Nauta Installation and Configuration

Administration Guide, Enterprise Edition 1.1

Document Revision 1.1: October 2019

Document Revision History

Document Revision Number	Date	Comments
1.1	October 2019	Initial Release of Enterprise Edition, 1.1 Release

Terms and Conditions

Contents

Figures

Tables

Nauta Overview

This Nauta Installation, Configuration, and Administration guide provides step-by-step instructions for installing and configuring Nauta. This guide also provides an overview of Nauta requirements, such as configuration management and operating system requirements. This guide is divided into three main sections: Preinstallation Information, Installation Tasks, and User and Account Management.

Note: For instructions on configuring the Nauta client, refer to the *Nauta User Guide*.

Nauta is a software suite that provides a multi-user, distributed computing environment for running deep learning model training experiments. Results of experiments can be viewed and monitored using a Command Line Interface (CLI), Web UI and/or TensorBoard. You can use existing data sets, your own data, or downloaded data from online sources, as well as create public or private folders to make collaboration among teams easier.

Nauta runs on Kubernetes and Docker for scalability and ease of management. Nauta uses customizable templates to remove the complexities of creating and running single and multi-node deep learning training experiments without all the system's complexity and scripting needed with standard container environments.

Purpose of this Guide

This guide describes how to install Nauta and discusses the following topics main topics:

01

Preinstallation Information

Nauta System Software Components Requisite

Nauta relies on several external software components, listed in the table below. These components are included in Nauta, and are installed automatically as part of the Nauta installation process. As shown in Table 1, these are the packages included in the install package.

List of External Software Components

Table 1 shows Nauta's external software components and their versions.

Table 1: Nauta External Software Components

Name	Version	Project link
addon-resizer	1.13.6	https://github.com/kubernetes/autoscaler/tree/master/addon-resizer
dashboard	1.8.3	https://github.com/kubernetes/dashboard
defaultbackend	1.4	https://github.com/kubernetes/ingress-nginx
dnsmasq-nanny	1.14.8	https://github.com/kubernetes/dns
dns-sidecar	1.14.8	https://github.com/kubernetes/dns
etcd	3.3.13	https://github.com/coreos/etcd
elasticsearch	7.3.0	https://github.com/elastic/elasticsearch
flannel	0.11.0	https://github.com/coreos/flannel
fluentd	1.2.5	https://www.fluentd.org
helm	2.11.0	https://github.com/helm/helm
heapster	1.4.3	https://github.com/kubernetes/heapster
ingress	0.24.0	http://quay.io/kubernetes-ingress-controller/nginx-ingress-controller
kubectl	1.15.3	https://github.com/kubernetes/kubernetes/tree/master/pkg/kubectl
kube-dns	1.14.12	https://github.com/kubernetes/dns
kube-proxy	1.15.3	http://gcr.io/google-containers/kube-proxy-amd64
mkl-dnn	0.18	https://github.com/intel/mkl-dnn
nginx	1.14.0	https://www.nginx.com
openvino	2019_R1.1	https://github.com/opencv/dldt
pause	3.1	http://gcr.io/google-containers/pause-amd64
pytorch	1.2.0	https://github.com/pytorch/pytorch
redsocks	0.5	https://github.com/darkk/redsocks
registry	2.7	https://github.com/docker/distribution
tensorflow	1.14.0	https://github.com/tensorflow/tensorflow

Key Components

The Nauta complete bundle (including installation scripts) contains the following:

1. A package that installs a vanilla Kubernetes cluster, including necessary OS-level software components, on provisioned Hardware.
2. Nauta components (containerized components, their configuration, integration method, and so on) on the above-mentioned Kubernetes cluster.

The Nauta software components are optimized for AI containers with Nauta-optimized libraries.

Nauta Installer System Requirements

A separate system (installer system) is required for installing Nauta. This section shows the requirements for that system. For more information, refer to the Installation Process, page 29.

Nauta Supported Operating Systems

Nauta supports the following Operating Systems:

- Red Hat Enterprise Linux 7.6 or CentOS 7.6
- Ubuntu16.04

Red Hat Enterprise Linux 7.6

Required on system, software requirements:

- Python 2.7 and `/usr/bin/python` available
- Python 3.5
- sshpass (when password authentication is used)
- Helm 2.9.1 (the version of a Helm client *must be* the same as Helm server used by the platform)

CentOS 7.6

Required on system, software requirements:

- Python 2.7 and `/usr/bin/python` available
- Python 3.5
- sshpass (when password authentication is used)
- Helm 2.9.1 (the version of a Helm client *must be* the same as Helm server used by the platform)

Ubuntu 16.04

Required on system, software requirements:

Python 2.7 and `/usr/bin/python` available

- Python 3.5
- apt required packages:
  - python3-pip
  - build-essential
  - libffi-dev
  - libssl-dev
  - sshpass
- Helm 2.9.1 (version of a Helm client *must be* the same as Helm server used by the platform)

Target Host Requirements

For the *Target host*, install Nauta **on bare metal only** with Red Hat Enterprise Linux 7.6 (this can be preinstalled).

- Configured access to master host over SSH.
    - This is configured access from your *Installer Machine* to your *Target Host* (master).
- Full network connectivity between target hosts is required. In addition, Installer connectivity is only required to the master node.

Red Hat Enterprise Linux 7.6

Red Hat Enterprise Linux 7.6 is required, as well as the following required packages:

- byacc
- cifs-utils
- ebtables
- ethtool
- gcc
- gcc-c++
- git
- iproute
- iptables >= 1.4.21
- libcgroup
- libcgroup-devel
- libcgroup-tools
- libffi-devel
- libseccomp-devel
- libtool-ltdl-devel
- make
- nfs-utils
- openssh
- openssh-clients
- openssl
- openssl-devel
- policycoreutils-python
- python
- python-backports
- python-backports-ssl_match_hostname
- python-devel

- python-ipaddress
- python-setuptools
- rsync
- selinux-policy >= 3.13.1-23
- selinux-policy-base >= 3.13.1-102
- selinux-policy-targeted >= 3.13.1-102
- socat
- systemd-libs
- util-linux
- vim
- wget

Valid Repositories

If the operating system is installed and configured with a valid repository that contains the required packages, an Administrator *does not* need to install the packages manually. However, if the repository *is not* valid, the Installer attempts to install the package automatically. If this fails, an error message is displayed.

Repositories List

Use the following command to check your repository list: `yum repolist all`

A list of **required** enabled repositories for RHEL 7.6, is:

- Extra Packages for Enterprise Linux 7 - x86_64
- Red Hat Enterprise Linux 7 Server - x86_64
- Red Hat Enterprise Linux 7 Server (High Availability) - x86_64
- Red Hat Enterprise Linux 7 Server (Optional) - x86_64
- Red Hat Enterprise Linux 7 Server (Supplementary) - x86_64

A list of **required** enabled repositories for Centos 7.6, is:

- CentOS-7 - Base
- CentOS-7 - Extras
- CentOS-7 - Updates
- Extra Packages for Enterprise (epel)

Inventory File Tasks

Nauta uses Ansible (refer to Ansible Overview) for certain provisioning tasks during installation.
You *must* create (or modify) an Ansible inventory file to match your hardware configuration. Nauta will look for your inventory file at the location defined in the `ENV_INVENTORY` environment variable (refer to the Installation Process, page 29 for more information).

Your Nauta cluster will contain one Master node and one or more Worker nodes. Each of these nodes *must be* specified in the Inventory file. For Configuration file information Configuration File Tasks, page 16 for more information.

Inventory Configuration File Example

Below is an example of Inventory file and shows one Master Node and five Worker nodes. Your configuration may differ from the example shown. However, you can copy and modify the information to create your own Ansible inventory file.

Note: Ansible uses the YAML format. Refer to YAML Format Overview for more information.

[master]

master-0 ansible_ssh_host=**192.168.100** ansible_ssh_user=root ansible_ssh_pass=YourPassword internal_interface=em2 data_interface=em2 external_interface=em3 local_data_device=/dev/sdb1

[worker]

worker-0 ansible_ssh_host= **192.168.100.61** ansible_ssh_user=root ansible_ssh_pass=YourPassword internal_interface=p3p1 data_interface=p3p1 external_interface=em1

worker-1 ansible_ssh_host= **192.168.100.55** ansible_ssh_user=root ansible_ssh_pass=YourPassword internal_interface=p3p1 data_interface=p3p1 external_interface=em1

worker-3 ansible_ssh_host= **192.168.100.106** ansible_ssh_user=root ansible_ssh_ pass=YourPassword internal_interface=p3p1 data_interface=p3p1 external_interface=em1

worker-4 ansible_ssh_host= **192.168.100.107** ansible_ssh_user=root ansible_ssh_ pass=YourPassword internal_interface=p3p1 data_interface=p3p1 external_interface=em1

Inventory File Structure

The file contains two sections, master and worker:

- **[master]** Contains a description of a master node. This section *must* contain **exactly one row**.
- **[worker]** Contains descriptions of workers. Each worker is described in one row. In this section, it can have one or many rows depending on a structure of a cluster.

Each row describes a server (playing either the role of Master or Worker depending on which section the row is in). For each server, the inventory file must define a series of values that tells Nauta where to find the server, how to log into it, and so on.

The format for each row is as follows:

```
SERVER_NAME] [VAR_NAME1]=[VAR_VALUE1] [VAR_NAME2]=[VAR_VALUE2]
[VAR_NAME3]=[VAR_VALUE3]...
```

Standard Hosting Name Rules

The `SERVER_NAME` *must* conform to standard host naming rules and each element of the hostname *must be* from 1 to 63 characters long. The entire hostname, including the dots *must not* exceed 253 characters long.

Valid characters for hostnames are ASCII(7) letters from a to z (lowercase), the digits from 0 to 9, and a hyphen. However, ***do not*** start a hostname with a **hyphen**.

Per-node Inventory Variable

Table 2 lists all the variables understood by Nauta's inventory system. Some variables are required for all servers in the inventory and some variables are optional.

Table 2: Inventory File Structure Variables

Variable Name	Description	Req?	Type	Default	Used When	Value
`ansible_ssh_user`	The user name *must be* the same for master and worker nodes. **Note:** If an Administrator decides to choose something other than root for Ansible SSH user, the user *must be*configured in sudoers file with NOPASSWD option. Refer to the official Ansible Inventory Documentation for more information.	Yes	string	none	always	username

Variable Name	Description	Req?	Type	Default	Used When	Value
`ansible_ssh_pass`	The SSH password to use.	Yes	string	none	always	Password
`ansible_ssh_host`	The name (DNS name) or IP Address of the host to connect to.	Yes	IPaddr	none	always	IP Address
`ansible_ssh_port`	The SSH port number, if not defined 22 is used.	No	int	22	not using 22	Port Address
`ansible_ssh_private_key_file`	This is a Private Key file used by SSH.	No	string	none	using a keyfile	filename
`internal_interface`	This is used for internal communication between Kubernetes processes and pods. All interfaces (both external and internal) are Ethernet interfaces.	Yes	string	none	always used for both for master and worker nodes	Interface name
`local_data_device`	This device is used for Nauta internal data and NFS data in case of local NFS.	Yes	string	none	used with master nodes	Path to block device
`local_device`	This device is used for Nauta internal data and NFS data in case of local NFS.	Yes	string	none	used with master nodes	Path to block device
`local_data_path`	This is used as the mountpoint for `local_data_device`	No	string	none	used with master nodes	Absolute path where data is located in file system
`external_interface`	This is used for external network communication.	Yes	string	none	always used for both for master and worker nodes	Interface name

Configuration File Tasks

Nauta's configuration is specified by a *YAML Configuration file*. Nauta will look for this file at the location defined in the `ENV_CONFIG` environment variable (explained in Installation Process, page 29), as well as shown example configuration file is also shown below. This configuration file specifies network proxies, DNS server locations, and other Nauta related parameters listed below. For Inventory file information, refer to Inventory File Tasks, page 13 for more information.

Parameter Color Indicators

In the examples shown, **Bold Green** indicates parameter name and ***Bold Italic Blue*** indicates exemplary parameter value.

Configuration Variables Indicators

Some configuration variables are of the *dictionary* type, and for these: `{}` indicates an empty dictionary. Likewise, some variables are of the *list* type, and for these: `[]` indicates an empty list.

Configuration Considerations

Host Name Considerations

Host names *must* conform to standard host naming rules and each element of the hostname *must* be from 1 to 63 characters long. The entire host name, including the dots *must not* exceed 253 characters long. Valid characters for host and domain names are ASCII (7) letters from a to z (lowercase), the digits from 0 to 9, and a hyphen. Furthermore, ***do not*** start a host and domain names with a ***hyphen***.

Proxy Value Settings

All parameters present in the configuration file must have values. Setting the configuration file with no value causes errors, for example, `proxy:`.

Example Configuration File

This is an *example* file, containing dummy values for a few of the supported configuration values. For a complete list, refer to the section after the YAML file example below. For YAML file information, refer to YAML Format Overview.

1. In the YAML file, the *list of items* is a sequence of elements starting on a new line with a dash at the beginning. For example, an empty list: [ ]. In an abbreviated form, elements can also be enclosed on a single line.
2. In the YAML file, a dictionary is a sequence of pairs for the element's *key: value*. It can also be presented with each pair on a new line or abbreviated on a single line.

YAML (Configuration) File Example

Below is an example YAML file that provides examples of proxy settings, DNS server settings, Kubernetes, and so on. For detailed instructions on how to complete this, see the example Inventory File Tasks, page 13 and Configuration File Tasks, page 16 for more information.

```
# Proxy Settings
proxy:
  http_proxy: http://<your proxy address and port>
  ftp_proxy: http://<your proxy address and port>
  https_proxy: http://<your proxy address and port>
  no_proxy: <localhost, any other addresses>, 10.0.0.1,localhost,.nauta
# This is a list of DNS servers used for resolution: a maximum of three entries.
dns_servers:
  - 8.8.8.8
  - 8.8.4.4
# This is a domain used as part of a domain search list.
dns_search_domains:
  - example.com
# This is place to define additional domain names for cluster to allow secure
communication.
dns_names_for_certificate:
  DNS.7: "mycluster1.domain.name"
  DNS.8: "mycluster2.domain.name"
# This is the Internal domain name.
domain: nauta
# Internal subdomain for infrastructure
nodes_domain: lab007
# This is the Internal Subdomain for Kubernetes resources.
k8s_domain: kubernetes
# This is the Network Range for Kubernetes pods.
kubernetes_pod_subnet: 10.3.0.0/16
# This is the Network Range for Kubernetes services.
kubernetes_svc_subnet: 10.4.0.0/16
```

Default Value of an Empty Dictionary

For empty dictionaries, there are two defaults:

1. If a parameter *is not* included in configuration file, the default value is utilized.
2. If a parameter is present in configuration file with a default value included, it appears as shown below.
   - `proxy: {}`

Deciding to Leave the Proxy Parameter Empty

There may be reasons to leave the proxy parameter set with an empty dictionary. Should you decide to do this, it may be for the following reasons:

- You *do not* need a proxy because you *do not* have one in your network.
- You intentionally *do not* want to use a proxy to connect your cluster as an external network to keep it isolated from the Internet.

Configuration Variable Details

proxy

- **Description:** These are the Proxy settings for internal applications.
- **Default value:** {}

```
proxy:
  http_proxy: http://<your proxy address and port>
  ftp_proxy: http://<your proxy address and port>
  https_proxy: http://<your proxy address and port>
  no_proxy: <localhost, any other addresses>, 10.0.0.1,localhost,.nauta
  HTTP_PROXY: http://<your proxy address and port>
  FTP_PROXY: http://<your proxy address and port>
  HTTPS_PROXY: http://<your proxy address and port>
  NO_PROXY: .<localhost, any other addresses>, 10.0.0.1,localhost,.nauta
```

Note: Proxy addresses should be replaced by a specific value by a client.

dns_servers

Description: This is a list of DNS servers used for resolution: a maximum of three entries.

Default value: []

```
dns_servers:
  - 8.8.8.8
  - 8.8.4.4
```

dns_names_for_certificate

Description: This is a list of DNS names of the cluster. The values from this parameter use acceptable addresses when accessing a cluster in securely. Key values have to be in the format `DNS.X` where `X` is number greater than 6 (internally allocated symbolic addresses).

Default value: []

dns_search_domains

Description: This is a domain used as part of a domain search list.

Default value: []

```
dns_search_domains:
  - example.com
```

domain

Description: This is the *Internal* domain name. This variable and *nodes_domain* (defined below) together form the *full domain* (`<nodes_domain>.<domain>`) for Nauta's internal domain. For example, if `domain` is *nauta* and `sub_domain` is *infra*, the full domain is `infra.nauta`.

Default value: nauta

```
domain: nauta
```

nodes_domain

Description: Internal subdomain for infrastructure. Full domain for an infrastructure is:

Default value: infra

```
nodes_domain: lab007
```

Note: These IP addresses *should not* conflict with Internal IP address ranges.

k8s_domain

Description: This is the internal subdomain for Kubernetes resources. Full domain for infrastructure is:

```
<k8s_domain>.<domain>
```

Default value: kubernetes

```
k8s_domain: kubernetes
```

kubernetes_pod_subnet

Description: This is the Network Range for Kubernetes pods.

Default value: 10.3.0.0/16

```
kubernetes_pod_subnet:10.3.0.0/16
```

kubernetes_svc_subnet

Description: This is the Network Range for Kubernetes services.

Default value: 10.4.0.0/16

```
kubernetes_svc_subnet: 10.4.0.0/16
```

apiserver_audit_log_maxage

Description: Maximum age in days for Kubernetes apiserver audit logs.

Default value: 7

```
apiserver_audit_log_maxage: 7
```

apiserver_audit_log_maxbackup

Description: Maximum number of log files kept for Kubernetes apiserver audit.

Default value: 10

```
apiserver_audit_log_maxbackup: 10
```

apiserver_audit_log_maxsize

Description: Maximum audit log file size in MB.

Default value: 1024

```
apiserver_audit_log_maxsize: 1024
```

insecure_registries

Description: This is a list of insecure Docker registries added to configuration.

Default value: []

Note: This refers to Docker registries only.

```
insecure_registries:
- my.company.registry:9876
```

enabled_plugins

Description: This is a list of enabled Yum plugins.

Default value: []

```
enabled_plugins:
- presto
```

disabled_plugins

Description: This is a list of disabled Yum plugins.

Default value: []

```
disabled_plugins:
- presto
```

use_system_enabled_plugins

Description: This defines if Yum should use system-enabled plugins.

Default value: False

```
use_system_enabled_plugins:
False
```

enabled_repos

Description: This is a list of enabled repositories, and is used for external dependencies installation.

Default value: []

```
enabled_repos:
- rhel
```

disabled_repos

Description: This is a list of disabled repositories, and is used for external dependencies installation.

Default value: []

```
disabled_repos:
- rhel
```

use_system_enabled_repos

Description: This defines if the default system repositories should be enabled in external dependencies installation.

Default value: True

```
use_system_enabled_repos:
True
```

input_nfs

Description: Definition of *input* NFS mounts for Samba. By default, internal NFS provisioner is used.

Default value: {}

Fields

- **path:** NFS path to mount
- **server:** NFS server

output_nfs

Description: This is the definition of *output* NFS mounts for Samba. By default, internal NFS provisioner is used.

Default value: {}

Fields

- **path:** NFS path to mount
- **server:** NFS server
- **Note** For NFS information, see below.

```
nauta_configuration:
  input_nfs:
    path: "{{ nfs_base_path }}/input"
    server: "{{ nfs_server }}"
  output_nfs:
    path: "{{ nfs_base_path }}/output"
    server: "{{ nfs_server }}"
```

Docker Log Driver Settings

This is the Docker log driver settings for controlling rotation of a containers' logs on *bare metal environments*. In case of cloud deployments, such as Google Cloud Platform*, instead of changing this parameter, refer to your cloud provider instructions for log rotation configuration.

- Refer to Docker Log Driver Settings for more information.
- **Default value:**

```
docker_log_driver_settings:
  max_size: 5g
```

```
max_file: 1
```

- **Fields**
  - **max_size:** Maximum size of log file
  - **max_file:** Maximum count of present log files

Network File System Overview

The Network File System (NFS) allows a system to share directories and files with others over a network. The advantage of using NFS is that end-users, as well as programs can access files on remote systems in the same way as local files. In addition, NFS uses less disk space, as it can store data on a single machine while remaining accessible to others over a network.

Optional Features: Redsocks and NFS Installation

Either NFS or Redsocks* is installed and configured during the installation process (see example below). By default, Redsocks *is not* installed; however, NFS ***is*** installed by default.

Example NFS Configuration Settings

Either NFS or Redsocks is installed and configured during the installation process. By default, Redsocks *is not* installed; however, NFS is installed by default.

```
features:
  nfs: True
  redsocks: True
```

Features List (NFS Default Settings)

- **NFS:** default: True
- **Redsocks:** default: False

Redsocks Overview

Redsocks is the tool that allows you to redirect network traffic through a Socket (for example: SOCKS4, SOCKS5 or HTTPs proxy server). Redsocks operates through a proxy server, and as a result it is refered to as a transparent proxy redirector.

- Refer to How to transparently use a proxy with any application (Docker) using Iptables and Redsocks for more information.

Redsocks Configuration

Redsocks configuration is an **optional** part of the installer; ; however, if you choose this option then configure Redsocks appropriately in your environment/organization.

Redsocks *is not* enabled during the installation, as the default is set to *False* (shown in the example below). Therefore, if you decide to install Redsocks you ***must*** set the feature switch to **True**.

WARNING: After the Nauta installation completes, should you decide to install Redsocks, you will need to redo the entire installation to include Redsocks and set the feature switch to True. It ***cannot*** be changed to False in your configuration file after the initial install. Redsocks and NFS are independent of each other, so use judgment when initially setting these feature switches.

How to Enable Features

Additional features can be enabled using features in the configuration, as shown below.

```
features:
  redsocks: False
```

Feature Plugin Configuration

```
features:
  redsocks: True
features_config:
  redsocks:
    IP: 10.217.247.236
    Port: 1080
```

Redsocks Configuration Parameters

IP

Description: This is the IP address of Socks5 proxy.

```
Required: True
```

Port

Description: This is the port address of Socks5 proxy.

```
Required: True
```

Interfaces

Description: Comma-separated list of interfaces from which traffic should be managed by RedSocks.

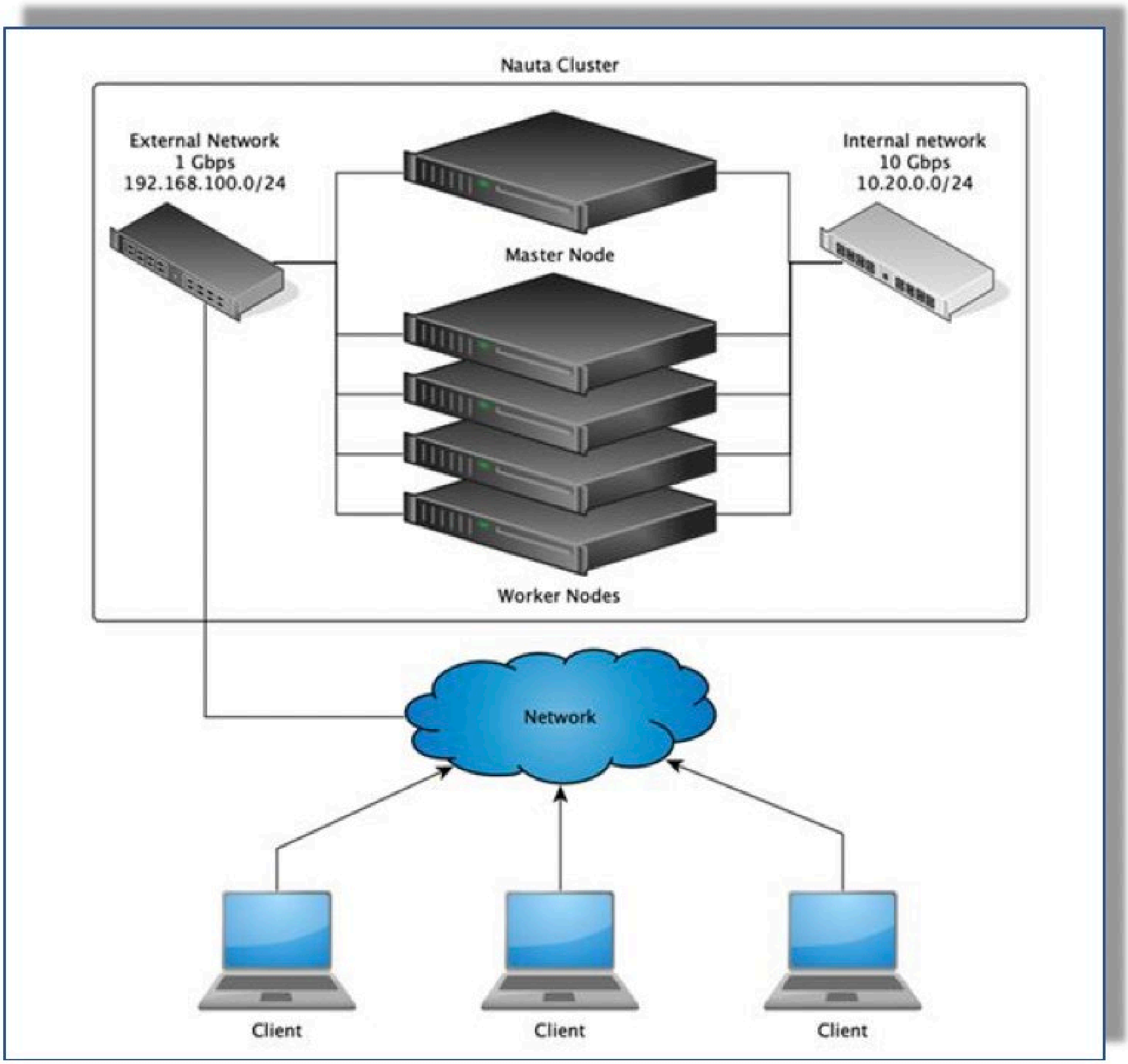
Description: This is the IP address of Socks5 proxy.

```
Required: False
Default: cni0
```

Networking Configuration Example

The figure below shows an example Nauta Networking Diagram. While you can build a cluster with 1 machine to run all the examples it is suggested to utilize at least 4 worker nodes (as shown in the example). The worker nodes should run Red Hat Enterprise Linux 7.6. All interfaces (both external and internal) are Ethernet interfaces.

Figure 1: Nauta Networking Diagram Example



02 Installation Tasks

Installation Requirements

Nauta Package: Extraction from Local Package

Copy the package to the installer machine, then untar it using the following command.

```
nauta-1.1.0-ent-20191010050128.tar.tar.gz -C <destination>
```

Nauta Structure

In the extracted archive, the following appears:

- **Files:**
  - **installer.sh:** sh script
  - **ansible.cfg:** configuration file for Ansible
- **Folders:**
  - **bin:** binary directory
  - **docs:** documentation
  - **libs:** contains various scripts that are used by the installer
  - **nauta:** Nauta Enterprise applications deployer
  - **platform:** Kubernetes platform deployer

Components Version

To see the list of installed components and their versions, refer to: List of External Software Components, page 8.

Installation Process

Before proceeding with this step, you *must* create an Inventory and Configuration file. If you are unsure how to do this, see Inventory Configuration File Example, page 13 and Example Configuration File page 16.

Creating an Inventory or Configuration File

The *Inventory* file defines where your master and worker nodes reside, and how to access them. The *Configuration* file defines your proxy settings, network quirks and filesystem preferences. Should you need to create an *Inventory* and/or *Configuration* follow the instructions are below.

Inventory File Creation

To create Nauta's *Inventory file*, do the following:

1. Copy the Inventory file example information: Inventory Configuration File Example, page 13.
2. Modify the newly created Inventory file to suit your needs using your favorite text editor (vi for example).

Configuration File Creation

To modify Nauta's Configuration file to define your proxy, network quirks, and file system preferences. To create the configuration file, follow these steps:

1. Copy the Configuration file example information: Example Configuration File page 16.
2. Modify the newly created Configuration file to suit your needs using your favorite text editor (vi for example).

Kernel Upgrade

If you run Linux kernel prior to 4.x version it is recommended that you upgrade it on all nodes of a cluster before performing a platform installation.

Running heavy training jobs on workers with the operating system kernel older than 4.x may lead to hanging the worker node.

- See Red Hat Bugzilla – Bug 1507149 for more information.

This may occur when a memory limit for a training job is set to a value close to the maximum amount of memory installed on this node(s). These problems are caused by errors in handling memory limits in older versions of the kernel.

The following kernel was verified as a viable fix for this issue (see link below).

- Index of /linux/kernel/el7/x86_64/RPMS

To install the new kernel refer to: CHAPTER 5. MANUALLY UPGRADING THE KERNEL in Red Hat's Kernel Administration Guide.

Note: The above kernel *does not* include Red Hat's optimizations and hardware drivers.

Nauta Installation Procedure

To install Nauta, follow these steps:

1. Set the following environment variables that point to the configuration and inventory file on the *Installer Machine*.

   ENV_INVENTORY (mandatory): Inventory file location, for example:

   ```
   export ENV_INVENTORY=$<absolute path inventory file>
   ```

   ENV_CONFIG (mandatory): Configuration file location, for example:

   ```
   export ENV_CONFIG=$<absolute path config file>
   ```

2. Run the installation:

   ```
   ./installer.sh install
   ```

Installation Script Options

Invoke `./installer.sh` with one of the following options:

- **install:** Use this script to install Kubernetes and Nauta as part of your installation.
- **platform-install:** Use this script to install **Kubernetes only**.
- **nauta-install:** Use this script to install **Nauta only**.
  - **Note:** If you select this option, it is *assumed* that Kubernetes is already installed. In addition, this requires the same procedure for Nauta upgrades (see below).
- **nauta-upgrade:** Nauta installation upgrade (refer to the Installation Process, page 29 for Nauta upgrade procedures).

Installation Output

Nauta will be installed on cluster nodes in the following folders: `/opt/nauta`, `/usr/bin`, `/etc/nauta-cluster`.

Access Files for the kubectl Client

On *Installer Machine*, the following files are created in the Installation folder. These *files* are access files used to connect to the cluster using `kubectl` client.

As an output of the Nauta installation, As an output of Kubernetes installation, `platform-admin.config` file is created and the installation log is written into the log folder in the main installation directory:

```
platform-admin.config - cluster admin config file
```

As an output of the Nauta installation, As an output of Kubernetes installation, `platform-admin.config` file is created and the installation log is written into the log folder in the main installation directory:

```
nauta-admin.config - NAUTA admin config file
```

Upgrading Nauta

As an administrator, you may be required to upgrade Nauta to gain new features, implement new networking configurations, or stay up-to-date with current versions, and so on.

Upgrading Nauta Procedure

To upgrade Nauta, do the following:

1. Set the following environment variables that point to the configuration, inventory and configuration file on the *Installer Machine*:

   ENV_INVENTORY (mandatory): Inventory file location, for example:

   ```
   export ENV_INVENTORY=<absolute path to inventory file>
   ```

   ENV_CONFIG (mandatory): Configuration file location, for example:

   ```
   export ENV_CONFIG=<absolute path to config file>
   ```

   NAUTA_KUBECONFIG (mandatory): Nauta admin file location, for example:

   ```
   export ENV_INVENTORY=<absolute path to inventory file>
   ```

2. Call the installer with nauta-upgrade option:

   ```
   ./installer.sh nauta-upgrade
   ```

   Note: It is recommended that you *do not use* the cluster during an upgrade.

3. After successful execution of platform upgrade you need to also upgrade all users by running:

   ```
   nctl user upgrade
   ```

   Note: This command can be run only by an administrator.

This completes the Nauta Upgrade Process.

03

User and Account Management

User and Account Management

As an Administrator there are certain tasks you need to perform, such as creating and deleting user accounts, as well as understanding the limitations around these tasks.

User Account Creation

When you create a new user account it creates a *user account configuration file* with *kubectl configuration* files.

Setting up an Administrator Environment

Before creating user accounts, you *must* complete the following steps:

1. Install nctl using the description in the *Nauta User Guide*.
2. Copy the `nauta-admin.config` file to the machine where `nctl` resides.
    - An Administrator will need the `nauta-admin.config` file on their machine (the machine where nctl resides).
3. Set up the `KUBECONFIG` variable to point to the full path of `nauta-admin.config` to where you copied the file in step number 2.
4. Follow the instructions below (*Creating a User*) to create users.

```
export KUBECONFIG=<PATH>/nauta-admin.config
```

Creating a User Account

The following steps are used to create a Nauta user account, not a Nauta Administrator account. Only a Nauta Administrator account has permissions to create user accounts. The Nauta Administrator account file is created as a result of the Nauta installation and should be obtained from the person that completed the installation.

Administrator Tasks

1. The Nauta `user create` command sets up a namespace and associated roles for the named user on the cluster. It sets up home directories, named after the username, on the input and output network shares with file-system level access privileges. To create a user, execute the following command:

```
nctl user create <username>
```

2. This command also creates a configuration file named: `<username>.config` and places this file in the user's home directory. To verify that your new user has been created, execute the following command:

```
nctl user list
```

3. This lists all users, including the new user just added, but *does not* show Administrators. A partial example is shown below.

```
| Name     | Creation date          | Date of last submitted job   |
|---------+-----------------------+-----------------------------+
| user1    | 2019-03-12 08:30:45 PM | 2019-02-27 07:55:13 PM       |
| user2    | 2019-03-12 09:50:50 PM |                              |
| user3    | 2019-03-12 09:51:31 PM |                              |
```

The above command lists all users, including any new user just added.

User Tasks

1. As Administrator, you need to provide `<username>.config` file to the Nauta user. The user *must* save this file to an appropriate location of their choosing on the machine that is running Nauta; for example, their `home directory` using the following command:

```
cp <username>.config ~/
```

2. Use the export command to set this variable for the user:

```
export KUBECONFIG=~/<username>.config
```

Limitations

Users with the same name *cannot* be created directly after being removed. In addition, user names are limited to a 32-character maximum and there are no special characters except for hyphens. However, all names *must* start with a letter *not* a number. You can use a hyphen to join user names, for example: john-doe (see Troubleshooting, page 38 for more details).

Delete a User Account

Only an Administrator can delete user accounts and deleting a user removes that user's account from the Nauta software; therefore, that user *will not* be able to log in to the system. This will halt and remove all experiments and pods; however, all artifacts related to that user's account, such as the users input and output folders and all data related to past experiments they have submitted remains.

Remove a User

To remove a user, execute the following command:

```
nctl user delete <username>
```

Respond to this question to confirm the previous step.

```
Do you want to continue? [y/N]: press y to confirm deletion.
```

The command may take up to 30 seconds to delete the user and you may receive the message: `User is still being deleted`. Check the status of the user after a few minutes. Recheck as desired.

Purging Process

To permanently remove (*Purge*) all artifacts associated with the user and all data related to past experiments submitted by that user (but **excluding** the contents of the user's input and output folders) , execute the following command:

```
nctl user delete <username>  -p/--purge
```

Respond to this question to confirm the previous step.

```
Do you want to continue? [y/N]: press y to confirm deletion.
```

Limitations

The `nauta user delete` command may take up to 30 seconds to delete the user. A new user with the same user name *cannot* be created until after the delete command confirms that the first user with the same name has been deleted (see Troubleshooting, page 38 for more details).

Launching the Web UI

To review the Resources Dashboard, launch the Web UI from the Command Line Interface (CLI). Refer to the Nauta User Guide for more information.

Do the following to launch the Web UI:

1. Execute the following command:

   ```
   nctl launch webui
   ```

2. Your default Web browser opens and displays the Web UI. For Administrators, the Web UI displays empty a list experiments and shows the following message:

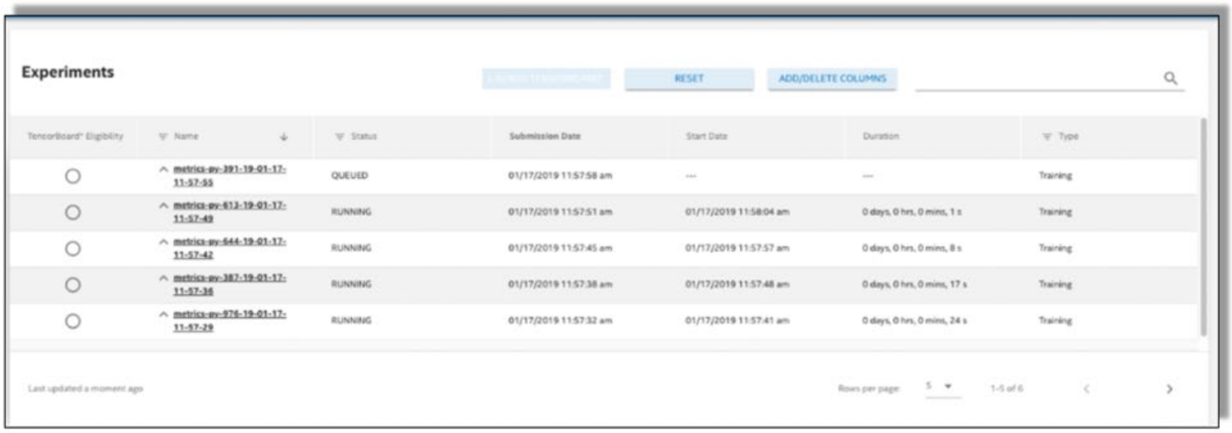
   ```
   No data for currently signed user. Click here to load all users data.
   ```

   The data is loaded and displays.

Nauta Web UI

Figure 2 shows the Nauta Web UI. This UI contains experiment information that can be sorted by name, status, and various dates.). Refer to the Nauta User Guide for more information.

Figure 2: Nauta Web UI



Kubernetes Resource Dashboard Overview

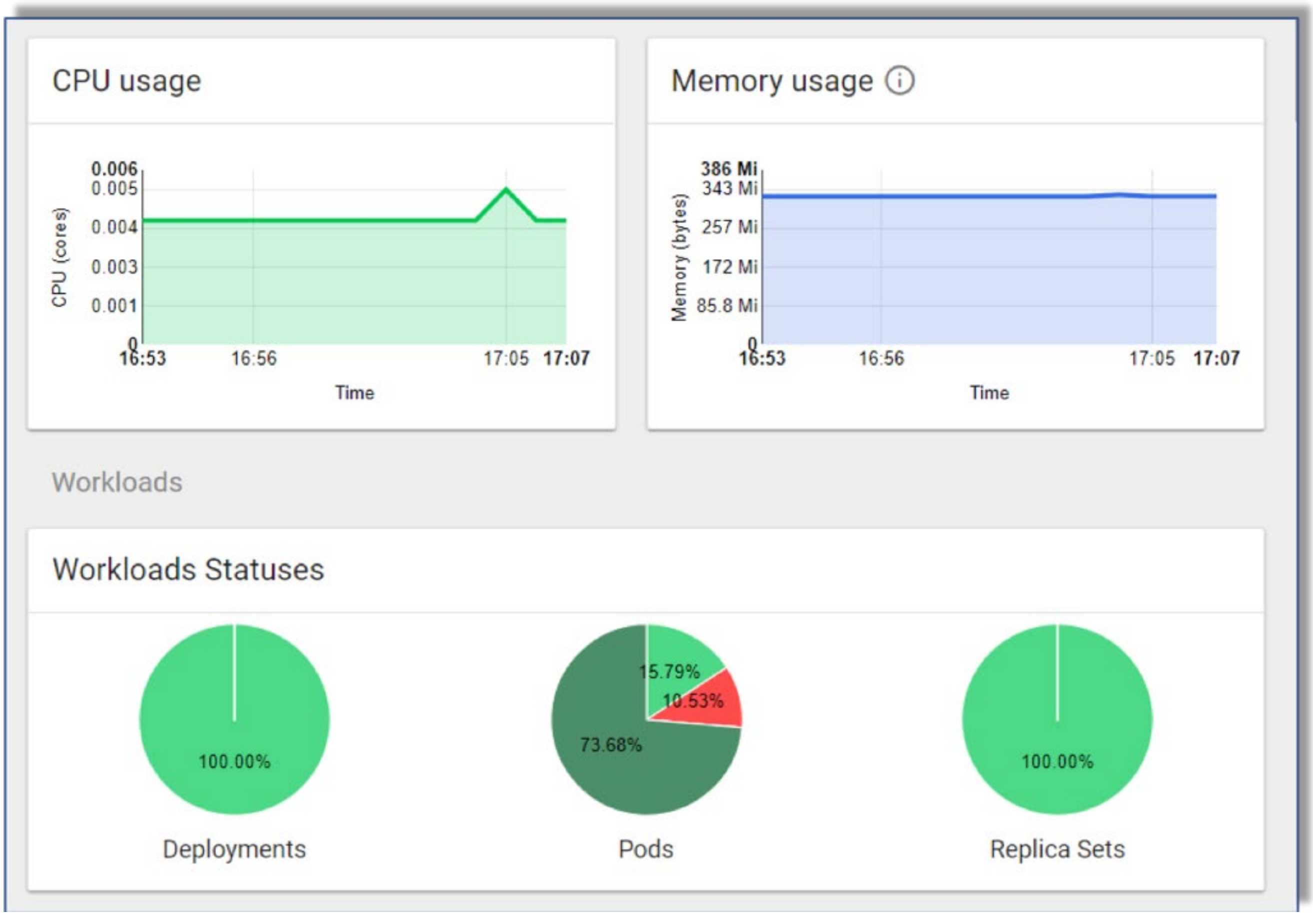
As an Administrator you may need to access the Kubernetes Dashboard. Kubernetes provides a way to manage containerized workloads and services, to manage resources given to a particular experiment and monitor workload statuses and resource consumption. Refer to Kubernetes Web UI (Dashboard) for detailed Kubernetes information.

To access Kubernetes:

1. Click the **Hamburger Menu**  at the far left of the UI to open a left frame.
2. Click **Resources Dashboard** to open the Kubernetes resources dashboard in a new browser window/tab.

Figure 3, page 37 shows an example Kubernetes Dashboard.

Figure 3: Kubernetes Dashboard—Example Only



Troubleshooting

This section provides information related to Nauta-related issues, descriptions, and workarounds. Before contacting customer support or filing a ticket, refer to the information below.

This section discusses the following main topics

Jupyter Error 1

Saving a file causes the following error:

```
Creating file failed. An error occurred while creating a new file.
Unexpected error while saving file: input/home/filename [Errno 2]
No such file or directory: '/mnt/input/home/filename'
```

The error appears when a user tries to save file in `/input/home` folder which is a read-only folder. In

Jupyter Error 1 Workaround

Jupyter, select the `/output/home` folder to correct this issue.

Jupyter Error 2

Closing the Jupyter notebook window in a Web browser causes experiments to stop executing. Attaching to the same Jupyter session still shows the stopped experiment.

Note: This is a known issue in Jupyter (refer to: keep notebook running after the browser tab closed #1647 on GitHub for more information).

Jupyter Error 2 Workaround

Currently, there *is no* workaround.

Docker Error

The Docker installation on the client machine takes up significant space and contains a large amount of container images. Refer to Official Docker Documentation for more information.

Docker Error Workaround

Docker takes a conservative approach to cleaning up unused objects (often referred to as garbage collection), such as images, containers, volumes, and networks. These objects are generally *not* removed unless you explicitly ask Docker to do so.

Note: Refer to the following information for detailed instructions on how to prune unused Docker images: Prune unused Docker objects.

Removal of Docker Images

Due to known errors in Docker Garbage Collector making automatic removal of Docker images is laborious and error-prone.

Periodic Registry Clean Up

Before running the Docker Garbage Collector, the administrator should remove images that are no longer needed and perform Docker's registry cleanup periodically.

If there are too many images in registry it may negatively impact the submission of experiments: submitting of experiments works much slower than usual and eventually a user *is not* able to submit an experiment. To prevent this, administrators should perform this cleanup periodically.

Removal of Docker Images Procedure and Workaround

1. Expose the internal Docker registry's API by exposing locally port 5000, exposed by *`nauta-docker-registry service`* located in the *nauta* namespace. This can be done, for example by issuing the following command on a machine that has access to Nauta: `kubectl port-forward svc/nauta-docker-registry 5000 -n nauta`
2. Get a list of images stored in the internal registry by issuing the following command (it is assumed that port 5000 is exposed locally): curl http://localhost:5000/v2/_catalog
    - For more information on Docker Images, refer to: docker image ls.
3. From the list of images received in the previous step (step 2), choose those that should be removed. For each chosen image, execute the following steps:
    a. Get the list of tags belonging to the chosen image by issuing the following command:
    curl http://localhost:5000/v2/<image_name>/tags/list
    b. For each tag, get a digest related to this tag:
    curl --header "Accept: application/vnd.docker.distribution.manifest.v2+json" http://localhost:5000/v2/<image_name>/manifests/<tag>

    Digest is returned in a header of a response under the Docker-Content-Digest key
    c. Remove the image from the registry by issuing the following command:
    curl -X "DELETE" --header "Accept: application/vnd.docker.distribution.manifest.v2+json" http://localhost:5000/v2/<image_name>/manifests/<digest>
4. Run Docker Garbage Collector by issuing the following command:

```
kubectl exec -it $(kubectl get --no-headers=true pods -l app=docker-registry -n
nauta -o custom-columns=:metadata.name) -n nauta registry garbage-collect
/etc/docker/registry/config.yml
```

5. Restart system's Docker registry. This can be done by deleting the pod with label:
`nauta_app_name=docker-registry`

User Management Error

Users witht the Same Name Cannot be Created

After deleting a user name and verifying that the user name *is not* on the list of user names, it *is not* possible to create a new user with the same name within a short period of time (*roughly a few minutes*). This is due to a user's-related Kubernetes objects, which are deleted asynchronously by Kubernetes. Due to these deletions, it can take time to resolve.

User Management Error Workaround

To resolve, wait a few minutes before creating a user with the same name.

Nauta Connection Error

Launching TensorBoard instance and launching Web UI *does not* work.

After running Nauta to launch the Web UI (`nctl launch webui`) or the `nauta launch tb <experiment_name>` commands, a connection error message may be visible. During the usage of these commands, a proxy tunnel to the cluster is created.

As a result, a connection error can be caused by an incorrect `user-config` generated by Administrator or by incorrect proxy settings in a local machine.

Nauta Connection Error Workaround

To prevent this, ensure that a valid `user-config` is used and check the proxy settings. In addition, ensure that the current proxy settings *do not* block any connection to a local machine.

DNS Server has Changed or Missed in Installation Step

To change DNS settings in the installation, make the changes on the master node:

- Stop consul service: `systemctl stop consul`
- Change the file with your favorite text editor, for example: `vim vim /etc/consul/dns.json`
- In recursor provide proper DNS server, for example: `"recursors" : ["8.8.8.8","8.8.4.4"]`
- Start consul service: `systemctl start consul`

Insufficient Resources Causes Experiment Failures

An experiment fails just after submission, even if the script itself is correct.

If a Kubernetes cluster *does not* have enough resources, the pods used in experiments are evicted. This results in failure of the whole experiment, even if there are no other reasons for this failure, such as those caused by users (like lack of access to data, errors in scripts and so on).

Insufficient Resources Causes Experiments Failures Workaround

It is recommended that the Administrator investigate the failures to determine a course of action. For example, determine why have all the resources been consumed. Once determined, try to free them.

Experiment's Pods Stuck in Terminating State on Node Failure

There may be cases where a node suddenly becomes unavailable, for example due to a power failure. Experiments using TFJob templates, which were running on such a node, will stay *Running in Nauta* and pods will terminate indefinitely. Furthermore, an experiment *is not* rescheduled automatically.

An example of a visible occurrence using the `kubectl` tool is shown below.

```
                             user@ubuntu:~$ kubectl get nodes
NAME                                    STATUS      ROLES     AGE    VERSION
worker0.node.infra.nauta                 NotReady    Worker    7d     v1.10.6
worker1.node.infra.nauta                 Ready       Worker    7d     v1.10.6
master.node.infra.nauta                  Ready       Master    7d     v1.10.6

                              user@ubuntu:~$ kubectl get pods
NAME                                       READY    STATUS         RESTARTS    AGE
experiment-master-0                        1/1      Terminating    0           45m
tensorboard-service-5f48964d54-tr7xf       2/2      Terminating    0           49m
tensorboard-service-5f48964d54-wsr6q       2/2      Running        0           43m
tiller-deploy-5c7f4fcb69-vz6gp             1/1      Running        0           49m
```

Experiment's Pods Stuck in Terminating State on Node Failure Workaround

To solve this issue, manually resubmit the experiment using Nauta. This is related to unresolved issues found in Kubernetes and design of TF-operator. For more information, see GitHub links below.

- Pod stuck in unknown status when kubernetes node is down #720
- Pods are not moved when Node in NotReady state #55713

A Multinode, Horovod-based Experiment Receives a FAILED Status after a Pod's Failure

If during the execution of a multinode, (and a Horovod-based experiment) one of the pods fails, then the whole experiment gets the FAILED status. Such behavior is caused by a construction of a Horovod framework. This framework uses an Open MPI framework to handle multiple tasks instead of using Kubernetes features. Therefore, it cannot rely also on other Kubernetes features such as restarting pods in case of failures.

A Multinode Horovod Experiment Receives a FAILED Status Workaround

This results in training jobs using Horovod does not resurrect failed pods. The Nauta system also does not restart these training jobs. If a user encounters this, they should rerun the experiment manually. The construction of a multinode TFJob-based experiment is different, as it uses Kubernetes features. Thus, training jobs based on TFJobs restart failing pods so an experiment can be continued after their failure without a need to be restarted manually by a user.

Experiments Still in RUNNING State Even if they Finish with Success

This can happen due to a known issue in Kubernetes client library it may happen, especially when there are a lot of experiments running at the same time. As a result, some single-node experiments _may still be_ in RUNNING status, even if scripts run within such experiments are finished with success. Our analysis indicates that this problem may affect roughly 1% of experiments.

Experiments Still in RUNNING State Even if they Finish with Success Workaround

Until a resolution of the problem on library's side is found, monitor the statuses of experiments and check, whether they *are not* running too long relative to predicted duration. If there are such cases, cancel an experiment *without* purging it. The results are still available on shares.