Nuage Networks Metro Automation Engine (MetroAE)

Version: 4.4.0

MetroAE is an automation engine that can be used to

- Install
- Health check
- Upgrade
- Configure
- Backup/restore
- Shutdown/restart
- Failover/failback

Nuage Networks components. You specify the individual details of your target platform, then let MetroAE do the rest.

Documentation

The Documentation directory contains the following guides to assist you in successfully working with MetroAE. The current documentation covers using MetroAE CLI only.

FILE NAME	DESCRIPTION
RELEASE_NOTES	New features, resolved issues and known limitations and issues
GETTING_STARTED	MetroAE Quick Start Guide

FILE NAME	DESCRIPTION
DOCKER	Installing and using MetroAE Docker container
SETUP	Set up your environment by cloning the repo, installing packages and configuring access.
CUSTOMIZE	Populate variable files for a deployment and unzip Nuage software.
VAULT_ENCRYPT	Safeguard sensitive data
DEPLOY	Deploy all VSP components or choose components individually.
DESTROY	Remove existing deployment(s) and start over.
CONFIG	MetroAE Config, the template-driven VSD configuration tool.
UPGRADE_SA	Upgrade component(s) from one release to the next in a standalone environment.
UPGRADE_HA	Upgrade component(s) from one release to the next in a clustered environment.
VSD_SERVICES	Using MetroAE to control VSD Services during maintenance
VSD_ROLLBACK_RESTORE	Roll Back or Restore VSD to original version outside normal upgrade path

FILE NAME	DESCRIPTION
VSD_CLUSTER_FAILOVER	Managing VSD cluster failover and failback
NSGV_BOOTSTRAP	Using MetroAE to bootstrap NSGv components
SD-WAN_PORTAL	Deploying SD-WAN portal using MetroAE
AWS	Using MetroAE to deploy on Amazon Web Services (AWS)
TERRAFORM	Using the Terraform framework in conjunction with MetroAE
HOOKS_AND_SKIPACTIONS	Configure ability to run custom commands in between playbooks and skip playbooks
PLUGINS	Add or remove plugins to MetroAE
AUTOGENERATE_DEPLOYMENTS	Generate MetroAE deployment configurations automatically via script
CONTRIBUTING	Contribute to MetroAE development
LICENSE	License statement for MetroAE

Important Notes

You can now run `python run_wizard.py` to let MetroAE help you setup your server and create or edit a deployment. The wizard will ask you questions about what you'd like to do and then create the proper files on disk. run_wizard.py is most useful when you are working with a clone of the nuage-metroae repo. It

can be used to generate deployments that can be used with the Docker container version of MetroAE, but those deployments would need to be copied manually from the nuage-metroae repo clone to the container's metroae_data directory.

Please see RELEASE_NOTES.md for all the details about what has changed or been added in this release.

All MetroAE operations, including Docker container management, use a command `metroae` for consistent usage and syntax. Please see DOCKER.md for details on configuration and use of the container version of MetroAE.

Supported Components for Deployment and Upgrade

MetroAE supports deployment and upgrade of the following components as VMs on the target server. These are the same target server types that are supported on the VSP platform.

Component	Install	Upgrade	SA / HA	Active / Standby Cluster
VSD (Virtualized Services Directory)	✓	✓	SA / HA	✓
VSTAT stats-in (Elasticsearch)	✓	✓	SA / HA	✓
VSC (Virtualized Services Controller)	✓	✓	SA / HA	
NUH (Network Utility Host)	✓		SA / HA	
VCIN (vCenter Integration Node)	✓		SA / HA	
VNSUTIL (Virtualized Network Services-Utility)	✓		SA Only	
VRS (Virtual Routing & Switching)	✓		-	
NSGv (Virtual Network Services Gateway)	✓		-	

Supported Dataplane components (installed as package/agent):

COMPONENT	KVM (EL6, EL7,UBUNTU 14.04/16.04)	ESXI
VRS (Virtual Routing & Switching)	X	(upgrade only)

Required Auxiliary Services

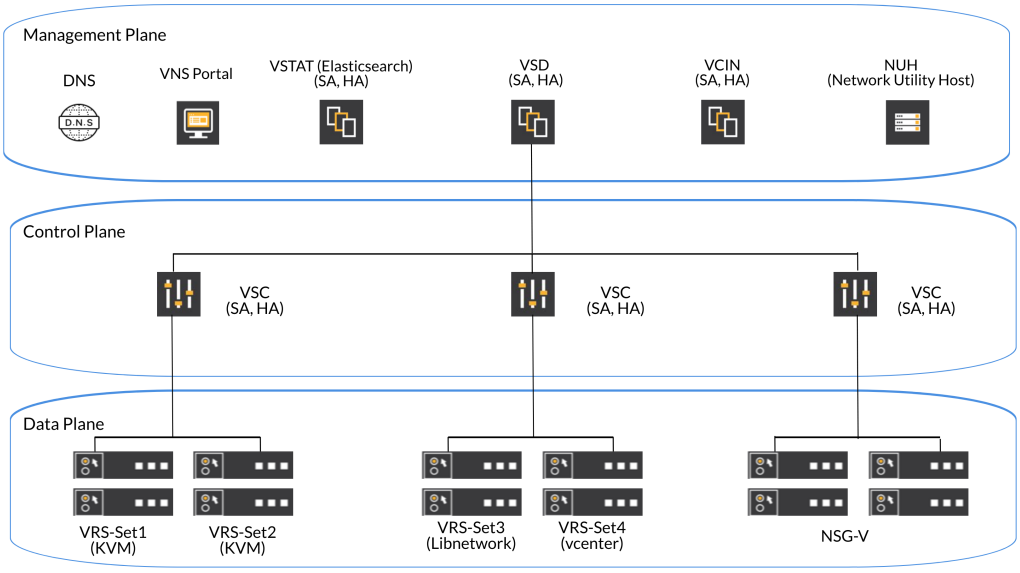
DNS/NTP

Supported Target Servers

MetroAE supports the deployment and upgrade of Nuage VSP components on the following target servers.

Component	KVM	VMware	OpenStack	AWS
VSD (Virtualized Services Directory)	✓	✓	✓	✓
VSTAT stats-in (Elasticsearch)	✓	✓	✓	✓
VSC (Virtualized Services Controller)	✓	✓	✓	
NUH (Network Utility Host)	✓			
VCIN (vCenter Integration Node)	✓	✓	✓	
VNSUTIL (Virtualized Network Services-Utility)	✓	✓		
VRS (Virtual Routing & Switching)	✓			
NSGv (Virtual Network Services Gateway)	✓	✓		✓

Typical Nuage Topology



Main Steps for Using MetroAE

1. Setup the MetroAE host. Setup prepares the host for running MetroAE, including retrieving the repository, installing prerequisite packages and setting up SSH access. You also have the option of installing MetroAE in a container, and then working with it via CLI or the GUI.

2. Obtain the proper Nuage binary files for your deployment. These can be downloaded from Nuage/Nokia online customer support.

3. Customize your deployment to match your network topology, and describe your Nuage Networks specifics.

4. Deploy new components, upgrade a standalone or clustered deployment, or run a health check on your system.

5. If things did not work out as expected, destroy your environment and redeploy.

MetroAE Workflows

MetroAE workflows are the operations that can be performed against a specified deployment. All supported workflows can be listed via:

```
metroae --list
```

Workflows fall into the following categories:

Standard Workflows

Standard workflows in MetroAE perform the following operations:

WORKFLOW	OPERATION DESCRIPTION
Predeploy	prepares infrastructure with necessary packages and makes the component(s) reachable
Deploy	installs and configures component(s)
Postdeploy	performs integration checks, and some basic commissioning tests
Health	checks health for a running component without assuming it was deployed with MetroAE
Destroy	removes component(s) from the infrastructure
Upgrade	upgrades component(s) from one release to another
Services	controls services for component(s) for maintenance

The following workflows are examples that combine together several of the above operations into simple to use groups:

install everything - Deploys all components specified in a deployment.

destroy everything - Destroys all components specified in a deployment.

health - Checks the health of all components specified in a deployment.

Special Workflows

Special workflows in MetroAE perform certain specific operations. Some of them are shown below.

WORKFLOW	OPERATION DESCRIPTION
tools copy qcow	Copy qcow2 images to target server
tools encrypt credentials	Encrypt credentials.yml in your deployment
tools unzip images	Unzip Nuage images
tools convert csv	Convert spreadsheet (CSV) to deployment
wizard	Run MetroAE Wizard
vsd certificates renew	Renew certificates on VSDs
vsc harden	Harden the VSC configuration

Collecting Triage Collateral : get_debug.py

In the event that you contact the MetroAE team for help, often via the team's email address, devops@nuagenetworks.net, you might be asked to provide a set of files that provide the collateral our engineers need to triage the situation. `get_debug.py` is provided to make the process of gathering files easier for you. `get_debug.py` will create a zip archive that contains the following files and folders:

ansible.log

deployments folder

src/inventory

If you run `get_debug.py` without arguments, it will include the entire contents of the `deployments` folder, the entire contents

of the `src/inventory` folder, and the file `ansible.log`, in a zip file in the current directory. The name of the file, by default, will be of the form `debug-<timestamp>.tar.gz`.

Optionally, you can pass `get_debug.py` 1 or 2 parameters:

> tarFileName: The name of the zip file to create. If not specified, the default value is of the form `debug-<timestamp>.tar.gz`
>
> deploymentName: The name of the deployment folder under `deployments` to include in the zip file. If not specified, the default action is to include all folders under `deployments`.

Command To run the script:
python get_debug.py [–tarFileName name-of-file] [–deploymentName name-of-deployment]

Python-based Ansible Operations Tool

MetroAE is based off of the Python-based Ansible operations tool.

Ansible provides a method to easily define one or more actions to be performed on one or more computers. These tasks can target the local system Ansible is running from, as well as other systems that Ansible can reach over the network. The Ansible engine has minimal installation requirements. Python, with a few additional libraries, is all that is needed for the core engine. MetroAE includes a few custom Python modules and scripts. Agent software is not required on the hosts to be managed. Communication with target hosts defaults to SSH. Ansible does not require the use of a persistent state engine. Every Ansible run determines state as it goes, and adjusts as necessary given the action requirements. Running Ansible requires only an inventory of potential targets, state directives, either expressed as an ad hoc action, or a series coded in a YAML file, and the credentials necessary to communicate with the target.

Playbooks are the language by which Ansible orchestrates, configures, administers and deploys systems. They are YAML-formatted files that collect one or more plays. Plays are one or more tasks linked to the hosts that they are to be executed on.

Roles build on the idea of include files and combine them to form clean, reusable abstractions. Roles are ways of automatically loading certain vars files, tasks, and handlers based on a known file structure.

Nomenclature

Ansible Host: The host where MetroAE runs. Ansible and the required packages are installed on this host. The Ansible Host must run el7 Linux host, e.g. CentOS 7.* or RHEL 7.*.

MetroAE User: The user who runs MetroAE to deploy and upgrade components.

Target Server: The hypervisor on which one or more VSP components are installed as VMs. Each deployment may contain more than one Target Server.

Questions, Feedback, and Contributing

Ask questions and get support on the MetroAE site. You may also contact us directly. Outside Nokia: devops@nuagenetworks.net Internal Nokia: nuage-metro-interest@list.nokia.com

Report bugs you find and suggest new features and enhancements via the GitHub Issues feature.

You may also contribute to MetroAE by submitting your own code to the project.

License

LICENSE

Metro Automation Engine Release Notes

Release info

MetroAE Version 4.4.0

Nuage Release Alignment 20.10

Date of Release 14-April-2021

Release Contents

Feature Enhancements

Fix up the image paths to root at /metroae_data/ when in the container

Added support for bootstraping NSGv using NUH external interfaces

Support multiple KVM bridges for NUH external interfaces

Added multi cpu core support for NSGV (METROAE-309)

Improve VSD security hardening to monitor service shutdown (METROAE-285)

Add next-hop to VSC VPRN config (METROAE-319)

Add VCIN SA upgrade support for Major Minor and Inplace Upgrade(METROAE-317)

Adding support for sending usage back to MetroAE team

Support for configuring NUH for stats-out proxy (METROAE-288)

Add plugin examples and documentation to MetroAE (METROAE-354)

Added support for instantiating and bootstrapping NSGv on AWS with VSP components deployed outside AWS

(METROAE-355)

Resolved Issues

Enhance several license file descriptions to include the file name

Updated SDWAN portal to support 20.11 version

Force rebuild after missing files (METROAE-298)

Updated SDWAN portal unzip task to copy the container tar file to correct folder

Fixed VSD deploy task to be idempotent (METROAE-304)

Fixed sshpass quotes for passwords that has special characters (METROAE-312)

Add option to override VSC config (METROAE-314)

Allow portal install without VSTAT fqdn (METROAE-301)

Fix VSR deploy errors (METROAE-310)

Fix vsd-install script parameters (METROAE-347)

Fix DNS dns_mgmt and dns_data to relax requirement of having specific length. (METROAE-315)

Add vstat-health role to vstat_health playbook (METROAE-322)

Hide log output in vsd-node-info (METROAE-332)

Remove vstat-vsd-health role from vstat_health playbook (METROAE-322)

Removed extra DEBUG line from nsgv-postdeploy (METROAE-334)

Removed DEBUG tasks which are not working in upgrade shutdown playbooks (METROAE-358)

Allowing metroae to continue execution if the vsd_continue_on_failure is set to true (METROAE-324)

Fix webfilter install required bridges issue (METROAE-371)

Test Matrix

This release was tested according to the following test matrix. Other combinations and versions have been tested in previous releases of MetroAE and are likely to work. We encourage you to test MetroAE in your lab before you apply it in production.

WORKFLOW	TARGET SERVER	VERSION
CONFIGURE	GCP	SA-20.10.R1
CONFIGURE	GCP	SA-5.4.1
CONFIGURE	GCP	SA-6.0.3
INSTALL	GCP	GEO-20.10.R1
INSTALL	GCP	HA-20.10.R1
INSTALL	GCP	HA-20.10.R1-ACTIVE-STANDBY-ES
INSTALL	GCP	HA-20.10.R1-IPv6
INSTALL	GCP	HA-20.10.R1-NO-VSC-FALLOCATE
INSTALL	GCP	HA-5.4.1
INSTALL	GCP	HA-6.0.3
INSTALL	GCP	SA-20.10.R1
INSTALL	GCP	SA-20.10.R1-ACTIVE-STANDBY-ES
INSTALL	GCP	SA-20.10.R1-ADD-VSC
INSTALL	GCP	SA-20.10.R1-CONTAINER
INSTALL	GCP	SA-20.10.R1-CPU-CORE
INSTALL	GCP	SA-20.10.R1-CPU-CORES-CPU_PINNING

WORKFLOW	TARGET SERVER	VERSION
INSTALL	GCP	SA-20.10.R1-CPU_PINNING
INSTALL	GCP	SA-20.10.R1-E2E
INSTALL	GCP	SA-20.10.R1-E2E-MUTLI-UPLINKS
INSTALL	GCP	SA-20.10.R1-FEATURES
INSTALL	GCP	SA-20.10.R1-IPV6
INSTALL	GCP	SA-20.10.R1-PLUGINS
INSTALL	GCP	SA-20.10.R1-TERRAFORM
INSTALL	GCP	SA-5.4.1
INSTALL	GCP	SA-6.0.3
INSTALL	GCP	STATS-OUT-20.10.R1
INSTALL	OPENSTACK	HA-20.10.R1
INSTALL	OPENSTACK	HA-6.0.3
INSTALL	OPENSTACK	SA-20.10.R1
INSTALL	OPENSTACK	SA-6.0.3
INSTALL	OPENSTACK	SA-CONTAINER-6.0.3
INSTALL	VCENTER	20.10.R1-HYBRID-VCIN
INSTALL	VCENTER	HA-20.10.R1
INSTALL	VCENTER	HA-20.10.R1-CHANGE-VSDPASS
INSTALL	VCENTER	HA-20.10.R1-VCIN
INSTALL	VCENTER	HA-5.4.1
INSTALL	VCENTER	HA-6.0.3
INSTALL	VCENTER	SA-20.10.R1

WORKFLOW	TARGET SERVER	VERSION
INSTALL	VCENTER	SA-20.10.R1-CHANGE-VSDPASS
INSTALL	VCENTER	SA-20.10.R1-VENV
INSTALL	VCENTER	SA-5.4.1
INSTALL	VCENTER	SA-6.0.3
RESTORE	GCP	HA-6.0.3
RESTORE	GCP	SA-6.0.3
UPGRADE	GCP	GEO-5.4.1-6.0.3
UPGRADE	GCP	GEO-6.0.3-20.10.R1
UPGRADE	GCP	GEO-6.0.3-6.0.7-INPLACE
UPGRADE	GCP	HA-5.4.1-5.4.1U5-INPLACE
UPGRADE	GCP	HA-5.4.1-6.0.3
UPGRADE	GCP	HA-5.4.1-6.0.3-HARDENED
UPGRADE	GCP	HA-6.0.3-20.10.R1
UPGRADE	GCP	HA-6.0.3-20.10.R1-HARDENED
UPGRADE	GCP	HA-6.0.3-6.0.7-INPLACE
UPGRADE	GCP	SA-5.4.1-6.0.3
UPGRADE	GCP	SA-5.4.1-6.0.3-HARDENED
UPGRADE	GCP	SA-5.4.1-6.0.3-VSD-SECURITY
UPGRADE	GCP	SA-6.0.3-20.10.R1
UPGRADE	GCP	SA-6.0.3-6.0.7-INPLACE
UPGRADE	OPENSTACK	HA-5.4.1-6.0.3

WORKFLOW	TARGET SERVER	VERSION
UPGRADE	OPENSTACK	SA-5.4.1-6.0.3
UPGRADE	VCENTER	HA-5.4.1-6.0.3
UPGRADE	VCENTER	HA-5.4.1-6.0.3-HARDENED
UPGRADE	VCENTER	HA-6.0.3-20.10.R1
UPGRADE	VCENTER	HA-6.0.3-20.10.R1-HARDENED
UPGRADE	VCENTER	HA-INPLACE-5.4.1-5.4.1U5
UPGRADE	VCENTER	HA-INPLACE-6.0.3-6.0.7
UPGRADE	VCENTER	SA-5.4.1-6.0.3
UPGRADE	VCENTER	SA-5.4.1-6.0.3-CONTAINER
UPGRADE	VCENTER	SA-6.0.3-20.10.R1
UPGRADE	VCENTER	SA-6.0.3-20.10.R1-HARDENED

MetroAE Quick Start Guide

1. Read documentation

1.1 Readme for information on supported components
1.2 Setup for setting up the MetroAE host and enabling SSH
1.3 Customize for customizing user data and files
1.4 Release Notes for information on the latest features

2. Setup MetroAE Host

What's a MetroAE Host?

It can be a VM, physical server or container.

It requires CentOS 7.x or RHEL 7.x with basic packages.

We recommend that you dedicate a server or VM that has following requirements:

Minimum of 2 CPUs, 4 GBs memory, and 40 GB disk

A read/write NFS mount for accessing the Nuage software images

2.1 Clone the master branch of the repo onto the **MetroAE Host**. Read Setup for details.

```
git clone https://github.com/nuagenetworks/nuage-metroae.git
```

2.2 Install the required packages. Run as root or sudo. Read Setup for details.

```
$ sudo ./setup.sh
```

3. Enable SSH Access

Passwordless SSH must be configured between the MetroAE host and all target servers, a.k.a. hypervisors. This is accomplished by generating SSH keys for the MetroAE user, then copying those keys to the authorized_keys files for the `target_server_username` on every `target_server`. The following steps should be executed on the MetroAE server as the MetroAE user.

Please note that this is only a requirement for target servers that are KVMs. Passwordless SSH is not required for these other target-server types: AWS, Openstack, and vCenter.

3.1 Generate keys for the MetroAE user

3.1.1 As MetroAE User on the MetroAE server, generate SSH keys:
`ssh-keygen`

3.2 Copy public key to each `target_server`

3.2.1 When you are going to run as 'root' on each `target_server`

As MetroAE User on the MetroAE server, copy SSH public key:
`ssh-copy-id root@<target_server>`

3.2.2 When you are going to run as `target_server_username` on each `target_server`

As MetroAE User on the MetroAE server, copy SSH public key:
`ssh-copy-id <target_server_username>@<target_server>`.

See Setup for more details about enabling SSH Access.

4. Install ovftool (for VMware only)

Download and install the ovftool from VMware. MetroAE uses ovftool for OVA operations. Note that MetroAE is tested using

ovftool version 4.3. ovftool version 4.3 is required for proper operation.

Note that running the metroae Docker container for VMware installations and upgrades requires special handling of the location of the ovftool command. Please see SETUP.md for details.

5. Prepare your environment

5.1 Unzip Nuage files

Execute: `metroae tools unzip images <zipped_directory> <unzip_directory>`

See SETUP.md for details.

Be sure that Nuage packages (tar.gz) are available on localhost (MetroAE host),

either in a native directory or NFS-mounted.

5.2 Checklists for Target Servers

KVM

- [ ] MetroAE host has ability to do a password*less SSH as root to the target server.
- [ ] Sufficient disk space / resources exist to create VMs.
- [ ] KVM is installed.
- [ ] All required management and data bridges are created.

vCenter

- [ ] User specified has required permissions to create and configure a VM.
- [ ] ovftool has been downloaded from VMware onto the MetroAE Host.

[ ] pyvmomi has been installed on MetroAE Host: `pip install pyvmomi`.

5.3 Reachability

MetroAE host must be able to resolve the host names of the Nuage components into their correct management IP addresses. This is required so that MetroAE can operate on each component in the deploy, post-deploy, and health workflows.

Next Steps

Refer to the list of documents in README.md for guidance on deploying, upgrading, etc.

Questions, Feedback, and Contributing

Get support via the forums on the MetroAE site.
Ask questions and contact us directly at devops@nuagenetworks.net.

Report bugs you find and suggest new features and enhancements via the GitHub Issues feature.

You may also contribute to MetroAE by submitting your own code to the project.

Deploying Components with the MetroAE Docker Container

This file describes many of the details of the commands used for managing MetroAE distributed via container. For information on how to setup Docker and the MetroAE Docker container, see SETUP.md.

The metroae Command

The metroae command is at the heart of interacting with the MetroAE container. It is used both to manage the container and to execute MetroAE inside the container. You can access all of the command options via `metroae <workflow> <component> [action] [deployment] [options]`.

metroae Container Management Command Options

The following command options are supported by the metroae command:

help -displays the help text for the command

pull - pulls the MetroAE container from the registry. By default, the latest container is pulled. You can also specify a valid container tag to pull another version, e.g. `metroae container pull 1.0.0`.

download - pulls the MetroAE container in tar format. This allows you to copy the tar file to systems behind firewalls and convert to Docker images by using `docker load` command.

setup - setup completes the steps necessary to get the MetroAE container up and running. It prompts you for the full paths to a data directory that the container uses on your local disk. The

setup action will create the subdirectory `metroae_data` on disk, then mount it as `/metroae_data/` inside the container. When using the MetroAE container, you must provide paths relative to this path as seen from inside the container. For example, if you tell setup to use `/tmp` for the data directory, setup will create `/tmp/metroae_data` on your host. The setup will also mount this directory inside the container as `/metroae_data/`. If you copy your tar.gz files to `/tmp/metroae_data/images/6.0.1/` on the host, the container sees this as `/data/images/6.0.1/`. So, when using unzip or setting `nuage_unzipped_files_dir` in common.yml, you would specify `/metroae_data/images/6.0.1/` as your path.

Running setup multiple times replaces the existing container, but it does not remove the data on your local disk.

metroae container start - starts the container using the settings from setup

metroae container stop - stops the running container

metroae container status - displays the container status along with container settings

metroae container destroy - stops and removes the metroae container along with the container image. Use this command when you want to replace an existing container with a new one, no need to run setup again afterwards.

metroae container update - upgrades the container to the latest available release image. You don’t need to run setup again after an upgrade.

metroae tools unzip images - unzips Nuage Networks tar.gz files into the images directory specified during the setup operation. Use of this command requires that the tar.gz files be placed in either the data or images directory that you specified during setup. See the current values of the data and images directories by executing the status command.

metroae tools generate example - generates an example for the specified schema and puts it in the examples directory under the data directory that you specified during setup. You can get the current values of the data and images directories by executing the status command.

metroae tools encrypt credentials - sets the encryption credentials for encrypting secure user data, e.g. passwords

metroae container ssh copyid - copies the container's public key into the ssh authorized_keys file on the specified server. This key is required for passwordless ssh access to all target servers. Usage: `metroae container ssh copyid user@host_or_ip`

metroae --list - lists the workflows that are supported by MetroAE

metroae --ansible-help - shows the help options for the underlying Ansible engine

metroae Workflow Command Options

The MetroAE container is designed so that you run MetroAE workflows, e.g. install, from the command line using the metroae command. The format of the command line is:

```
`metroae <workflow> <component> [operation] [deployment]
[options]`
```

Troubleshooting

SSH connection problems

If MetroAE is unable to authenticate with your target server, chances are that passwordless ssh has not been configured properly. The public key of the container must be copied to the authorized_keys file on the target server. Use the `copy-ssh-id`

command option, e.g. `metroae container copy-ssh-id user@host_name_or_ip`.

Where is my data directory?

You can find out about the current state of your container, including the path to the metroae_data directory, by executing the container status command, `metroae container status`.

General errors

metroae.log and ansible.log are located in the data directory you specified during setup.

Manually use the container without the script (Nokia internal support only)

Pull the container

```
docker pull registry.mv.nuagenetworks.net:5001/metroae:1.0
```

Run the container

docker run -e USER_NAME='user name for the container' -e GROUP_NAME='group name for the container' -d $networkArgs -v 'path to the data mount point':/data:Z -v 'path to images mount point':/images:Z --name metroae registry.mv.nuagenetworks.net:5001/metroae:1.0

For Linux host

```
networkArgs is '--network host'
```

For Mac host

```
networkArgs is '-p "UI Port":5001'
```

Execute MetroAE Commands

```
docker exec 'running container id' /source/nuage-
metroae/metroae playbook deployment
```

Stop the container

```
docker stop 'running container id'
```

Remove the container

```
docker rm 'container id'
```

Remove MetroAE image

```
docker rmi 'image id'
```

Questions, Feedback, and Contributing

Get support via the forums on the MetroAE site. Ask questions and contact us directly at devops@nuagenetworks.net.

Report bugs you find and suggest new features and enhancements via the GitHub Issues feature.

You may also contribute to MetroAE by submitting your own code to the project.

Setting Up the Environment

You can set up the MetroAE host environment either with a Docker container or with a GitHub clone.

Environment

Method One: Set up Host Environment Using Docker Container

Using a Docker container results in a similar setup as a GitHub clone, plus it delivers the following features:

- All prerequisites are satisfied by the container. Your only requirement for your server is that it run Docker engine.
- Your data is located in the file system of the host where Docker is running. You don't need to get inside the container.
- A future release will include Day 0 Configuration capabilities.

System (and Other) Requirements

- Operating System: Enterprise Linux 7 (EL7) CentOS 7.4 or greater or RHEL 7.4 or greater
- Locally available image files for VCS or VNS deployments
- Docker Engine 1.13.1 or greater installed and running
- Container operations may need to be performed with elevated privileges (*root*, *sudo*)

Steps

1. Get a copy of the metroae script from the github repo

```
https://github.com/nuagenetworks/nuage-
metroae/blob/master/metroae
```

You can also copy the script out of a cloned workspace on your local machine. You can copy the metroae script to any directory of your choosing, e.g. `/usr/local/bin`. Note that you may need to set the script to be executeable via `chmod +x`.

2. Pull the latest Docker container using the following command:

```
metroae container pull
```

3. Setup and start the Docker container using the following command:

```
metroae container setup [path to data directory]
```

You can optionally specify the data directory path. If you don't specify the data directory on the command line, you will be prompted to enter one during setup. This path is required for container operation. The data directory is the place where docs, examples, Nuage images, and your deployment files will be kept and edited. Note that setup will create a subdirectory beneath the data directory you specify, `metroae_data`. For example, if you specify `/tmp` for your data directory path during setup, setup will create `/tmp/metroae_data` for you. Setup will copy docs, logs, and deployment files to `/tmp/metroae_data`. Inside the container itself, setup will mount `/tmp/metroae_data` as `/metroae_data/`. Therefore, when you specify path names for metroae when using the container, you should always specify the container-relative path. For example, if you copy your tar.gz files to `/tmp/metroae_data/6.0.1` on the host, this will appear as `/metroae_data/6.0.1` inside the container. When you use the unzip-files action on the container, then, you would specify a source path as `/metroae_data/6.0.1`. When you complete the nuage_unzipped_files_dir variable in common.yml, you would also specify `/metroae_data/6.0.1`.

Note that you can run setup multiple times and setup will not destroy or modify the data you have on disk. If you specify the same data and imafges directories that you had specified on

earlier runs, metroae will pick up the existing data. Thus you can update the container as often as you like and your deployments will be preserved.

Note that you can stop and start the MetroAE container at any time using these commands:

```
metroae container stop
metroae container start
```

4. For KVM Only, copy the container ssh keys to your KVM target servers (aka hypervisors) using the following command:

```
metroae container ssh copyid
[target_server_username]@[target_server]
```

This command copies the container's public key into the ssh authorized_keys file on the specified target server. This key is required for passwordless ssh access from the container to the target servers. The command must be run once for every KVM target server. This step should be skipped if your target-server type is anything but KVM, e.g. vCenter or OpenStack.

5. For ESXi / vCenter Only, install ovftool and copy to metroae_data directory

When running the MetroAE Docker container, the container will need to have access to the ovftool command installed on the Docker host. The following steps are suggested:

5.1. Install ovftool

Download and install the ovftool from VMware.

5.2. Copy ovftool installation to metroae_data directory

The ovftool command and supporting files are usually installed in the /usr/lib/vmware-ovftool on the host. In order to the metroae container to be able to access these files, you must copy the entire folder to the metroae_data directory on the host. For example, if you have configured the container to use `/tmp/metroae_data` on your host, you would copy `/usr/lib/vmware-ovftool` to `/tmp/metroae_data/vmware-ovftool`. Note: Docker does not support following symlinks. You must copy the files as instructed.

5.3. Configure the ovftool path in your deployment

The path to the ovftool is configured in your deployment in the common.yml file. Uncomment and set the variable 'vcenter_ovftool' to the container-relative path to where you copied the `/usr/lib/vmware-ovftool` folder. This is required because metroae will attempt to execute ovftool from within the container. From inside the container, metroae can only access paths that have been mounted from the host. In this case, this is the metroae_data directory which is mounted inside the container as `/metroae_data`. For our example, in common.yml you would set `vcenter_ovftool: /metroae_data/vmware-ovftool/ovftool`.

6. You can check the status of the container at any time using the following command:

```
metroae container status
```

That's it! Container configuration data and logs will now appear in the newly created `/opt/metroae` directory. Documentation, examples, deployments, and the ansible.log file will appear in the data directory configured during setup, `/tmp/metroae_data` in our examples, above. See DOCKER.md for specfic details of each command and container management command options. Now you're ready to customize for your topology.

Note: You will continue to use the `metroae` script to run commands, but for best results using the container you should make sure your working directory is *not* the root of a nuage-metroaegit clone workspace. The `metroae` script can be used for both the container and the git clone workspace versions of MetroAE. At run time, the script checks its working directory. If it finds that the current working directory is the root of a git cloned workspace, it assumes that you are running locally. It will *not* execute commands inside the container. When using the container, you should run `metroae` from a working directory other than a git clone workspace. For example, if the git clone workspace is `/home/username/nuage/nuage-metroae`, you can `cd /home/username/nuage` and then invoke the command as `./nuage-metroae/metroae install vsds`.

Method Two: Set up Host Environment Using GitHub Clone

If you prefer not to use a Docker container you can set up your environment with a GitHub clone instead.

System (and Other) Requirements

Operating System: Enterprise Linux 7 (EL7) CentOS 7.4 or greater or RHEL 7.4 or greater

Locally available image files for VCS or VNS deployments

Optionally set up and activate python virtual environment

If you would like to run MetroAE within a python virtual environment, please follow the steps below.

1. Install pip

If pip isn't installed on the host, install it with the following command.

```
(sudo) yum install -y python-pip
```

2. Install the virtualenv package

Install the virtualenv package using the following command.

```
pip install virtualenv
```

3. Set up and activate python virtual environment

The python virtual environment can be created as follows.

```
virtualenv [path_to_virtual_environment_or_name]
```

After which the environment can be activated.

```
source [path_to_virtual_environment_or_name]/bin/activate
```

After activating the virtual environment, you can proceed with the rest of the document. The steps will then be performed within the virtual environment. Virtual environments can be exited/deactivated with this command.

```
deactivate
```

Steps

1. Clone Repository

If Git is not already installed on the host, install it with the following command.

```
yum install -y git
```

Clone the repo with the following command.

```
git clone https://github.com/nuagenetworks/nuage-metroae.git
```

Once the nuage-metroae repo is cloned, you can skip the rest of this procedure by running the MetroAE wizard, run_wizard.py. You can use the wizard to automatically handle the rest of the steps described in this document plus the steps described in customize.

```
python run_wizard.py
```

If you don't run the wizard, please continue with the rest of the steps in this document.

2. Install Packages

MetroAE code includes a setup script which installs required packages and modules. If any of the packages or modules are already present, the script does not upgrade or overwrite them. You can run the script multiple times without affecting the system. To install the required packages and modules, run the following command.

```
sudo ./setup.sh
```

The script writes a detailed log into *setup.log* for your reference. A *Setup complete!* messages appears when the packages have been successfully installed.

If you are using a python virtual environment, you will need to run the following command to install the required pip packages.

```
pip install -r pip_requirements.txt
```

Please ensure that the pip selinux package and the yum libselinux-python packages are successfully installed before

running MetroAE within a python virtual environment.

3. For ESXi / vCenter Only, install additional packages

PACKAGE	COMMAND
pyvmomi	`pip install pyvmomi==6.7.3`
jmespath	`pip install jmespath`

Note that we have specified version 6.7.3 for pyvmomi. We test with this version. Newer versions of pyvmomi may cause package conflicts.

If you are installing VSP components in a VMware environment (ESXi/vCenter) download and install the ovftool from VMware. MetroAE uses ovftool for OVA operations.

4. For OpenStack Only, install additional packages

```
pip install --upgrade -r openstack_requirements.txt
```

5. Copy ssh keys

Communication between the MetroAE Host and the target servers (hypervisors) occurs via SSH. For every target server, run the following command to copy the current user's ssh key to the authorized_keys file on the target server:

```
ssh-copy-id [target_server_username]@[target_server]
```

6. Configure NTP sync

For proper operation Nuage components require clock synchronization with NTP. Best practice is to synchronize time on the target servers that Nuage VMs are deployed on, preferrably to the same NTP server as used by the components themselves.

Unzip Nuage Networks tar.gz files

Ensure that the required unzipped Nuage software files (QCOW2, OVA, and Linux Package files) are available for the components to be installed. Use one of the two methods below.

Method One: Automatically

Run the command below, replacing [zipped_directory] and [nuage_unzipped_files_dir] with the actual paths:

```
metroae tools unzip images [zipped_directory]
[nuage_unzipped_files_dir]
```

Note: After completing setup you will customize for your deployment, and you'll need to add this unzipped files directory path to `common.yml`.

Method Two: Manually

Alternatively, you can create the directories under the [nuage_unzipped_files_dir] directory and manually copy or unzip the appropriate files to that location. MetroAE uses `find` to locate the files under [nuage_unzipped_files_dir], so the precise location under that directory is not significant. For reference, the automatic unzip in Method One puts files in the following locations:

```
<nuage_unzipped_files_dir>/vsd/qcow2/
<nuage_unzipped_files_dir>/vsd/ova/ (for VMware)
<nuage_unzipped_files_dir>/vsc/
<nuage_unzipped_files_dir>/vrs/el7/
<nuage_unzipped_files_dir>/vrs/ul16_04/
<nuage_unzipped_files_dir>/vrs/vmware/
<nuage_unzipped_files_dir>/vrs/hyperv/
<nuage_unzipped_files_dir>/vstat/
<nuage_unzipped_files_dir>/vns/nsg/
<nuage_unzipped_files_dir>/vns/nuh/
<nuage_unzipped_files_dir>/vns/util/
```

Note: After completing setup you will customize for your deployment, and you'll need to add this unzipped files directory path to `common.yml`.

Next Steps

After you've set up your environment you're ready to customize for your topology.

You May Also Be Interested in

Encrypting Sensitive Data in MetroAE Deploying Components in AWS

Questions, Feedback, and Contributing

Get support via the forums on the MetroAE site.

Ask questions and contact us directly at devops@nuagenetworks.net.

Report bugs you find and suggest new features and enhancements via the GitHub Issues feature.

You may also contribute to MetroAE by submitting your own code to the project.

Customizing Components for a Deployment

Prerequisites / Requirements

If you have not already set up your MetroAE Host environment, see SETUP.md before proceeding.

What is a Deployment?

Deployments are component configuration sets. You can have one or more deployments in your setup. When you are working with the MetroAE container, you can find the deployment files under the data directory you specified during `metroae container setup`. For example, if you specified `/tmp` as your data directory path, `metroae container setup` created `/tmp/metroae_data` and copied the default deployment to `/tmp/metroae_data/deployments`. When you are working with MetroAE from a workspace you created using `git clone`, deployments are stored in the workspace directory `nuage-metroae/deployments/<deployment_name>/`. In both cases, the files within each deployment directory describe all of the components you want to install or upgrade.

If you issue:

```
./metroae install everything
```

The files under nuage-metroae/deployments/default will be used to do an install.

If you issue:

```
./metroae install everything mydeployment
```

The files under `nuage-metroae/deployments/mydeployment` will be used to do an install. This allows for different sets of component definitions for various projects.

Each time you issue MetroaÆ, the inventory will be completely rebuilt from the deployment name specified. This will overwrite any previous inventory, so it will reflect exactly what is configured in the deployment that was specified.

Customize Your Own Deployment

You can customize the deployment files for your workflows using any of the following methods:

- Edit the files in the `default` deployment
- Edit the files in a new deployment directory that you have created
- Run `run_wizard.py` to let MetroAE create or edit your deployment
- Create your deployment using the MetroAE spreadsheet (CSV file)

Based on your network topology and the specific components you plan on deploying, you will configure several files. Setting deployment files correctly ensures that when you subsequently execute workflows they configure components as intended. Precise syntax is crucial.

When a workflow is executed, each deployment file is validated against a data schema which ensures that all required fields are present and in the correct syntax. These schemas are located in the schemas/ directory. They follow the json-schema.org standard.

You have the option of configuring the default deployment files provided in the deployments/default/ sub-directory or creating your own sub-directories under the deployments/ directory. You

can find examples of deployment files for different deployments in the examples/ directory. Unless you specify a different deployment sub-directory name, the default deployment is used when a workflow is executed. This method allows MetroAE to support many deployments (different configurations) in parallel and the ability to switch between them as required. See below for the supported deployment files that you can specify in a deployments sub-directory.

Note that you can edit the deployment files manually at any time. MetroAE comes with its own wizard for automating the creation and editing of deployment files when you are working with a clone of the nuage-metroae repo. To start the wizard:

```
python run_wizard.py
```

You can use the wizard to setup your MetroAE environment, if you wish. Or you can skip the setup step and go directly to the creation and editing of your deployment.

You can also use the MetroAE spreadsheet to create your deployment. You can find the MetroAE CSV template in `deployment_spreadsheet_template.csv`. When you finish customizing the spreadsheet, save it to a CSV file of your own naming. Then you can either convert the CSV directly to a deployment using this syntax:

```
convert_csv_to_deployment.py deployment_spreadsheetname.csv
your_deployment_name
```

or you can let `metroae` do the conversion for you by specifying the name of the CSV file instead of the name of your deployment:

```
metroae deployment_spreadsheet_name.csv
```

This will create or update a deployment with the same name as the CSV file - without the extension.

The deployment files that can be configured using the wizard, spreadhseet, or edited manually are listed, below.

`common.yml`

`common.yml` contains the common configuration parameters for the deployment that are used for all components and workflows. This file is always required for any workflow.

Notes for `common.yml`

> `nuage_unzipped_files_dir` is a required parameter that points to the location of the binary image and package files used for Install and Upgrade workflows. We require that you have obtained those files from Nuage/Nokia online customer support prior to running MetroAE. When running MetroAE in a container, this parameter should *not* begin with a '/' and be set equal to the relative path from the images path you configured when you installed the container. For example, if you set the images path for the container to `/home/username/images` and you will unzip the files you are going to use into `/home/username/images/6.0.1`, set `nuage_unzipped_files_dir` to `6.0.1`. MetroAE will concatenate the two paths to access your files. If, however, you are operating without the container and, instead, cloned the nuage-metroae repo to your disk, set `nuage_unzipped_files_dir` to the full, absolute path to the images directory, `/home/username/images/6.0.1` in the example, above.
>
> For best performance, `ntp_server_list` should include the same servers that the target servers will be using. This will help to ensure NTP synchronization.

`credentials.yml`

`credentials.yml` contains user credentials for command-line access to individual components, e.g. VSD, authentication parameters for HTTP and hypervisor access, and the passwords for internal VSD services. All of the credentials in this file are

optional. MetroAE will use default parameters when these are not specified. This file does not require modification if you are not using non-default credentials.

`nsgvs.yml`

`nsgvs.yml` contains the definition of the NSGvs to be operated on in this deployment. This file should be present in your deployment only if you are specifying NSGvs. If not provided, no NSGvs will be operated on. This file is of yaml list type and may contain as many NSGv definitions as required.

Notes for `nsgvs.yml`

ZFB support is included in the nsgv schema and supporting files. In the beta release of MetroAE 3, however, ZFB is not supported.

`upgrade.yml`

`upgrade.yml` contains the configuration parameters for an upgrade workflow. This file is only required when performing an upgrade.

`vcins.yml`

`vcins.yml` contains the definition of the VCINs to be operated on in this deployment. This file should be present in your deployment only if you are specifying VCINs. If not provided, no VCINs will be operated on. This file is of yaml list type and may contain as many VCIN definitions as required.

`vnsutils.yml`

`vnsutils.yml` contains the definition of the VNSUTILs to be operated on in this deployment. This file should be present in your deployment only if you are specifying VNSUTILs. If not provided, no VNSUTILs will be operated on. This file is of yaml list type and may contain as many VNSUTILs definitions as you require, though one is usually sufficient.

nuhs.yml

`nuhs.yml` contains the definition of the NUHs to be operated on in this deployment. This file should be present in your deployment only if you are specifying NUHs. If not provided, no NUHs will be operated on. This file is of yaml list type and may contain as many NUHs definitions as you require, though one is usually sufficient.

webfilters.yml

`webfilters.yml` contains the definition of the Webfilters to be operated on in this deployment. This file should be present in your deployment only if you are specifying Webfilters. If not provided, no Webfilters will be operated on. This file is of yaml list type and may contain as many Webfilters definitions as you require, though one is usually sufficient.

vrss.yml

`vrss.yml` contains the definition of the VRSs to be operated on in this deployment. This file should be present in your deployment only if you are specifying VRSs. If not provided, no VRSs will be operated on. This file is of yaml list type and may contain as many VRS defintions as you require.

vscs.yml

`vscs.yml` contains the definition of the VSCs to be operated on in this deployment. This file should be present in your deployment only if you are specifying VSCs. If not provided, no VSCs will be operated on. This file is of yaml list type and may contain as many VSC definitions as you require.

vsds.yml

`vsds.yml` contains the definition of the VSDs to be operated on in this deployment. This file should be present in your deployment only if you are specifying VSDs. If not provided, no

VSDs will be operated on. This file is of yaml list type and must contain either 0, 1, 3, or 6 VSD definitions. 1 VSD is for stand-alone VSD operation. 3 VSDs are for a single cluster operation. 6 VSDs are defined for active-standby, geo-redundant operation.

Notes on `vsds.yml` for active-standby, geo-redundant deployment and upgrade

When installing or upgrading an active-standby, geo-redundant cluster, all 6 VSDs must be defined in the `vsds.yml` file in your deployment. The first 3 VSDs are assumed to be `active` and the second 3 VSDs are assumed to be `standby`.

`vstats.yml`

`vstats.yml` contains the definition of the VSTATs (VSD Statistics) to be operated on in this deployment. This file should be present in your deployment only if you are specifying VSTATs. If not provided, no VSTATs will be operated on. This file is of yaml list type. If it contains exactly 3 VSTAT definitions, a cluster installation or upgrade will be executed. Any other number of VSTAT definitions will result in 1 or more stand-alone VSTATs being installed or upgraded.

VSD Disk Performance Testing

You can use MetroAE to verify that your VSD setup has sufficient disk performance (IOPS). By default, the disk performance test will run at the beginning of the VSD deploy step, prior to installing the VSD software. The parameters that you can use to control the operation of the test are available in 'common.yml'. You can skip the test, specify the total size of all the files used in the test, and modify the minimum threshold requirement in IOPS for the test. Note that to minimize the effects of file sstem caching, the total file size must exceed the total RAM on the VSD. If MetroAE finds that the test is enabled and the disk performance is below the threshold, an error will occur and installation will stop. The default values that are provided for the test are recommended for best VSD performance in most cases.

Your specific situation may require different values or to skip the test entirely.

In addition to the automatic execution that takes place in the VSD deploy step, you can run the VSD disk performance test at any time using `metroae vsd test disk`.

Enabling post-installation security features

You can use MetroAE to enable optional post-installation security features to 'harden' the installation. Your deployment contains a number of optional variables that can be used to accomplish this. These variables are described, below. For more detail, please see the Nuage VSP Install Guide.

`vsds.yml`

`ca_certificate_path` is an optional parameter that points to the location of the certificate of the signing authority.

`certificate_path` is an optional parameter that points to the location of the certificate pem file for the vsd.

`intermediate_certificate_path` is an optional parameter that points to the location of the chain certificate.

`failed_login_attempts` is an optional parameter to set the number of failed login attempts.

`failed_login_lockout_time` is an optional parameter that set the lockout time after reaching the max number of failed login attempts.

`advanced_api_access_logging` is an optional parameter to enable adding custom header to access log.

`tls_version` is an optional parameter to set the minimum TLS version to use.

`vscs.yml`

`ca_certificate_path` is an optional parameter that points to the location of the certificate of the signing authority.

`certificate_path` is an optional parameter that points to the location of the certificate pem file for the vsc.

`private_key_path` is an optional parameter that points to the location of the certificate private key pem file for the vsc.

`ejabberd_id` is an optional parameter that defines the ejabberd username used to when creating the certificate.

`vrss.yml`

`ca_certificate_path` is an optional parameter that points to the location of the certificate of the signing authority.

`certificate_path` is an optional parameter that points to the location of the certificate pem file for the vrs.

`private_key_path` is an optional parameter that points to the location of the certificate private key pem file for the vrs.

`vnsutils`

Currently post install security hardening is not supported for VNSUTILs(proxy) using MetroAE. If using custom certificates, they need to copied into /opt/proxy/config/keys and supervisord needs to be restarted.

Operating using the Default Deployment

The Default deployment is provided as a starting place for your workflows. It is located in a subdirectory named `Default` within the Deployments directory. You can operate MetroAE by simply editing the contents of the Default deployment. Follow these steps:

1. Edit the files for the components you will operate on that already exist in the Default subdirectory

2. Remove the files for the components you will *not* operate on that already exist in the Default subdirectory
3. To add components to the default directory, or replace ones you previously deleted, copy and then edit files found in the examples directory.

Adding a new Deployment

To create a new deployment:

1. Create a new subdirectory under deployments.
2. Copy the contents of an existing deployment subdirectory, e.g. deployments/default, to the new subdirectory.
3. Edit the files in the new subdirectory.
4. If you'd like to add components that are not included, you can copy a *blank* file from the examples directory.

Hosting your deployment files outside of the repo

When you are contributing code, or pulling new versions of MetroAE quite often, it may make sense to host your variable files in a separate directory outside of `nuage-metroae/deployments/`. A deployment directory in any location can be specified instead of a deployment name when issuing the `metroae` command.

Generating example deployment configuration files

A sample of the deployment configuration files are provided in the deployments/default/ directory and also in examples/. If these are overwritten or deleted or if a "no frills" version of the files with only the minimum required parameters are desired, they can be generated with the following command:

```
metroae tools generate example --schema <schema_filename> [--
no-comments]
```

This will print an example of the deployment file specified by <schema_filename> under the schemas/ directory to the screen. The optional `--no-comments` will print the minimum required parameters (with no documentation).

Example:

```
metroae tools generate example --schema vsds >
deployments/new/vsds.yml
```

Creates an example vsds configuration file under the “new” deployment.

Running MetroAE using a Proxy VM

MetroAE version 3.4.0 onwards supports using a proxy VM for deployment of VSC, VSD and VSTATS(ES). In this configuration, the host on which MetroAE is run on, does not have direct access to the VSD, VSC and VSTATS. In such a case, a proxy VM can be set between the host and individual components which has access to all the components. To operate in such a manner, user can edit their `deployments\<deployment_name>\common.yml` parameters to indicate the proper `ssh_proxy_username` and `ssh_proxy_host` and then run the MetroAE commands as usual.

Next Steps

The next step is to deploy your components. See DEPLOY.md for guidance.

Questions, Feedback, and Contributing

Get support via the forums on the MetroAE site.
Ask questions and contact us directly at
devops@nuagenetworks.net.

Report bugs you find and suggest new features and
enhancements via the GitHub Issues feature.

You may also contribute to MetroAE by submitting your own code
to the project.

Encrypting Sensitive Data in MetroAE

You can safeguard sensitive data in MetroAE by encrypting files with MetroAE's encryption tool. See the steps below for instructions on how to encrypt `credentials.yml`. It uses Ansible's vault encoding in the background. More details about the vault feature can be found in documentation provided by Ansible.

1. Create the credentials file to be encrypted

In your MetroAE deployment folder, create or edit the `credentials.yml` to store credentials required for various Nuage components. This file will be encrypted.

2. Encrypt `credentials.yml`

To encrypt `credentials.yml`, run the following command:

```
metroae tools encrypt credentials [deployment_name]
```

The default deployment name is `default` if not specified. This command will prompt for master passcode to encrypt the file and will also prompt for confirming passcode. Note: All user comments and unsupported fields in the credentials file will be lost.

3. Running MetroAE with encrypted credentials

While running MetroAE commands you can supply the MetroAE passcode via prompt or by setting an environment variable

```
metroae <workflow> <component> [action] [deployment_name]
```

This command prompts you to enter the master passcode that you used to encrypt the credentials file. Alternatively, if you have the environment variable METROAE_PASSWORD set to the right passcode, MetroAE does not prompt for the passcode.

Questions, Feedback, and Contributing

Get support via the forums on the MetroAE site.
Ask questions and contact us directly at devops@nuagenetworks.net.

Report bugs you find and suggest new features and enhancements via the GitHub Issues feature.

You may also contribute to MetroAE by submitting your own code to the project.

Deploying Components with MetroAE

You can execute MetroAE workflows to perform the following installations:

- Deploy All Components
- Deploy Individual Modules
- Install a Particular Role or Host
- Copy QCOW2 files
- Deploy Standby Cluster
- Setup a Health Monitoring Agent
- Debugging

Prerequisites / Requirements

Before deploying any components, you must have previously set up your Nuage MetroAE environment and customized the environment for your target platform.

Make sure you have unzipped copies of all the Nuage Networks files you are going to need for installation or upgrade. These are generally distributed as `*.tar.gz` files that are downloaded by you from Nokia OLCS/ALED. There are a few ways you can use to unzip:

- If you are running MetroAE via a clone of the nuage-metroae repo, you can unzip these files by using the nuage-unzip shell script `nuage-unzip.sh` which will place the files in subdirectories under the path specified for the `nuage_unzipped_files_dir` variable in `common.yml`.

  If you are running MetroAE via the MetroAE container, you can unzip these files using the metroae command. During the setup, you were promoted for the location of an data

directory on the host. This data directory is mounted in the container as `/data`. Therefore, for using the unzip action, you must 1) copy your tar.gz files to a directory under the directory you specified at setup time and 2) you must specify a container-relative path on the unzip command line. For example, if you specified the data directory as `/tmp`, setup created the directory `/data/metroae_data` on your host and mounted that same directory as `/data` in the container. Assuming you copied your tar.gz files to `/tmp/metroae_data/6.0.1` on the docker host, your unzip command line would be as follows: `metroae tools unzip images /data/6.0.1/ /data/6.0.1`.

You can also unzip the files manually and copy them to their proper locations by hand. For details of this process, including the subdirectory layout that MetroAE expects, see SETUP.md.

Use of MetroAE Command Line

MetroAE can perform a workflow using the command-line tool as follows:

```
metroae <workflow> <componment> [deployment] [options]
```

`workflow`: Name of the workflow to perform, e.g. 'install' or 'upgrade'. Supported workflows can be listed with --list option.

`component`: Name of the component to apply the workflow to, e.g. 'vsds', 'vscs', 'everything', etc.

`deployment`: Name of the deployment directory containing configuration files. See CUSTOMIZE.md

`options`: Other options for the tool. These can be shown using --help. Also, any options not directed to the metroae tool are passed to Ansible.

The following are some examples:

```
metroae install everything
```

Installs all components described in deployments/default/.

```
metroae destroy vsds east_network
```

Takes down only the VSD components described by deployments/east_network/vsds.yml. Additional output will be displayed with 3 levels of verbosity.

Deploy All Components

MetroAE workflows operate on components as you have defined them in your deployment. If you run a workflow for a component not specified, the workflow skips all tasks associated with that component and runs to completion without error. Thus, if you run the `install everything` workflow when only VRS configuration is present, the workflow deploys VRS successfully while ignoring the tasks for the other components not specified. Deploy all specified components with one command as follows:

```
metroae install everything
```

Note: `metroae` is a shell script that executes `ansible-playbook` with the proper includes and command line switches. Use `metroae` (instead of `ansible-playbook`) when running any of the workflows provided herein.

Deploy Individual Modules

MetroAE offers modular execution models in case you don't want to deploy all components together. See modules below.

MODULE	COMMAND	DESCRIPTION

MODULE	COMMAND	DESCRIPTION
VCS	`metroae install vsds`	Installs VSD components
VNS	`metroae install vscs`	Installs VSC components

Install a Particular Role or Host

MetroAE has a complete library of workflows, which are directly linked to each individual role. You can limit your deployment to a particular role or component, or you can skip steps you are confident need not be repeated. For example, to deploy only the VSD VM-images and get them ready for VSD software installation, run:

```
metroae install vsds predeploy
```

To limit your deployment to a particular host, just add `--limit` parameter:

```
metroae install vsds predeploy --limit "vsd1.example.com"
```

VSD predeploy can take a long time. If you are **vCenter user** you may want to monitor progress via the vCenter console.

Note: If you have an issue with a VM and would like to reinstall it, you must destroy it before you replace it. Otherwise, the install will find the first one still running and skip the new install.

Copy QCOW2 Files before Deployment

When installing or upgrading in a KVM environment, MetroAE copies the QCOW2 image files to the target file server during the predeploy phase. As an option, you can pre-position the qcow2 files for all the components by running copy_qcow2_files. This

gives the ability to copy the required images files first and then run install or upgrade later.

When QCOW2 files are pre-positioned, you must add a command-line variable, 'skip_copy_images', to indicate that copying QCOW2 files should be skipped. Otherwise, the QCOW2 files will be copied again. An extra-vars 'skip_copy_images' needs to be passed on the command line during the deployment phase to skip copying of the image files again. For example, to pre-position the QCOW2 images, run:

```
metroae tools copy qcow
```

Then, to skip the image copy during the install:

```
metroae install everything --extra-vars skip_copy_images=True
```

Deploy the Standby Clusters

MetroAE can be used to bring up the Standby VSD and VSTAT(ES) cluster in situations where the active has already been deployed. This can be done using the following commands. For VSD Standby deploy

```
metroae install vsds standby predeploy
metroae install vsds standby deploy
```

For Standby VSTATs(ES)

```
metroae install vstats standby predeploy
metroae install vstats standby deploy
```

Setup a Health Monitoring Agent

A health monitoring agent can be setup on compatible components during the deploy step. Currently this support includes the Zabbix agent. An optional parameter `health_monitoring_agent` can be specified on each component in the deployment files to enable setup. During each component deploy step when enabled, the agent will be downloaded, installed and configured to be operational. The agent can be installed separately, outside of the deploy role, using the following command:

```
metroae health monitoring setup
```

Debugging

By default, ansible.cfg tells ansible to log to ./ansible.log.

Ansible supports different levels of verbosity, specified with one of the following command line flags: -v -vv -vvv -vvvv

More letters means more verbose, usually for debugging. The highest level, -vvvv, provides SSH connectivity information.

Running individual workflows is also useful for debugging. For example, `vsd_predeploy`, `vsd_deploy`, and `vsd_postdeploy`.

If you would like to remove an entire deployment, or individual components, and start over, see DESTROY.md for details.

Next Steps

After you have successfully deployed Nuage Networks VSP components, you may want to upgrade to a newer version at some point in the future. See UPGRADE_SA.md for standalone deployments and UPGRADE_HA.md for clustered deployments.

Questions, Feedback, and Contributing

Get support via the forums on the MetroAE site.
Ask questions and contact us directly at
devops@nuagenetworks.net.

Report bugs you find and suggest new features and
enhancements via the GitHub Issues feature.

You may also contribute to MetroAE by submitting your own code
to the project.

Removing Components with MetroAE

To remove a deployment:

Prerequisites / Requirements

Use this procedure when you have previously deployed VSP components and would like to remove all or some of the components and start over.

1. Check Existing Deployment

Ensure that all components you wish to be destroyed are specified under your deployment. See CUSTOMIZE.md for details about specifying components in a deployment. If you have already used a deployment configuration to do a deploy or upgrade, the existing configuration does not need to be changed to destroy said deployment.

2. Remove Components

You have the option of removing the entire deployment or only specified individual components.

Remove All Components

Remove the entire existing deployment with one command as follows:

```
metroae destroy everything
```

Note: you may alternate between `metroae install everything` and `metroae destroy everything` as needed.

Remove Individual Components

Alternatively, you can remove individual components (VSD, VSC, VRS, etc) as needed. See VSC example below for details.

Example Sequence for VSC:

Configure components under your deployment
Run `metroae install everything` to deploy VSD, VSC, VRS, etc.
Discover that something needs to be changed in the VSCs
Run `metroae destroy vscs` to tear down just the VSCs
Edit `vscs.yml` in your deployment to fix the problem
Run `metroae install vsc predeploy`, `metroae install vscs deploy`, and `metroae install vscs postdeploy` to get the VSCs up and running again.

Questions, Feedback, and Contributing

Get support via the forums on the MetroAE site.
Ask questions and contact us directly at devops@nuagenetworks.net.

Report bugs you find and suggest new features and enhancements via the GitHub Issues feature.

You may also contribute to MetroAE by submitting your own code to the project.

MetroAE Config

MetroAE Config is a template-driven VSD configuration tool. It utilizes the VSD API along with a set of common Feature Templates to create configuration in the VSD. The user will create a yaml or json data file with the necessary configuration parameters for the particular feature and execute simple CLI commands to push the configuration into VSD.

MetroAE Config is available via the MetroAE container. Once pulled and setup, additional documentation will be available.

Configuration Engine Installation

MetroAE Configuration engine is provided as one of the capabilities of the MetroAE Docker container. The installation of the container is handled via the metroae script. Along with the configuration container we also require some additional data.

On a host where the configuration engine will be installed the following artifacts will be installed:

- Docker container for configuration
- Collection of configuration Templates
- VSD API Specification

System Requirements

- Operating System: RHEL/Centos 7.4+
- Docker: Docker Engine 1.13.1+
- Network Access: Internal and Public
- Storage: At least 800MB

Operating system

The primary requirement for the configuration container is Docker Engine, however the installation, container operation and functionality is only tested on RHEL/Centos 7.4. Many other operating systems will support Docker Engine, however support for these is not currently provided and would be considered experimental only. A manual set of installation steps is provided for these cases.

Docker Engine

The configuration engine is packaged into the MetroAE Docker container. This ensures that all package and library requirements are self-contained with no other host dependencies. To support this, Docker Engine must be installed on the host. The configuration container requirements, however, are quite minimal. Primarily, the Docker container mounts a local path on the host as a volume while ensuring that any templates and user data are maintained only on the host. The user never needs to interact directly with the container.

Network Access

Currently the configuration container is hosted in an internal Docker registry and the public network as a secondary option, while the Templates and API Spec are hosted only publicly. The install script manages the location of these resources. The user does not need any further information. However, without public network access the installation will fail to complete.

Storage

The configuration container along with the templates requires 800MB of local disk space. Note that the container itself is ~750MB, thus it is recommended that during install a good network connection is available.

User Privileges

The user must have elevated privileges on the host system.

Installation

Execute the following operations on the Docker host:

1. Install Docker Engine

   Various instructions for installing and enabling Docker are available on the Internet. One reliable source of information for Docker CE is hosted here:

   https://docs.docker.com/engine/install/centos/

2. Add the user to the wheel and docker groups on the Docker host.

3. Move or Copy the “metroae” script from the nuage-metroae repo to /usr/bin and set permissions correctly to make the script executable.

4. Switch to the user that will operate "metroae config".

5. Setup the container using the metroae script.

   We are going to pull the image and setup the metro container in one command below. During the install we will be prompted for a location for the container data directory. This is the location where our user data, templates and VSD API specs will be installed and created and a volume mount for the container will be created. However this can occur orthogonally via “pull” and “setup” running at separate times which can be useful depending on available network bandwidth.

   ```
   [caso@metroae-host ~]$ metroae container setup
   ```

   The MetroAE container needs access to your user data. It gets access by internally mounting a directory from the host. We refer to this as the 'data directory’. The data directory is where you will have deployments, templates, documentation, and other useful files. You will be prompted to provide the path to a local directory on the host that will

be mounted inside the container. You will be prompted during setup for this path.

The MetroAE container can be setup in one of the following configurations:

Config only

Deploy only

Both config and deploy

During setup, you will be prompted to choose one of these options.

6. Follow additional insturctions found in the documentaion that is copied to the Docker host during setup.

Complete documentation will be made available in the data directory you specify during setup. The complete documentation includes how to configure your environment, usage information for the tool, and more.

Upgrading a Standalone Deployment with MetroAE

Prerequisites / Requirements / Notes

Before upgrading any components, you must have previously set up your MetroAE environment and customized the upgrade environment for your target platform.

Ensure that you have added upgrade.yml to your deployment and specified `upgrade_from_version` and `upgrade_to_version`. MetroAE uses these values to determine whether it is to perform a patch upgrade, a major upgrade or a minor upgrade. Failure to populate these variables correctly could cause the wrong type of upgrde to be attempted, possibly resulting in an error. If a minor upgrade is treated as a major upgrade, for example, you may get stuck in the turn-on-api step which should not be executed for minor upgrades.

Note that if your existing VSP components were not installed using MetroAE or were installed using a different MetroAE host, you can still use MetroAE to do the upgrade. You must manually copy the MetroAE host user's ssh public key to each of the VSP Linux-based components (VSD, VSTAT, VNSUTIL) and to any KVM-based hypervisors used for VSP components and any KVM-based hypervisors where data-plane endpoints (VRS, NSGv) have been installed. This will allow passwordless ssh between the Ansible host and the Linux nodes in the deployment. Passwordless ssh to these nodes is a requirement for proper MetroAE operation for health and upgrade.

By default, the special enterprise called Shared Infrastructure is created on the VSD. When putting domains in maintenance mode prior to an upgrade, MetroAE skips Shared Infrastructure domains because they cannot be modified.

Patch Upgrade for VSD, AKA in-place upgrade

A patch upgrade is applicable to the VSD cluster when upgrading from one 'u' release to another. A patch upgrade is also referred to as an in-place upgrade. The existing VSDs will remain in service. The migration ISO will be mounted and the migration script will be executed on each VSD. A patch upgrade is:

> Supported beginning in VSD version 5.4.1.
>
> `upgrade_from_version` and `upgrade_to_version` variables must be set to 'u' versions of the same release, e.g. 5.4.1 and 5.4.1u1, 5.4.1u1 and 5.4.1u4, etc.

Note that MetroAE only supports patch upgrades for VSD using the `upgrade_vsds` play. Attempting to do a patch release upgrade via any other method will result in an error.

Example Deployment

For this example, our standalone (SA) deployment consists of:

> one VSD node
>
> one VSC node
>
> VRS instance(s)
>
> one VSTAT (Elasticsearch) node

Upgrading Automatically

If your topology does not include VRS you can upgrade everything with one command. If it does includes VRS you can upgrade everything with two commands. MetroAE also gives you the option of upgrading individual components with a single command for each. If you prefer to have more control over each step in the upgrade process proceed to Upgrading By Individual Steps for instructions.

Upgrade All Components (without VRS)

```
metroae upgrade everything
```

Issuing this workflow will detect if components are clustered (HA) or not and will upgrade all components that are defined in the deployment. This option does not pause until completion to allow VRS(s) to be upgraded. If VRS(s) need to be upgraded, the following option should be performed instead.

Upgrade All Components (with VRS)

```
metroae upgrade beforevrs

( Upgrade the VRS(s) )

metroae upgrade aftervrs
```

Issuing the above workflows will detect if components are clustered (HA) or not and will upgrade all components that are defined in the deployment. This option allows the VRS(s) to be upgraded in-between other components.

Upgrade Individual Components

```
metroae upgrade preupgrade health

metroae upgrade vsds

metroae upgrade vscs beforevrs

( Upgrade the VRS(s) )

metroae upgrade vscs aftervrs

metroae upgrade vstats

metroae upgrade postdeploy

metroae upgrade postupgrade health
```

Issuing the above workflows will detect if components are clustered (HA) or not and will upgrade all components that are

defined in the deployment. This option allows the VRS(s) to be upgraded in-between other components. Performing individual workflows can allow specific components to be skipped or upgraded at different times.

Upgrading By Individual Steps

The following workflows will upgrade each component in individual steps. The steps listed below are only applicable for stand-alone (SA) deployments. Performing an upgrade in this way allows full control of the timing of the upgrade process.

Preupgrade Preparations

1. Run health checks on VSD, VSC and VSTAT.

   ```
   metroae upgrade preupgrade health
   ```

   Check the health reports carefully for any reported errors before proceeding. You can run health checks at any time during the upgrade process.

2. Backup the VSD node database.

   ```
   metroae upgrade sa vsd dbbackup
   ```

   The VSD node database is backed up.

   Troubleshooting: If you experience a failure you can re-execute the command.

   Note MetroAE provides a simple tool for optionally cleaning up the backup files that are generated during the upgrade process. The tool deletes the backup files for both VSD and VSC. There are two modes foe clean-up, the first one deletes all the backups and the second one deletes only the latest backup. By default the tool deletes only the latest backup. If you'd like to clean-up the backup files, you can simply run below commands: Clean up all the backup files: `metroae vsp_upgrade_cleanup -e`

`delete_all_backups=true` Clean up the latest backup files: `metroae vsp_upgrade_cleanup`

Upgrade VSD

1. Power off the VSD node.

   ```
   metroae upgrade sa vsd shutdown
   ```

   VSD is shut down; it is not deleted. (The new node is brought up with the `upgrade_vmname` you previously specified.) You have the option of powering down VSD manually instead.

   Troubleshooting: If you experience a failure you can re-execute the command or power off the VM manually.

2. Predeploy the new VSD node.

   ```
   metroae install vsds predeploy
   ```

   The new VSD node is now up and running; it is not yet configured.

   Troubleshooting: If you experience a failure, delete the new node by executing the command `metroae upgrade destroy sa vsd`, then re-execute the predeploy command. Do NOT run `metroae destroy vsds` as this command destroys the “old” VM which is not what we want to do here.

3. Deploy the new VSD node.

   ```
   metroae upgrade sa vsd deploy
   ```

   The VSD node is upgraded.

   Troubleshooting: If you experience a failure before the VSD install script runs, re-execute the command. If it fails a second time or if the failure occurs after the VSD install script runs, destroy the VMs (either manually or with the command `metroae upgrade destroy sa vsd`) then re-execute

the deploy command. Do NOT run `metroae destroy vsds` for this step.

4. Set the VSD upgrade complete flag.

   ```
   metroae upgrade sa vsd complete
   ```

   The upgrade flag is set to complete.

   Troubleshooting: If you experience a failure, you can re-execute the command.

5. Log into the VSD and verify the new version.

Upgrade VSC

This example is for one VSC node. If your topology has more than one VSC node, proceed to the **Upgrade VSC Node One** section of UPGRADE_HA.md and follow those instructions through to the end.

1. Run VSC health check (optional).

   ```
   metroae upgrade sa vsc health -e
   report_filename=vsc_preupgrade_health.txt
   ```

   You performed health checks during preupgrade preparations, but it is good practice to run the check here as well to make sure the VSD upgrade has not caused any problems.

2. Backup and prepare the VSC node.

   ```
   metroae upgrade sa vsc backup
   ```

   Troubleshooting: If you experience failure, you can re-execute the command.

3. Deploy VSC.

   ```
   metroae upgrade sa vsc deploy
   ```

   The VSC is upgraded.

Troubleshooting: If you experience a failure, you can re-execute the command. If it fails a second time, manually copy a valid .tim file to the VSC to affect the deployment. If that fails, deploy a new VSC using the old version, or recover the VM from a backup. You can use MetroAE for the deployment (vsc_predeploy, vsc_deploy, vsc_postdeploy…).

4. Run VSC postdeploy.

   ```
   metroae upgrade sa vsc postdeploy
   ```

   VSC upgrade is complete.

   Troubleshooting: If you experience a failure, you can re-execute the command. If it fails a second time, manually copy a valid .tim file to the VSC to affect the deployment. If that fails, deploy a new VSC using the old version, or recover the VM from a backup. You can use MetroAE for the deployment (vsc_predeploy, vsc_deploy, vsc_postdeploy…).

Upgrade VRS

Upgrade your VRS(s) and then continue with this procedure. Do not proceed without completing this step.

Upgrade VSTAT

Our example includes a VSTAT node. If your topology does not include one, proceed to *Finalize the Upgrade* below.

1. Run VSTAT health check (optional).

   ```
   metroae upgrade sa vstat health -e
   report_filename=vstat_preupgrade_health.txt
   ```

   You performed health checks during preupgrade preparations, but it is good practice to run the check here as well to make sure the VSD upgrade has not caused any problems.

2. Prepare the VSTAT node for upgrade.

```
metroae upgrade sa vstat prep
```

Sets up SSH and disables stats collection.

3. Upgrade the VSTAT node.

```
metroae upgrade sa vstat inplace
```

Performs an in-place upgrade of the VSTAT.

4. Complete VSTAT upgrade and perform post-upgrade checks.

```
metroae upgrade sa vstat wrapup
```

Completes the upgrade process, renables stats and performs a series of checks to ensure the VSTAT is healthy.

Finalize the Upgrade

1. Finalize the settings.

```
metroae upgrade postdeploy
```

The final steps for the upgrade are executed.

Troubleshooting: If you experience a failure, you can re-execute the command.

2. Run a health check.

```
metroae upgrade postupgrade health
```

Health reports are created that can be compared with the ones produced during preupgrade preparations. Investigate carefully any errors or discrepancies.

Questions, Feedback, and Contributing

Get support via the forums on the MetroAE site.
Ask questions and contact us directly at devops@nuagenetworks.net.

Report bugs you find and suggest new features and enhancements via the GitHub Issues feature.

You may also contribute to MetroAE by submitting your own code to the project.

Upgrading a Clustered Deployment with MetroAE

Prerequisites / Requirements / Notes

Before upgrading any components, you must have previously set up your MetroAE environment and customized the upgrade environment for your target platform.

Ensure that you have added upgrade.yml to your deployment and specified `upgrade_from_version` and `upgrade_to_version`. MetroAE uses these values to determine whether it is to perform a patch upgrade, a major upgrade or a minor upgrade. Failure to populate these variables correctly could cause the wrong type of upgrde to be attempted, possibly resulting in an error. If a minor upgrade is treated as a major upgrade, for example, you may get stuck in the turn-on-api step which should not be executed for minor upgrades.

Note that if your existing VSP components were not installed using MetroAE or were installed using a different MetroAE host, you can still use MetroAE to do the upgrade. You must manually copy the MetroAE host user's ssh public key to each of the VSP Linux-based components (VSD, VSTAT, VNSUTIL) and to any KVM-based hypervisors used for VSP components and any KVM-based hypervisors where data-plane endpoints (VRS, NSGv) have been installed. This will allow passwordless ssh between the Ansible host and the Linux nodes in the deployment. Passwordless ssh to these nodes is a requirement for proper MetroAE operation for health and upgrade.

By default, the special enterprise called Shared Infrastructure is created on the VSD. When putting domains in maintenance mode prior to an upgrade, MetroAE skips Shared Infrastructure domains because they cannot be modified.

Patch Upgrade for VSD, AKA in-place upgrade

A patch upgrade is applicable to the VSD cluster when upgrading from one 'u' release to another. A patch upgrade is also referred to as an in-place upgrade. The existing VSDs will remain in service. The migration ISO will be mounted and the migration script will be executed on each VSD. A patch upgrade is:

> Supported beginning in VSD version 5.4.1.
>
> `upgrade_from_version` and `upgrade_to_version` variables must be set to 'u' versions of the same release, e.g. 5.4.1 and 5.4.1u1, 5.4.1u1 and 5.4.1u4, etc.

Note that MetroAE only supports patch upgrades for VSD using the `upgrade_vsds` play. Attempting to do a patch release upgrade via any other method will result in an error.

Active/Standby cluster upgrade

You can use MetroAE to upgrade Active/Standby VSD clusters, also known as geo-redundant clusters. You can also use MetroAE to upgrade Active/Standby VSTAT (ES) clusters. The support for this is built into the `upgrade_everything`, `upgrade_vsds`, and `upgrade_vstats` plays. A step-by-step manual procedure is supported, but is not documented here. See Upgrading By Individual Steps for more information.

Example Deployment

For this example, our clustered (HA) deployment consists of:

> three VSD nodes in a cluster
>
> two VSC nodes
>
> VRS instance(s)
>
> one VSTAT (Elasticsearch) node

Upgrading Automatically

If your upgrade plans do not include upgrading VRSs or other dataplane components, you can upgrade everything with one command. If your upgrade plans do include VRSs or other dataplane components, you can upgrade everything with two commands. MetroAE also gives you the option of upgrading all instances of a component type, e.g. VSC, with a single command for each component type. If you prefer to have more control over each step in the upgrade process proceed to Upgrading By Individual Steps for instructions.

Upgrade All Components including Active/Standby clusters (does not pause for external VRS/dataplane upgrade)

```
metroae upgrade everything
```

Issuing this workflow will detect if components are clustered (HA) or not and will upgrade all components that are defined in the deployment. This option does not pause until completion to allow VRSs and other dataplane components to be upgraded. If dataplane components need to be upgraded, the following option should be performed instead.

Upgrade All Components including Active/Standby clusters (includes pause for VRS)

```
metroae upgrade beforevrs

( Upgrade the VRSs andn other dataplane components )

metroae upgrade aftervrs
```

Issuing the above workflows will detect if components are clustered (HA) or not and will upgrade all components that are defined in the deployment. This option allows the VRSs and other dataplane components to be upgraded between other components.

Upgrade Individual Components including Active Standby clusters

```
metroae upgrade preupgrade health

metroae upgrade vsds

metroae upgrade vscs beforevrs

( Upgrade the VRS(s) )

metroae upgrade vscs aftervrs

metroae upgrade vstats

metroae upgrade postdeploy

metroae upgrade postupgrade health
```

Issuing the above workflows will detect if components are clustered (HA) or not and will upgrade all components that are defined in the deployment. This option allows the VRS(s) to be upgraded in-between other components. Performing individual workflows can allow specific components to be skipped or upgraded at different times.

Upgrading By Individual Steps not including Active/Standby clusters

The following workflows will upgrade each component in individual steps. Performing an upgrade in this way allows full control of the timing of the upgrade process and provides opportunities for you to add custom steps to the overall process. Note that the steps listed below are only applicable for clustered (HA) deployments. Active/Standby cluster upgrades require additional steps that are not documented here. See the component playbooks `src\playbooks\with_build\upgrade_vsds.yml` and `src\playbooks\with_build\upgrade_vstats.yml` for the full list of steps.

Preupgrade Preparations

1. Run health checks on VSD, VSC and VSTAT.

   ```
   metroae upgrade preupgrade health
   ```

   Check the health reports carefully for any reported errors before proceeding. You can run health checks at any time during the upgrade process.

2. Backup the database and decouple the VSD cluster.

   ```
   metroae upgrade ha vsd dbbackup
   ```

   `vsd_node1` has been decoupled from the cluster and is running in standalone (SA) mode.

   Troubleshooting: If you experience a failure, recovery depends on the state of `vsd_node1`. If it is still in the cluster, you can re-execute the command. If not, you must redeploy `vsd_node1` from a backup or otherwise recover.

   Note MetroAE provides a simple tool for optionally cleaning up the backup files that are generated during the upgrade process. The tool deletes the backup files for both VSD and VSC. There are two modes foe clean-up, the first one deletes all the backups and the second one deletes only the latest backup. By default the tool deletes only the latest backup. If you'd like to clean-up the backup files, you can simply run below commands: Clean up all the backup files: `metroae upgrade cleanup -e delete_all_backups=true` Clean up the latest backup files: `metroae upgrade cleanup`

Upgrade VSD

1. Power off VSD nodes two and three.

   ```
   metroae upgrade ha vsd shutdown23
   ```

`vsd_node2` and `vsd_node3` are shut down; they are not deleted. The new nodes are brought up with the upgrade vmnames you previously specified. You have the option of powering down VSDs manually instead.

Troubleshooting: If you experience a failure you can re-execute the command or power off the VM manually.

2. Predeploy new VSD nodes two and three.

   ```
   metroae upgrade ha vsd predeploy23
   ```

   The new `vsd_node2` and `vsd_node3` are now up and running; they are not yet configured.

   Troubleshooting: If you experience a failure, delete the newly-created nodes by executing the command `metroae upgrade destroy ha vsd23`, then re-execute the predeploy command. Do NOT run `metroae destroy vsds` as this command destroys the “old” VM which is not what we want to do here.

3. Deploy new VSD nodes two and three.

   ```
   metroae upgrade ha vsd deploy23
   ```

   The VSD nodes have been upgraded.

   Troubleshooting: If you experience a failure before the VSD install script runs, re-execute the command. If it fails a second time or if the failure occurs after the VSD install script runs, destroy the VMs (either manually or with the command `metroae upgrade destroy ha vsd23`) then re-execute the deploy command. Do NOT run `metroae destroy vsds` as this command destroys the “old” VM.

4. Power off VSD node one.

   ```
   metroae upgrade ha vsd shutdown1
   ```

   `vsd_node1` shuts down; it is not deleted. The new node is brought up with the `upgrade_vmname` you previously

specified. You have the option of powering down VSD manually instead.

Troubleshooting: If you experience a failure you can re-execute the command or power off the VM manually.

5. Predeploy the new VSD node one.

   ```
   metroae upgrade ha vsd predeploy1
   ```

   The new VSD node one is now up and running; it is not yet configured.

   Troubleshooting: If you experience a failure, delete the newly-created node by executing the command `metroae upgrade destroy ha vsd1`, then re-execute the predeploy command. Do NOT run `vsd_destroy` as this command destroys the "old" VM.

6. Deploy the new VSD node one.

   ```
   metroae upgrade ha vsd deploy1
   ```

   All three VSD nodes are upgraded.

   Troubleshooting: If you experience a failure before the VSD install script runs, re-execute the command. If it fails a second time or if the failure occurs after the VSD install script runs, destroy the VMs (either manually or with the command `metroae upgrade destroy ha vsd1`) then re-execute the deploy command. Do NOT run `metroae destroy vsds` as this command destroys the "old" VM.

7. Set the VSD upgrade complete flag.

   ```
   metroae upgrade ha vsd complete
   ```

   The upgrade flag is set to complete.

   Troubleshooting: If you experience a failure, you can re-execute the command.

8. Log into the VSDs and verify the new versions.

Upgrade VSC Node One

1. Run VSC health check (optional).

   ```
   metroae upgrade ha vsc health -e
   report_filename=vsc_preupgrade_health.txt
   ```

   You performed health checks during preupgrade preparations, but it is good practice to run the check here as well to make sure the VSD upgrade has not caused any problems.

2. Backup and prepare VSC node one.

   ```
   metroae upgrade ha vsc backup1
   ```

   Troubleshooting: If you experience a failure, you can re-execute the command.

3. Deploy VSC node one.

   ```
   metroae upgrade ha vsc deploy1
   ```

   VSC node one has been upgraded.

   Troubleshooting: If you experience a failure, you can re-execute the command. If it fails a second time, manually copy (via scp) the .tim file, bof.cfg, and config.cfg (that were backed up in the previous step), to the VSC. Then reboot the VSC.

4. Run postdeploy for VSC node one.

   ```
   metroae upgrade ha vsc postdeploy1
   ```

   One VSC is now running the **old** version, and one is running the **new** version.

   Troubleshooting: If you experience a failure, you can re-execute the command. If it fails a second time, manually copy (via scp) the .tim file, bof.cfg, and config.cfg (that were

backed up before beginning VSC upgrade), to the VSC. Then reboot the VSC.

Upgrade VRS

Upgrade your VRS(s) and then continue with this procedure. Do not proceed without completing this step.

Upgrade VSC Node Two

1. Backup and prepare VSC node two.

   ```
   metroae upgrade ha vsc backup2
   ```

   Troubleshooting: If you experience a failure, you can re-execute the command.

2. Deploy VSC node two.

   ```
   metroae upgrade ha vsc deploy2
   ```

   VSC node two has been upgraded.

   Troubleshooting: If you experience a failure, you can re-execute the command. If it fails a second time, manually copy (via scp) the .tim file, bof.cfg, and config.cfg (that were backed up before beginning VSC upgrade), to the VSC. Then reboot the VSC.

3. Run postdeploy for VSC node two.

   ```
   metroae upgrade ha vsc postdeploy2
   ```

   Both VSCs are now running the **new** version.

   Troubleshooting: If you experience a failure, you can re-execute the command. If it fails a second time, manually copy (via scp) the .tim file, bof.cfg, and config.cfg (that were backed up before beginning VSC upgrade), to the VSC. Then reboot the VSC.

Upgrade VSTAT

Our example includes a VSTAT node. If your topology does not include one, proceed to *Finalize the Upgrade* below.

1. Run VSTAT health check (optional).

   ```
   metroae upgrade ha vstat health -e
   report_filename=vstat_preupgrade_health.txt
   ```

   You performed health checks during preupgrade preparations, but it is good practice to run the check here as well to make sure the VSD upgrade has not caused any problems.

2. Prepare the VSTAT nodes for upgrade.

   ```
   metroae upgrade ha vstat prep
   ```

   Sets up SSH and disables stats collection.

3. Upgrade the VSTAT nodes.

   ```
   metroae upgrade ha vstat inplace
   ```

   Performs an in-place upgrade of the VSTAT nodes.

4. Complete VSTAT upgrade and perform post-upgrade checks.

   ```
   metroae upgrade ha vstat wrapup
   ```

   Completes the upgrade process, renables stats and performs a series of checks to ensure the VSTAT nodes are healthy.

Finalize the Upgrade

1. Finalize the settings

   ```
   metroae upgrade postdeploy
   ```

   The final steps for the upgrade are executed.

   Troubleshooting: If you experience a failure, you can re-execute the command.

2. Run a health check.

   ```
   metroae upgrade postupgrade health
   ```

   Health reports are created that can be compared with the ones produced during preupgrade preparations. Investigate carefully any errors or discrepancies.

Questions, Feedback, and Contributing

Get support via the forums on the MetroAE site.
Ask questions and contact us directly at devops@nuagenetworks.net.

Report bugs you find and suggest new features and enhancements via the GitHub Issues feature.

You may also contribute to MetroAE by submitting your own code to the project.

Controlling VSD Services

There are times when you need to stop, start, or restart VSD services. For example, if you have a maintenance window in which you want to move the server that hosts a VSD, MetroAE support for VSD Service manipulation will allow you to cleanly shut down VSD services before the move, then bring the VSD services up after the move. You can use the MetroAE workflows that are provided for these purposes.

Prerequisites / Requirements

Before attempting to control the VSD services using MetroAE, you must configure a deployment with information about the VSDs. You also must have previously set up your Nuage MetroAE environment and customized the environment for your target platform.

Use of MetroAE Command Line

MetroAE can perform any of the following VSD service workflows using the command-line tool as follows:

```
metroae vsd services stop [deployment] [options]
metroae vsd services start [deployment] [options]
metroae vsd services restart [deployment] [options]
```

> `deployment`: Name of the deployment directory containing configuration files. See CUSTOMIZE.md
>
> `options`: Other options for the tool. These can be shown using --help. Also, any options not directed to the metroae tool are passed to Ansible.

Note: The VSD services workflows can be used even if you didn't use MetroAE to install or upgrade your VSD deployment.

Questions, Feedback, and Contributing

Get support via the forums on the MetroAE site. Ask questions and contact us directly at devops@nuagenetworks.net.

Report bugs you find and suggest new features and enhancements via the GitHub Issues feature.

You may also contribute to MetroAE by submitting your own code to the project.

Rolling Back or Restoring VSD

In certain cases, you can use MetroAE to rollback or restore a stand-alone VSD or VSD cluster to its original version outside of the normal upgrade path.

Use Cases

A. Your upgrade has failed and you want to roll back to the pre-upgrade original version of the VSD without losing your VSD configuration.

B. Your existing VSD has failed and you want to restore it to the original version without losing your VSD configuration.

In both of these use cases, if the prerequisites, below, are met, MetroAE can help.

Prerequisites

Before rolling back or restoring a VSD configuration, you must meet the following prerequisites:

- You must have a recent backup of the original VSD database available.
  - If you were using MetroAE for the upgrade, the upgrade must have proceeeded past the point where the VSD database was backed up.
    - By default, MetroAE stores your VSD backup in the “backup” subdirectory under the path you specified for the `nuage_unzipped_files_dir` variable in common.yml. If you find a backup in this location, you can proceed with the rollback/restore.

      If you have configured the variable `metro_backup_root` in *upgrade.yml*, you can check for the backup in that location. `metro_backup_root` is an optional variable you

might not have defined. If you find the backup, you can proceed with the rollback/restore.

If you were not using MetroAE, you must have a backup available that was generated by some other method.

You must have the VSD image file, e.g. qcow2, for the original version to restore.

You must configure a deployment for MetroAE to operate on.

If you were using MetroAE for the upgrade, you can start with the deployment that you configured for the upgrade - or make a copy of it and start there.

If you were not using MetroAE, you must create a new deployment.

The `nuage_unzipped_files_dir` variable in your deployment's `common.yml` file must be set to point to the location where the original VSD image file can be found, e.g. 6.0.3 images.

If the original VMs are still on the hypervisor, the VSD VM names in your deployment's `vsds.yml` file must be set to new, unique values. If you have completely destroyed the original VMs, you can use the original VM names.

Steps

If all of these prerequisites and assumptions are true, perform the following steps to rollback/restore a VSD configuration:

1. Shut down all running VSDs.

The VSDs that are being rolled back or restored cannot be running. Shut them down manually using your hypervisor's controls. Optionally undefine or delete them from disk.

2. Configure or create a new deployment as if you intend to install the original version.

Set the `nuage_unzipped_files_dir` variable in `common.yml` to point to the original version image.

If necessary, provide unique VM names in `vsds.yml`.

3. Run the following command to bring up new copies of the original VMs (e.g. 5.4.1).

```
metroae install vsds predeploy <deployment name>
```

4. Manually copy (via scp) the pre-upgrade backup to /opt/vsd/data on VSD 1.

5. Run the following command to start the installation of the VSD software on the VSD VM. This will also restore the backup that you copied to the first VSD.

```
metroae install vsds deploy <deployment name>
```

6. Run the following command to run sanity and connectivity checks on the VSDs:

```
metroae install vsds postdeploy <deployment
name>
```

At this point, your original VSD configuration should be restored and up and running.

Questions, Feedback, and Contributing

Get support via the forums on the MetroAE site.
Ask questions and contact us directly at devops@nuagenetworks.net.

Report bugs you find and suggest new features and enhancements via the GitHub Issues feature.

You may also contribute to MetroAE by submitting your own code to the project.

VSD cluster failover using MetroAE

You can use this procedure to make the current `standby` VSD cluster the `active` VSD cluster. You can promote the `standby` cluster to `active` when both the `active` and `standby` clusters are in good health. Using the `vsd_force_cluser_failover` flag (see below), you can promote the `standby` cluster even when the `primary` cluster is unhealthy or unreachable.

When you use MetroAE to execute the failover procedure, the vsds.yml file in your deployment must be configured with data for 6 VSDs. The first 3 VSDs (VSDs 1-3 in the vsds.yml file) must correspond to the 3 VSDs that are currently `active` VSD cluster and the second 3 VSDs (VSDs 4-6 in the vsds.yml file) must correspond to the 3 VSDs that are currently `standby`. You can create a new deployment for the failover operation or you can re-use the same deployment that was used by MetroAE to install the active/standby cluster in the first place. It is not required that MetroAE be used to install the active/standby clusters in the first place.

Note that the deployment must be configured for the *current* active/standby situation. You can use this procedure to switch back and forth (failover and failback) your clusters, but the order of VSDs in vsds.yml must be changed between runs *or* you can create separate deployments: One for deactivating the `active` cluster and promoting the `standby` cluster to `active` and one for reversing the operation.

When your deployment is correct, use the following command to deactivate the current `active` cluster and promote the `standby` cluster to `active`:

```
metroae vsd failover [deployment_name]
```

If both clusters are in good shape, your failover will complete successfully. If the current `primary` cluster is in poor health or unreachable, MetroAE will print an error and quit. If you want to ignore these errors and force the current `standby` cluster to become `primary`, use following command:

```
metroae vsd failover [deployment_name] -e
vsd_force_cluster_failover=yes
```

MetroAE will also execute a health check on the current `standby` cluster. If the health check fails on the `standby` cluster, MetroAE will print an error and quit. If you want to ignore these errors and promote the `standby` cluster anyway, use the following command:

```
metroae vsd failover [deployment_name] -e
skip_health_check=true
```

Note that you can use both `vsd_force_cluster_failover` and `skip_health_check` on one command:

```
metroae vsd failover [deployment_name] -e
vsd_force_cluster_failover=yes -e skip_health_check=true
```

Questions, Feedback, and Contributing

Get support via the forums on the MetroAE site.
Ask questions and contact us directly at devops@nuagenetworks.net.

Report bugs you find and suggest new features and enhancements via the GitHub Issues feature.

You may also contribute to MetroAE by submitting your own code to the project.

Bootstrapping NSGvs

MetroAE supports the bootstrap of NSGv components while deploying them. There are two supported methods for bootstrap.

Supported NSGv Bootstrap Methods

- MetroAE Bootstrap
- External ISO

MetroAE Bootstrap

MetroAE can perform all of the tasks necessary to bootstrap NSGvs. A global and per NSGv configuration will be applied to the VSD. An ISO file is then generated and downloaded for each NSGv. MetroAE applies each ISO to the corresponding NSGv for bootstrapping during the predeploy role.

To use MetroAE bootstrapping, specify the `bootstrap_method` parameter to be `zfb_metro` in the `nsgvs.yml` deployment file for each NSGv to be bootstrapped by MetroAE. When using this mode, parameters within the `NSGv Zero-Factor Bootstrap` section of the deployment file must be filled out to provide the required information for the per-NSGv VSD configuration. In addition, a second deployment file `nsgv_bootstrap.yml` containing the global VSD configuration is required to be provided. This file contains the global VSD configuration related to bootstrapping such as defining the proxy user, NSG template and VSC infrastructure profile.

External ISO

An NSGv may be bootstrapped using a provided 3rd-party ISO file. In this mode, MetroAE assumes any required VSD configuration is already in place. In the `nsgvs.yml` deployment file, specify the `bootstrap_method` parameter to be `zfb_external` for each

NSGv using this mode. The parameters `iso_path` and `iso_file` are required to provide the path and filename of the ISO file on the MetroAE host. During the predeploy phase of the NSGv, the provided ISO will be used for bootstrapping.

Activation Link

If you would like to use an activation link for bootstrapping the NSGv, MetroAE will spin up with NSGv without configuring it for bootstrap. Bootstrap configuration will be left to the user. To use an activation link as the bootstrap method, you can specify the `bootstrap_method` parameter to be `activation_link` in the `nsgvs.yml` deployment file. You will not need the `nsgv_bootstrap.yml` deployment file if you specify activation link as your bootstrap method.

Questions, Feedback, and Contributing

Get support via the forums on the MetroAE site.
Ask questions and contact us directly at devops@nuagenetworks.net.

Report bugs you find and suggest new features and enhancements via the GitHub Issues feature.

You may also contribute to MetroAE by submitting your own code to the project.

SD-WAN Portal deployment using MetroÆ

MetroÆ now supports the deployment of SD-WAN Portal for Nuage VNS. For more information about SD-WAN Portal and the deployment requirements, please consult the product documentation

SD-WAN Portal deployment requires a RHEL7/CentOS 7 image as well as docker images with the product software

Supported deployment:

- KVM hypervisor(s)
- SD-WAN Portal versions 6.0.1+

Currently the following workflows are supported:

- metroae install portal - Deploy Portal VM(s) on KVM hypervisor and install the application
- metroae install portal predeploy - Prepares the HV and deploys Portal VMs
- metroae install portal deploy - Installs Docker-CE, SD-WAN Portal on the already prepared VMs
- metroae install portal postdeploy - To be updated. Includes a restart and license update task
- metroae install portal license - Copies the license file to the Portal VM(s) and restarts the Portal(s)
- metroae destroy portal - Destroys Portal VMs and cleans up the files from hypervisor(s)

Example deployment files are available under examples/kvm_portal_install

1. Configure `common.yml`

In your MetroAE deployment folder, create or edit the `common.yml` configuring the necessary attributes. Portal specific attributes include:

portal_fqdn_global - SD-WAN Portal Global FQDN. Typically a public (external) FQDN resolvable to the Portal endpoint on a Proxy/LB in both standalone and HA deployments. For standalone - FQDN of a single Portal node

vsd_port_global - Used with vsd_global_fqdn by the SD-WAN Portal to connect to the VSD cluster. Defaults to 8443.

vstat_fqdn_global - ElasticSearch cluster FQDN. SD-WAN Portal accesses the ES cluster to retrieve statistics for visualization and reports. For standalone ES - FQDN of a single ES node.

yum_proxy - Portal is using the same Proxy for Docker to pull the images from Docker hub. Optional if using the pre-downloaded SW package.

portal_license_file - SD-WAN Portal license. Request through ASLM.

portal_ram - Amount of Portal RAM to allocate, in GB.

2. Configure `credentials.yml`

Create or edit the `credentials.yml` configuring the necessary attributes. Portal specific attributes include:

portal_username - VSD username reserved for the Portal. (Optional)

portal_password - VSD password for the user configured in `portal_username`. (Optional)

portal_custom_username - CLI user to log in to the Portal VM

portal_custom_password - CLI user password to log in to the Portal VM

smtp_auth_username - SMTP server username used by Portal Messaging app to send Portal user management emails

smtp_auth_password - Password for SMTP server user.

3. Configure `portals.yml`

Create or edit the `portals.yml` configuring the necessary attributes. For standalone deployment, only one Portal section is needed. 3 Portal sections are required for HA deployment (see example deployment file). Standard MetroÆ attributes are used with some unique to SD-WAN Portal ones listed below:

password_reset_email; new_account_email; forgot_password_email - Sender address to be used in Portal user management emails

smtp_fqdn - SMTP server FQDN

smtp_port - SMTP server port

smtp_secure - Specifies whether user/password authentication is required. If True, configure username and password in credentials.yml

sdwan_portal_secure - Enables SSL for the SD-WAN Portal

portal_version - if using Docker hub to pull the images, specifies the tag of the Docker image

---

Deploying Nuage Networks Components in Amazon Web Services (AWS) with MetroAE (limited support)

Supported Components

MetroAE supports deployment of the following components in AWS.

- VSD
- VSC (as KVM running on an AWS bare-metal instance)
- VSTAT (ElasticSearch)
- VNS Utils
- NSGv

Main Steps for Deploying in AWS

1. Install Libraries

MetroAE uses the CloudFormation Ansible module for deploying components in AWS. This module requires the python-boto and python-boto3 python libraries. Use one of the following three methods to install these required libraries on the MetroAE host.

Method 1: pip

```
pip install boto
pip install boto3
```

Method 2: yum

```
yum install python-boto
yum install python-boto3
```

Method 3: apt

```
apt-get install python-boto
apt-get install python-boto3
```

2. Upload or Import AMIs

Amazon Machine Images (AMIs) are used to run instances in EC2. For each Nuage Networks component that you want to deploy (except VSC), you'll need to upload or import an AMI to AWS. The AMI identifiers are provided to MetroAE for deployment. VSC is not supported as an AMI. It must be deployed as KVM running on an AWS bare-metal instance.

3. Setup Virtual Private Cloud

Before installing Nuage Networks components, you must define and deploy a virtual private cloud (VPC) in AWS. An example file (aws-example-vpc.yml) of a basic VPC is provided in the examples directory. This VPC must define the network interfaces that will be used by each component. The VPC should also provide connectivity between various components and Internet access (either directly or outgoing only through NAT). We strongly recommend that you define security policies, IP addressing and DNS. The recommended subnets for each component are defined

below. Note that the access subnet is expected to have direct Internet access and the management subnet to have outgoing only access.

COMPONENT	SUBNET1	SUBNET2
VSD	Mgmt	
VSC	Mgmt	Data
VSTAT	Mgmt	
VNS Util	Mgmt	Data
NSGv	Access	Data

4. Setup Bare Metal Host (for VSC Only)

Deploying VSC as a standard AWS component is not supported. Because it relies on the VxWorks operating system, the VSC image cannot be converted to an AMI. Instead, you can run VSC as a KVM instance within an AWS bare-metal server. Follow the steps below to setup the bare-metal host.

1. Install a Linux AMI on the server.

2. Install the libvirt KVM libraries on the server.

3. Start the libvirtd daemon.

4. Setup network connectivity to the VSC.

The AWS bare-metal server does not support bridge interfaces, PCI passthrough, or macvtap. To make connections use the routed network option. The routed networks must be defined in libvirt on the host. Multiple addresses can be supported on a single bare-metal interface by adding secondary IP addresses via the EC2 console and using SNAT and DNAT iptables rules.

5. Configure Components

Configuring components for AWS is similar to configuring for other server types. See CUSTOMIZE.md for details on standard deployments. The configuration files for AWS deployments require a few additional specifications.

credentials.yml

AWS access can be specified as `aws_access_key` and secret keys can be specified as `aws_secret_key`. If AWS access is not specified, values are taken from the environment variables `AWS_ACCESS_KEY` and `AWS_SECRET_KEY`.

vsds.yml and vstats.yml

For components other than VSC, set `target_server_type` to "aws".

AWS requires that the following fields be specified for all components, except VSC.

aws_region: The AWS region (i.e. us-east-1)

aws_ami_id: Identifier for the AMI image to be used for the component

aws_instance_type: The AWS instance type for the image (i.e. t2.medium)

aws_key_name: The name of the key pair used for access to the component

aws_mgmt_eni/aws_data_eni/aws_access_eni: The elastic network interface identifiers from the deployed VPC for each required subnet for the component.

vscs.yml

VSC is not supported as a direct AWS component, but it can be deployed as a bare-metal server by specifying several fields in `vscs.yml` as shown below.

Set `target_server_type` to “kvm” and `target_server` to the address(es) of the bare-metal host(s).

To support routed network connectivity, specify the following fields.

`internal_mgmt_ip`: The ip address to be assigned to the management interfaces on the VSC itself. This internal address can be NATed to the real address of the bare-metal host using iptables rules.

`internal_ctrl_ip`: The ip address to be assigned to the data interfaces on the VSC itself. This internal address can be NATed to the real address of the bare-metal host using iptables rules.

`internal_data_gateway_ip`: The ip address of the data network gateway for the VSC. This ip address is used for VSC to connect to NSG and other components via static route addition on VSC

nsgv_bootstrap.yml

Bootstrapping of NSGvs deployed to AWS is supported through the normal bootstrapping process. See NSGV_BOOTSTRAP.md for details.

Alternative Specification for NSGv Only Deployments

If you'd like to deploy only NSGv (no other components), then MetroAE can optionally provision a suitable VPC. You will need to configure `nsgvs.yml` in your deployments subdirectory. For the automatic creation of a test VPC on AWS to host your NSGv, the following parameters must be provided in `nsgvs.yml` for each NSGv:

provision_vpc_cidr

provision_vpc_nsg_wan_subnet_cidr

provision_vpc_nsg_lan_subnet_cidr

provision_vpc_private_subnet_cidr

The CIDRs for the VPC, WAN interface, LAN interface and private subnet must be specified. When provisioning a VPC in this way, the elastic network interface identifiers `aws_data_eni` and `aws_access_eni` for the NSGv do not need to be specified as they are discovered from the created VPC.

6. Deploy Components

After you have set up the environment and configured your components, you can use MetroAE to deploy your components with a single command.

```
metroae install everything
```

Alternatively, you can deploy individual components or perform individual tasks such as predeploy, deploy and postdeploy. See DEPLOY.md for details.

Questions, Feedback, and Contributing

Get support via the forum on the MetroAE site.
Ask questions and contact us directly at devops@nuagenetworks.net.

Report bugs you find and suggest new features and enhancements via the GitHub Issues feature.

You may also contribute to MetroAE by submitting your own code to the project.

Deploying Nuage Networks Components using Terraform and MetroAE

Nuage Network components can be deployed using Terraform by allowing the tool to perform the predeploy role while utilizing MetroAE for deploy and postdeploy roles. The predeploy role performs the steps to define and spin up VM resources. The deploy role installs, configures and activates the Nuage Network software on the VM resources. The postdeploy role provides health checking to ensure that the software is indeed running properly.

Steps for Deploying using Terraform and MetroAE

1. Install Terraform and MetroAE

Both Terraform and MetroAE are required to be installed using standard installation procedures. Terraform can be installed via the installation tutorial. MetroAE is installed via the setup guide.

2. Customize MetroAE Deployment

A set of MetroAE deployment files is required to provide configuration for MetroAE to perform the deploy steps. These should be written as if MetroAE were to be performing all of the

roles of the deployment using the steps described in the customize deployment guide.

3. Create Terraform Configuration

The Terraform configuration files are required to describe the required providers and resource definitions to allocate compute nodes for components. These can be written according to the build infrastructure guide.

Note that Terraform will need to perform all of the steps that MetroAE would have done during the predeploy role. These include the following for most VM machine types:

- Define VM resources including number of CPU cores and memory
- Create required dependencies (i.e. disks, networks, etc.)
- Disable cloud-init for the component
- Configure networking for the component
- Set the hostname for the component
- Copy in authorized SSH keys
- Configure autostart for the VM
- Start the VM

4. Add Terraform Provisioners and Dependencies

Terraform will be providing the predeploy role, however, MetroAE will need to be triggered to perform the deploy and postdeploy roles. This is accomplished using provisioners.

```
resource "<provider>" "vsd1" {
    resource definitions...

    provisioner "local-exec" {
        command = "./metroae install vsds deploy"
        working_dir = "<metro-directory>/nuage-metroae"
```

```
    }

    provisioner "local-exec" {
        command = "./metroae install vsds postdeploy"
        working_dir = "<metro-directory>/nuage-metroae"
    }
}
```

The provisioners will trigger MetroAE to execute the deploy and postdeploy roles after the component VM is running. Provisioners will need to be added for each component to be deployed.

MetroAE should not be run in parallel and playbooks need to be run in the correct order (i.e. VSDs, VSCs, VSTATS…). To ensure that this is the case, resources need to be defined using dependencies as follows.

```
resource "<provider>" "vsd1" {
    resource definitions...
}
resource "<provider>" "vsc1" {
    resource definitions...
    depends_on = [<provider>.vsd1]
}
resource "<provider>" "vstat1" {
    resource definitions...
    depends_on = [<provider>.vsd1, <provider>.vsc1]
}
```

5. Apply Using Terraform

With all of the tools installed and properly configured, Terraform can perform the entire deployment using apply as described by change infrastructure.

```
terraform plan
terraform apply
```

Questions, Feedback, and Contributing

Get support via the forums on the MetroAE site. You may also contact us directly. Ask questions and contact us directly at devops@nuagenetworks.net.

Report bugs you find and suggest new features and enhancements via the GitHub Issues feature.

You may also contribute to MetroAE by submitting your own code to the project.

Hooks and Skip Actions

MetroAE allows fine-grained control of operations via its hierarchical design. You have the option to execute workflows such as `install_everything` when you just want the simplicity of allowing MetroAE to handle the details. In some situations, however, you may want to take advantage of fine-grained control such as individually executing the workflows `vsd_predeploy`, `vsd_deploy`, and `vsd_postdeploy`.

One of the reasons to take advantage of fine-grained control is so that you can execute your own scripts for verification and configuration between MetroAE's steps. Another is so that you can skip a workflow entirely, using your own tooling to bring up the VSD VM instead of using `vsd_predeploy`, for example. Hooks have been added to simplify and automate the former. Skips have been added to simplify and automate the latter.

Hooks

Hooks are predefined locations that preceed specific MetroAE worflows. They can be thought of as places in the code where MetroAE can execute commands on your behalf. These commands can be any executable available on the Ansible host. If the command is a script, it has access to the full range of MetroAE's Ansible variable, e.g. `echo {{ hostvars[inventory_hostname].name }}`. As you can see, your script should use standard Ansible Jinja2 variable syntax. Each predefined location comes immediately before selected low-level MetroAE workflows. Defining a command or script to run at a specific location means that the command or script will run before the workflow is executed.

Hooks File

A Hooks file is a YAML file that contains the name of the MetroAE workflow that is the location of the hook and the command or

script that will be executed before the workflow is executed. The path to a Hooks file is specified in the optional `hooks` section of the deployment file for the component being operated on, e.g. `vsds.yml`. If more than one command needs to be run at a specific location, multiple Hooks files must be created, each specifying the same location and a unique command. When creating multiple files for the same location MetroAE does not guarantee the execution of the files in the order defined in the deployment file. If order matters the best practice is to create a wrapper script that calls the commands in the preferred order. If any one command execution fails, the MetroAE workflow will fail with an error.

Here is an example of a Hooks file that specifies a script to execute prior to running the `vstat_predeploy` workflow:

```
location: vstat_predeploy
command: "/tmp/myfiles/myscript.sh"
```

Workflow Hook Locations

The following Nuage component workflows have hook support.

Installation Workflow Hook Locations

```
vsd_predeploy
vsd_deploy
vsd_postdeploy
vsc_predeploy
vsc_deploy
vsc_postdeploy
vstat_predeploy
vstat_deploy
vstat_postdeploy
vnsutil_predeploy
vnsutil_deploy
vnsutil_postdeploy
nsgv_predeploy
nsgv_postdeploy
vcin_predeploy
vcin_deploy
vcin_destroy
vrs_predeploy
```

```
vrs_deploy
vrs_postdeploy
vrs_destroy
```

Upgrade Workflow Hook Locations

```
upgrade_vsds
upgrade_vsds_inplace
upgrade_vsds_inplace_dbbackup
vsc_ha_upgrade_backup_and_prep_1
vsc_ha_upgrade_backup_and_prep_2
vsc_ha_upgrade_deploy_1
vsc_ha_upgrade_deploy_2
vsc_ha_upgrade_postdeploy_1
vsc_ha_upgrade_postdeploy_2
vsc_sa_upgrade_backup_and_prep
vsc_sa_upgrade_deploy
vsc_sa_upgrade_postdeploy
vsd_ha_upgrade_database_backup_and_decouple
vsd_ha_upgrade_deploy_2_and_3
vsd_ha_upgrade_predeploy_2_and_3
vsd_ha_upgrade_shutdown_2_and_3
vsd_upgrade_complete
vsp_upgrade_postdeploy
vstat_upgrade
vstat_upgrade_prep
vstat_upgrade_wrapup
```

Skip Actions

A Skip Action is invoked when you want a particular workflow to be skipped. For example, defining `vsd_predeploy` in the skip actions list for a component will cause MetroAE to skip this workflow even when running a workflow that encapsulates it, e.g. `install_vsds`. In such a case, running `install_everything` or `install_vsds` will skip the `vsd_predeploy` workflow. Skip actions need to specified in common.yml file. skip actions in common.yml takes a list. Example of skip_actions

```
skip_actions:
  - vsd_predeploy
  - vstat_predeploy
```

Note: In the current release, Skip Actions are supported for the following workflows:

```
vsd_predeploy
vsc_predeploy
vstat_predeploy
nsgv_predeploy
vnsutil_predeploy
vrs_predeploy
```

MetroAE Plugins

Plugins are a way for you to extend the functionality of the Metro Automation Engine without modifying MetroAE code. This allows you to create your own workflows as Ansible roles and have them executed by MetroAE.

Overview

The overview of the process is:

1. Create the proper files and subdirectories (see below) in a directory of your choosing
2. Package the plugin using the `package-plugin.sh` script
3. Install the plugin using the `metroae plugin install` command
4. Add the proper data files, if required, to your deployment directory
5. Execute your role

This process remains the same whether you are using the MetroAE Container or working with a github clone workspace.

Note that if you are working with the MetroAE Container and you update the container, you will need to reinstall the plugin.

Operations

MetroAE supports the following operations in support of plugins

Package a Plugin

To package a plugin, issue:

```
./package-plugin.sh <plugin-directory>
```

This will create a tarball of the plugin ready for distribution to users.

Install a Plugin

Users who wish to install the plugin can issue:

```
./metroae plugin install <plugin-tarball-or-directory>
```

This should be issued from the nuage-metro repo or container. Note that a tarball or unzipped directory can both be installed.

Uninstall a Plugin

To uninstall a plugin, the user can issue:

```
./metroae plugin uninstall <plugin-name>
```

This should be issued from the nuage-metro repo or container. Uninstall is by plugin name as all installed files were recorded and will be rolled back.

Examples

Examples of plugins can be found in the MetroAE `examples` directory. They range from `examples/plugins/simple` which contains a minimal example that does not require user input, e.g., just running a simple Ansible task, to `examples/plugins/full` which takes advantage of MetroAE's schema validation of user inputs, Ansible inventory generation, and more. You can choose the level of support your plugin requires. Ther eis also an intermediate example, `examples/plugins/medium`.

Authoring Plugins

Create Workspace

In any location you choose, create a base directory that has the same name as the name of the plugin. This base directory will serve as the 'root' of your plugin development work. The directory and plugin name should not contain spaces or special characters. The following directories should be created under the base plugin directory:

playbooks: Set of playbooks for user to issue to initiate plugin features. (required)

roles: Ansible roles implementing the plugin features. (optional)

schemas: Set of schemas defining the deployment data required from the user. (optional)

Upon installation of the plugin, the files in the above directories will be copied and merged into the MetroAE codebase, becoming part of the MetroAE suite.

Note that the name of the playbooks and roles need not match the name of the plugin. Note also that a single plugin can contain multiple playbooks and roles.

It is best practice that your playbook names match your role names. By convention, MetroAE has adopted the practice of naming playbooks with _ separators and roles with – seperators. For example, `vsd_deploy.yml` is the name of the playbook, and `vsd-deploy` is the name of the corresponding role.

All file names must be unique to the plugin to prevent conflict with other components of the MetroAE suite. For example, the vsd_deploy.yml playbook already exists within MetroAE. To avoid clobbering this exitsing playbook, your playbook names should be unique.

Create Plugin Configuration

Each plugin needs a configuration file that describes the general parameters of the plugin. It must be called `plugin-cfg.yml` and be contained in the base plugin directory.

For the `simple` form of plugin, one that does not require user input or inventory generation, the plugin configuration file can be as simple as:

```
plugin_name: simple
description: |
  Plugin for running ls
version: 1.0.0
required_metro_version: 4.1.0
schemas: []
hooks: []
```

This simple plugin has no defined schemas or hooks, and therefore no user inputs.

A more complete or `full` formm of plugin takes advantage of the many MetroAE features, such as user input validation via JSON schema, Ansible inventory generation, built-in common roles, and more.

A more complete plugin configuration has the following form:

```
plugin_name: full
description: |
  Plugin for demonstration purposes. Installs a Demo VM using
only plugin
  functionality.
version: 1.0.0
required_metro_version: 4.1.0
schemas:
  - name: "demovms"
    is_list: yes
    required: no
    encrypted: no
hooks:
  - location: build
    role: common
    tasks: demovm-process-vars
```

Note that this `full` plugin configuration specifies that there will be user inputs that will be validated by a JSON schema with the name `demovms`. This implies that there will be a role created named `demovms` and a user input deployment file named `demovms.yml` that will be parsed at run time.

The specifics of the elements of the plugin configuration are as follows:

plugin_name: The name of the plugin and the base directory. (required)

description: Human-readable description describing the purpose of the plugin. (optional)

version: Version number describing the plugin. Should be in the form major.minor.patch (required)

required_metro_version: Minimum required version of MetroAE required for this plugin to operate. (required)

schemas: Listing of the schemas provided by the plugin to describe deployment data required from user. (Optional)

name: Base file name of the schema (no extension)

is_list: Whether or not the schema describes a list of data sets or is a single object data set.

required: Whether or not the data from the schema must always be present in the user's deployment set.

encrypted: Determines whether or not the data contains encrypted fields.

hooks: Describes the MetroAE hook locations where the plugin roles will be executed. (optional)

location: The name of the MetroAE hook location to execute the role.

role: The name of the role directory where tasks will be executed.

tasks: The name of the tasks file where the tasks are defined.

Note that when not specified, `schemas` and `hooks` must be shown as empty lists, i.e., `hooks: []`.

Create Schema for Required User Data

MetroAE configures workflows using user data sets called `deployments`. The `deployments` are sets of YAML files

grouped together in directories. The contents of which are to be filled out by the user and validated by json schemas. The plugin can recieve required user data from the user by defining a set of one or more schemas for the data. These schemas should be placed under the `schemas` directory under the base plugin directory and then listed in the `schemas` section of the `plugin-cfg.yml` file as described in the above section. Care should be taken to keep schema names unique so they do not conflict with the MetroAE Suite or other plugins. The data from the user will be available during the `build` role and can be used to generate the required Ansible `inventory` for the plugin.

A sample schema is provided in the MetroAE examples.

Generate Plugin Inventory

In general, there are 3 categories of plugins: Those that do not require additional user inputs to be added to the inventory, those that introduce new hosts that require additional user inputs added to the inventory for those hosts, and those that do not introduce new hosts but need to add custom variables to the inventory for existing hosts.

If your plugin does not require additional user inputs, there is nothing for you to do here.

Generate Plugin Inventory for New Host Types

If your plugin introduces new host types that require user input, Ansible requires host_vars files called the `inventory` to provide the configuration for the playbooks that are executed. You can find an example of this process in `examples\plugins\full`. In MetroAE, the inventory is generated dynamically during the build workflow. The build workflow is executed automatically by the `metroae` script whenever changes to the deployment files are detected. Any host_vars inventory required by the plugin must be generated by a playbook you provide. Your process-vars playbook will be executed via the `build` hook. This can be accomplished by defining the following hook in the plugin config:

```
hooks:
  - location: build
    role: common
    tasks: <plugin_device_name>-process-vars
```

The `<plugin_device_name>-process-vars` task file must be defined under the `roles/common` directory under the plugin base directory and should implement creating the inventory files for the plugin. Any data supplied by the user that was defined by schemas is made available to Ansible here.

Device specific host files can be created by the usual means of using role templates and writing them to the host_vars directory. The `write-host-files` role is paricularly useful for this job. Special care should be taken to keep any inventory files written unique as the plugin should not conflict with existing MetroAE infrastructure or other plugins. The master `hosts` file should not be directly edited by the plugin. However, special functionality has been added where blocks can be appened into master file by setting the special variable `plugin_hosts`. The contents of `plugin_hosts` will be automatically appened into the master `hosts` file after the build hook for the plugins have been run. In order to play nicely with other plugins, each plugin should append their contents rather than replace. Example code:

```
- name: Append content to plugin hosts
  set_fact:
    plugin_hosts: |
      {{ plugin_hosts }}

      {{ this_plugin_device_hosts }}
```

Append Plugin Inventory for Existing Hosts

If your plugin does not add a new host type that requires their own custom inputs but it does require custom inputs that should be applied to an existing host type, you can append the required variables to the inventory for an existing host type. An example of this can be found in `examples\plugins\medium` folder.

In the `medium` example, a new user is added to the VSD. This new user is added via a custom variable `my_new_vsd_user` in the `newcredentials` schema file and is available for use in the custom playbooks and roles. Important thing to note here is that the process-vars playbook that you deliver with the the appending of credentials to the appropriate host_vars file as described here in the

```
    - name: Append credentials to the component host vars
      blockinfile:
        path: "{{ inventory_dir }}/host_vars/{{
newcredentials.name }}"
        block: "{{ lookup('file', '{{ inventory_dir
}}/host_vars/newcredentials') }}"
        insertafter: EOF
```

In this case, then, the Plugin will be executed with `hosts` set to the host type that the role shoudl modify, e.g., `vsds`.

Create Roles for Plugin

The Ansible roles are where any tasks required for the plugin are to be implemented. The roles should all be defined under the `roles` directory under the plugin base directory. These roles will be merged into the MetroAE Suite codebase for the user during plugin installation. As such, care should be taken that role file names do not conflict with existing roles in MetroAE or that of other plugins. Although, directory names can be common and overlapping directories will be merged rather than replaced. Directories can be nested under each role, this is very common such as `tasks`, `vars` or `templates`.

Roles in plugins act as any other role in Ansible. The roles can even call other roles from the plugin or from existing MetroAE roles. Existing roles in the MetroAE engine cannot be edited by a plugin. Plugins must be exclusively additive. The roles can be triggered by plugin playbooks or by the hooks functionality.

Example roles are provided in the examples.

Call Plugin Roles by Playbook or Hooks

The roles of the plugin can be called by playbooks. Any `workflow` actions that the user can execute from the plugin should be implemented as an Ansible playbook and placed in the `playbooks` directory under the base plugin directory. Care should be taken that the playbook names are unique so they do not conflict with existing MetroAE engine playbooks or playbooks of other plugins. The playbooks for the plugin act as any other playbook in Ansible. They can contain hosts from the dynamically created plugin inventory, or existing MetroAE hosts. They can issue roles from the plugin itself or from existing MetroAE roles.

Existing roles within the MetroAE engine cannot be modified by plugins. However,plugin roles can be issued by existing roles using hooks. Hooks can be defined in the `plugin-cfg.yml` file which define which role and tasks file should beexecuted and the hook location where it should be executed. The MetroAE documentation has more information about hooks, although plugins extend the functionality to seamlessly issue plugin roles rather than executing shell commands.

Define Menus for Plugin Playbooks

MetroAE defines a menuing system for workflows. The menus organize content and provide context-specific help. Plugins can merge their workflows to the master menu by providing a `menu` file under the base plugin directory. Use of menu options is optional. Care should be taken that menu items do not conflict with existing MetroAE engine menus or menus of other plugins. The menu items are shell variables that define each menu level. An example format is as follows:

```
    MENU+=(',install,demovms'  'Install Demonstration VM'
'playbook'  'demovm_predeploy'  ',install')
    MENU+=(',install,demovms,predeploy'  'Pre-deploy install
step for Demonstration VM'  'playbook'    'demovm_predeploy'
',install,demovms')
```

First item: A comma-separated string of each level of the menu item.

Second item: The context-specific help string for the menu item

Third item: The action to perform. This will almost always be `playbook` to issue a playbook.

Forth item: The parameter for the action. This will almost always be the playbook name to issue.

Fifth item: The help level to show context-specific help.

Package and Distribute Plugin

Once all of the steps of plugin development have been successfully developed, the plugin can be packaged. The packaging is described at the top of this document. The result of packaging is a zipped tarball. The tarball can be distributed to users by any means. The user can install or uninstall the plugin using the procedure defined at the top of this document.

Automatically Generating Deployment Files

If you would like to automatically generate deployment files, MetroAE provides jinja2 templates for this purpose. The templates are automatically generated from the schemas by the MetroAE team. They are always up to date. You can find these jinja2 templates in the src/deployment_templates directory in your workspace. You can use one, monolithic YAML file as input to generate all deployment files. You can use multiple input files, one for each deployment file you wish to generate. Or some combination of the two. To generate the files, you can use a Python script to call jinja2 to convert the YAML input file into a deployment file.

Example code

Below is example code that can be used to create deployment files from a single user input file.

```
def write_deployment(var_dict, deployment_name,
jinja_template_file, deployment_file):

    deployment_dir = os.path.join(DEPLOYMENTS_DIRECTORY,
deployment_name)
    if not os.path.exists(deployment_dir):
        os.makedirs(deployment_dir)

    write_deployment_file(jinja_template_file,
                          deployment_file,
                          var_dict)


def write_deployment_file(template_file, to_file, var_dict):
    with open(template_file, "r") as file:
        template_string = file.read().decode("utf-8")

    template = jinja2.Template(template_string,
                               autoescape=False,

undefined=jinja2.StrictUndefined)

    var_file_contents = template.render(var_dict)
```

```
    with open(to_file, "w") as file:
        file.write(var_file_contents.encode("utf-8"))
```

Example user data

```
nuage_unzipped_files_dir: "/my/unzipped/filedir"
dns_domain: "company.com"
vsd_fqdn_global: "vsd1.company.com"
mgmt_bridge: "br0"
data_bridge: "br1"
ntp_server_list: [ "5.5.5.5", "2.2.2.2", ]
dns_server_list: [ "10.1.0.2", ]
deployment_name: ""
deployment_description: ""
credentials:
  -
    name: ""
vsds:
  -
    hostname: "vsd1.company.com"
    mgmt_ip: "192.168.110.30"
    mgmt_ip_prefix: 24
    mgmt_gateway: "192.168.110.1"
    target_server_type: "kvm"
    target_server: "10.105.1.102"
```

Contributing to MetroAE

MetroAE is built on a community model. The code has been architected to make contribution simple and straightforward. You are welcome to join in the community to help us continuously improve MetroAE.

We Use Github Flow, So All Code Changes Happen Through Pull Requests

Pull requests are the best way to propose changes to the codebase (we use Github Flow). We actively welcome your pull requests:

1. Fork the repo and create your branch from `dev`.
2. If you've added code that should be tested, add tests.
3. Make sure your code passes flake8.
4. Issue that pull request!

Prerequisites / Requirements

All contributions must be consistent with the design of the existing workflows.

All contrinbutions must be submitted as pull requests to the *dev* branch, reviewed, updated, and merged into the nuage-metroae repo.

You must have a github.com account and have been added as a collaborator to the nuage-metroae repo.

Contributing your code

1. Developing your code.

The manner in which you develop the code contribution depends on the extent of the changes. Are you enhancing an existing playbook or role, or are you adding one or more new roles? Making changes to what already exists is simple. Just make your changes to the files that are already there.

Adding a new component or feature is a bit more involved. For example, if you are adding support for the installation of a new component, the following elements would be expected unless otherwise agreed upon by the repo owners:

1. A new user-input schema for the component must be created. See the exitsing files in the `schemas` directory.
2. A new deployment template must be created. See the existing files in the `src/deployment_templates` directory.
3. Add to the example data. All deployment templates and examples are auto-generated. The data in `src/raw_example_data` is used by the automatic generation to populate the examples properly. Also see the examples that have been auto-generated in the `examples/` directory.
4. Add your component and associated file references to `src/workflows.yml`.
5. Add your schema to `src/roles/common/vars/main.yml`.
6. Execute `src/generate_all_from_schemas.sh` to create all the required files for your component.
7. Create the proper roles. The following roles are required unless otherwise agreed to by the repo owners: *newcomponent-predeploy*, *newcomponent-deploy*, *newcomponent-postdeploy*, *newcomponent-health*, and *newcomponent-destroy* should be created under `src/roles/`
8. Create the proper playbooks to execute the roles: *newcomponent_predeploy.yml*, *newcomponent_deploy.yml*,

newcomponent_postdeploy.yml, *newcomponent_health.yml*, and *newcomponent_destroy.yml* should be created under `src/playbooks/with_build`

9. Test, modify, and retest until your code is working perfectly.

2. Test all proposed contributions on the appropriate hypervisors in the `metro-fork` directory. If you choose not to provide support for one or more supported hypervisors, you must provide graceful error handling for those types.

3. All python files modified or submitted must successfully pass a `flake8 --ignore=E501` test.

4. Add a brief description of your bug fix or enhancement to `RELEASE_NOTES.md`.

Any contributions you make will be under the APACHE 2.0 Software License

In short, when you submit code changes, your submissions are understood to be under the same APACHE License 2.0 that covers the project. Feel free to contact the maintainers if that's a concern.

Report bugs using Github's issues

We use GitHub issues to track public bugs.

Write bug reports with detail, background, and sample code

Great Bug Reports tend to have:

A quick summary and/or background

Steps to reproduce

    Be specific!

    Give sample code if you can.

- What you expected would happen
- What actually happens
- Notes (possibly including why you think this might be happening, or stuff you tried that didn't work)

Use a Consistent Coding Style

- 4 spaces for indentation rather than tabs
- 80 character line length
- TODO

License

By contributing, you agree that your contributions will be licensed under its APACHE 2.0 License.

Questions and Feedback

Ask questions and get support on the MetroAE site.
You may also contact us directly.
Outside Nokia: devops@nuagenetworks.net.
Internal Nokia: nuage-metro-interest@list.nokia.com.

References

This document was adapted from the open-source contribution guidelines for Facebook's Draft

Apache License
Version 2.0, January 2004
https://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as

indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   © You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a “NOTICE” text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file

distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT,

MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Contributing to MetroAE

MetroAE is built on a community model. The code has been architected to make contribution simple and straightforward. You are welcome to join in the community to help us continuously improve MetroAE.

We Use Github Flow, So All Code Changes Happen Through Pull Requests

Pull requests are the best way to propose changes to the codebase (we use Github Flow). We actively welcome your pull requests:

1. Fork the repo and create your branch from `dev`.
2. If you've added code that should be tested, add tests.
3. Make sure your code passes flake8.
4. Issue that pull request!

Prerequisites / Requirements

All contributions must be consistent with the design of the existing workflows.

All contrinbutions must be submitted as pull requests to the *dev* branch, reviewed, updated, and merged into the nuage-metroae repo.

You must have a github.com account and have been added as a collaborator to the nuage-metroae repo.

Contributing your code

1. Developing your code.

The manner in which you develop the code contribution depends on the extent of the changes. Are you enhancing an existing playbook or role, or are you adding one or more new roles? Making changes to what already exists is simple. Just make your changes to the files that are already there.

Adding a new component or feature is a bit more involved. For example, if you are adding support for the installation of a new component, the following elements would be expected unless otherwise agreed upon by the repo owners:

1. A new user-input schema for the component must be created. See the exitsing files in the `schemas` directory.
2. A new deployment template must be created. See the existing files in the `src/deployment_templates` directory.
3. Add to the example data. All deployment templates and examples are auto-generated. The data in `src/raw_example_data` is used by the automatic generation to populate the examples properly. Also see the examples that have been auto-generated in the `examples/` directory.
4. Add your component and associated file references to `src/workflows.yml`.
5. Add your schema to `src/roles/common/vars/main.yml`.
6. Execute `src/generate_all_from_schemas.sh` to create all the required files for your component.
7. Create the proper roles. The following roles are required unless otherwise agreed to by the repo owners: *newcomponent-predeploy*, *newcomponent-deploy*, *newcomponent-postdeploy*, *newcomponent-health*, and *newcomponent-destroy* should be created under `src/roles/`
8. Create the proper playbooks to execute the roles: *newcomponent_predeploy.yml*, *newcomponent_deploy.yml*,

newcomponent_postdeploy.yml, *newcomponent_health.yml*, and *newcomponent_destroy.yml* should be created under `src/playbooks/with_build`

9. Test, modify, and retest until your code is working perfectly.

2. Test all proposed contributions on the appropriate hypervisors in the `metro-fork` directory. If you choose not to provide support for one or more supported hypervisors, you must provide graceful error handling for those types.

3. All python files modified or submitted must successfully pass a `flake8 --ignore=E501` test.

4. Add a brief description of your bug fix or enhancement to `RELEASE_NOTES.md`.

Any contributions you make will be under the APACHE 2.0 Software License

In short, when you submit code changes, your submissions are understood to be under the same APACHE License 2.0 that covers the project. Feel free to contact the maintainers if that's a concern.

Report bugs using Github's issues

We use GitHub issues to track public bugs.

Write bug reports with detail, background, and sample code

Great Bug Reports tend to have:

A quick summary and/or background

Steps to reproduce

Be specific!

Give sample code if you can.

What you expected would happen

What actually happens

Notes (possibly including why you think this might be happening, or stuff you tried that didn't work)

Use a Consistent Coding Style

4 spaces for indentation rather than tabs

80 character line length

TODO

License

By contributing, you agree that your contributions will be licensed under its APACHE 2.0 License.

Questions and Feedback

Ask questions and get support on the MetroAE site.
You may also contact us directly.
Outside Nokia: devops@nuagenetworks.net.
Internal Nokia: nuage-metro-interest@list.nokia.com.

References

This document was adapted from the open-source contribution guidelines for Facebook's Draft