User Guide

Network and Edge Cloud Reference System Architecture Release 23.07

Authors

Jan Benedikt

Joseph Gasparakis

Michael Pedersen

Abhijit Sinha

Mykyta Zernov

1 Introduction

1.1 Purpose and Scope

The **Network and Edge Cloud Reference System Architecture (Cloud RA)** is one of the three deployment models offered by the Network and Edge Reference System Architecture. The Cloud RA provides the means to develop and deploy applications in a Cloud Service Provider (CSP) environment (public cloud) while leveraging Intel® technology benefits. The Cloud RA uses a CSP's Intel based instances for running cloud-native applications in the Cloud. In this type of deployment, the CSP handles the control plane. The worker instances software stack additions are provided per the demands of specific deployable workloads. More tools and deployment targets will be added in future releases.

This document discusses the latest release of the Cloud RA and describes how the Network and Edge Cloud Reference System Architecture (Cloud RA) is provisioned, deployed, and used.

See Network and Edge Reference System Architectures Portfolio User Manual for an overview of the Network and Edge Reference System Architectures.

1.2 Version 23.07 Release Information

This release of the Cloud RA (23.07) is built upon previous Cloud RA releases. The release supports Microsoft* Azure Kubernetes Service (AKS) and Amazon Elastic Kubernetes Service* (Amazon EKS). Release highlights:

- Support for Cilium eBPF dataplane on Azure Kubernetes Service
- Updated scripts to configure and deploy Cloud RA
    - Default Kubernetes version: 1.26 (AKS), 1.26 (Amazon EKS)
    - Updated Terraform providers
    - Improved deployment speed

Experience Kits, the collaterals that explain in detail the technologies enabled in Cloud RA release 23.07, including benchmark information, are available in the following location: Network & Edge Platform Experience Kits. For NDA material, contact your regional Intel representative.

Table of Contents

Figures

Tables

Document Revision History

Revision	Date	Description
001	September 2022	Initial release.
002	December 2022	Updated for Reference System Architecture Release 22.11; Included deployment of Microsoft Azure Kubernetes Service
003	March 2023	Updated for Reference System Architecture Release 23.02; Support for "Cloud" CLI and CPU features (Intel® Software Guard Extensions (Intel® SGX) and static CPU Management Policy).
004	July 2023	Updated for Reference System Architecture Release 23.07; Support for Cilium eBPF Dataplane on Microsoft Azure* and updates to Kubernetes version and tools used to deploy on Azure and AWS*.

2 Overview

2.1.1 Amazon Web Services* (AWS) Deployment

The Cloud RA can be used to deploy a cluster on AWS using EKS (Elastic Kubernetes Service). Figure 1 shows the high-level view of this deployment.

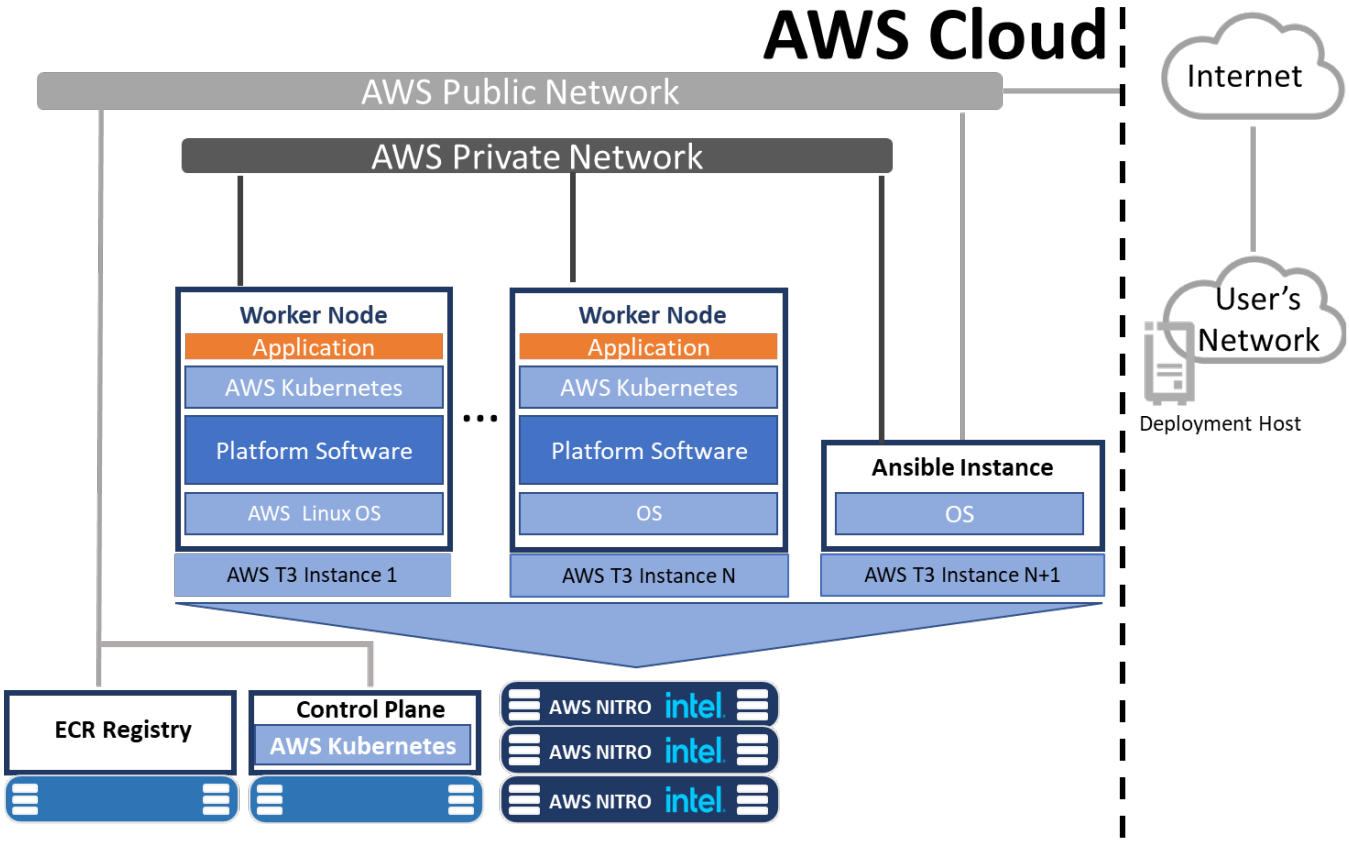

Figure 1. Cloud RA AWS High-level Deployment

In a nutshell, the user needs to use the deployment host, which is a computer that runs Linux OS (Ubuntu and Fedora have been tested) and is connected to the Internet and is able to access AWS servers. That computer needs to be further prepared based on the instructions provided in sections Prerequisites and Preparation of the Deployment Process.

When properly prepared, the user can use it to leverage the Cloud RA to deploy a user defined number of AWS EKS worker nodes using the user defined type of instance (T3.large by default). Cloud RA also automatically deploys the Control Plane of the EKS Cluster and an ECR Registry to allow the user to store containers and deploy them on the worker nodes.

The deployment can be performed in an automated way (by running one single command) or in a manual way (by running several steps), depending on the level of control the user requires. Automated way makes things simpler to the user, but manual allows the user to possibly make further additions and/or modifications on the deployment during different stages.

Either way, the user will need to prepare their deployment host as described in Preparation of the Deployment Host.

Next, the user will need to define what exactly they want to deploy (for example the number of the worker node instances). They will also need to define some parameters based on their environment (like their public IP address they use). More details are available in section Defining Deployment.

Then, the user can follow the Automated Deployment Process or the Manual Deployment Process.

For this release, AWS "Cloud" CLI or HashiCorp* Terraform can be used as the back-end tool for deployments.

2.1.2 Microsoft Azure* Deployment

The Cloud RA can be used to deploy a cluster on Microsoft Azure using AKS (Azure Kubernetes Services). Figure 2 shows the high-level view of this deployment.

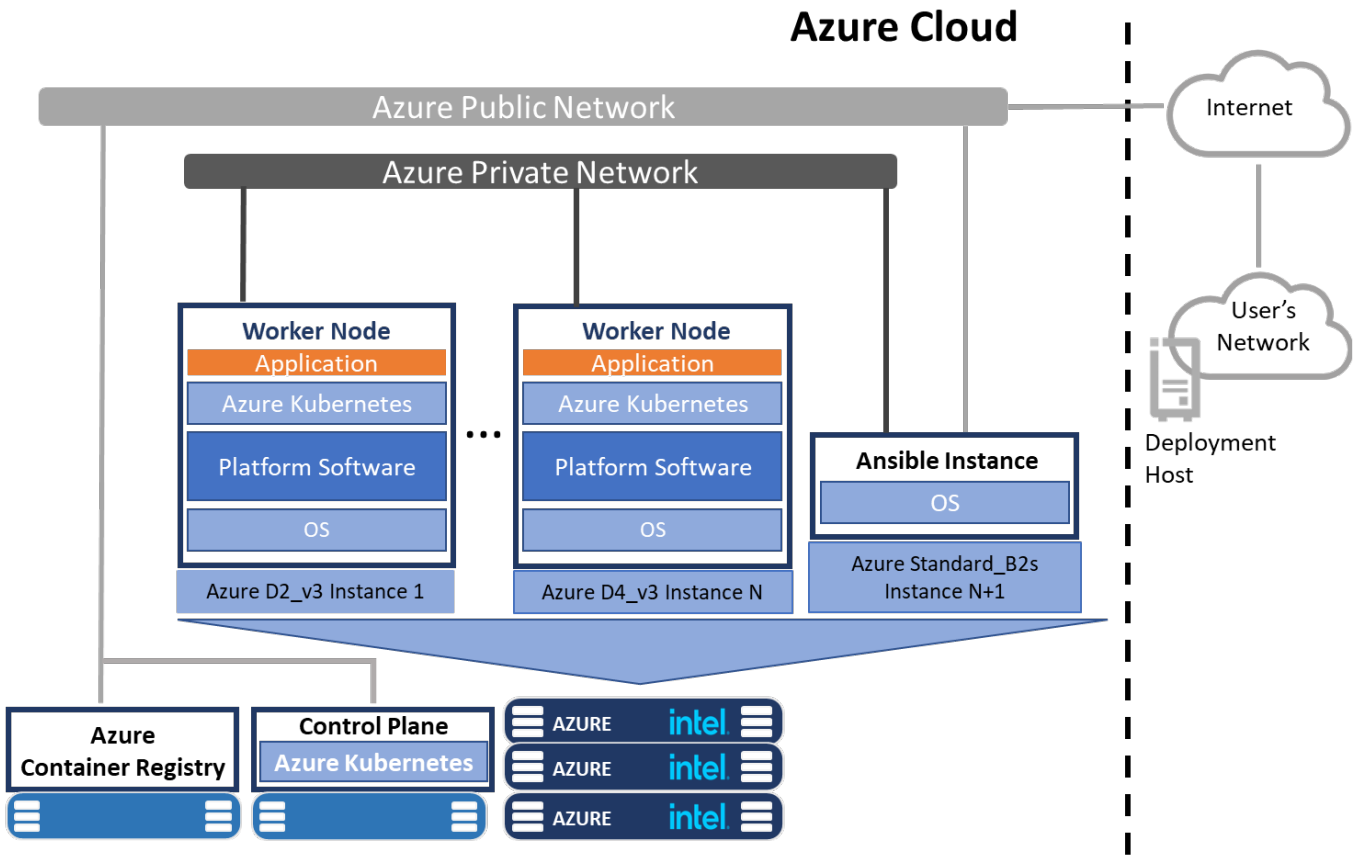

Figure 2. Cloud RA AKS High-level Deployment

Azure instances together with the Ansible* instance, the Azure Container Registry, and the control plane. For Azure deployment, follow the same Prerequisites and Preparation of the Deployment Process.

For this release, AWS "Cloud" CLI or HashiCorp* Terraform can be used as the back-end tool for deployments.

3 Preparation of the Deployment Host

3.1 Prerequisites

We are assuming the user to have an Ubuntu 22.04 development machine that has access to the Internet, with the software packages described in Software Requirements installed.

- AWS and/or Azure account is needed.
- Initially, the user needs to describe their AWS EKS or Azure AKS worker instances and their networking parameters and the software to be deployed.
- These choices need to be input in two *yaml* files as described in Defining Deployment.

3.2 Software Requirements

Install the following software before running Cloud RA:

- Python* 3.8
- AWS CLI 2.12.7 (https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html)
- Azure CLI 2.50.0 (https://learn.microsoft.com/en-us/cli/azure/install-azure-cli-linux?pivots=apt)
- Terraform 1.5.2 (https://www.terraform.io/downloads)
- Docker* 20.10.17

Install the following packages - If you are using a proxy server to configure your package manager to use it:

- python3
- python3-pip
- git
- netcat (needed specifically for SSH in a proxy environment)

Also, further guidelines for setting up the proxy can be found in Proxy Configurations.

Docker and HashiCorp Terraform require distro specific instructions. Please follow the ones needed for yours based on this instruction: https://docs.docker.com/engine/install/ .

Also, make sure to follow the instructions on the Proxy Configurations for some further configuration on Docker.

Finally, install HashiCorp Terraform from here: https://learn.hashicorp.com/tutorials/terraform/install-cli. Go to the "Install Terraform" section and choose your Linux distro. Follow the instructions on the website.

3.3 Docker* Configuration

Depending on your environment, a few additional steps might be needed to configure Docker.

If you plan to use a non-root account for deploying Cloud RA, you will first have to create a Docker group, and add the current user to the group:

```
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
```

At this point, either log out and log in back as the user, or run the following command:

```
$ newgrp docker
```

Make sure that the Docker service is running on the system:

```
$ sudo service docker start
(OR) $ sudo systemctl start docker.service
```

Verify that Docker is running, and can be accessed as the user:

```
$ docker ps
CONTAINER ID     COMMAND    CREATED    STATUS     PORTS      NAMES
```

The above steps, plus additional information about configuring Docker, can be found here:
https://docs.docker.com/engine/install/linux-postinstall/#configure-docker-to-start-on-boot

3.4 Proxy Configurations

If you are not using a proxy, skip this section and move to the next one.

The following steps are based on Ubuntu 22.04, in an environment where proxy is needed to reach the internet. Depending on your setup, the steps might slightly vary. The below examples should be valid in most cases.

Eventually, you will need to have working SSH, Docker, and remote connectivity with the package manager.

Start by exporting proxy variables for the system environment:

```
export http_proxy=http://<proxy server>:<proxy port>
export https_proxy=http://<proxy server>:<proxy port>
export no_proxy=<depends on your network>
export HTTP_PROXY=http://<proxy server>:<proxy port>
export HTTPS_PROXY=http://<proxy server>:<proxy port>
export NO_PROXY=<depends on your network>
```

Configure Docker to use proxy settings. On Ubuntu 22.04 (using Systemd), perform the following command:

```
$ sudo mkdir -p /etc/systemd/system/docker.service.d
$ sudo nano /etc/systemd/system/docker.service.d/http-proxy.conf

[Service]
Environment="HTTP_PROXY=http://<proxy server>:<proxy port>"
Environment="HTTPS_PROXY=https://<proxy server>:<proxy port>"

$ sudo systemctl daemon-reload
$ sudo systemctl restart docker
```

If you are using a setup without Systemd, such as Ubuntu on Windows Subsystem for Linux 2 (WSL2), the steps to configure Docker changes:

```
$ sudo nano /etc/default/docker
```

```
export http_proxy=http://<proxy server>:<proxy port>
export https_proxy=http://<proxy server>:<proxy port>
export ftp_proxy=http://<proxy server>:<proxy port>
export socks_proxy=<proxy server>:<proxy port>
export no_proxy=<depends on your network>
export HTTP_PROXY=http://<proxy server>:<proxy port>
export HTTPS_PROXY=http://<proxy server>:<proxy port>
export FTP_PROXY=http://<proxy server>:<proxy port>
export SOCKS_PROXY=<proxy server>:<proxy port>
export NO_PROXY=<depends on your network>

$ sudo service docker restart
```

If you also need a proxy for SSH connections, you can add a global proxy configuration, which will be used during Cloud RA deployment:

```
$ nano ~/.ssh/config

Host *
  ProxyCommand nc -X 5 -x <proxy server>:<proxy port> %h %p
```

4 Preparation of the Deployment Process

This section describes how to obtain the code for the Cloud RA and how to configure the AWS CLI that is being used during the deployment phase.

4.1 Obtaining Cloud RA

Cloud RA is provided as part of Network and Edge Reference System Architecture. It can be obtained by cloning the GitHub repository:

```
$ git clone https://github.com/intel/container-experience-kits.git
$ cd container-experience-kits
$ git checkout v23.07
```

Install the requirements for the container-experience-kits:

```
$ pip install -r requirements.txt
```

Now change to the directory containing the Cloud RA content:

```
$ cd container-experience-kits/cloud
```

Unless otherwise noted, the rest of this User Guide assumes that you are in this directory.

Install the additional requirements needed for Cloud RA:

```
$ pip install -r requirements.txt
```

Create a deployment directory in this folder, which will be used by Cloud RA scripts to assist with provisioning:

```
$ mkdir deployment
```

4.2 CSP Command Line Interface (CLI) Configuration

Before starting the deployment process, the system must be configured with valid CSP logins.

4.2.1 AWS Command Line Interface (CLI) Configuration

Before configuring the AWS CLI, make sure that you have a user prepared with sufficient permissions configured in AWS Identity and Access Management (IAM). For the default configuration method, you will need an access ID and access key for the user. The default region does not matter as the deployment configuration specifies the region.

To configure the AWS CLI, run the following command and provide details based on your AWS account:

```
$ aws configure
```

4.2.2 Azure Command Line Interface (CLI) Configuration

To configure the Azure CLI, run the following command:

```
$ az login
```

Follow the steps on the screen to login. A browser window will open asking you to sign in.

5 Prepare the Cloud Reference Architecture

The following steps will configure the Cloud Reference Architecture, which is needed before running the deployment scripts.

Start by returning to the container-experience-kits folder if you are currently in the cloud directory:

```
$ cd container-experience-kits
```

5.1 Generate Configuration Templates

Generate the configuration files for the profile being deployed. Here the Build-Your-Own configuration profile is used. Note the architecture (ARCH) option, which must match the earlier architecture of the CSP instance types being used. For the example instance types included here, the instances are either 1st Gen Intel® Xeon® Scalable processors (skl) or 2nd Gen Intel® Xeon® Scalable processors (clx), so the architecture should be set to "skl".

To generate the configuration files, run:

```
$ make cloud-profile PROFILE=build_your_own ARCH=skl
```

For more information on the Configuration Profiles for Container Bare Metal Reference System Architecture (BMRA), refer to *section 2.2 Configuration Profiles* in the Network and Edge Container Bare Metal Reference System Architecture User Guide.

5.2 Update Ansible* Host and Group Variables

After generating the configuration templates, two new folders and files will be created:

- group_vars/all.yml: Contains cluster level configuration options.
- host_vars/node1.yml: Contains node level configuration options. The options here will be used for all instances that are part of the CSP managed Kubernetes cluster.

The options in both these files are based on the choice of profile when the configuration templates were generated. Values can be updated manually at this point. However, be aware that changing any of them can cause issues with the deployment.

6 Defining Deployment

The user is required to define a "Hardware Configuration Profile" and a "Software Configuration Profile". Both are YAML files. The first describes the desired instance and networking deployment, and latter describes the software components and versions that will be deployed on the worker instances. Below are the schemas of the two files.

6.1 Hardware Configuration Profile

The hardware configuration, 'cwdf.yaml', specifies details of the public cloud environment to be deployed. This section provides examples for both AWS and Azure. These examples are also available in the 'cloud' directory of the Network and Edge Reference System Architecture source code.

Place the 'cwdf.yaml' file in the previously created deployment directory as follows:

```
container-experience-kits/cloud/deployment/cwdf.yaml
```

6.1.1 Example of cwdf.yaml for AWS

```
cloudProvider: aws
awsConfig:
  profile: default
  region: eu-central-1
  vpc_cidr_block: "10.21.0.0/16"
  extra_tags:
    Owner: "some_user"
  subnets:
    - name: "subnet_a"
      az: eu-central-1a
      cidr_block: "10.21.1.0/24"
    - name: "subnet_b"
      az: eu-central-1b
      cidr_block: "10.21.2.0/24"
  sg_whitelist_cidr_blocks: []
  ecr_repositories:
    - "example-repo"
  eks:
    kubernetes_version: "1.26"
    subnets: ["subnet_a", "subnet_b"]
    custom_ami: "ubuntu" # Comment out this line to use Amazon Linux 2 OS
    node_groups:
      - name: "default"
```

```
        instance_type: "t3.large"
        vm_count: 3
```

6.1.2 Example of cwdf.yaml for Azure

```
cloudProvider: azure
azureConfig:
  location: "West Europe"
  vpc_cidr_block: "10.21.0.0/16"
  extra_tags:
    Owner: "some_user"
  subnets:
    - name: "subnet_a"
      cidr_block: "10.21.1.0/24"
    - name: "subnet_b"
      cidr_block: "10.21.2.0/24"
    - name: "subnet_c"
      cidr_block: "10.21.3.0/24"
  sg_whitelist_cidr_blocks: []
  enable_proximity_placement: true
  aks:
    kubernetes_version: "1.26"
    cni: "kubenet" #  Possible values are: kubenet, cilium
    enable_sgx: false # Requires DCsv series instances in one of node pools
    default_node_pool:
      subnet_name: "subnet_a"
      vm_count: 1
      vm_size: "Standard_D2_v3"
    additional_node_pools:
      - name: "large"
        subnet_name: "subnet_b"
        vm_count: 1
        vm_size: "Standard_D4_v3"
      - name: "burstable"
        subnet_name: "subnet_c"
        vm_count: 1
        vm_size: "Standard_B2ms"
```

6.2 Software Configuration Profile

The software configuration, 'sw.yaml', specifies details of the software configuration to be used. Below are examples for both AWS and Azure. These examples are also available in the 'cloud/sw_deployment' directory of the Network and Edge Reference System Architecture source code.

The example shown in Section 6.2.1 works for both AWS and Azure. The only differences are the values in 'cloud_settings'. Examples for both AWS and Azure are also available in the 'cloud/sw_deployment' directory of the Network and Edge Reference System Architecture source code.

You can ignore several options in the software configuration (marked with x's) by using the automated deployment process. These options will be populated as part of the hardware provisioning.

Place the 'sw.yaml' file in the previously created deployment directory as follows:

```
container-experience-kits/cloud/deployment/sw.yaml
```

6.2.1 Example of sw.yaml for AWS and Azure

```
ansible_host_ip: xxx.xxx.xxx.xxx # This value can be ignored for automatic deployment
cloud_settings:
  provider: azure # Use the value from `profile` in the cwdf.yaml file
  region: "West Europe" # Use the value from `region` in the cwdf.yaml file
controller_ips:
  - 127.0.0.1
exec_containers: []
replicate_from_container_registry: https://registry.hub.docker.com
replicate_to_container_registry: xxxxx # This value can be ignored for automatic deployment
ssh_key: ../deployment/ssh/id_rsa # Leave this value as is for automatic deployment
worker_ips: # These values can be ignored for automatic deployment
  - xxx.xxx.xxx.xxx
```

```
  - xxx.xxx.xxx.xxx
  - xxx.xxx.xxx.xxx
```

For more information on the Configuration Profiles for Container Bare Metal Reference System Architecture (BMRA), refer to *section 2.2 Configuration Profiles* in the Network and Edge Container Bare Metal Reference System Architecture User Guide.

7 Deployment Process

This section describes two deployment process. It is strongly recommended that the user should not open ports on the Internet side. The deployments should have robust ACLs to prevent users (with Internet connection) from scanning and attacking the cloud deployment. With the standard deployment, port 22 (SSH) is opened on the Ansible instance to allow the user to access the deployment.

7.1 Automated Deployment Process

Using the automated deployment, the user can invoke the `deployer.py` to let it deploy all the instances and software described in the two files mentioned above.

There are two internal ways how to deploy infrastructure in cloud. First (default) tool is Terraform. Second tool is Bash script using cli tool of selected cloud provider.

The parameter is specified for the selection of the deployment tool:

```
--provisioner_tool <terraform/cloudcli>
```

The use of this parameter is optional. If this parameter is not specified, the Terraform tool is automatically selected.

The below command assumes that the configuration files have been added to the "deployment" directory:

```
$ python3 deployer.py deploy --deployment_dir=deployment
```

The automated deployment process will deploy an additional Ansible instance and container registry alongside the cluster.

After infrastructure provisioning is completed, deployer.py will run discovery and save the results in discovery results folder both on the local machine and on the Ansible host.

To destroy the deployment and shut down the cloud instances, run the following:

```
$ python3 deployer.py cleanup --deployment_dir=deployment
```

If a deployment has failed and the cleanup script fails to remove resources, the manual steps in 7.2.4 can be utilized.

After running the cleanup script or manually removing resources, some Terraform files will remain in the deployment directory. Either delete these files manually (including a few hidden files and folder) or remove the entire deployment directory. If you plan on redeploying later, it might be useful to back up the `cwdf.yaml` and `sw.yaml` files before deleting the directory.

7.2 Manual Deployment Process

The manual deployment has more steps but allows the user to have more control on each step of the deployment. The process can be summarized in the following four steps:

7.2.1 Step 1 - Prepare the Deployment

Similar to Defining Deployment, the user needs to define the hardware and software configuration profiles.

7.2.2 Step 2 - Create the Instances for the Cluster of Workers

7.2.2.1 Using Terraform

Generate the Terraform manifests from the Hardware Configuration Profile by invoking `cwfd.py`. The following assumes a directory "deployment" created under the main directory:

```
$ python3 cwdf.py generate-terraform \
  --cwdf_config=deployment/cwdf.yaml \
  --ssh_public_key=deployment/ssh/id_rsa.pub \ #SSH key should already exist
  --job_id=manual \
  --create_ansible_host=True \
  --create_container_registry=True \
  > deployment/main.tf
```

This should generate `main.tf`. Then, deploy this manifest using HashiCorp Terraform. Change directory into the deployment directory and start the deployment using the following command:

```
$ cd deployment
$ terraform init
$ terraform apply
```

When running "terraform apply" or "terraform destroy" in Azure environment you may run into the error: "Kubernetes cluster unreachable" or "Service Unavailable". This is caused due to a bug in Terraform Helm provider when the AKS authorization token expires.

To work around this issue, note AKS cluster name and Azure resource group name by running the following commands from the deployment directory:

```
$ terraform output aks_cluster_name
$ terraform output resource_group_name
```

Then, run the following commands to authenticate against AKS cluster:

```
$ az aks install-cli # If kubectl is not installed on your machine
$ az aks get-credentials -n <aks_cluster_name> -g <resource_group_name>
```

Now "terraform apply" or "terraform destroy" command should succeed. These steps are performed automatically when using automated deployment process.

7.2.2.2 Using CloudCLI

Generate the CloudCLI bash scripts from from the Hardware Configuration Profile by invoking `cwfd.py`. The following assumes a directory "deployment" created under the main directory:

```
python3 cwdf.py generate-cloudcli \
    --deployment_dir=./deployment/ \
    --cwdf_config=./deployment/cwdf.yaml \
    --job_id=manual \
    --create_ansible_host=True \
    --create_container_registry=True
```

This should generate two bash scripts. First with name `<cloud_provider>_cloudcli_provisioning.sh` is for deployment of the new cloud infrastructure. Second script is intended for cleaning.

The desired instances and their networking should be deployed. At this instance, user must note the following information:

- IP addresses of the worker instances
- IP address of the Ansible Instance
- AWS ECR or Azure Registry URL

7.2.3 Step 3 - Deploy the Software on Worker Nodes and Execute the Desired Containers

Update "deployment/sw.yaml" with the output information from section 7.2.2. The file should already contain all of the variables shown in section 6.2.1.

```
$ nano deployment/sw.yaml

(existing content skipped)
ansible_host_ip: < public IP of the Ansible instance (ansible_host_public_ip) >
replicate_to_container_registry: < URL of the ECR resource (ecr_url) >
ssh_key: < path to the private key matching the public key (ssh_public_key) used for generating
the Terraform manifests >
worker_ips: # List of private IPs of the EKS worker instances (eks_worker_instances.private_ip)
- < worker 1 private IP >
- < worker 2 private IP >
- < worker 3 private IP >
```

Run the sw_deployment_tool.py like this:

```
$ cd sw_deployment # from the cloud/ directory
$ python3 sw_deployment_tool.py
```

7.2.4 Step 4 - Clean up

7.2.4.1 Using Terraform

To shut down and delete the deployed instances by Terraform, use the following command:

```
$ cd deployment # from the cloud/ directory
$ terraform destroy
```

7.2.4.2 Using CloudCLI

To shut down and delete the deployed instances by CloudCLI, use the following command:

```
$ cd deployment # from the cloud/ directory
$ <cloud_provider>_cloudcli_cleanup.sh
```

8 Key Terms

Table 1 lists the key terms used throughout the portfolio. These terms are specific to Network and Edge Reference System Architectures Portfolio deployments.

Table 1. Terms Used

TERM	DESCRIPTION
Experience Kits	Guidelines delivered in the form of—manuals, user guides, application notes, solution briefs, training videos—for best-practice implementation of cloud native and Kubernetes technologies to ease developments and deployments.
Network and Edge Reference System Architectures Portfolio	A templated system-level blueprint for a range of locations in enterprise and cloud infrastructure with automated deployment tools. The portfolio integrates the latest Intel platforms and cloud-native technologies for multiple deployment models to simplify and accelerate deployments of key workloads across a service infrastructure.
Deployment Model	Provides flexibility to deploy solutions according to IT needs. The portfolio offers three deployment models: • **Container Bare Metal Reference System Architecture (BMRA)** – A deployment model of a Kubernetes cluster with containers on a bare metal platform. • **Virtual Machine Reference System Architecture (VMRA)**– A deployment model of a virtual cluster on a physical node. The virtual cluster can be a Kubernetes containers-based cluster. • **Cloud Reference System Architecture (Cloud RA)** – A deployment model of a cluster on a public Cloud Service Provider. The cluster can be Kubernetes with containers based.
Configuration Profiles	A prescribed set of components—hardware, software modules, hardware/software configuration specifications, designed for a deployment for specific workloads at a network location (such as Access Edge). Configuration Profiles define the components for optimized performance, usability, and cost per network location and workload needs. In addition, generic Configuration Profiles are available to developers for flexible deployments.
Reference System Architecture Flavor	An instance of reference architecture generated by implementing a Configuration Profile specification.
Ansible Playbook	A set of validated scripts that prepare, configure, and deploy a Reference System Architecture Flavor per Configuration Profile specification.
Configuration Profile Ansible Scripts	Automates quick, repeatable, and predictive deployments using Ansible playbooks. Various Configuration Profiles and Ansible scripts allow automated installations that are application-ready, depending on the workload and network location.
Kubernetes cluster	A deployment that installs at least one worker node running containerized applications. Pods are the components of the application workload that are hosted on worker nodes. Control nodes manage the pods and worker nodes.
Intel® Platforms	Prescribes Intel platforms for optimized operations. The platforms are based on 3rd Gen and 4th Gen Intel® Xeon® Scalable processors. The platforms integrate Intel® Ethernet 700 and 800 Series, Intel® QuickAssist Technology (Intel® QAT), Intel® Server GPU (Graphic Processor Unit), Intel® Optane™ technology, and more.

In addition to key terms, portfolio deployment procedures follow a hardware and software configuration taxonomy. Table 2 describes the taxonomy used throughout this document.

Table 2. Software Configuration Taxonomy

TERM	DESCRIPTION
Software Taxonomy	
TRUE	Feature is included and enabled by default
FALSE	Feature is included but disabled by default - can be enabled and configured by user
N/A	Feature is not included and cannot be enabled or configured

9 Reference Documentation

The *Network and Edge Reference System Architectures Portfolio User Manual* contains a complete list of reference documents. Additionally, a bare metal-based reference system architecture (BMRA) deployment allows creation of a Kubernetes cluster on multiple nodes. The *Network and Edge Container Bare Metal Reference System Architecture User Guide* provides information and installation instructions for a BMRA. A virtual machine-based reference architecture (VMRA) deployment allows creation of a Kubernetes cluster for a Configuration Profile on a virtualized infrastructure. The *Network and Edge Virtual Machine Reference System Architecture User Guide* provides information and installation instructions for a VMRA.

Collaterals, including technical guides and solution briefs that explain in detail the technologies enabled in Cloud RA release v23.07, are available in the following location: Network & Edge Platform Experience Kits.

Appendix A Cloud RA Release Notes

This section lists the notable changes from the previous releases, including new features, bug fixes, and known issues.[1]

A.1 Cloud RA 23.07 Release Updates

New Components/Features:

- Support for Cilium eBPF dataplane on Azure Kubernetes Service

Updates/Changes:

- Support for Kubernetes 1.25, 1.26 on Amazon EKS
- Support for Kubernetes 1.26 on AKS

Removed Support:

- Cilium using bring-your-own CNI deployment on AKS
- Kubernetes 1.22, 1.23 on Amazon EKS
- Kubernetes 1.23, 1.24 on AKS

A.2 Cloud RA 23.02 Release Updates

New Components/Features:

- Support for using Amazon Web Services (AWS) and Azure "Cloud" CLIs as an alternative to Terraform
- Azure Kubernetes Service (AKS) support for static CPU Management Policy and Intel® CPU Control Plane Plugin for Kubernetes
- Intel® Software Guard Extensions (Intel® SGX) on AKS

Updates/Changes:

- Supported Kubernetes versions updated for AKS and Amazon EKS
- Ubuntu images updated for AKS and Amazon EKS
- Ability to deploy more RA software components on Azure and AWS
  - Elasticsearch
  - Kibana

Removed Support:

- full_nfv profile

A.3 Cloud RA 22.11 Release Updates

New Components/Features:

- Support for Azure AKS deployments on top of previous support for AWS EKS
- Support for Cilium with kube-proxy and eBPF CNI on Azure
- Proximity Placement Groups for Azure
- Enhanced discovery mechanism
- Support for generating and deploying configuration profiles

Updates/Changes:

- Ability to deploy more software components on both Azure and AWS, namely:
  - Node Feature Discovery (NFD)
  - Userspace CNIs and OVS-DPDK
  - Telemetry (Prometheus, Telegraf, OpenTelemetry)
  - Jaeger
  - Istio
  - Multus
  - Traffic Analytics Development Kit (TADK) Web Application Firewall (WAF) workload

Known Limitations/Restrictions:

- Cilium CNI with eBPF is not as performant as kubenet CNI. This is because in Azure there is an extra VXLAN encapsulation that happens outside of the deployment control. As a result it has a negative impact on performance. The positive side is that Cilium CNI with either eBPF or kube-proxy allows network policy to be put in place.

A.4 Cloud RA 22.08 Release Updates

This is the first release of Cloud RA.

[1] Workloads and configurations. Results may vary.

Appendix B Abbreviations

The following abbreviations are used in this document.

Table 3. Abbreviations

TERM	DESCRIPTION
AWS	Amazon Web Services
AKS	Azure Kubernetes Services
ACL	Access Control List
BMRA	Bare Metal Reference Architecture
CLI	Command Line Interface
ECR	Elastic Container Registry
IAM	Identity and Access Management
I/O	Input/Output
K8s	Kubernetes
OS	Operating System
RA	Reference Architecture
SSH	Secure Shell Protocol
VMRA	Virtual Machine Reference Architecture

0723/DN/WIT/PDF 739730-004US