+<PROJECT-CERTS>+$(pwd)/workspace/fraud-detection/cert+g"
   workspace/fraud-detection/swop/swop*_profile.yaml
   sed -i "s+<PROJECT-CACERTS>+$(pwd)/workspace/fraud-detection/cert/ca/
   capath+g" workspace/fraud-detection/swop/swop*_profile.yaml
   ```

7. Create a docker volume and copy Swarm Learning wheel file.

   ```
   docker volume rm sl-cli-lib
   docker volume create sl-cli-lib
   docker container create --name helper -v sl-cli-lib:/data hello-world
   docker cp lib/swarmlearning-client-py3-none-manylinux_2_24_x86_64.whl helper:/
   data
   docker rm helper
   ```

8. Run Swarm Network node (SN1) - sentinel node.

   ```
   ./scripts/bin/run-sn -d --rm --name=sn1 --network=host-1-net --host-ip=$
   {HOST_IP} \
   --sentinel --sn-api-port=${SN_API_PORT} \
   --key=workspace/fraud-detection/cert/sn-1-key.pem \
   --cert=workspace/fraud-detection/cert/sn-1-cert.pem \
   --capath=workspace/fraud-detection/cert/ca/capath \
   --apls-ip=${APLS_IP}
   ```

Use the Docker logs command to monitor the Sentinel SN node and wait for the node to finish initializing. The Sentinel node is ready when the following messages appear in the log output:

```
swarm.blCnt : INFO : Starting SWARM-API-SERVER on port: 30304
```

9. Run Swarm Operator node (SWOP1).

NOTE: If required, modify the proxy, according to the environment, either in the following command or in the swop profile file under `workspace/fraud-detection/swop` folder.

```
./scripts/bin/run-swop -d --rm --name=swop1 --network=host-1-net \
--usr-dir=workspace/fraud-detection/swop \
--profile-file-name=swop1_profile.yaml --sn-ip=${SN_IP} --sn-api-port=$
{SN_API_PORT} \
--key=workspace/fraud-detection/cert/swop-1-key.pem \
--cert=workspace/fraud-detection/cert/swop-1-cert.pem \
--capath=workspace/fraud-detection/cert/ca/capath \
-e http_proxy= -e https_proxy= --apls-ip=${APLS_IP}
```

10. Run SWCI node (SWCI1). It creates, finalizes, and assigns below tasks to task-framework for sequential execution:

    - `user_env_tf_build_task`: Builds TensorFlow based Docker image for ML node to run model training.
    - `swarm_fd_task`: Creates containers out of ML image, and mounts model and data path to run Swarm training.

NOTE: Note: If required, according to environment, modify SN IP in `workspace/fraud-detection/swci/swci-init` file.

```
./scripts/bin/run-swci --rm --name=swci1 --network=host-1-net \
--usr-dir=workspace/fraud-detection/swci --init-script-name=swci-init \
--key=workspace/fraud-detection/cert/swci-1-key.pem \
--cert=workspace/fraud-detection/cert/swci-1-cert.pem \
--capath=workspace/fraud-detection/cert/ca/capath \
-e http_proxy= -e https_proxy= --apls-ip=${APLS_IP}
```

11. Four nodes of Swarm training are automatically started when the run task (`swarm_fd_task`) gets assigned and executed. Open a new terminal on host-1 and monitor the Docker logs of ML nodes for Swarm training. Swarm training ends with the following log message:

```
SwarmCallback : INFO : All peers and Swarm training rounds finished. Final Swarm model was
loaded.
```

Final Swarm model is saved in each node specific scratch directory, which is `workspace/fraud-detection/data-and-scratch<n>/user<n>` directory. All the dynamically spawned SL and ML nodes exits after Swarm training. The SN and SWOP nodes continues to run.

12. To clean up, run the `scripts/bin/stop-swarm` script on all the systems to stop and remove the container nodes of the previous run. If required, backup the container logs and remove Docker network (`host-1-net`) and Docker volume (`sl-cli-lib`), and delete the workspace directory.

National Institutes of Health Chest X-Ray Dataset

The purpose of this example is to show case Swarm use case with real world National Institutes of Health Chest X-Ray Dataset (NIH) dataset[8].

[8] Kaggle NIH Chest X-rays : **NIH Chest X-rays**

NIH dataset is a well-known health care dataset. NIH dataset has complex x-ray images. It is a challenging task to meet better metrics on this dataset.

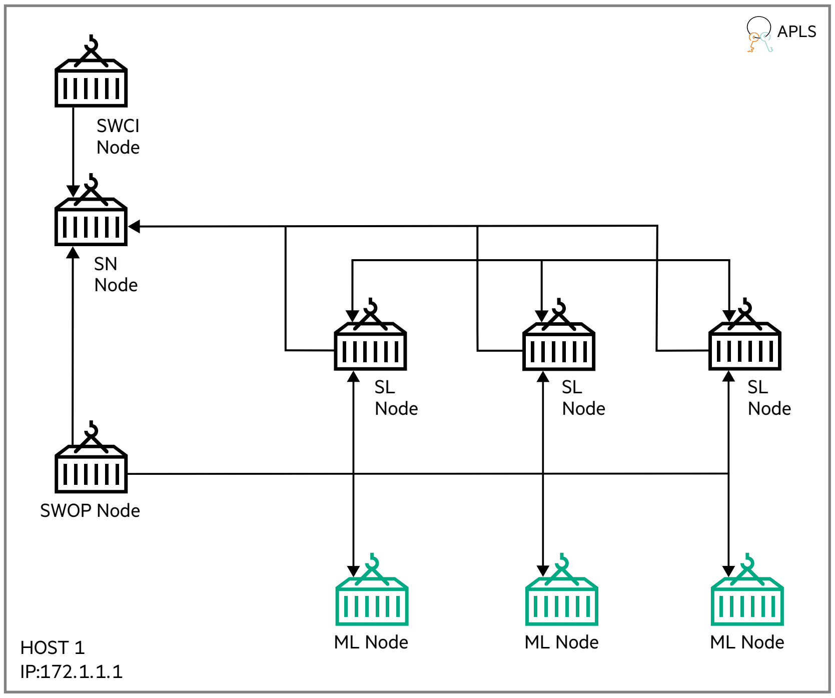
The data processing and model code for this example are taken from multiple Kaggle kernels and modified to run on a Swarm Learning platform.

This example uses node specific biased data and one common test data. The Machine Learning program, after conversion to Swarm Learning for the TensorFlow-based Keras platform, is in `examples/nih/model`. The TensorFlow-based file is called `nih_tf.py`.

This example shows the Swarm training of the NIH model using three ML nodes. ML nodes are automatically spawned by Swarm Operator (SWOP) node running on a single host. Swarm training is initiated by SWCI node and orchestrated by one SN node running on a single host. This example also shows how private data and shared model can be mounted on ML nodes for Swarm training. For more information, see the profile files and task definition files placed under `examples/nih/swop` and `examples/nih/swci` folders, respectively.

Cluster Setup

The following image illustrates the cluster setup for the NIH example which uses one host: host-1: 172.1.1.1.



1. This example uses one SN node. The names of the docker containers representing these two nodes are sn-1. sn-1 is the Sentinel Node. sn-1 runs on host 172.1.1.1.
2. SL and ML nodes are automatically spawned by SWOP nodes during training and removed after training.
3. This example uses one SWOP node named swop-1. swop-1 runs on host 172.1.1.1.
4. Training is initiated by SWCI node (swci-1) that runs on host 172.1.1.1.
5. This example assumes that License Server already runs on host 172.1.1.1. All Swarm nodes connect to the License Server, on its default port 5814.

Running the NIH example

1. Navigate to `swarm-learning` folder (that is, parent to `examples` directory).

```
cd swarm-learning
```

2. Create a temporary `workspace` directory and copy `nih` example and `gen-cert` utility.

```
mkdir workspace cp -r examples/nih workspace/ cp -r examples/utils/gen-
cert workspace/nih/
```

3. Run the `gen-cert` utility to generate certificates for each Swarm component using the command: `gen-cert -e <EXAMPLE-NAME> -i <HOST-INDEX>`

```
./workspace/nih/gen-cert -e nih -i 1
```

4. Create train, test data for each node.

   a. Navigate to `workspace/nih/data` folder.

   b. Execute data processing script.

   c. Verify Node1, Node2, Node3, Test folders are generated.

   d. Navigate back to `swarm-learning` folder.

NOTE: `nih_nodes_data_processing.py` and `references data_generator.yaml` get the path of NIH data. Path provided in `data_generator.yaml` must contain `images` folder (with nih images as .png files) and master sheet of data description (Data_Entry_2017.csv which has information about all the images). For more information, see *NIH Chest X-rays*.

```
cd workspace/nih/data python3 nih_nodes_data_processing.py cd ../../..
```

5. Create a docker network for SN, SWOP, SWCI, SL and user containers running on a host.

```
docker network create host-1-net
```

6. Declare and assign values to the variables like APLS_IP, SN_IP, HOST_IP and SN_API_PORT. For example,

```
APLS_IP=172.1.1.1 SN_1_IP=172.1.1.1 HOST_1_IP=172.1.1.1 SN_API_PORT=30304
SN_P2P_PORT=30303
```

NOTE: User must use the appropriate values as per their Swarm network.

7. Search and replace all occurrences of placeholders and replace them with appropriate values.

```
sed -i "s+<PROJECT-MODEL>+$(pwd)/workspace/nih+g" workspace/nih/swci/taskdefs/swarm_nih_task.yaml
sed -i "s+<PROJECT-MODEL>+$(pwd)/workspace/nih+g" workspace/nih/swci/taskdefs/swarm_ind_task.yaml
sed -i "s+<SWARM-NETWORK>+host-1-net+g" workspace/nih/swop/swop1_profile.yaml
sed -i "s+<HOST_ADDRESS>+${HOST_1_IP}+g" workspace/nih/swop/swop1_profile.yaml
sed -i "s+<LICENSE-SERVER-ADDRESS>+${APLS_IP}+g" workspace/nih/swop/swop*_profile.yaml
sed -i "s+<PROJECT>+$(pwd)/workspace/nih+g" workspace/nih/swop/swop*_profile.yaml
sed -i "s+<PROJECT-CERTS>+$(pwd)/workspace/nih/cert+g" workspace/nih/swop/swop*_profile.yaml
sed -i "s+<PROJECT-CACERTS>+$(pwd)/workspace/nih/cert/ca/capath+g" workspace/nih/swop/
swop*_profile.yaml
```

8. Create a docker volume and copy Swarm Learning wheel file.

```
docker volume rm sl-cli-lib docker volume create sl-cli-lib docker
container create --name helper -v sl-cli-lib:/data hello-world docker cp
lib/swarmlearning-client-py3-none-manylinux_2_24_x86_64.whl helper:/data
docker rm helper
```

9. Run Swarm Network node (SN1) - sentinel node.

```
./scripts/bin/run-sn -d --rm --name=sn1 --network=host-1-net --host-ip=$
{HOST_1_IP} --sentinel \ --sn-p2p-port=${SN_P2P_PORT} --sn-api-port=$
{SN_API_PORT} --key=workspace/nih/cert/sn-1-key.pem \ --
cert=workspace/nih/cert/sn-1-cert.pem --capath=workspace/nih/cert/ca/
capath --apls-ip=${APLS_IP}
```

Use the Docker logs command to monitor the Sentinel SN node and wait for the node to finish initializing. The Sentinel node is ready when the following messages appear in the log output:

```
swarm.blCnt : INFO : Starting SWARM-API-SERVER on port: 30304
```

10. Run Swarm Operator node (SWOP1).

NOTE: If required, modify the proxy, according to the environment, either in the following command or in the swop profile file under `workspace/nih/swop` folder.

```
./scripts/bin/run-swop -d --rm --name=swop1 --network=host-1-net --sn-ip=${SN_1_IP} --sn-api-port=$
{SN_API_PORT}       \
--usr-dir=workspace/nih/swop --profile-file-name=swop1_profile.yaml --key=workspace/nih/cert/swop-1-key.pem        \
--cert=workspace/nih/cert/swop-1-cert.pem --capath=workspace/nih/cert/ca/capath -e http_proxy= -e https_proxy=     \
-e SWOP_KEEP_CONTAINERS=True --apls-ip=${APLS_IP}
```

11. Run SWCI node (SWCI1). It creates, finalizes, and assigns below tasks to task-framework for sequential execution:

- `user_env_tf_build_task`: Builds Tensorflow based docker image for ML node to run model training.
- `swarm_nih_task`: Creates containers out of ML image and mount model and data path to run Swarm training with NIH data.
- `swarm_ind_task`: Create containers out of ML image and mount model and data path to run individual training of each node using NIH data.

NOTE: If required, modify IP, according to environment, in `workspace/nih/swci/swci-init` file.

```
./scripts/bin/run-swci --rm --name=swci1 --network=host-1-net --usr-
dir=workspace/nih/swci --init-script-name=swci-init        \
--key=workspace/nih/cert/swci-1-key.pem --cert=workspace/nih/cert/swci-1-
cert.pem --capath=workspace/nih/cert/ca/capath    \
-e http_proxy= -e https_proxy= --apls-ip=${APLS_IP}
```

12. Three nodes of Swarm training are automatically started when the run task gets assigned and executed. Open a new terminal and monitor the Docker logs of ML nodes for Swarm training. Swarm training ends with the following log message:

```
SwarmCallback : INFO : All peers and Swarm training rounds finished. Final
Swarm model was loaded.
```

Final Swarm model is saved inside `<PROJECT>` directory which is user's specific as `workspace/nih/<userN>` directory. All the dynamically spawned SL and ML nodes exit after Swarm training. The SN and SWOP nodes continue to run.

13. To clean up, run the `scripts/bin/stop-swarm` script on all the systems to stop and remove the container nodes of the previous run. If required, backup the container logs and remove Docker network (`host-1-net`) and Docker volume (`sl-cli-lib`), and delete the `workspace` directory.

Examples using reverse proxy

Prerequisite for running reverse proxy examples

1. Reverse proxy examples use NGINX and BIND9 as custom Docker images to build it. Both of these images use Ubuntu 22.04 as the base image, along with some image-specific apt-get packages. Ensure that these packages and images are downloaded properly while running the scripts within these examples.
2. Before running these examples, ensure that the APLS is already running and the wheel file is placed under the `lib` directory.
3. Reverse proxy examples run the swarm containers with fixed IP addresses. This is required because the users pre-configure the NGINX with the IP addresses of swarm containers before the containers start. For default bridge networks, there are more chances of IP conflicts; Or, the Docker command will not consider the IP passed. Hence, HPE recommends creating a separate non-default (custom) bridge network to start the containers with predefined IP addresses.

   To create the own network, use the following command:

   ```
   docker network create --subnet=<subnet-ip>/16 <network-name>
   ```

   In this command, `<subnet-ip>` is an IP other than default Docker bridge network's subnet. For example, if the default bridge uses 172.18.0.0 subnet, then you can use 192.18.0.0 subnet. `<network-name>` is a string which is used as an argument to the run scripts of these examples.
4. By default, the NGINX container started by these examples will use 443 port. Ensure that there is no other service is already running on this 443 port.
5. In these reverse proxy examples, build task pulls docker images along with some packages and run task pulls datasets from the external sites. Ensure that the necessary proxy configurations are added in the SWOP profiles before running the automated scripts.

NOTE: All reverse proxy-based examples are in the `examples/reverse-proxy` folder. NGINX and BIND9 Docker files are specified in the `examples/reverse-proxy/common` folder. The configuration file of NGINX, called `nginx.conf`, is present in the respective example of the reverse proxy.

Working of reverse proxy examples

1. DNS and Proxy modifications are needed to run Swarm Learning in reverse proxy mode. As these modifications are plenty, these examples are curated to handle these modifications automatically.
2. BIND9, as a Docker container, is used as a DNS server for these examples to resolve the FQDN to an IP address.
3. If the user wants to use their local DNS server instead of BIND9 as a container, then they should have enough privilege to add IP and FQDN mapping to the local `named.conf` file.

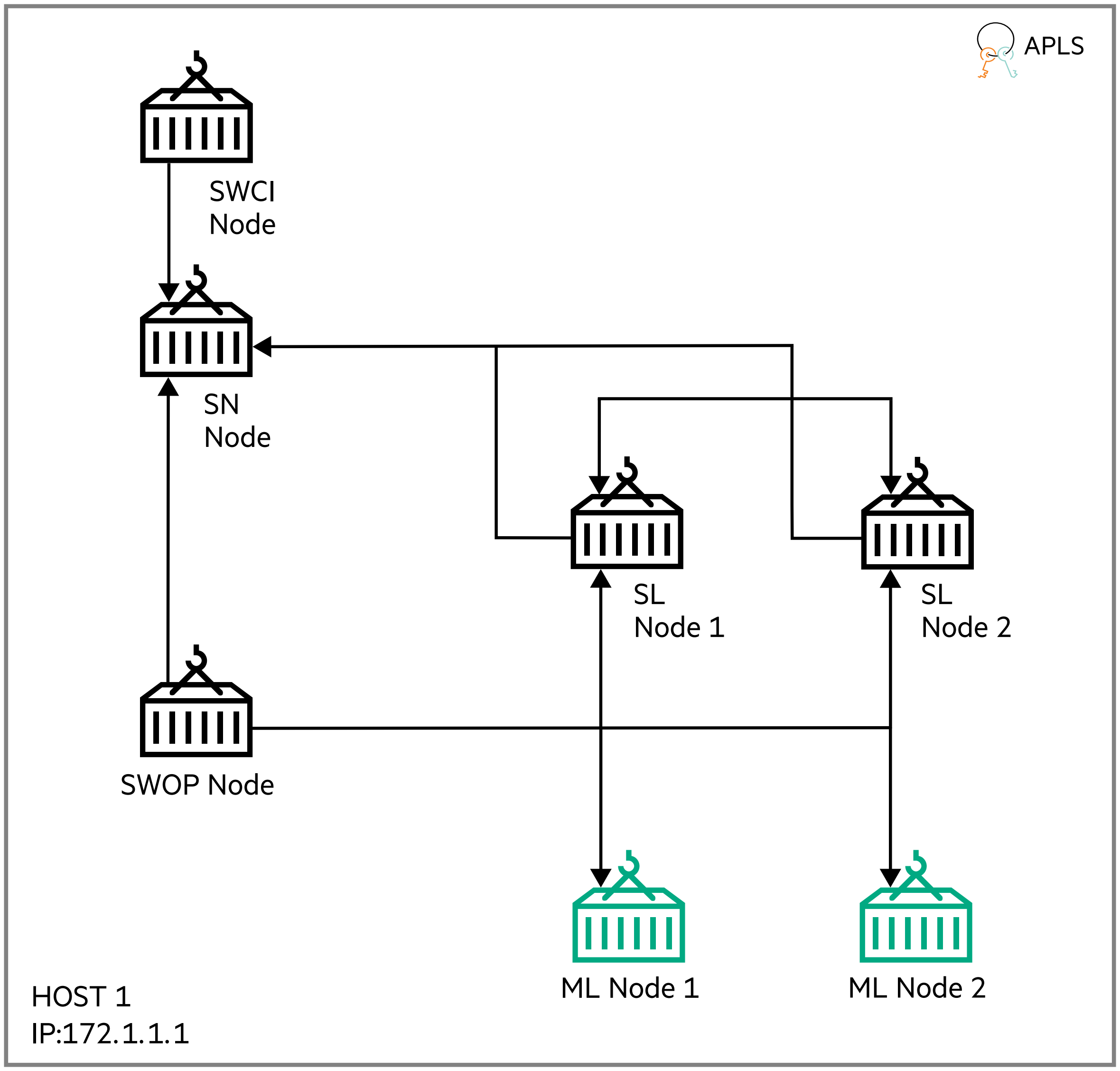
   NOTE: The location of this file is specific to the type of operating system.
4. BIND9 container created from these examples has a built-in function called `add-dns` to add the mapping into the running DNS BIND9 container.
5. Alternatively, routing or tunneling of requests using a reverse proxy is managed by running the NGINX as a Docker container.
6. Routing configuration in NGINX is managed by using `nginx.conf` file and is volume mounted before staring the NGINX container.

7. These examples run the container with pre-configured `nginx.conf` files. The container IP addresses used in this configuration file are the IPs adjacent to the BIND9 container. Hence to these examples, HPE recommends to keep these local IPs available.
8. If the user wants to use custom IP addresses, then they must alter their `nginx.conf` file according to their requirements and start the NGINX container.

MNIST using Reverse Proxy

This is single host MNIST example using reverse proxy service parameters with 1 SN node, 1 SWOP node, 1 SWCI node and 2 SL nodes.

The following image illustrates a cluster setup for the Reverse Proxy with MNIST example:



1. This example uses a SN node. This node is named as sn-1 and is the Sentinel Node. sn-1 runs on host 172.1.1.1.
2. SL and ML nodes are automatically spawned by SWOP node during training.

3. This example uses a SWOP node that connects to a SN node. The name of the docker container representing this SWOP node is swop-1. swop-1 container also runs on host 172.1.1.1.
4. Training is initiated by SWCI node (swci-1) that also runs on host 172.1.1.1.
5. This example assumes that the License Server already runs on the same host 172.1.1.1. All Swarm nodes connect to the License Server, on its default port 5814.

This example runs Reverse Proxy with MNIST[9] on the Swarm Learning platform with an ML program. It uses TensorFlow as the backend. The ML code for this example is taken from[10] and modified to run on a Swarm Learning platform.

This example shows the MNIST using reverse proxy to mimic real-world behavior. This example uses BIND9 as the DNS server and NGINX as the reverse proxy server and builds both the Docker images with suitable configurations. (For more information, see respective Docker files). For user convenience, this example has automated the end-to-end running of MNIST which includes starting of the BIND9 and NGINX containers. For more information on arguments passed to the respective run scripts of swarm components, see `run-all` script.

This example uses one training batch and one test batch. Both batch files are stored in an archive file called `mnist.npz`. The ML program, after conversion to Swarm Learning for the TensorFlow-based Keras platform, is in `examples/reverse-proxy/mnist/model` and the TensorFlow-based file name is `mnist_tf.py`.

This example shows the Swarm training of MNIST model using two ML nodes. ML nodes are automatically spawned by SWOP node. Swarm training is initiated by SWCI node and orchestrated by a SN node. This example also shows how private data and shared model can be mounted to ML nodes for Swarm training. For more information, see the profile files and task definition files placed under `examples/reverse-proxy/mnist/swop` and `examples/reverse-proxy/mnist/swci` folders, respectively.

For example, if the 192.18.0.0 subnet is used in the network that is created as part of the reverse proxy example prerequisite, the IP addresses of BIND9 and NGINX may be 192.18.0.1 and 192.18.0.2, respectively. The following table shows how the corresponding swarm components are incremented by 1 in the last octet of this IP address.

NOTE: These are the container IP addresses.

Table 2: Container IP addresses

SNo	Container	IP Address
1	SN-1-IP	192.18.0.3
2	SWOP-1-IP	192.18.0.4
3	SWCI-1-IP	192.18.0.5
4	SL-1-IP	192.18.0.6
5	ML-1-IP	192.18.0.7
6	SL-2-IP	192.18.0.8
7	ML-2-IP	192.18.0.9

[9] **https://yann.lecun.com/exdb/mnist/**
[10] **https://www.tensorflow.org/tutorials/quickstart/beginner**

Table 3: IP mapping with FQDNs in Bind9 Container

SNo	FQDN	IP Address
1	`api.sn-1.swarm`	172.1.1.1
2	`p2p.sn-1.swarm`	172.1.1.1
3	`fs.sl-1.swarm`	172.1.1.1
4	`fs.sl-2.swarm`	172.1.1.1

Table 4: NGINX Configuration

SNo	FQDN	IP with Port
1	`api.sn-1.swarm`	192.18.0.3:30304
2	`p2p.sn-1.swarm`	192.18.0.3:30303
3	`fs.sl-1.swarm`	192.18.0.6:30305
4	`fs.sl-2.swarm`	192.18.0.8:30305

Running the MNIST example using Reverse Proxy

1. On host-1, navigate to swarm-learning folder.

   ```
   cd swarm-learning
   ```

   NOTE: Ensure that the prerequisites are met. For more information about prerequisite, see **Prerequisite for running reverse Proxy examples**.

2. Run the `run-all` script from the `swarm-learning` folder with `APLS IP` argument and `Network name`. Assume as part of **Prerequisite for running reverse Proxy examples**, the network created is `rp-network`.

   ```
   ./examples/reverse-proxy/mnist/run-all 172.1.1.1 rp-network
   ```

   First argument `APLS IP` is the IP address of the APLS. This is a mandatory parameter. This argument is passed as the current host IP by assuming that the APLS is running on the same host.

   Second argument `Host_IP` is the IP address of the current host. This argument is used for the NGINX and DNS configurations.

   This script is used for starting BIND9 container, NGINX container, and rest all of the swarm containers in a sequential manner. All the run-script commands now uses FQDN's as its service parameter arguments instead of IP and Ports.

   NOTE: SN-P2P-PORT still needs 30303 port in the host machine, where SN container runs.

3. On host-1, Swarm training is automatically started when the run task (`swarm_mnist_task`) gets assigned and executed. Open a new terminal on host-1 and monitor the Docker logs of ML nodes for Swarm training. Swarm training will end with the following log message.

   ```
   SwarmCallback : INFO : All peers and Swarm training rounds finished. Final Swarm model was loaded.
   ```

   The final Swarm model is saved inside `workspace/reverse-proxy/mnist/model` directory on host-1. All the dynamically spawned SL and ML nodes exit after Swarm training.

4. On host-1, to clean-up, run the `scripts/bin/stop-swarm` script to stop and remove the swarm container nodes of the previous run. If required, backup the container logs. This example builds and starts BIND9 and NGINX, so

remove their respective images and containers. Then, remove docker volume (`sl-cli-lib`) and delete the `workspace` directory.

CIFAR-10 using Reverse Proxy

This example runs Reverse Proxy with CIFAR-10 [11] on a Swarm Learning platform. It uses TensorFlow as the backend.

This example uses CIFAR-10 dataset distributed along with tensorflow package. The ML program, after conversion to Swarm Learning, is in `swarm-learning/examples/reverse-proxy/cifar10/model` and is called `cifar10.py`. It contains a tiny ML model for the purpose of showing steps of converting ML code for Swarm Learning.

This example shows the CIFAR-10 example using reverse proxy to mimic the real-world behavior. This example uses BIND9 [12] as the DNS server and NGINX [13] as the reverse proxy server and builds both the docker images with suitable configurations. (For more information, see respective Docker files). For user convenience, this example has automated the flow of running CIFAR-10 example which includes starting of the BIND9 and NGINX containers. For more information on arguments passed to the respective run scripts of swarm components, see `run-on-host-1` and `run-on-host-2` scripts.

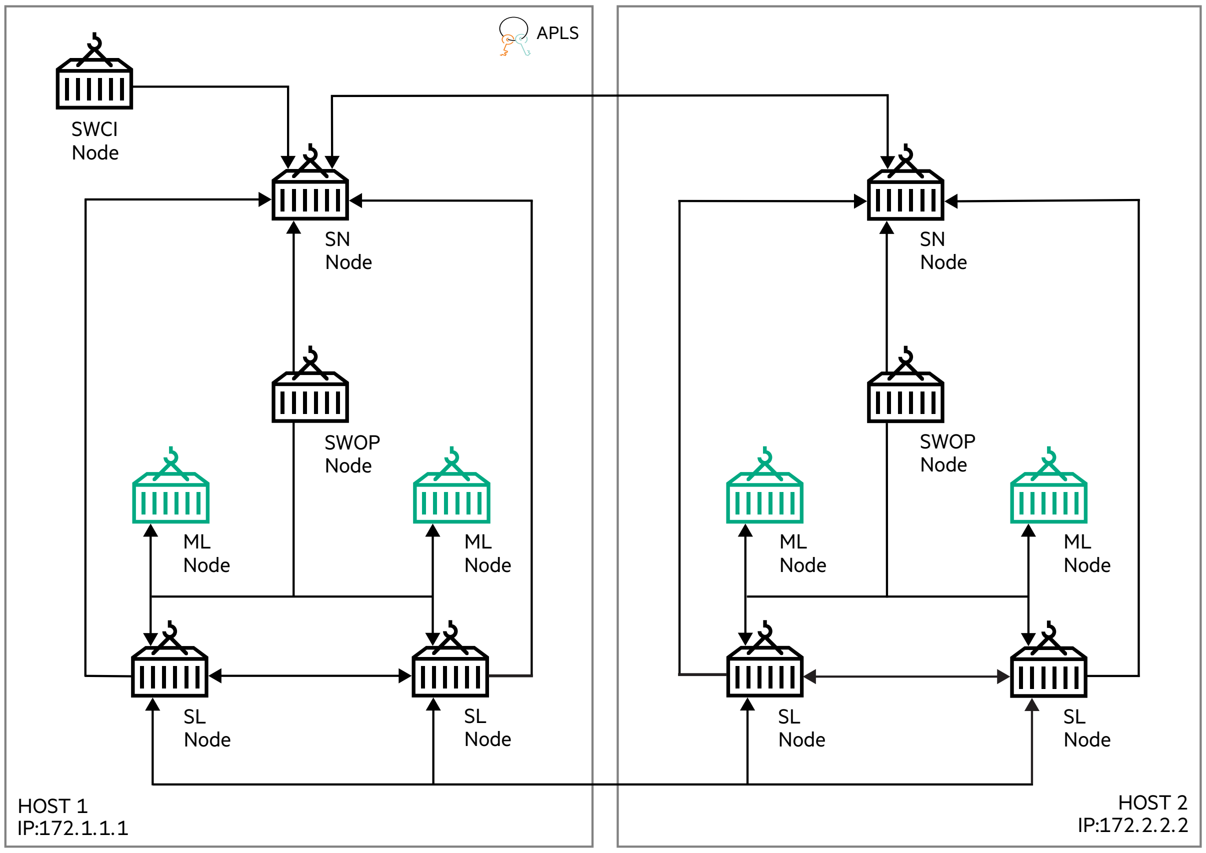
This example shows the Swarm training of CIFAR-10 model using four ML nodes. ML nodes are automatically spawned by SWOP nodes running on two different hosts. Swarm training is initiated by SWCI node and orchestrated by two SN nodes.

Cluster Setup

The following image illustrates the cluster setup for the CIFAR-10 example which uses two hosts:

- host-1: 172.1.1.1
- host-2: 172.2.2.2

[11] **https://www.cs.toronto.edu/%7Ekriz/cifar.html**
[12] **https://www.isc.org/bind/** and **https://bind9.readthedocs.io/**
[13] **https://www.nginx.com/** and **https://nginx.org/en/docs/**



1. This example uses two SN nodes. The names of the docker containers representing these two nodes are sn-1 and sn-2, where sn-1 is the Sentinel Node and sn-2 is a Non-Sentinel Node. sn-1 runs on host 172.1.1.1 and sn-2 runs on host 172.2.2.2.
2. SL and ML nodes are automatically spawned by SWOP nodes during training.
3. This example uses two SWOP nodes - one connects to each SN node. The names of the docker containers representing these two SWOP nodes are swop-1 and swop-2. swop-1 runs on host 172.1.1.1 and swop-2 runs on host 172.2.2.2.
4. Training is initiated by SWCI node (swci-1) that runs on host 172.1.1.1.
5. This example assumes that License Server already runs on host 172.1.1.1. All Swarm nodes connect to the License Server, on its default port 5814.

For example, if the 192.18.0.0 subnet is used in the network that is created on host-1 as part of reverse proxy example prerequisite, then the IP addresses of BIND9 and NGINX may be 192.18.0.1 and 192.18.0.2, respectively. The following table shows how the corresponding swarm components are incremented by 1 in the last octect of this IP address.

NOTE: These are the container IP addresses.

SNo	Container	IP Address
1	SN-1-IP	192.18.0.3
2	SWOP-1-IP	192.18.0.4
3	SWCI-1-IP	192.18.0.5

Table Continued

SNo	Container	IP Address
4	SL-1-IP	192.18.0.6
5	ML-1-IP	192.18.0.7
6	SL-2-IP	192.18.0.8
7	ML-2-IP	192.18.0.9

Similarly, if the 192.19.0.0 subnet is used in the network that is created on host-2 as part of reverse proxy example prerequisite, then the IP addresses of BIND9 and NGINX may be 192.19.0.1 and 192.19.0.2, respectively. The following table shows how the corresponding swarm components are incremented by 1 in the last octect of this IP address.

SNo	Container	IP Address
1	SN-2-IP	192.19.0.3
2	SWOP-2-IP	192.19.0.4
3	SL-3-IP	192.19.0.5
4	ML-3-IP	192.19.0.6
5	SL-4-IP	192.19.0.7
6	ML-4-IP	192.19.0.8

Table 5: DNS Configuration on both hosts

SNo	FQDN	IP Address
1	`api.sn-1.swarm`	172.1.1.1
2	`p2p.sn-1.swarm`	172.1.1.1
3	`fs.sl-1.swarm`	172.1.1.1
4	`fs.sl-2.swarm`	172.1.1.1
5	`api.sn-2.swarm`	172.2.2.2
6	`p2p.sn-2.swarm`	172.2.2.2
7	`fs.sl-3.swarm`	172.2.2.2
8	`fs.sl-4.swarm`	172.2.2.2

Table 6: NGINX Configuration on Host-1

SNo	FQDN	IP Address
1	`api.sn-1.swarm`	192.18.0.3:30304
2	`p2p.sn-1.swarm`	192.18.0.3:30303
3	`fs.sl-1.swarm`	192.18.0.6:30305
4	`fs.sl-2.swarm`	192.18.0.8:30305

Table 7: NGINX Configuration on Host-2

SNo	FQDN	IP Address
1	`api.sn-2.swarm`	192.19.0.3:30304
2	`p2p.sn-2.swarm`	192.19.0.3:30303
3	`fs.sl-3.swarm`	192.19.0.5:30305
4	`fs.sl-4.swarm`	192.19.0.7:30305

Prerequisite for running CIFAR-10 example using Reverse Proxy

1. SN-P2P-Service still relies on the 30303 port. Ensure that the 30303 port is open between both the hosts.
2. Ensure that the password-less SSH setup is done on both the machines. These automated scripts use SSH and SCP to check and transfer certificate-related PEM files between the hosts.

Running the CIFAR-10 example using Reverse Proxy

1. On both host-1 and host-2, navigate to `swarm-learning` folder (that is, parent to `examples` directory).

   ```
   cd swarm-learning
   ```

   NOTE: Ensure that the prerequisites are met. For more information about prerequisite, see **Prerequisite for running reverse Proxy examples**.

2. On host-1, run the `run-on-host-1` script from the `swarm-learning` folder with arguments `APLS_IP`, `Host_1_IP`, `Host_2_IP`, `Host_1_DNS_IP`, `Host_2_USER`, `Host_2_INSTALL_DIR` and `Network_Name`.

   - `APLS_IP` is the IP address of the APLS.
   - `Host_1_IP` is the IP address of the host-1.
   - `Host_2_IP` is the IP address of the host-2.
   - `Host_1_DNS_IP` is the DNS IP of the host-1. This ensures that the SL and ML containers use two DNS IP's for name resolution. One is the host-1 DNS IP and another one is the IP of the host-1 BIND9 container.
   - `Host_2_USER` is the current user on the host-2 machine. If the `Host_2_USER` is empty, then it uses the default user.
   - `Host_2_INSTALL_DIR` is the location where SL is installed on host-2 (For example, `/home/test2/swarm-learning`). If `Host_2_INSTALL_DIR` is not set, then the SL is installed in the default installation directory of swarm, that is, `/opt/hpe/swarm-learning`.
   - `Network_Name` is the custom bridge network which is created as part of the prerequisites, to run the reverse proxy examples. For more information on prerequisite, see **Prerequisite for running reverse Proxy examples**.

     For instance, if the `Host_1_DNS_IP` is 172.3.3.3;

     `Host_2_USER` is test2;

     `Host_2_INSTALL_DIR` is `/home/test2/swarm-learning`; and

     `Network_Name` is rp-network-1. This is the network created in host 1.

Then, the Run command displays as follows:

```
./examples/reverse-proxy/cifar10/run-on-host-1 172.1.1.1 172.1.1.1
172.2.2.2 172.3.3.3 test2 /home/test2/swarm-learning rp-network-1
```

This step creates the workspace directory, moves files from examples to workspace, creates a common path between hosts for the ML program, generates certificates, creates a volume for the wheel file, and shares certificate-related PEM files. This step also starts the BIND9 container, the NGINX container, and the rest of all the swarm containers specific to host-1 in a sequential manner. All the run-script commands now uses FQDNs as service parameter arguments instead of ports.

3. On host-2, run the `run-on-host-2` script from the `swarm-learning` folder with arguments `APLS_IP`, `Host_1_IP`, `Host_2_IP`, `Host_2_DNS_IP`, `Host_1_USER` and `Host_1_INSTALL_DIR`.

   - `APLS_IP` is the IP address of the APLS.
   - `Host_1_IP` is the IP address of the host-1.
   - `Host_2_IP` is the IP address of the host-2.
   - `Host_2_DNS_IP` is the DNS IP of the host-2. This ensures that the SL and ML containers use two DNS IP's for name resolution. One is the host-2 DNS IP and another one is the IP of the host-2 BIND9 container.
   - `Host_1_USER` is the current user on the host-1 machine. If the `Host_1_USER` is empty, then it uses the default user.
   - `Host_1_INSTALL_DIR` is the location where SL is installed on host-1 (for example, `/home/test1/swarm-learning`). If `Host_1_INSTALL_DIR` is not set, then the SL is installed in the default installation directory of swarm, that is, `/opt/hpe/swarm-learning`.
   - `Network_Name` is the custom bridge network created as part of the prerequisites to run the reverse proxy examples. For more information about prerequisite, see **Prerequisite for running reverse Proxy examples**.

     For instance, if `Host_2_DNS_IP` is 172.4.4.4;

     `Host_1_USER` is test1;

     `Host_1_INSTALL_DIR` is `/home/test2/swarm-learning`; and

     `Network_Name` is rp-network-2. This is the network created in host 2.

     Then, the Run command displays as follows:

     ```
     ./examples/reverse-proxy/cifar10/run-on-host-2 172.1.1.1 172.1.1.1
     172.2.2.2 172.4.4.4 test1 /home/test1/swarm-learning rp-network-2
     ```

     This step creates the workspace directory, moves files from examples to workspace, creates a common path between hosts for the ML program, generates certificates, creates a volume for the wheel file, and shares certificate-related PEM files. This step also starts the BIND9 container, the NDINX container, and the rest of all the swarm containers specific to host-2 in a sequential manner. All the run-script commands will now use FQDNs as service parameter arguments instead of ports.

4. Swarm training is automatically started when the run task (`swarm_mnist_task`) gets assigned and executed. Open a new terminal on either host-1 or host-2 and monitor the docker logs of ML nodes for Swarm training. Swarm training will end with the following log message:

   ```
   SwarmCallback : INFO : All peers and Swarm training rounds finished. Final Swarm model was loaded.
   ```

Final Swarm model is saved inside `/tmp/reverse-proxy/cifar10/model` directory on both host-1 and host-2. All the dynamically spawned SL and ML nodes exit after Swarm training. The SN and SWOP nodes continue to run.

5. To clean up, run the `scripts/bin/stop-swarm` script on the host system to stop and remove the swarm container nodes of the previous run. If needed, backup the container logs. This example builds and starts BIND9 and NGINX, so remove their respective images and containers. Then, remove docker volume (`sl-cli-lib`) and delete the `workspace` directory.

Examples using SPIRE

CIFAR-10 using SPIRE

This example runs CIFAR-10 [14] example using Spire [15] as a certificate manager on the Swarm Learning platform. It uses TensorFlow as the backend.

This example uses CIFAR-10 dataset distributed along with tensorflow package. The ML program, after conversion to Swarm Learning, is in `swarm-learning/examples/spire/cifar10/model` and is called `cifar10.py`. It contains a tiny ML model for the purpose of showing steps of converting ML code for Swarm Learning.

For user convenience, this example has automated the flow of running CIFAR-10 example which includes starting of the Spire server and Spire agent containers. For more information on arguments passed to the respective run scripts of swarm components, see `run-on-host-1` and `run-on-host-2` scripts.

This example shows the Swarm training of CIFAR-10 model using four ML nodes. ML nodes are automatically spawned by SWOP nodes running on two different hosts. Swarm training is initiated by SWCI node and orchestrated by two SN nodes.

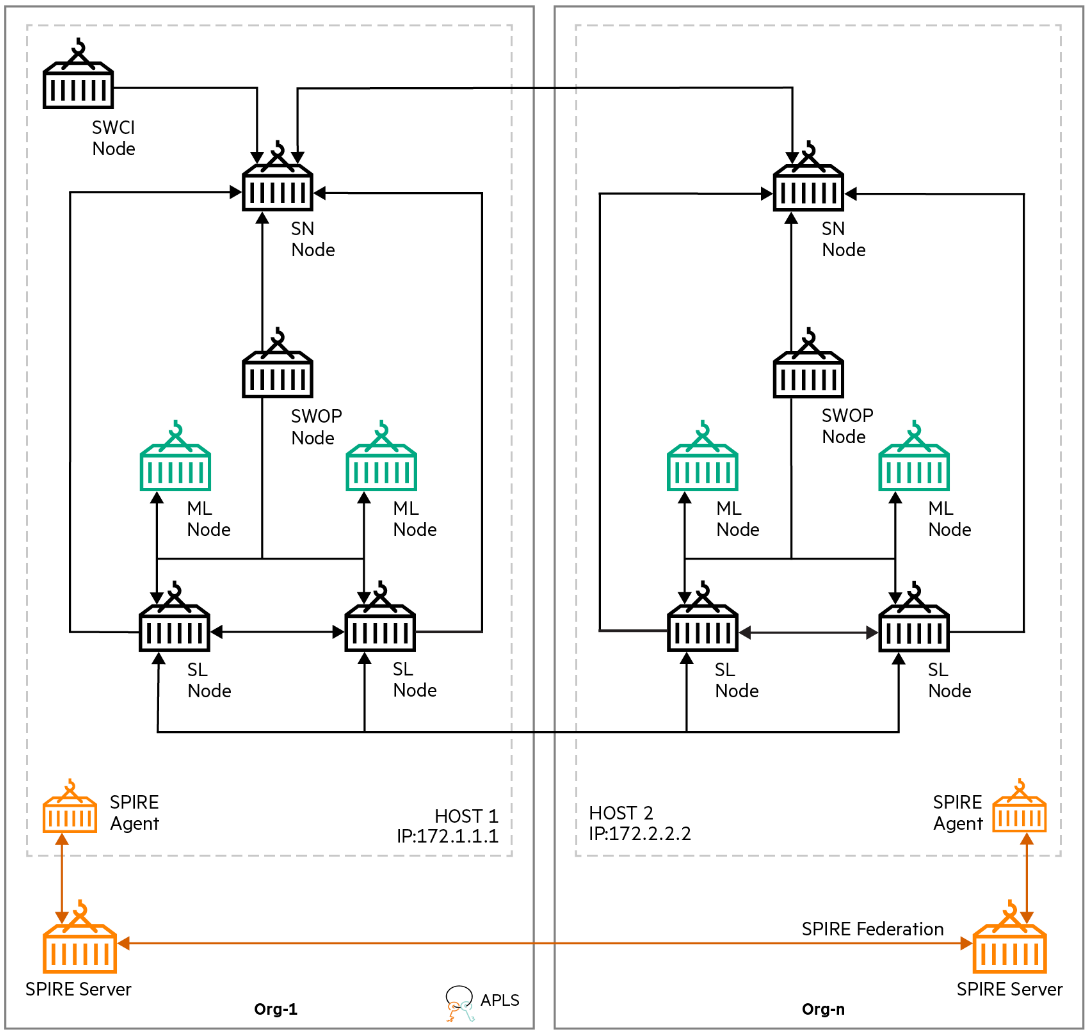
Cluster Setup

The following image illustrates the cluster setup for the CIFAR-10 example which uses two hosts:

- host-1: 172.1.1.1
- host-2: 172.2.2.2

[14] V.N. a. G. H. Alex Krizhevsky, "CIFAR-10 and CIFAR-100 datasets," [Online]. Available: **https://www.cs.toronto.edu/~kriz/cifar.html**

[15] **https://spiffe.io/docs/latest/spire-about/spire-concepts/**



1. This example uses two SN nodes. The names of the docker containers representing these two nodes are sn-1 and sn-2, where sn-1 is the Sentinel Node and sn-2 is a Non-Sentinel Node. sn-1 runs on host 172.1.1.1 and sn-2 runs on host 172.2.2.2.
2. SL and ML nodes are automatically spawned by SWOP nodes during training.
3. This example uses two SWOP nodes - one connects to each SN node. The names of the docker containers representing these two SWOP nodes are swop-1 and swop-2. swop-1 runs on host 172.1.1.1 and swop-2 runs on host 172.2.2.2.
4. Training is initiated by SWCI node (swci-1) that runs on host 172.1.1.1.
5. This example assumes that License Server already runs on host 172.1.1.1. All Swarm nodes connect to the License Server, on its default port 5814.

Prerequisite for running CIFAR-10 example using SPIRE

1. Hosts must be able to pull spire-server and spire-agent docker images from the git hub container registry[16].
2. Ensure that the password-less SSH setup is done on both the machines. These automated scripts use SSH and SCP to check and transfer certificate-related PEM files between the hosts.

[16] **https://docs.github.com/en/packages/working-with-a-github-packages-registry/working-with-the-container-registry**

Running the CIFAR-10 example using Spire

1. On both host-1 and host-2, navigate to `swarm-learning` folder (that is, parent to `examples` directory).

   ```
   cd swarm-learning
   ```

2. On host-1, run the `run-on-host-1` script from the `swarm-learning` folder with arguments `APLS_IP`, `Host_1_IP`, `Host_2_IP`, `Host_2_USER`, and `Host_2_INSTALL_DIR`.

   - `APLS_IP` is the IP address of the APLS.
   - `Host_1_IP` is the IP address of the host-1.
   - `Host_2_IP` is the IP address of the host-2.
   - `Host_2_USER` is the current user on the host-2 machine. If the `Host_2_USER` is empty, then it uses the default user.
   - `Host_2_INSTALL_DIR` is the location where SL is installed on host-2 (For example, `/home/test2/swarm-learning`). If `Host_2_INSTALL_DIR` is not set, then the SL is installed in the default installation directory of swarm, that is, `/opt/hpe/swarm-learning`.

     For instance, if `Host_2_USER` is test2;

     `Host_2_INSTALL_DIR` is `/home/test2/swarm-learning`.

     Then, the Run command displays as follows:

     ```
     ./examples/spire/cifar10/run-on-host-1 172.1.1.1 172.1.1.1 172.2.2.2 test2 /home/test2/swarm-learning
     ```

     This step creates the `workspace` directory, moves files from `examples` to `workspace`, creates a common path between hosts for the ML program, creates a volume for the wheel file, and shares certificate-related bundle files. This step also starts the spire server container, the spire agent container, and the rest of all the swarm containers specific to host-1 in a sequential manner.

3. On host-2, run the `run-on-host-2` script from the `swarm-learning` folder with arguments `APLS_IP`, `Host_1_IP`, `Host_2_IP`, `Host_1_USER` and `Host_1_INSTALL_DIR`.

   - `APLS_IP` is the IP address of the APLS.
   - `Host_1_IP` is the IP address of the host-1.
   - `Host_2_IP` is the IP address of the host-2.
   - `Host_1_USER` is the current user on the host-1 machine. If the `Host_1_USER` is empty, then it uses default user.
   - `Host_1_INSTALL_DIR` is the location where SL is installed on host-1 (for example, `/home/test1/swarm-learning`). If `Host_1_INSTALL_DIR` is not set, then the SL is installed in the default installation directory of swarm, that is, `/opt/hpe/swarm-learning`.

     For instance, if `Host_1_USER` is test1;

     `Host_1_INSTALL_DIR` is `/home/test1/swarm-learning`.

     Then, the Run command displays as follows:

     ```
     ./examples/spire/cifar10/run-on-host-2 172.1.1.1 172.1.1.1 172.2.2.2 test1 /home/test1/swarm-learning
     ```

     This step creates the `workspace` directory, moves files from `examples` to `workspace`, creates a common path between hosts for the ML program, creates a volume for the wheel file, and shares certificate-related bundle files. This step also starts the spire server container, the spire agent container and the rest of all the swarm containers specific to host-2 in a sequential manner.

4. Swarm training is automatically started when the run task (`swarm_mnist_task`) gets assigned and executed.

   Open a new terminal on either host-1 or host-2 and monitor the docker logs of ML nodes for Swarm training. Swarm training will end with the following log message:

   ```
   SwarmCallback : INFO : All peers and Swarm training rounds finished. Final Swarm model was loaded.
   ```

   Final Swarm model is saved inside `/tmp/spire/cifar10/model/saved_models` directory on both host-1 and host-2. All the dynamically spawned SL and ML nodes exit after Swarm training. The SN and SWOP nodes continue to run.

5. To clean up, run the `scripts/bin/stop-swarm` script on the host system to stop and remove the swarm container nodes of the previous run. If needed, backup the container logs. This example builds and starts spire server and spire agent containers, so remove their respective images and containers. Then, remove docker volume (`sl-cli-lib`) and delete the `workspace` directory.

Frequently asked questions

Generic

What is Swarm Learning?

Swarm Learning is a decentralized, privacy-preserving Machine Learning framework. This framework utilizes the computing power at, or near, the distributed data sources to run the Machine Learning algorithms that train the models. It uses the security of a blockchain platform to share learnings with peers in a safe and secure manner.

What are the components of Swarm Learning?

Swarm Learning has 4 types of components that form a network. They are Swarm Learning nodes, Swarm Network nodes, SWCI nodes, and SWOP nodes.

What is the License server node?

The License server (APLS) node is a special node running the HPE AutoPass license server. It is responsible for validating the licenses of the Swarm Learning framework. There is typically one instance of this node running in the Swarm.

How do we resolve license related issues?

User can raise the support ticket through this URL: **https://myenterpriselicense.hpe.com/cwp-ui/contact-us**.

How do you run Swarm Learning on GPU?

The SL, SN, SWOP, and SWCI nodes utilize only the CPUs. However, the user ML nodes can run on Nvidia GPUs by using the GPU version of your underlying ML platform (Keras/PyTorch) and following the GPU specific instructions of your ML platform. For more information on starting SL and ML nodes, see *HPE Swarm Learning Installation and Configuration Guide*.

For Nvidia GPUS, you can set `--gpus` under `usrcontaineropts` section of the SWOP profile. For more information, see **https://docs.docker.com/config/containers/resource_constraints/#gpu**.

If you are starting the SL and ML nodes by using the `run-sl` script, then the GPUs can be specified as appropriate environment variables by using `--ml-e` option.

For AMD GPUs, you can set `usercontaineropts` and/or `usrenvvars` section of the SWOP profile. For more information, see **SWOP profile schema**.

If you are starting the SL and ML nodes by using the `run-sl` script, then the GPUs can be specified as appropriate parameters as specified in the User machine learning container parameters section. For more information on User machine learning container parameters, see *HPE Swarm Learning Installation and Configuration Guide*.

What all GPUs are supported?

Currently SWOP framework is designed to start ML nodes on Nvidia GPUs and AMD GPUs. In the future other GPUs may be supported.

You can set `--gpus` under `usrcontaineropts` section for the specific user container in the SWOP profile.

How can you determine if AMD GPUs are allocated for local training?

You can check for Cuda/Gpu availability from your application code.

PyTorch:

torch.cuda.is_available()

https://pytorch.org/docs/stable/generated/torch.cuda.is_available.html

In TensorFlow:

tensorflow.test.is_gpu_available()

https://www.tensorflow.org/api_docs/python/tf/test/is_gpu_available

What are the additional steps to be followed to enable GPU access for local training?

The additional steps to enable GPUs for local training are as follows:

1. Build the user container to enable GPU access in it. Use base image as tensorflow-gpu or Nvidia image with PyTorch installed on it as applicable to ML platform.
2. Update SWOP profile `usrcontaineropts` (or) provide run-scripts options as applicable to Nvidia or AMD.
3. Create application code to access GPU.

Can you have heterogeneous ML nodes, with some running on CPU and others on GPU?

Yes.

Each ML node by default runs on CPU. If you want to run on GPUs, specify it in the `usrcontaineropts` section in the SWOP profile.

Can you run multiple concurrent Model Trainings in the same Swarm Network?

Yes, it is supported only in the licensed version. User can even run one training session using Keras and another using PyTorch.

User need differents training contracts specified in the ML programs via `Swarmcallback` API.

If user is using SWOP to launch concurrent training, user need to have separate SWOP nodes each watching a different taskrunner, which is specified in their SWOP profiles.

Concurrent model training uses the same Swarm SN network across different model trainings, sharing the blockchain.

If user is using SLM-UI, then user needs to create different projects for each model training. User must create a new taskrunner and training contract in SLM-UI and use the same in the SWOP profile and ML program respectively, for each project.

User needs to spawn the SWOP nodes and execute the RUN-SWARM type task using the respective taskrunners of each project.

What is the IP address used in the run scripts?

The `--host-ip` and `slhostip` IP addresses in the run scripts and the SWOP profile are the IP addresses of the host machine, where the respective containers are running on the host machine. Based on access, user can even use the FQDN of the host system.

By default, Swarm Learning framework uses a Docker bridge network. For improved isolation, users can even use a user-defined bridge network.

While using the user-defined bridge network, the options `--ip` and `ip` field of `slnetworkopts` in SWOP profile are the IP addresses of the container themselves. This case is specific to the reverse proxy examples or scenarios where user wants to use the fixed IP addresses for containers.

Where are the log files?

The system log files are the docker logs. By default, the docker containers that run the SN, SL, ML, SWOP, and SWCI nodes are not removed after they exit. Log output produced by these containers can be retrieved using the docker logs command.

For a SL node, a subset of the log output is stored with the name `<program-name>_sw.log`, in the model directory.

The ML program can produce additional log output. To do so, it should be modified to write this output to files in the model directory.

What network ports does Swarm Learning use? Can they be customized?

Each SN node requires two network ports for incoming connections from other SN, SL, SWCI, and SWOP nodes.

- One SN to SN peer to peer communication port - is meant for peer-to-peer communication using the underlying blockchain platform's protocols. By default, port 30303 is used.
- One SN API server port - is meant for running a REST-based API server on each SN node. By default, port 30304 is used.

Each SL node requires one network port for incoming connections from other SL nodes.

A SL file server port - is meant for running a file server on each SL node. By default, port 30305 is used.

Each License Server node requires one network port for incoming connections from other nodes.

A License Server API server port - is meant for running a REST-based API server. By default, port 5814 is used.

(Optional) A SWCI API server port - is used by the SWCI node to run a REST-based API service. By default, port 30306 is used.

The port numbers can be customized by using the corresponding `swarm-learning/scripts/bin/run-sn`, `swarm-learning/scripts/bin/run-sl`, and `swarm-learning/scripts/bin/run-swci` scripts that are supplied with the Swarm Learning package. Use the `–help` option on the above scripts to get exact details.

For configuring the license server API port, see *AutoPass License Server User Guide*.

Do you need sudo/root privileges to run Swarm Learning?

`sudo` is not required to launch the container, if docker is configured to run as a non-root user. Refer Manage Docker as a non-root user If docker is not configured to run as a non-root user, the scripts will automatically prefix docker commands with `sudo`. If the user does not have `sudo` privileges, an error will result.

NOTE: Effective user inside the docker container should be root.

How do you uninstall Swarm learning?

Use the docker log command to save any container log output that you want to preserve. Use a directory outside the Swarm Learning installation directory. Also, consider cleaning the model directories by removing unnecessary files and sub-directories.

Use the `swarm-learning/scripts/bin/uninstall` script to uninstall the Swarm Learning package. This script does not accept any command line parameters. It should run on every node where Swarm Learning package is installed. While running, it stops all Swarm Learning components that are running on that host, removes the docker container images, and deletes the Swarm Learning installation directory.

When to pass proxy as environment variables while running Swarm Learning?

All Swarm components accepts proxy as environment variables. If the network configuration has a proxy server, user can easily pass it through the `-e http_proxy`, `-e https_proxy`, and `-e no_proxy` values from UI or from CLI commands. SWOP being an orchestrator, performs tasks that sometimes needs proxy for building images and starting SL-ML pairs. In this case, SWOP additionally accepts the proxy values from the SWOP profile under `env` section. These are available on secured environments.

In non-proxy environments, user can use these variables with empty values so that Swarm containers can also start with the same non-proxy configuration.

Can any user in the system run swarm training?

HPE recommends to run with the same user who installed Swarm. Otherwise, ensure adequate access privilege for the 'other user' on the directories used to run the swarm training.

Swarm Network (SN) node

What is the Sentinel node?

The Sentinel node is a special Swarm Network node. It is responsible for initializing the blockchain network and deploying the smart contracts on the blockchain. For this reason, the Sentinel node should be the very first Swarm Network node that is started in the Swarm Learning framework. Once the blockchain network has been initialized, there is no difference between the functioning of the Sentinel node and that of the other Swarm Network nodes.

How do you know if Swarm Network node started successfully?

Look for following message after executing `run-sn` command to confirm successful starting of Swarm Network node. It might take a few minutes before this message appears.

`swarm.blCnt : INFO : Starting SWARM-API-SERVER on port :30304` (30304 is the default port).

This message does not show up if APLS server is not configured correctly.

What are the possible reasons for "unable to contact API server"?

A swarm container could be unable to reach an SN node for several reasons. For more information, see **`Unable to contact API-Server`**.

How many SLs can connect with a SN?

It depends on several factors like, the available system resources, the ML algorithm complexity, how often it does parameter merging and so on.

On a Proliant XL Gen 9 system with 8 Xeon CPUs and 32 GB memory HPE has tested and found up to 16 SLs could connect with 1 SN, when running a MNIST training with 100 epochs.

HPE recommends starting up to 4 SLs to 1 SN and scale it up slowly if needed.

When you start SWCI, you do not specify any IP/name for SN. How does it know which SN to connect to?

SWCI is designed to work with several swarm networks at once. Therefore, you can create a context and switch to that context to execute commands. Each context identifies which SN the SWCI must connect to.

Why is blockchain required? Can you use a different blockchain network?

Swarm Learning uses a blockchain network primarily to provide a consistent system state to all the nodes without requiring any central coordinator.

The current implementation runs an open-source version of Ethereum but, more platforms might be added in the future. At the time of initialization, the framework spawns its own blockchain network with a custom set of parameter values. Hence it cannot be replaced with any other blockchain network. This applies even when the blockchain platform is a supported one.

Where is the blockchain stored on SN container?

By default, blockchain is stored in the `/platform/swarm/SMLNODE` path inside SN containers. Blockchain is not preserved by default. To preserve it, user have to mount `/platform/swarm/SMLNODE` on a persistent volume on the host while starting SN. For more information, see *Starting Sentinel node* section in *HPE Swarm Learning Installation and Configuration Guide*.

SL and ML

What are the supported machine learning platforms?

Swarm Learning supports Python3 based Machine Learning models that uses PyTorch and Keras (based on TensorFlow 2).

What models work with swarm learning?

Currently, Swarm Learning works only with **parametric** machine learning models. For example, NN, CNN, RNN, LSTM, and many more. Its also supports Transfer Learning (models which includes mix of trainable and non-trainable parameters).

Support for other ML models is part of Swarm roadmap.

What are the supported Python packages in ML node?

Any Python package can be used to build the ML container.

If SWOP framework is used, packages must be specified in the build-task definition file.

What is the guidance on minPeers in the application vs "WITH PEERS in the assign run task"?

`minPeers` specifies the minimum number of ML peers (quorum) that must be available (and able to communicate to each other) to continue the Swarm training. Otherwise, the Swarm training gets blocked indefinitely.

`WITH peersNeeded` in the `assign task` is used to start the number of SL, ML pairs using the SWOP framework. Also, this count is used to decide the overall status of the task.

`WITH peersNeeded` in the `assign task` must be equal to or greater than the `minPeers`. It can be closer to the total number of ML peers in the Swarm training.

What happens if a node runs slowly or drops out of the network?

Swarm Learning has a configurable parameter called `minPeers`, which is the minimum number of nodes essential at each sync step for the model training to continue. The framework ensures that a node can contribute in a sync step only if it is up to date with the model derived from the previous sync step.

The scenario of a node running at a slower rate than the others or completely dropping out of the network can lead to two situations:

- The number of remaining nodes is greater than or equal to `minPeers`.
- The number of remaining nodes is less than `minPeers`.

In the first case where the number of remaining nodes is greater than or equal to `minPeers`, the training will continue post the sync step with the remaining nodes. Once the dropped node rejoins the network, it will update its model to the latest one. It will then resume contributing to model training from the succeeding sync steps.

In case of a slow running node, however, the training will continue with contributions from the remaining nodes. The contributions from the slow node are merged periodically using a patented logic.

In the case where the number of nodes remaining in the network is less than `minPeers`, the training will pause at the sync step till the minimum number is met again. This can occur either when a dropped node rejoins the network or, when a slow node reaches the sync step.

Can I add new nodes into the network?

Yes. New nodes can be added in the network at any point in the training. Just like a dropped node, a new node will resume model training from the latest model derived from the last sync step.

What are the supported merge methods? Can I specify a custom merge method?

Swarm Learning uses weighted mean as the default merge method. User can specify one of the merge methods such as mean, coordinate wise median, and geometric wise median. Currently, users cannot specify their own merge method. This will be supported in a later release.

How to choose the suitable merge method among mean, coordinate median and geometric median methods?

User must consider this merge method as one of the Swarm hyper parameters and expected to experiment with it. Mean is the default merge method. It works well for most of the use cases. HPE recommends you to try coordinate median and geometric median methods under the following circumstances.

- The coordinate or geometric median merge methods are recommended for scenarios where model is relatively complex and takes longer duration to run each epoch. For such complex models, the rate at which the model converges over epochs is better in median-based merge methods. If user has the flexibility to run ML model for higher number of epochs, then all the merge methods may give similar results.
- The strong bias nature of the datasets tends to have non-uniform weights and biases in the intermediate models across ML nodes. For such biased datasets, coordinate median and geometric median methods are known to perform better than the mean method.

For more information, see *Merge Methods in Swarm Learning Whitepaper*.

Before enabling Swarm Learning, how to confirm the standalone user application has no issues and runs?

Run the user container with `SWARM_LOOPBACK` set to `TRUE`, this bypasses Swarm Learning to help you quickly develop, integrate, and test your model code with Swarm Learning package. If your code runs to completion and saves the local model it would indicate that the ML application may not have any issues.

If `SWARM_LOOPBACK` is set to `TRUE`, all Swarm functionality is bypassed, except parameter validation.

This can help you to verify and test integration of the model code with Swarm without spawning any Swarm Learning containers.

How to run user container as non-root?

By default, when user ML container is run through SWOP or using the `run-sl` script, the user ML container is run with current user's UID and GID of the host machine. If the current user on the host is non-root, the user container also runs as non-root.

How to mount data/example/file to ML container if “Swarm install directory" or "source file path” is different across hosts?

SWOP profile supports mounts with private data. If the installation path or any file path is different across hosts, then `privatedata` field of SWOP profile can be used to mount. User can specify different values for `privatedata` field specific to each ML container. Mount target path is in the `PrivateContent` field in the run task definition. It is the same for all ML containers, and hence ML applications can access these files in the same manner.

Can each Swarm Learning node run a different ML program and parameters?

No. The program and parameters should be the same across all the Swarm Learning nodes.

Swarm management

What are the supported SWCI commands?

SWCI has a built-in inline help, that lists all supported commands and further one can see help for each command.

For Example,

```
SWCI:0 > HELP
    ASSIGN TASK
    CD
    CREATE CONTEXT
    CREATE CONTRACT
    …

SWCI:1 > HELP CREATE CONTRACT
    CREATE CONTRACT <TrainingContractName : string>
Registers the specified SL Training Contract into the Swarm Learning Network.
```

How to debug error with command "ASSIGN TASK TO TASKRUNNER"?

Use SWCI command "GET TASKRUNNER STATUS" to know the overall status of the TASK execution.

One can also use "GET TASKRUNNER PEER STATUS" to display the status for the individual SWOP PEERs that are listening on this TASKRUNNER.

- For RUN_SWARM task type, the status summary reports SWOP node UID, Number of SL PEERs this SWOP has spawned, and list of all SL node information (UID, Status, Description). For all other types of tasks, the status summary reports SWOP node status (UID, Status, Description).
- If there are failed PEERs, using its node UID, one can identify the container name/id from 'LIST NODES' command. With container name/id, user can debug the error with docker logs command.

How to assign task to all available SWOP peers?

Use `ASSIGN TASK <task-name> TO <taskrunner name> WITH ALL PEERS` to trigger a task to run on ***all available*** SWOP peers listening on the specified Taskrunner. The syntax is used to run a given task on all available nodes at that point of time, without manually counting on how many nodes are available across the consortium.

For more information on triggering this task from SLM-UI, see *Executing a task* section in *HPE Swarm Learning Installation and Configuration Guide*. For more information on triggering this task from CLI mode, see **SWCI commands related to Taskrunner**.

How to target a task on a failed (task) SWOP peer?

Use `ASSIGN TASK <task-name> TO <taskrunner name> TARGET <SWOP-UID>` to trigger a task to run on a particular SWOP. User can use `LIST NODES` command to identify the SWOP-ID of the desired SWOP Target. This can be useful to take corrective action by re-running a task that had failed earlier on a particular node.

For more information on triggering this task from SLM-UI, see *Executing a task* section in *HPE Swarm Learning Installation and Configuration Guide*. For more information on triggering this task from CLI mode, see **SWCI commands related to Taskrunner**.

Swarm Learning Management UI (SLM-UI)

What is a concept of SLM-UI project?

Project in SLM-UI is a logical representation of a particular Swarm training. Projects help to view deployment topology and monitor the progress for the given Swarm training. They define what all Swarm nodes (and associated host nodes) a training will run, the model being used, the x.509 certificates, SWOP and Task yaml files for a particular training. Multiple Projects can be defined in a single instance of SLM-UI.

Project artifacts are created under the `swarm-learning/slm-ui/projects/<project number>` automatically once the project is saved.

How can user monitor training progress?

In Project Nodes under Projects tab, the system displays all running swarm nodes associated with the project, loss, model metric (for example, accuracy) and overall training progress for each SL-ML node pair. User can hover over the mouse on progress bar to view the total number of epochs and the total number of completed epochs.

When do we need to create multiple task runners and contracts (Training contracts)?

If you are running concurrent Swarm training, you need to create multiple task runners and contracts. If you are running a single training, the default task runner and contract would be good enough.

How to start SLM-UI manually?

1. Run `<swarm-learning>/slm-ui/scripts/run-postgres -pw" supersecretpassword"`. (`supersecretpassword` is a default database password. User can change this default database password using external tools like pgAdmin).
2. Then, run `<swarm-learning>/slm-ui/scripts/run-slm-ui -pw" supersecretpassword"`.

What are the limitations of SLM-UI?

If you are using reverse proxy setup, you cannot use SLM-UI.

If you are using SPIRE based certificates, you cannot use SLM-UI.

Troubleshooting

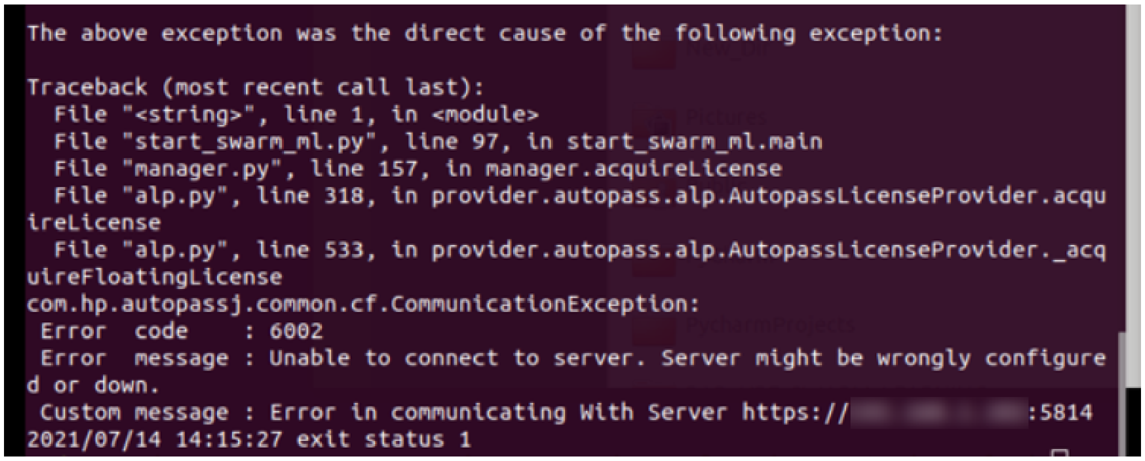
Troubleshooting provides solutions to commonly observed issues during Swarm Learning set up and execution.

Error code: 6002

```
> Error message: Unable to connect to server. Server might be wrongly configured or down.
> Custom message: Error in communicating with server https://HOST_SYSTEM_IP:5814 (default port)
```

Problem description

Error code: 6002, as shown in the following screenshot occurs when Swarm Learning components are not able to connect to the APLS server.



```
The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "start_swarm_ml.py", line 97, in start_swarm_ml.main
  File "manager.py", line 157, in manager.acquireLicense
  File "alp.py", line 318, in provider.autopass.alp.AutopassLicenseProvider.acqu
ireLicense
  File "alp.py", line 533, in provider.autopass.alp.AutopassLicenseProvider._acq
uireFloatingLicense
com.hp.autopassj.common.cf.CommunicationException:
 Error  code    : 6002
 Error  message : Unable to connect to server. Server might be wrongly configure
d or down.
 Custom message : Error in communicating With Server https://[redacted]:5814
2021/07/14 14:15:27 exit status 1
```

Resolution

1. Verify if License Server is running.

   On the License Server host, verify if it is running, if not, restart the License Server.
   For more information about restarting the License Server, see *AutoPass License Server User Guide*.

2. Access the APLS web management console. If the browser cannot connect, verify the network proxy settings, firewall policies, that are in effect. If required, work with your network administrator to resolve.

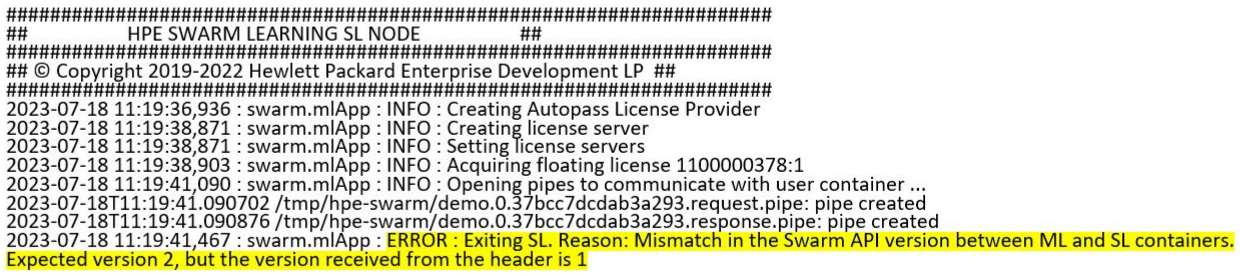
3. Verify if the Swarm licenses are installed using APLS web management console. For more information, see *HPE Swarm Learning Installation and Configuration Guide*.

Mismatch in the Swarm API version between ML and SL containers

```
Mismatch in the Swarm API version between ML and SL containers. Expected
version 2, but the version received from the header is 1.
```

Problem description

Error occurs as shown in the following screenshot when there is a mismatch in the Swarm API version between ML and SL containers.



```
###########################################################################
##                HPE SWARM LEARNING SL NODE                ##
###########################################################################
## © Copyright 2019-2022 Hewlett Packard Enterprise Development LP  ##
###########################################################################
2023-07-18 11:19:36,936 : swarm.mlApp : INFO : Creating Autopass License Provider
2023-07-18 11:19:38,871 : swarm.mlApp : INFO : Creating license server
2023-07-18 11:19:38,871 : swarm.mlApp : INFO : Setting license servers
2023-07-18 11:19:38,903 : swarm.mlApp : INFO : Acquiring floating license 1100000378:1
2023-07-18 11:19:41,090 : swarm.mlApp : INFO : Opening pipes to communicate with user container ...
2023-07-18T11:19:41.090702 /tmp/hpe-swarm/demo.0.37bcc7dcdab3a293.request.pipe: pipe created
2023-07-18T11:19:41.090876 /tmp/hpe-swarm/demo.0.37bcc7dcdab3a293.response.pipe: pipe created
2023-07-18 11:19:41,467 : swarm.mlApp : ERROR : Exiting SL. Reason: Mismatch in the Swarm API version between ML and SL containers.
Expected version 2, but the version received from the header is 1
```

SL and ML containers use Google's Protobuf protocol to verify the request and response parameters. Upon request from ML to SL container, SL checks the proto API version against the version in the ML container and vice versa. If it does not match, then this error message is displayed. When ML acts as a client and SL acts as a server (for example, when a callback session creates a request from ML to SL node) and there is a mismatch in the API version, this error is displayed in the SL container. When SL acts as a client and ML acts as a server (for example, when a global loss is transferred from SL to ML container) and there is a mismatch in API version, this error is displayed in the ML container.

Proto API version	Compatible releases
1	2.0.0 and lower versions
2	2.1.0 and higher versions

Resolution

1. If Swarm Learning images are latest, then verify if the Swarm Learning wheel file used to build the ML image is also latest or not. If not, then update to the latest Swarm Learning wheel file.
2. Generally, HPE recommends you to match the version of Swarm Learning images with the same version of Swarm Learning wheel file before starting the model training.

Unable to contact API-Server

```
Unable to contact API-Server
```

Resolution

A swarm container could be unable to reach an SN node for several reasons.

1. SN is not running. To confirm, check the Docker running state of the SN container.
2. SN node can be reached via SN container FQDN only in a single host custom bridge network. But for all other scenarios, IP address of the host machine must be used. Ensure the correctness of `--sn-ip` and `--sn-api-port` parameters.
3. Ensure that SN-API-port is allowed in your firewall settings. User can check this by running `sudo ufw status`. If the SN-API-port is not in the list, then add it by using `sudo ufw allow <SN-API-port>`. The same configuration is applicable for all other Swarm ports. Ignore this step if the `ufw status` is inactive, as this state allows all ports.
4. If the certificates get expired, then the other swarm components including non-sentinel SN are not able to reach SN. User can check the expiry date of their certificates and update them accordingly.
5. When network proxy is not set correctly, both `http_proxy` and `https_proxy` ENV variables need to be set as per the customers network policy.

API server is down

```
API server is down
```

Problem description

If sentinel SN dies, SLM-UI stops working.

Resolution

If sentinel SN crashes, it is better to delete SWCI container that is implicitly started (named as `slm-ui-<>-swci` and recreate SN to ensure SLM-UI continues to work. SLM-UI automatically spawns a new SWCI to work with the new SN.

Failed to start thread "GC Thread#0" - pthread_create failed

```
Failed to start thread "GC Thread#0" - pthread_create failed
```

Problem description

This error is displayed in the container logs when the Swarm containers are not able to start.

Resolution

User needs to upgrade the Docker version to the latest version.

sudo groupadd docker

```
sudo groupadd docker
```

Problem description

This error is displayed if the user is not part of `docker` group where the swarm learning is getting installed.

Resolution

User needs to run the following commands:

```
sudo groupadd docker

sudo usermod -aG docker [USER]
```

Emulate Docker CLI using podman

```
Emulate Docker CLI using podman
```

Problem description

UI container runs on port 80 which is privileged port and not allowed by default.

Resolution

User needs to choose different port during SLM-UI installation.

Rootlessport cannot expose privileged port 80

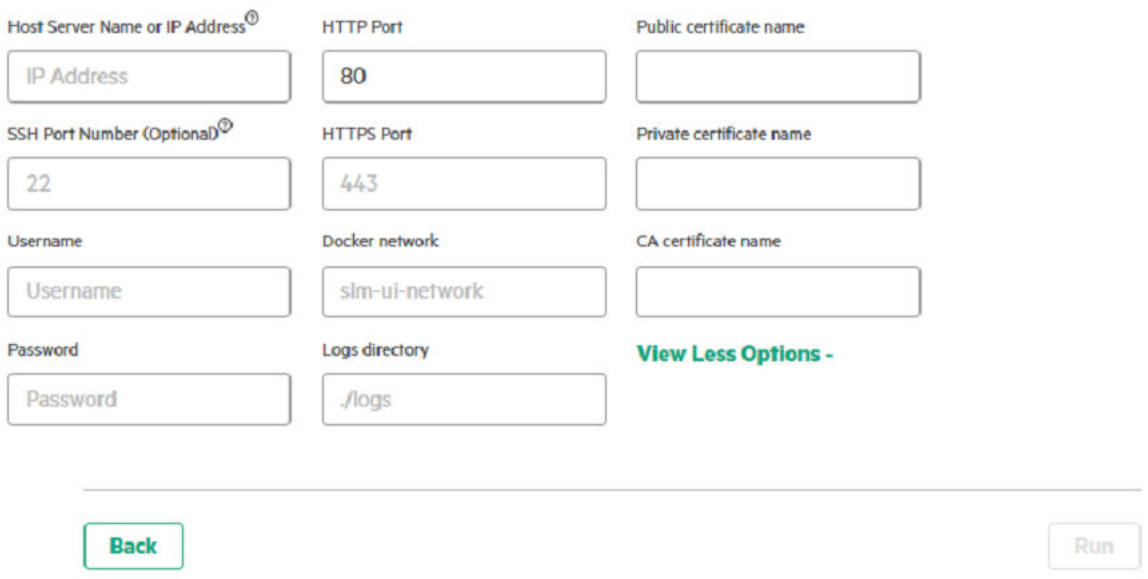
```
rootlessport cannot expose privileged port 80
```

Problem description

If the default **HTTP Port** `80` or **HTTPS Port** `443` is used, SLM-UI installation fails with this error in the Podman environment.

Resolution

While installing SLM-UI, user needs to provide different port numbers which are greater than 1024, for both **HTTP Port** and **HTTPS Port** as follows.



Host Server Name or IP Address	HTTP Port	Public certificate name
IP Address	80	
SSH Port Number (Optional)	HTTPS Port	Private certificate name
22	443	
Username	Docker network	CA certificate name
Username	slm-ui-network	
Password	Logs directory	View Less Options -
Password	./logs	

Swarm Learning log collector

The Swarm Learning log collector script is used to collect log and basic system information and create a tar archive file, which can be sent to HPE for troubleshooting any Swarm Learning related issues.

NOTE: This script collects the logs only from the current host. If the user runs Swarm Learning on multiple hosts, then the user must run the script on each host machine.

The tar archive file contains the following details:

```
- OS details
- nvidia details [if user running examples with GPU]
- Running and exited docker information
- docker logs and docker inspect of all artifacts [SN SWOP SL ML]
```

Syntax

```
./swarmLogCollector [OPTIONS]
```

Run the following command if you are using SWOP:

```
./swarmLogCollector "<DOCKER_HUB>" "workspace=<Swarm Installation DIR/swarm-learning workspace/exampleFolder>"
```

For example,

Run the following command if you are running the example from CLI:

```
./swarmLogCollector "hub.myenterpriselicense.hpe.com/hpe/swarmlearning"
"workspace=<Swarm Installation DIR>/workspace/fraud-detection/"
```

Run the following command if you are running the example from SLM-UI:

```
./swarmLogCollector "hub.myenterpriselicense.hpe.com/hpe/swarmlearning"
"workspace=<Swarm Installation DIR>/slm-ui/projects/1/"
```

Run the following command if you are using `run-sl` script:

```
./swarmLogCollector "<DOCKER_HUB>" "mlimage=<ml image name>"
```

For example,

```
./swarmLogCollector "hub.myenterpriselicense.hpe.com/hpe/swarm-learning" "mlimage=user-env-tf2.7.0-swop"
```

The following lists are the additional information that users can manually provide:

1. Issue description

   ```
   What is the Issue:
   Occurrence - consistent or rare:
   Commands used for starting containers:
   Details of ML platform used:
   ```

2. Quick Checklist: Respond [**`Yes`**/**`No`**]

   ```
   APLS server web GUI shows available Licenses?
   If Multiple systems are used, can each system access every other system?
   Is Password-less SSH configuration setup for all the systems?
   ```

3. Additional notes

   ```
   Are you running documented example without any modification?
   Add any additional information about use case or any notes which supports
   for issue investigation:
   ```

4. Use GPU diagnostics messages in the application code if you are running the examples with GPU. Like `cuda_available`, any python packages help to print the GPU statistics.
5. Provide Firewall related information by running the respective commands. For example, the following command provides the Firewall details in Ubuntu OS.

   ```
   sudo ufw status
   ```

6. Run the example with `SWARM_LOOPBACK` set to `True`. This is to confirm that the user application has no issues. For more information on confirming that the user application has no issues, see **FAQ**.

GNU General Public License

Support and other resources

Accessing Hewlett Packard Enterprise Support

- For live assistance, go to the Contact Hewlett Packard Enterprise Worldwide website:

  https://www.hpe.com/info/assistance

- To access documentation and support services, go to the Hewlett Packard Enterprise Support Center website:

  https://www.hpe.com/support/hpesc

Information to collect

- Technical support registration number (if applicable)
- Product name, model or version, and serial number
- Operating system name and version
- Firmware version
- Error messages
- Product-specific reports and logs
- Add-on products or components
- Third-party products or components

Accessing updates

- Some software products provide a mechanism for accessing software updates through the product interface. Review your product documentation to identify the recommended software update method.
- To download product updates:

  Hewlett Packard Enterprise Support Center

  https://www.hpe.com/support/hpesc

  My HPE Software Center

  https://www.hpe.com/software/hpesoftwarecenter

- To subscribe to eNewsletters and alerts:

  https://www.hpe.com/support/e-updates

- To view and update your entitlements, and to link your contracts and warranties with your profile, go to the Hewlett Packard Enterprise Support Center **More Information on Access to Support Materials** page:

  https://www.hpe.com/support/AccessToSupportMaterials

IMPORTANT: Access to some updates might require product entitlement when accessed through the Hewlett Packard Enterprise Support Center. You must have an HPE Account set up with relevant entitlements.

Remote support

Remote support is available with supported devices as part of your warranty or contractual support agreement. It provides intelligent event diagnosis, and automatic, secure submission of hardware event notifications to Hewlett Packard Enterprise, which initiates a fast and accurate resolution based on the service level of your product. Hewlett Packard Enterprise strongly recommends that you register your device for remote support.

If your product includes additional remote support details, use search to locate that information.

HPE Get Connected

https://www.hpe.com/services/getconnected

HPE Tech Care Service

https://www.hpe.com/services/techcare

HPE Complete Care

https://www.hpe.com/services/completecare

Warranty information

To view the warranty information for your product, see the **warranty check tool**.

Regulatory information

To view the regulatory information for your product, view the *Safety and Compliance Information for Server, Storage, Power, Networking, and Rack Products*, available at the Hewlett Packard Enterprise Support Center:

https://www.hpe.com/support/Safety-Compliance-EnterpriseProducts

Additional regulatory information

Hewlett Packard Enterprise is committed to providing our customers with information about the chemical substances in our products as needed to comply with legal requirements such as REACH (Regulation EC No 1907/2006 of the European Parliament and the Council). A chemical information report for this product can be found at:

https://www.hpe.com/info/reach

For Hewlett Packard Enterprise product environmental and safety information and compliance data, including RoHS and REACH, see:

https://www.hpe.com/info/ecodata

For Hewlett Packard Enterprise environmental information, including company programs, product recycling, and energy efficiency, see:

https://www.hpe.com/info/environment

Documentation feedback

Hewlett Packard Enterprise is committed to providing documentation that meets your needs. To help us improve the documentation, use the **Feedback** button and icons (at the bottom of an opened document) on the Hewlett Packard Enterprise Support Center portal (**https://www.hpe.com/support/hpesc**) to send any errors, suggestions, or comments. This process captures all document information.