aws

User Guide

AWS Telco Network Builder

AWS Telco Network Builder: User Guide

Table of Contents

What is AWS TNB? 1
New to AWS? 2
Who is AWS TNB for? 2
AWS TNB features 2
Accessing AWS TNB 3
Pricing for AWS TNB 4
What's next 4
How AWS TNB works 5
Architecture 5
Integration 6
Quotas 7
AWS TNB concepts 8
Lifecycle of a network function 8
Use standardized interfaces 9
Network function packages 10
AWS TNB network service descriptors 11
Management and operations 12
Network service descriptors 13
Setting up AWS TNB 15
Sign up for an AWS account 15
Create a user with administrative access 16
Choose an AWS Region 17
Note the service endpoint 17
(Optional) Install the AWS CLI 18
Set up AWS TNB roles 18
Getting started with AWS TNB 20
Prerequisites 20
Create a function package 21
Create a network package 21
Create and instantiate a network instance 22
Clean up 22
Function packages 24
Create 21
View 25

Download a package ... 26
Delete a package ... 26
AWS TNB network packages ... 28
Create ... 21
View ... 29
Download ... 30
Delete ... 30
Network ... 32
Life cycle operations ... 32
Create ... 22
Instantiate ... 34
Update a function instance ... 35
Update a network instance ... 36
Considerations ... 36
Parameters that you can update ... 36
Updating a network instance ... 50
View ... 51
Terminate and delete ... 51
Network operations ... 53
View ... 53
Cancel ... 54
TOSCA reference ... 55
VNFD template ... 55
Syntax ... 55
Topology template ... 55
AWS.VNF ... 56
AWS.Artifacts.Helm ... 57
NSD template ... 58
Syntax ... 58
Using defined parameters ... 59
VNFD import ... 59
Topology template ... 60
AWS.NS ... 61
AWS.Compute.EKS ... 62
AWS.Compute.EKS.AuthRole ... 66
AWS.Compute.EKSManagedNode ... 67

AWS.Compute.EKSSelfManagedNode ... 74
AWS.Compute.PlacementGroup ... 80
AWS.Compute.UserData ... 82
AWS.Networking.SecurityGroup ... 83
AWS.Networking.SecurityGroupEgressRule ... 85
AWS.Networking.SecurityGroupIngressRule ... 88
AWS.Resource.Import ... 91
AWS.Networking.ENI ... 92
AWS.HookExecution ... 94
AWS.Networking.InternetGateway ... 95
AWS.Networking.RouteTable ... 98
AWS.Networking.Subnet ... 99
AWS.Deployment.VNFDeployment ... 102
AWS.Networking.VPC ... 104
AWS.Networking.NATGateway ... 105
AWS.Networking.Route ... 107
Common nodes ... 108
AWS.HookDefinition.Bash ... 108
Security ... 111
Data protection ... 111
Data handling ... 112
Encryption at rest ... 113
Encryption in transit ... 113
Inter-network traffic privacy ... 113
Identity and access management ... 113
Audience ... 113
Authenticating with identities ... 114
Managing access using policies ... 117
How AWS TNB works with IAM ... 120
Identity-based policy examples ... 126
Troubleshooting ... 140
Compliance validation ... 142
Resilience ... 143
Infrastructure security ... 144
Network connectivity security model ... 145
IMDS version ... 145

Monitoring 146
CloudTrail logs 146
AWS TNB event examples 148
Deployment tasks 149
Quotas 152
Document history 153

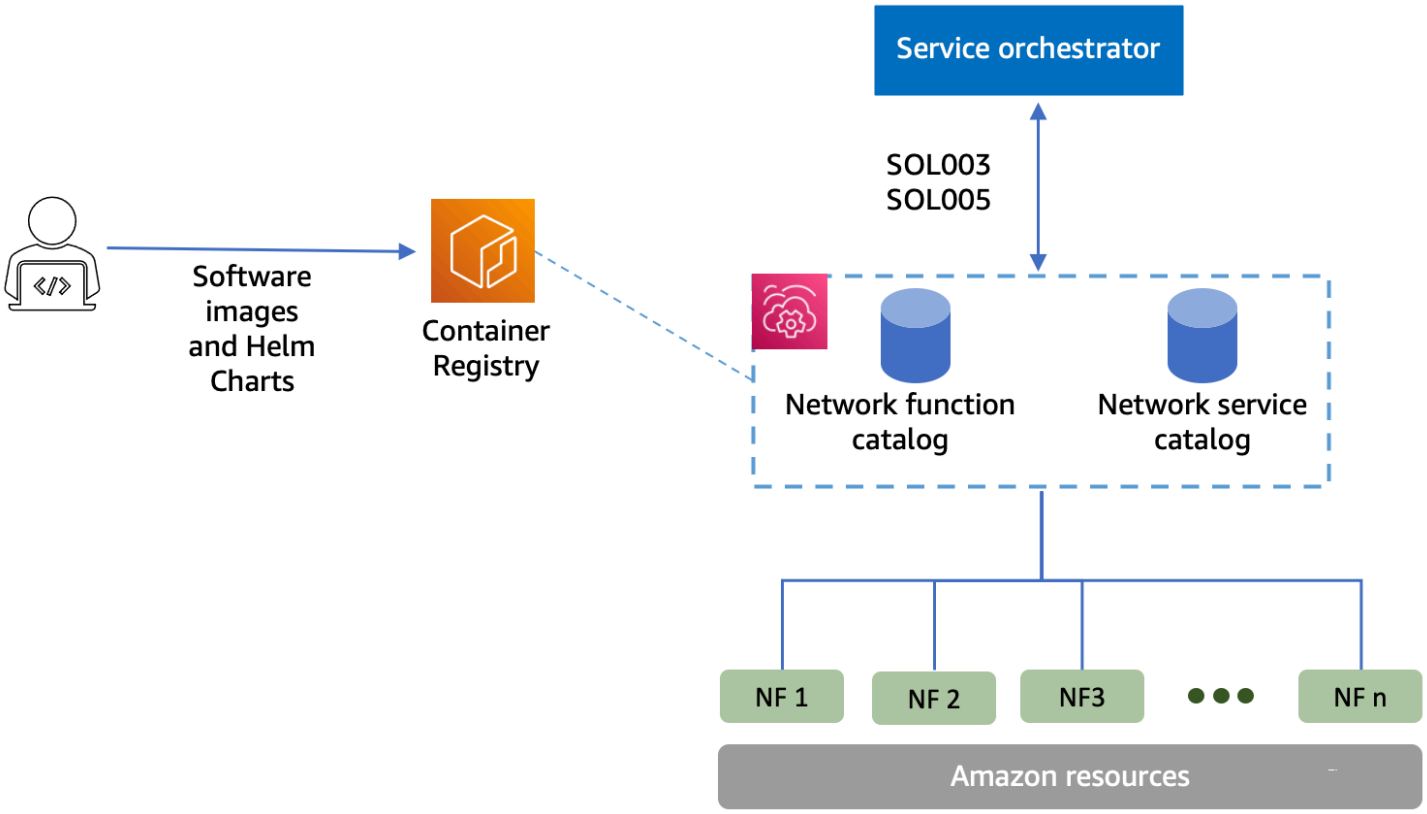
What is AWS Telco Network Builder?

AWS Telco Network Builder (AWS TNB) is an AWS service that provides communication service providers (CSPs) with an efficient way to deploy, manage, and scale 5G networks on AWS infrastructure.

With AWS TNB, you deploy scalable and secure 5G networks in the AWS Cloud using an image of your network in an automated manner. You don't need to learn new technologies, decide which compute service to use, or know how to provision and configure AWS resources.

Instead, you describe your network's infrastructure and provide the software images of network functions from your independent software vendor (ISV) partners. AWS TNB integrates with third-party service orchestrators and AWS services to automatically provision the necessary AWS infrastructure, deploy containerized network functions, and configure networking and access management to create a fully operational network service.

The following diagram illustrates the logical integrations between AWS TNB and service orchestrators to deploy network functions by using European Telecommunications Standards Institute (ETSI)-based standard interfaces.



Topics

- New to AWS?
- Who is AWS TNB for?
- AWS TNB features
- Accessing AWS TNB
- Pricing for AWS TNB
- What's next

New to AWS?

If you are new to AWS products and services, begin learning more with the following resources:

- Introduction to AWS
- Getting started with AWS

Who is AWS TNB for?

AWS TNB is for CSPs looking to take advantage of the cost-efficiencies, agility, and elasticity the AWS Cloud offers without writing and maintaining custom scripts and configurations to design, deploy, and manage network services. AWS TNB automatically provisions the necessary AWS infrastructure, deploys containerized network functions, and configures networking and access management to create fully operational network services based on the CSP-defined network service descriptors, and the network functions that the CSP wants to deploy.

AWS TNB features

The following are some of the reasons that a CSP would want to use AWS TNB:

Helps simplify tasks

Provide more efficiency to your network operations, such as deploying new services, updating and upgrading network functions, and changing network infrastructure topologies.

Integrates with orchestrators

AWS TNB integrates with popular third-party service orchestrators that are ETSI-compliant.

Scales

You can configure AWS TNB to scale underlying AWS resources to meet traffic demand, more efficiently perform network function updates, roll out network infrastructure topology changes, and reduce deployment time of new 5G services from days to hours.

Inspects and monitors AWS resources

AWS TNB lets you Inspect and monitor the AWS resources that support your network on a single dashboard, such as Amazon VPC, Amazon EC2, and Amazon EKS.

Supports service templates

AWS TNB allows you to create service templates for all telecom workloads (RAN, Core, IMS). You can create a new service definition, reuse an existing template, or integrate with a continuous integration and continuous delivery (CI/CD) pipeline to publish a new definition.

Tracks changes to network deployments

When you change the underlying configuration of a network function deployment, for example, changing the instance type of an Amazon EC2 instance type, you can track the changes in a repeatable and scalable manner. Doing so manually would require managing the state of the network, creating and deleting resources, and paying attention to the order of the changes needed. When you use AWS TNB to manage the lifecycle of your network function, you only make the changes to your network service descriptors describing the network function. AWS TNB will then automatically make the required changes in the correct order.

Simplifies the network function lifecycle

You can manage the first and all subsequent versions of a network function and specify when to upgrade. You can also manage your RAN, Core, IMS, and network applications in the same way.

Accessing AWS TNB

You can create, access, and manage your AWS TNB resources using any of the following interfaces:

- **AWS TNB console** — Provides a web interface for managing your network.
- **AWS TNB API** — Provides a RESTful API for performing AWS TNB actions. For more information see AWS TNB API Reference

- **AWS Command Line Interface (AWS CLI)** — Provides commands for a broad set of AWS services, including AWS TNB. It's supported on Windows, macOS, and Linux. For more information, see the AWS Command Line Interface User Guide.
- **AWS SDKs** – Provides language-specific APIs and completes many of the connection details. These include calculating signatures, handling request retries, and error handling. For more information, see AWS SDKs.

Pricing for AWS TNB

AWS TNB helps CSPs automate the deployment and management of their telecom networks on AWS. You pay for the following two dimensions when you use AWS TNB:

- By managed network function item (MNFI) hours.
- By number of API requests.

You also incur additional charges as you use other AWS services in conjunction with AWS TNB. For more information, see AWS TNB Pricing.

To view your bill, go to the **Billing and Cost Management Dashboard** in the AWS Billing and Cost Management console. Your bill contains links to usage reports that provide additional details about your bill. For more information about AWS account billing, see AWS Account Billing.

If you have questions concerning AWS billing, accounts, and events, contact AWS Support.

AWS Trusted Advisor is a service that you can use to help optimize the costs, security, and performance of your AWS environment. For more information, see AWS Trusted Advisor.

What's next

For more information about how to get started with AWS TNB, see the following topics:

- Setting up AWS TNB – Complete the prerequisite steps.
- Getting started with AWS TNB – Deploy your first network function, such as Centralized Unit (CU), Access and Mobility Management Function (AMF), User Plane Function (UPF), or a complete 5G Core.

How AWS TNB works

AWS TNB integrates with standardized end-to-end orchestrators and AWS resources to operate full 5G networks.

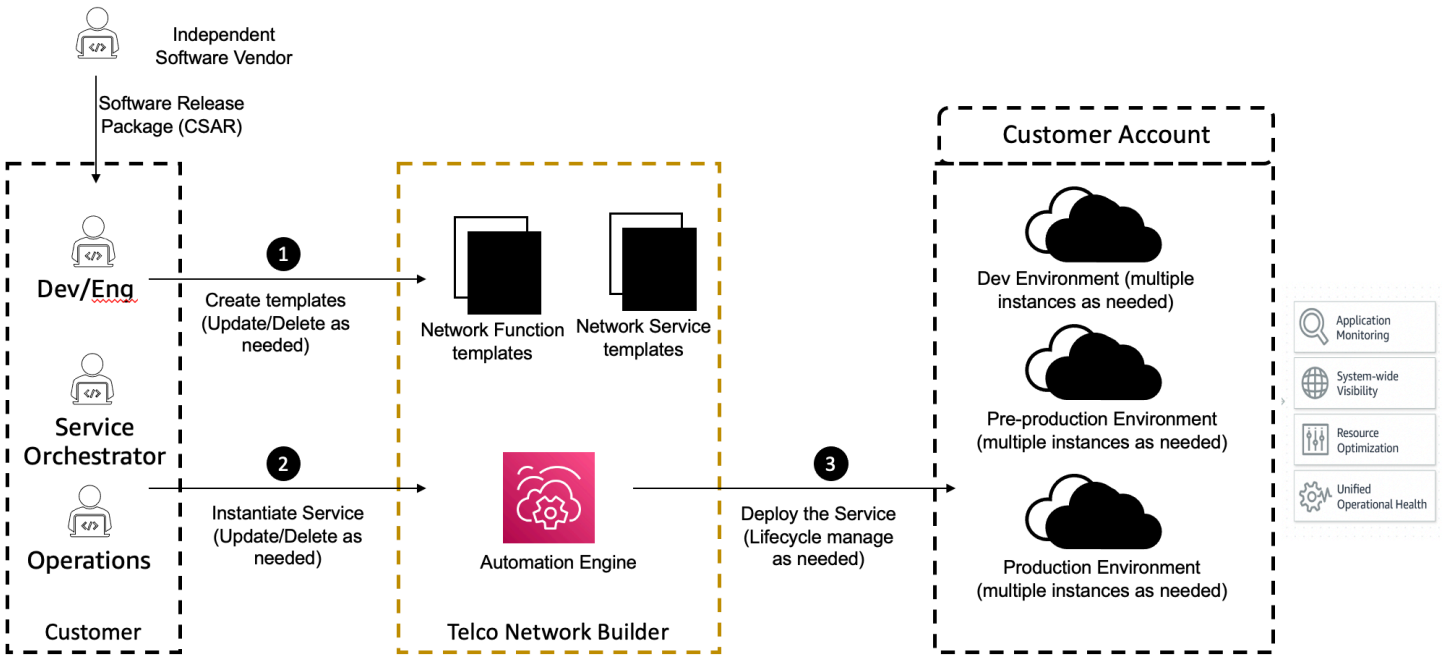
AWS TNB allows you to ingest network function packages and network service descriptors (NSDs) and provides you with the automation engine to operate your networks. You can use your end-to-end orchestrator and integrate with AWS TNB APIs, or use AWS TNB SDKs to build your own automation flow. For more information, see AWS TNB architecture .

Topics

- AWS TNB architecture
- Integration with AWS services
- AWS TNB resource quotas

AWS TNB architecture

AWS TNB provides you with the ability to perform lifecycle management operations through the AWS Management Console, AWS CLI, AWS TNB REST API, and SDKs. This allows the different CSP personas, such as members of the Engineering, Operations, and Programmatic System teams, to take advantage of AWS TNB. You create and upload a network function package as a Cloud Service Archive (CSAR) file. The CSAR file contains Helm charts, software images, and a Network Function Descriptor (NFD). You can use templates to repeatedly deploy multiple configurations of that package. You create network service templates defining the infrastructure and the network functions that you want to deploy. You can use parameter overrides to deploy different configurations in different locations. You can then instantiate a network, using the templates and deploy your network functions on AWS infrastructure. AWS TNB provides you with the visibility of your deployments.



Integration with AWS services

A 5G network is made up of a set of interconnected containerized network functions deployed across thousands of Kubernetes clusters. AWS TNB integrates with the following AWS services as telecom-specific APIs to create a fully operational network service:

- Amazon Elastic Container Registry (Amazon ECR) to store Independent Software Vendors (ISVs) network functions artifacts.
- Amazon Elastic Kubernetes Service (Amazon EKS) to set up clusters.
- Amazon VPC for networking constructs.
- Security groups using AWS CloudFormation.
- AWS CodePipeline for deployment targets across AWS Regions, AWS Local Zones, and AWS Outposts.
- IAM to define roles.
- AWS Organizations to control access to AWS TNB APIs.
- AWS Health Dashboard and AWS CloudTrail to monitor health and post metrics.

AWS TNB resource quotas

Your AWS account has default quotas, formerly referred to as limits, for each AWS service. Unless otherwise noted, each quota is specific to an AWS Region. You can request increases for some quotas, but not for all quotas.

To view the quotas for AWS TNB, open the Service Quotas console. In the navigation pane, choose **AWS services**, and select **AWS TNB**.

To request a quota increase, see Requesting a quota increase in the *Service Quotas User Guide*.

Your AWS account has the following quotas related to AWS TNB.

Resource quota	Description	Default value	Adjustable?
Network service instances	The maximum number of network service instances in one Region.	800	Yes
Concurrent ongoing network service operations	The maximum number of concurrent ongoing network service operations in one Region.	40	Yes
Network packages	The maximum number of network packages in one Region.	40	Yes
Function packages	The maximum number of function packages in one Region.	200	Yes

AWS TNB concepts

This topic describes essential concepts to help you start using AWS TNB.

Contents

- Lifecycle of a network function
- Use standardized interfaces
- Network function packages for AWS TNB
- Network service descriptors for AWS TNB
- Management and operations for AWS TNB
- Network service descriptors for AWS TNB

Lifecycle of a network function

AWS TNB helps you throughout the lifecycle of your network functions. The network function lifecycle includes the following stages and activities:

Planning

1. Plan your network by identifying the network functions to deploy.
2. Put the network function software images in a container image repository.
3. Create the CSAR packages to deploy or upgrade.
4. Use AWS TNB to upload the CSAR package that defines your network function (for example, CU AMF, and UPF), and integrate with a continuous integration and continuous delivery (CI/CD) pipeline that can help you create new versions of your CSAR package as new network function software images, or customer scripts, are available.

Configuration

1. Identify the information required for the deployment, such as compute type, network function version, IP information, and names of resources.
2. Use the information to create your network service descriptor (NSD).
3. Ingest NSDs that define your network functions and the resources required for the network function to instantiate.

Instantiation

1. Create the infrastructure required by the network functions.

2. Instantiate (or provision) the network function as defined in its NSD and start carrying traffic.
3. Validate the assets.

Production

During the lifecycle of the network function, you'll complete production operations, such as:

- Update the network function configuration, for example, update a value in the deployed network function.
- Update the network instance with a new network package and parameter values. For example, update the Amazon EKS `version` parameter in the network package.

Use standardized interfaces

AWS TNB integrates with European Telecommunications Standards Institute (ETSI) compliant service orchestrators allowing you to simplify the deployment of your network services. Service orchestrators can use AWS TNB SDKs, the CLI, or the APIs to initiate operations, such as instantiating or upgrading an network function to a new version.

AWS TNB supports the following specifications.

Specification	Release	Description
ETSI SOL001	v3.6.1	Defines standards for allowing TOSCA-based network function descriptors.
ETSI SOL002	v3.6.1	Defines models around network function management.
ETSI SOL003	v3.6.1	Defines standards for network function lifecycle management.
ETSI SOL004	v3.6.1	Defines CSAR standards for network function packages.
ETSI SOL005	v3.6.1	Defines standards for network service package and network service lifecycle management.
ETSI SOL007	v3.5.1	Defines standards for allowing TOSCA-based network service descriptors.

Network function packages for AWS TNB

With AWS TNB, you can store network function packages that comply with ETSI SOL001/SOL004 into a functions catalog. Then, you can upload Cloud Service Archive (CSAR) packages that contain artifacts describing your network function.

- Network function descriptor – Defines metadata for package onboarding and network function management
- Software Images – References network function Container Images. Amazon Elastic Container Registry (Amazon ECR) can act as your network function images repository.
- Additional Files – Use to manage the network function; for example, scripts and Helm charts.

The CSAR is a package defined by the OASIS TOSCA standard and includes a network/service descriptor that complies with the OASIS TOSCA YAML specification. For information about the required YAML specification, see TOSCA reference for AWS TNB.

The following is an example network function descriptor.

```
tosca_definitions_version: tnb_simple_yaml_1_0

topology_template:

  node_templates:

    SampleNF:
      type: tosca.nodes.AWS.VNF
      properties:
        descriptor_id: "SampleNF-descriptor-id"
        descriptor_version: "2.0.0"
        descriptor_name: "NF 1.0.0"
        provider: "SampleNF"
      requirements:
        helm: HelmChart

    HelmChart:
      type: tosca.nodes.AWS.Artifacts.Helm
      properties:
        implementation: "./SampleNF"
```

Network service descriptors for AWS TNB

AWS TNB stores network service descriptors (NSDs) about the network functions that you want to deploy and how you want to deploy them into the catalog. You can upload your YAML NSD file (`vnfd.yaml`), as described by ETSI SOL007 to include the following information:

- Network function that you want to deploy
- Networking instructions
- Compute instructions
- Lifecycle hooks (custom scripts)

AWS TNB supports ETSI standards for the modeling of resources, such as network, service, and function, in the TOSCA language. AWS TNB makes it more efficient for you to use AWS services by modeling them in a way that your ETSI-compliant service orchestrator can understand.

The following is a snippet of an NSD showing how to model AWS services. The network function will be deployed on an Amazon EKS cluster with Kubernetes version 1.27. The subnets for the applications are Subnet01 and Subnet02. You can then define the NodeGroups for your applications with an Amazon Machine Image (AMI), instance type, and autoscaling configuration.

```
tosca_definitions_version: tnb_simple_yaml_1_0

SampleNFEKS:
  type: tosca.nodes.AWS.Compute.EKS
  properties:
    version: "1.27"
    access: "ALL"
    cluster_role: "arn:aws:iam::${AWS::TNB::AccountId}:role/SampleClusterRole"
  capabilities:
    multus:
      properties:
        enabled: true
  requirements:
    subnets:
      - Subnet01
      - Subnet02

SampleNFEKSNode01:
  type: tosca.nodes.AWS.Compute.EKSManagedNode
```

```
  properties:
    node_role: "arn:aws:iam::${AWS::TNB::AccountId}:role/SampleNodeRole"
  capabilities:
    compute:
      properties:
        ami_type: "AL2_x86_64"
        instance_types:
          - "t3.xlarge"
        key_pair: "SampleKeyPair"
    scaling:
      properties:
        desired_size: 3
        min_size: 2
        max_size: 6
  requirements:
    cluster: SampleNFEKS
    subnets:
      - Subnet01
    network_interfaces:
      - ENI01
      - ENI02
```

Management and operations for AWS TNB

With AWS TNB, you can manage your network using standardized management operations in accordance with ETSI SOL003 and SOL005. You can use the AWS TNB APIs to perform lifecycle operations such as:

- Instantiating your network functions.
- Terminating your network functions.
- Updating your network functions to override Helm deployments.
- Updating an instantiated or updated network instance with a new network package and parameter values.
- Managing versions of your network functions packages.
- Managing versions of your NSDs.
- Retrieving information about your deployed network functions.

Network service descriptors for AWS TNB

A network service descriptor (NSD) is a `.yaml` file in a network package that uses the TOSCA standard to describe the network functions that you want to deploy, and the AWS infrastructure that you want to deploy the network functions on. To define your NSD and configure your underlying resources and network lifecycle operations, you must understand the NSD TOSCA Schema supported by AWS TNB.

Your NSD file is divided into the following parts:

1. **TOSCA definition version** – This is the first line of your NSD YAML file and contains the version information, shown in the following example.

```
tosca_definitions_version: tnb_simple_yaml_1_0
```

2. **VNFD** – The NSD contains the definition of the network function on which to perform lifecycle operations. Each network function must be identified by the following values:
   - A unique ID for `descriptor_id`. The ID must match the ID in the network function CSAR package.
   - A unique name for `namespace`. The name must be associated with a unique ID to more easily reference throughout your NSD YAML file, shown in the following example.

```
vnfds:
  - descriptor_id: "61465757-cb8f-44d8-92c2-b69ca0de025b"
    namespace: "amf"
```

3. **Topology template** – Defines the resources to be deployed, the network function deployment, and any customized scripts, such as lifecycle hooks. This is shown in the following example.

```
topology_template:

  node_templates:

    SampleNS:
      type: tosca.nodes.AWS.NS
      properties:
        descriptor_id: "<Sample Identifier>"
        descriptor_version: "<Sample nversion>"
        descriptor_name: "<Sample name>"
```

4. **Additional nodes** – Each modeled resource has sections for properties and requirements. The properties describe optional or mandatory attributes for a resource, such as the version. The requirements describe dependencies that must be provided as arguments. For example, to create an Amazon EKS Node Group Resource, it must be created within an Amazon EKS Cluster. This is shown in the following example.

```
SampleEKSNode:
  type: tosca.nodes.AWS.Compute.EKSManagedNode
  properties:
    node_role: "arn:aws:iam::${AWS::TNB::AccountId}:role/SampleRole"
  capabilities:
    compute:
      properties:
        ami_type: "AL2_x86_64"
        instance_types:
          - "t3.xlarge"
        key_pair: "SampleKeyPair"
    scaling:
      properties:
        desired_size: 1
        min_size: 1
        max_size: 1
  requirements:
    cluster: SampleEKS
    subnets:
      - SampleSubnet
    network_interfaces:
      - SampleENI01
      - SampleENI02
```

Setting up AWS TNB

Set up AWS TNB by completing the tasks described in this topic.

Tasks

- Sign up for an AWS account
- Create a user with administrative access
- Choose an AWS Region
- Note the service endpoint
- (Optional) Install the AWS CLI
- Set up AWS TNB roles

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Choose an AWS Region

To view the list of available Regions for AWS TNB, see the AWS Regional Services List. To view the list of endpoints for programmatic access, see AWS TNB endpoints in the *AWS General Reference*.

Note the service endpoint

To connect programmatically to an AWS service, you use an endpoint. In addition to the standard AWS endpoints, some AWS services offer FIPS endpoints in selected Regions. For more information, see AWS service endpoints.

Region Name	Region	Endpoint	Protocol	
US East (N. Virginia)	us-east-1	tnb.us-east-1.amazonaws.com	HTTPS	
US West (Oregon)	us-west-2	tnb.us-west-2.amazonaws.com	HTTPS	
Asia Pacific (Seoul)	ap-northeast-2	tnb.ap-northeast-2.amazonaws.com	HTTPS	
Asia Pacific (Sydney)	ap-southeast-2	tnb.ap-southeast-2.amazonaws.com	HTTPS	

Region Name	Region	Endpoint	Protocol
Canada (Central)	ca-central-1	tnb.ca-central-1.amazonaws.com	HTTPS
Europe (Frankfurt)	eu-central-1	tnb.eu-central-1.amazonaws.com	HTTPS
Europe (Paris)	eu-west-3	tnb.eu-west-3.amazonaws.com	HTTPS
Europe (Spain)	eu-south-2	tnb.eu-south-2.amazonaws.com	HTTPS
Europe (Stockholm)	eu-north-1	tnb.eu-north-1.amazonaws.com	HTTPS
South America (São Paulo)	sa-east-1	tnb.sa-east-1.amazonaws.com	HTTPS

(Optional) Install the AWS CLI

The AWS Command Line Interface (AWS CLI) provides commands for a broad set of AWS products, and is supported on Windows, macOS, and Linux. You can access AWS TNB using the AWS CLI. To get started, see the AWS Command Line Interface User Guide. For more information about the commands for AWS TNB, see tnb in the *AWS CLI Command Reference*.

Set up AWS TNB roles

You must create a IAM service role to manage different parts of your AWS TNB solution. AWS TNB service roles can make API calls to other AWS services, such as AWS CloudFormation, AWS

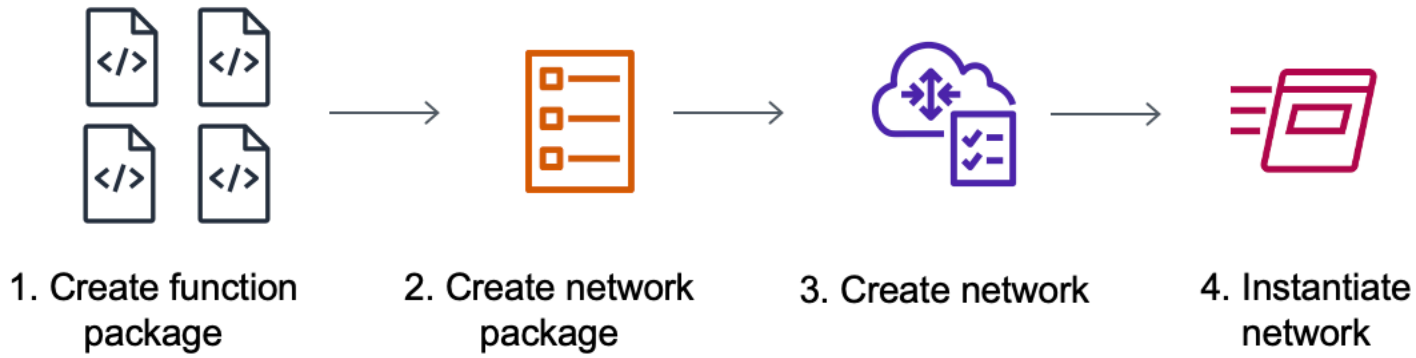
CodeBuild, and various compute and storage services, on your behalf, to instantiate and manage resources for your deployment.

For more information about the AWS TNB service role, see Identity and access management for AWS TNB.

Getting started with AWS TNB

This tutorial demonstrates how you use AWS TNB to deploy a network function, for example, the Centralized Unit (CU), Access and Mobility Management Function (AMF), or 5G User Plane Function (UPF).

The following diagram illustrates the deployment process:



Tasks

- Prerequisites
- Create a function package
- Create a network package
- Create and instantiate a network instance
- Clean up

Prerequisites

Before you can perform a successful deployment, you must have the following:

- An AWS Business Support plan.
- Permissions through IAM roles.
- A Network Function (NF) package that complies with ETSI SOL001/SOL004.
- Network Service Descriptor (NSD) templates that comply with ETSI SOL007.

You can use a sample function package or network package from the Sample packages for AWS TNB GitHub site.

Create a function package

A network function package is a Cloud Service Archive (CSAR) file. The CSAR file contains Helm charts, software images, and a Network Function Descriptor (NFD).

To create a function package

1. Open the AWS TNB console at https://console.aws.amazon.com/tnb/.
2. In the navigation pane, choose **Function packages**.
3. Choose **Create function package**.
4. Under **Upload function package**, choose **Choose files**, and upload each CSAR package as a `.zip` file. You can upload a maximum of 10 files.
5. (Optional) Under **Tags**, choose **Add new tag** and enter a key and value. You can use tags to search and filter your resources or track your AWS costs.
6. Choose **Next**.
7. Review the package details, and then choose **Create function package**.

Create a network package

A network package specifies the network functions that you want to deploy and how you want to deploy them into the catalog.

To create a network package

1. In the navigation pane, choose **Network packages**.
2. Choose **Create network package**.
3. Under **Upload network package**, choose **Choose files**, and upload each NSD as a `.zip` file. You can upload a maximum of 10 files.
4. (Optional) Under **Tags**, choose **Add new tag** and enter a key and value. You can use tags to search and filter your resources or track your AWS costs.
5. Choose **Next**.
6. Choose **Create network package**.

Create and instantiate a network instance

A network instance is a single network created in AWS TNB that can be deployed. You must create a network instance and instantiate it. When you instantiate a network instance, AWS TNB provisions the necessary AWS infrastructure, deploys containerized network functions, and configures networking and access management to create a fully operational network service.

To create and instantiate a network instance

1. In the navigation pane, choose **Networks**.
2. Choose **Create network instance**.
3. Enter a name and description for the network, and then choose **Next**.
4. Choose a network package. Verify the details and choose **Next**.
5. Choose **Create network instance**. The initial state is `Created`.

   The **Networks** page appears showing the new network instance in the `Not instantiated` state.
6. Select the network instance, choose **Actions** and **Instantiate**.

   The **Network instantiate** page appears.
7. Review details and update parameter values. Updates to the parameter values apply only to this network instance. The parameters in the NSD and VNFD packages do not change.
8. Choose **Instantiate network**.

   The **Deployment status** page appears.
9. Use the **Refresh** icon to track the deployment status of your network instance. You can also enable **Auto refresh** in the **Deployment tasks** section to track the progress of each task.

Clean up

You can now delete the resources that you created for this tutorial.

To clean up your resources

1. In the navigation pane, choose **Networks**.
2. Chose the ID of the network, and then choose **Terminate**.

3. When prompted for confirmation, enter the network ID, and then choose **Terminate**.
4. Use the **Refresh** icon to track the status of your network instance.
5. (Optional) Select the network, and choose **Delete**.

Function packages for AWS TNB

A function package is a .zip file in CSAR (Cloud Service Archive) format that contains a network function (an ETSI standard telecommunication application) and function package descriptor that uses the TOSCA standard to describe how the network functions should run on your network.

Tasks

- Create a function package in AWS TNB
- View a function package in AWS TNB
- Download a function package from AWS TNB
- Delete a function package from AWS TNB

Create a function package in AWS TNB

Learn how to create a function package in the AWS TNB network functions catalog. Creating a function package is the first step for creating a network in AWS TNB. After you've uploaded a function package, you can create a network package.

Console

To create a function package using the console

1. Open the AWS TNB console at https://console.aws.amazon.com/tnb/.
2. In the navigation pane, choose **Function packages**.
3. Choose **Create function package**.
4. Choose **Choose files** and upload each CSAR package as a `.zip` file. You can upload a maximum of 10 files.
5. Choose **Next**.
6. Review the package details.
7. Choose **Create function package**.

AWS CLI

To create a function package using the AWS CLI

1. Use the create-sol-function-package command to create a new function package:

```
aws tnb create-sol-function-package
```

2. Use the put-sol-function-package-content command to upload the function package content. For example:

```
aws tnb put-sol-function-package-content \
--vnf-pkg-id ^fp-[a-f0-9]{17}$ \
--content-type application/zip \
--file "fileb://valid-free5gc-udr.zip" \
--endpoint-url "https://tnb.us-west-2.amazonaws.com" \
--region us-west-2
```

View a function package in AWS TNB

Learn how to view the content of a function package.

Console

To view a function package using the console

1. Open the AWS TNB console at https://console.aws.amazon.com/tnb/.
2. In the navigation pane, choose **Function packages**.
3. Use search box to find the function package

AWS CLI

To view a function package using the AWS CLI

1. Use the list-sol-function-packages command to list your function packages.

```
aws tnb list-sol-function-packages
```

2. Use the get-sol-function-package command to view details about a function package.

```
aws tnb get-sol-function-package \
--vnf-pkg-id ^fp-[a-f0-9]{17}$ \
--endpoint-url "https://tnb.us-west-2.amazonaws.com" \
--region us-west-2
```

Download a function package from AWS TNB

Learn how to download a function package from the AWS TNB network functions catalog.

Console

To download a function package using the console

1. Open the AWS TNB console at https://console.aws.amazon.com/tnb/.
2. In the navigation pane on the left side of the console, choose **Function packages**.
3. Use search box to find the function package
4. Choose the function package
5. Choose **Actions**, **Download**.

AWS CLI

To download a function package using the AWS CLI

Use the get-sol-function-package-content command to download a function package.

```
aws tnb get-sol-function-package-content \
--vnf-pkg-id ^fp-[a-f0-9]{17}$ \
--accept "application/zip" \
--endpoint-url "https://tnb.us-west-2.amazonaws.com" \
--region us-west-2
```

Delete a function package from AWS TNB

Learn how to delete a function package from the AWS TNB network functions catalog. To delete a function package, the package must be in a disabled state.

Console

To delete a function package using the console

1. Open the AWS TNB console at https://console.aws.amazon.com/tnb/.
2. In the navigation pane, choose **Function packages**.
3. Use the search box to find the function package.
4. Choose a function package.
5. Choose **Actions**, **Disable**.
6. Choose **Actions**, **Delete**.

AWS CLI

To delete a function package using the AWS CLI

1. Use the update-sol-function-package command to disable a function package.

```
aws tnb update-sol-function-package --vnf-pkg-id ^fp-[a-f0-9]{17}$ ---
operational-state DISABLED
```

2. Use the delete-sol-function-package command to delete a function package.

```
aws tnb delete-sol-function-package \
--vnf-pkg-id ^fp-[a-f0-9]{17}$ \
--endpoint-url "https://tnb.us-west-2.amazonaws.com" \
--region us-west-2
```

Network packages for AWS TNB

A network package is a .zip file in CSAR (Cloud Service Archive) format defines the function packages you want to deploy and the AWS infrastructure you want to deploy them on.

Tasks

- Create a network package in AWS TNB
- View a network package in AWS TNB
- Download a network package from AWS TNB
- Delete a network package from AWS TNB

Create a network package in AWS TNB

A network package consists of a network service descriptor (NSD) file (required) and any additional files (optional), such as scripts specific to your needs. For example, if you have multiple function packages in your network package, you can use the NSD to define which network functions should run in certain VPCs, subnets, or Amazon EKS clusters.

Create a network package after creating function packages. Once you've created a network package, you need to create a network instance.

Console

To create a network package using the console

1. Open the AWS TNB console at https://console.aws.amazon.com/tnb/.
2. In the navigation pane, choose **Network packages**.
3. Choose **Create network package**.
4. Choose **Choose files** and upload each NSD as a `.zip` file. You can upload a maximum of 10 files.
5. Choose **Next**.
6. Review the package details.
7. Choose **Create network package**.

AWS CLI

To create a network package using the AWS CLI

1. Use the create-sol-network-package command to create a network package.

```
aws tnb create-sol-network-package
```

2. Use the put-sol-network-package-content command to upload network package content. For example:

```
aws tnb put-sol-network-package-content \
--nsd-info-id ^np-[a-f0-9]{17}$ \
--content-type application/zip \
--file "fileb://free5gc-core-1.0.9.zip" \
--endpoint-url "https://tnb.us-west-2.amazonaws.com" \
--region us-west-2
```

View a network package in AWS TNB

Learn how to view the content of a network package.

Console

To view a network package using the console

1. Open the AWS TNB console at https://console.aws.amazon.com/tnb/.
2. In the navigation pane, choose **Network packages**.
3. Use the search box to find the network package.

AWS CLI

To view a network package using the AWS CLI

1. Use the list-sol-network-packages command to list your network packages.

```
aws tnb list-sol-network-packages
```

2. Use the get-sol-network-package command to view details about a network package.

```
aws tnb get-sol-network-package \
--nsd-info-id ^np-[a-f0-9]{17}$ \
--endpoint-url "https://tnb.us-west-2.amazonaws.com" \
--region us-west-2
```

Download a network package from AWS TNB

Learn how to download a network package from the AWS TNB network service catalog.

Console

To download a network package using the console

1. Open the AWS TNB console at https://console.aws.amazon.com/tnb/.
2. In the navigation pane, choose **Network packages**.
3. Use the search box to find the network package
4. Choose the network package.
5. Choose **Actions**, **Download**.

AWS CLI

To download a network package using the AWS CLI

- Use the get-sol-network-package-content command to download a network package.

```
aws tnb get-sol-network-package-content \
--nsd-info-id ^np-[a-f0-9]{17}$ \
--accept "application/zip" \
--endpoint-url "https://tnb.us-west-2.amazonaws.com" \
--region us-west-2
```

Delete a network package from AWS TNB

Learn how to delete a network package from the AWS TNB network service catalog. To delete a network package, the package must be in a disable state.

Console

To delete a network package using the console

1. Open the AWS TNB console at https://console.aws.amazon.com/tnb/.
2. In the navigation pane, choose **Network packages**.
3. Use the search box to find the network package
4. Choose network package
5. Choose **Actions**, **Disable**.
6. Choose **Actions**, **Delete**.

AWS CLI

To delete a network package using the AWS CLI

1. Use the update-sol-network-package command to disable a network package.

```
aws tnb update-sol-network-package --nsd-info-id ^np-[a-f0-9]{17}$ --nsd-
operational-state DISABLED
```

2. Use the delete-sol-network-package command to delete a network package.

```
aws tnb delete-sol-network-package \
--nsd-info-id ^np-[a-f0-9]{17}$ \
--endpoint-url "https://tnb.us-west-2.amazonaws.com" \
--region us-west-2
```

Network instances for AWS TNB

A network instance is a single network created in AWS TNB that can be deployed.

Tasks

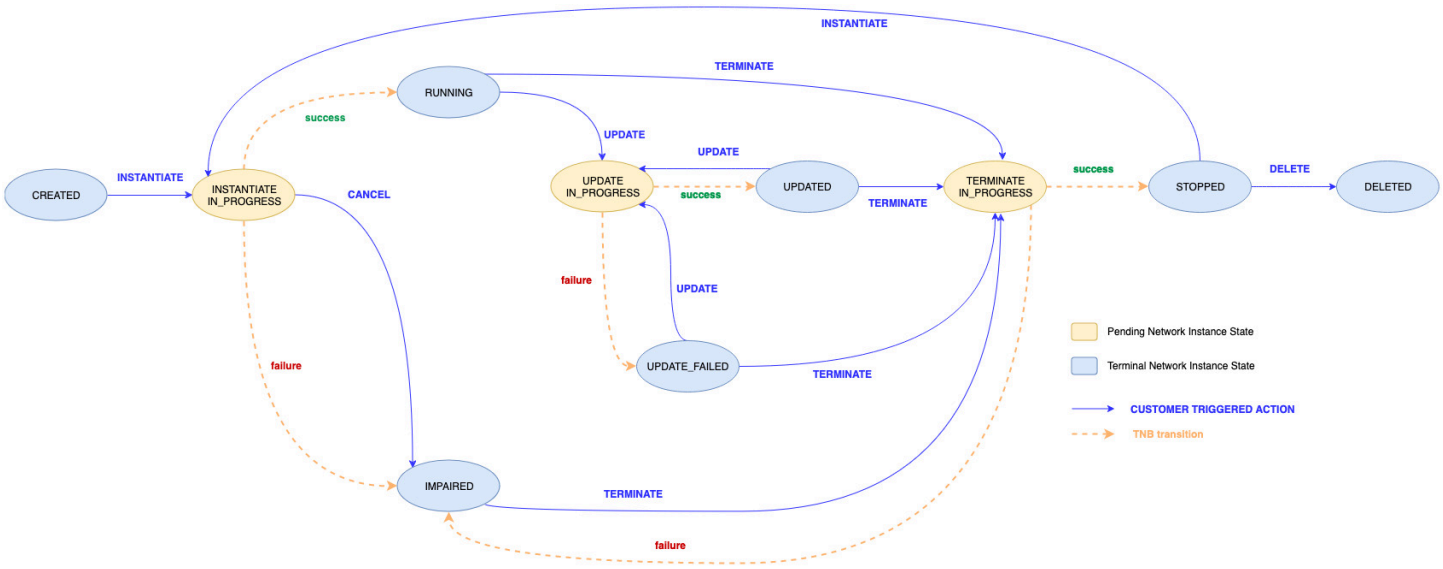
- Life cycle operations of a network instance
- Create a network instance using AWS TNB
- Instantiate a network instance using AWS TNB
- Update a function instance in AWS TNB
- Update a network instance in AWS TNB
- View a network instance in AWS TNB
- Terminate and delete a network instance from AWS TNB

Life cycle operations of a network instance

AWS TNB allows you to easily manage your network using standardized management operations inline with ETSI SOL003 and SOL005. You can perform the following life-cycle operations:

- Create the network
- Instantiate the network
- Update the network function
- Update the network instance
- View network details and status
- Terminate the network

The following image shows the network management operations:



Create a network instance using AWS TNB

You create a network instance after creating a network package. After you create a network instance, instantiate it.

Console

To create a network instance using the console

1. Open the AWS TNB console at https://console.aws.amazon.com/tnb/.
2. In the navigation pane, choose **Networks**.
3. Choose **Create network instance**.
4. Enter a name and description for the instance and then choose **Next**.
5. Select the network package, verify the details, and choose **Next**.
6. Choose **Create network instance**.

   The new network instance appears on the **Networks** page. Next, instantiate this network instance.

AWS CLI

To create a network instance using the AWS CLI

- Use the create-sol-network-instance command to create a network instance.

```
aws tnb create-sol-network-instance --nsd-info-id ^np-[a-f0-9]{17}$ --ns-name
 "SampleNs" --ns-description "Sample"
```

Next, instantiate this network instance.

Instantiate a network instance using AWS TNB

After you create a network instance, you must instantiate it. When you instantiate a network instance, AWS TNB provisions the necessary AWS infrastructure, deploys containerized network functions, and configures networking and access management to create a fully operational network service.

Console

To instantiate a network instance using the console

1. Open the AWS TNB console at https://console.aws.amazon.com/tnb/.
2. In the navigation pane, choose **Networks**.
3. Select the network instance that you want to instantiate.
4. Choose **Actions** and then **Instantiate**.
5. On the **Instantiate network** page, review details and optionally, update parameter values.

   Updates to the parameter values apply only to this network instance. The parameters in the NSD and VNFD packages do not change.
6. Choose **Instantiate network**.

   The **Deployment status** page appears.
7. Use the **Refresh** icon to track the deployment status of your network instance. You can also enable **Auto refresh** in the **Deployment tasks** section to track the progress of each task.

   When the deployment status changes to `Completed`, the network instance is instantiated.

AWS CLI

To instantiate a network instance using the AWS CLI

1. Use the instantiate-sol-network-instance command to instantiate the network instance.

```
aws tnb instantiate-sol-network-instance --ns-instance-id ^ni-[a-f0-9]{17}$ --
additional-params-for-ns "{\"param1\": \"value1\", \"param2\": \"value2\"}"
```

2. Next, view the network operation status.

Update a function instance in AWS TNB

After a network instance is instantiated, you can update a function package in the network instance.

Console

To update a function instance using the console

1. Open the AWS TNB console at https://console.aws.amazon.com/tnb/.
2. In the navigation pane, choose **Networks**.
3. Select the network instance. You can update a network instance only if its state is `Instantiated`.

   The network instance page appears.
4. From the **Functions** tab, select the function instance to update.
5. Choose **Update**.
6. Enter your update overrides.
7. Choose **Update**.

AWS CLI

Use the CLI to update a function instance

Use the update-sol-network-instance command with the `MODIFY_VNF_INFORMATION` update type to update a function instance in a network instance.

```
aws tnb update-sol-network-instance --ns-instance-id ^ni-[a-f0-9]{17}$ --update-type
 MODIFY_VNF_INFORMATION --modify-vnf-info ...
```

Update a network instance in AWS TNB

After a network instance is instantiated, you might need to update the infrastructure or application. To do so, you update the network package and parameter values for the network instance and deploy the update operation to apply the changes.

Considerations

- You can update a network instance that is in the `Instantiated` or `Updated` state.
- When you update a network instance, the `UpdateSolNetworkService` API uses the new network package and parameter values to update the topology of the network instance.
- AWS TNB verifies that the number of NSD and VNFD parameters in the network instance does not exceed 200. This limit is enforced to protect from bad actors passing erroneous or huge payloads that affect the service.

Parameters that you can update

You can update the following parameters when you update an instantiated network instance:

Parameter	Description	Example: Before	Exam After

Amazon EKS cluster version	You can update the value for the Amazon EKS cluster control plane `version` parameter to the next minor version. You cannot downgrade the version. Worker nodes are not updated.	`EKSCluster:`
		`  type: tosca.nod es.AWS.Compute.EKS`
		`  properties:`
		`  version: "1.28"`
			`EKSC r:`
			`typ tos es.A mput`

Parameter	Description	Example: Before	Exam After
			pro s: ver "1.

Parameter	Description	Example: Before	Exam After

Scaling properties	You can update the scaling properties of the EKSManagedNode and EKSSelfManagedNode TOSCA nodes.	`EKSNodeGroup01:`
		`  ...`
		`  scaling:`
		`  properties:`
		`  desired_size: 1`
		`  min_size: 1`
		`  max_size: 1`
			EKSNoup0
			...
			sca
			pros:
			desize:

Parameter	Description	Example: Before	Exam After
			mir max

Parameter	Description	Example: Before	Exam After

Amazon EBS CSI plugin properties	You can enable or disable the Amazon EBS CSI plugin on your Amazon EKS clusters. You can also change the plugin version.	`EKSCluster:`
		`  capabilities:`
		`  ...`
		`  ebs_csi:`
		`  properties:`
		`  enabled: false`
			EKSC
			r:
			cap
			ies:
			...
			ebs
			pro
			s:
			ena
			ver
			"v1
			e
			ksbu
			"

Parameter	Description	Example: Before	Exam After
VNF	You can reference the VNFs in the NSD and deploy them to the cluster created in NSD using VNFDeployment TOSCA node. As part of the update, you will be able to add, update, and delete VNFs to the network.	vnfds: - descriptor_id: "43c012fa-2616-41a8-a833-0dfd4c5a049e " namespace: " vnf1" - descriptor_id: "64222f98-ecd6-4871-bf94-7354b53f3ee5 " namespace: "vnf2" // Deleted VNF ... SampleVNF1HelmDeploy: type: tosca.nodes.AWS.Deployment.VNFDeployment requirements: cluster: EKSCluster vnfs: - vnf1.SampleVNF1 - vnf2.SampleVNF2	vnfd - des r_id "59 79e9 - be53 2ad0 " nam : "vr Upd VNF - des r_id "b7 839d -916 a166 " nam : "vr Add VNF .... Sa mple

Parameter	Description	Example: Before	Exam After
			elmD : typ tos es.A ploy VNFD ment req nts: clu EKS r vnf

Parameter	Description	Example: Before	Example: After
			- v leVN - v leVN

Parameter	Description	Example: Before	Exam After

Hooks	To run life cycle operations before and after you create a network function, add the `pre_create` and `post_create` hooks to the `VNFDeployment` node.
	
	In this example, the `PreCreateHook` hook will run before `vnf3.SampleVNF3` is instantiated and the `PostCreateHook` hook will run after `vnf3.SampleVNF3` is instantiated.
		`vnfds:`
		`  - descriptor_id: "43c012fa-2616-41a8-a833-0dfd4c5a049e "`
		`    namespace: " vnf1"`
		`  - descriptor_id: "64222f98-ecd6-4871-bf94-7354b53f3ee5 "`
		`    namespace: " vnf2"`
		`...`
		`SampleVNF1HelmDeploy:`
		`  type: tosca.nodes.AWS.Deployment.VNFDeployment`
		`  requirements:`
		`    cluster: EKSCluster`
		`    vnfs:`
		`      - vnf1.SampleVNF1`
		`      - vnf2.SampleVNF2  // Removed during update`
			vnfd
			
			-
			des
			r_id
			"43
			2616
			-
			a833
			d4c5
			"
			
			nam
			:
			"vn
			
			-
			des
			r_id
			"b7
			839d
			-916
			a166
			"
			
			nam
			:
			"vn
			....
			S
			ampl
			Helm
			y:
			
			typ
			tos

Parameter	Description	Example: Before	Exam After
			es.A ploy VNFD ment rec nts: clu EKS r vnf - v leVN No cha to thi fur as the nam and uui rem the sam

Parameter	Description	Example: Before	Exam After
			- v leVN New VNF as the nam , vnf was not pre y pre int s: Hoo pos te: eHoo pre e: Hook

Parameter	Description	Example: Before	Exam After

Hooks	To run life cycle operations before and after you update a network function, you can add the `pre_update` hook and the `post_update` hook to the `VNFDeployment` node.
	
	In this example, `PreUpdate Hook` will run before `vnf1.SampleVNF1` is updated and `PostUpdat eHook` will run after `vnf1.SampleVNF1` is updated to the `vnf` package indicated by the updated `uuid` for the namespace vnf1.
		`vnfds:`
		`  - descriptor_id:`
		`  "43c012fa-2616-41a8-a833-0dfd4c5a049e  "`
		`  namespace: " vnf1"`
		`  - descriptor_id:`
		`  "64222f98-ecd6-4871-bf94-7354b53f3ee5  "`
		`  namespace: " vnf2"`
		`...`
		
		`SampleVNF1HelmDeploy:`
		`  type: tosca.nod es.AWS.Deployment. VNFDeployment`
		`  requirements:`
		`  cluster: EKSCluster`
		`  vnfs:`
		`  - vnf1.SampleVNF1`
		`  - vnf2.Samp leVNF2`
			`vnfd`
			
			`  -`
			`  des r_id`
			`  "0e bd87 -`
			`  b8a1 4666`
			`  "`
			
			`  nam :`
			`  "vn`
			
			`  -`
			`  des r_id`
			`  "64 ecd6 -`
			`  bf94 4b53`
			`  "`
			
			`  nam :`
			`  "vn ...`
			`S ampl Helm y:`
			
			`  typ`

Parameter	Description	Example: Before	Exam After
			tos es.A ploy VNFD ment rec nts: clu EKS r vnf - v leVN A VNF upd as the uui cha for nam "vr - v

Parameter	Description	Example: Before	Exam After
			leVN No cha to thi fur as nam and uui rem the sam int s: Hoo pre e: Hook pos te: eHoo

Updating a network instance

Console

To update a network instance using the console

1. Open the AWS TNB console at https://console.aws.amazon.com/tnb/.
2. In the navigation pane, choose **Networks**.
3. Select the network instance. You can update a network instance only if its state is `Instantiated` or `Updated`.
4. Choose **Actions** and **Update**.

   The **Update instance** page appears with the network details and a list of parameters in the current infrastructure.
5. Choose a new network package.

   The parameters in the new network package appear in the **Updated parameters** section.
6. Optionally, update parameter values in the **Updated parameters** section. For the list of parameter values you can update, see Parameters that you can update.
7. Choose **Update network**.

   AWS TNB validates the request and starts the deployment. The **Deployment status** page appears.
8. Use the **Refresh** icon to track the deployment status of your network instance. You can also enable **Auto refresh** in the **Deployment tasks** section to track the progress of each task.

   When the deployment status changes to `Completed`, the network instance is updated.
9. - If validation fails, the network instance remains in the same state as it was before you requested the update - either `Instantiated` or `Updated`.
   - If the update fails, the network instance state shows `Update failed`. Choose the link for each failed task to determine the reason.
   - If the update succeeds, the network instance state shows `Updated`.

AWS CLI

Use the CLI to update a network instance

Use the update-sol-network-instance command with the UPDATE_NS update type to update a network instance.

```
aws tnb update-sol-network-instance --ns-instance-id ^ni-[a-f0-9]{17}$ --
update-type UPDATE_NS --update-ns "{\"nsdInfoId\":\"^np-[a-f0-9]{17}$\",
 \"additionalParamsForNs\": {\"param1\": \"value1\"}}"
```

View a network instance in AWS TNB

Learn how to view a network instance.

Console

To view a network instance using the console

1. Open the AWS TNB console at https://console.aws.amazon.com/tnb/.
2. In the navigation pane, choose **Network instances**.
3. Use the search box to find the network instance.

AWS CLI

To view a network instance using the AWS CLI

1. Use the list-sol-network-instances command to list your network instances.

   ```
   aws tnb list-sol-network-instances
   ```

2. Use the get-sol-network-instance command to view details about a specific network instance.

   ```
   aws tnb get-sol-network-instance --ns-instance-id ^ni-[a-f0-9]{17}$
   ```

Terminate and delete a network instance from AWS TNB

To delete a network instance, the instance must be in a terminated state.

Console

To terminate and delete a network instance using the console

1. Open the AWS TNB console at https://console.aws.amazon.com/tnb/.
2. In the navigation pane, choose **Networks**.
3. Select the ID of the network instance.
4. Choose **Terminate**.
5. When prompted for confirmation, enter the ID and choose **Terminate**.
6. Refresh to track the status of your network instance.
7. (Optional) Select the network instance and choose **Delete**.

AWS CLI

To terminate and delete a network instance using the AWS CLI

1. Use the terminate-sol-network-instance command to terminate a network instance.

```
aws tnb terminate-sol-network-instance --ns-instance-id ^ni-[a-f0-9]{17}$
```

2. (Optional) Use the delete-sol-network-instance command to delete a network instance.

```
aws tnb delete-sol-network-instance --ns-instance-id ^ni-[a-f0-9]{17}$
```

Network operations for AWS TNB

A network operation is any operation that is done to your network, such as network instance instantiation or termination.

Tasks

- View a AWS TNB network operation
- Cancel a AWS TNB network operation

View a AWS TNB network operation

View the details of a network operation, including the tasks involved in the network operation and the status of the tasks.

Console

To view a network operation using the console

1. Open the AWS TNB console at https://console.aws.amazon.com/tnb/.
2. In the navigation pane, choose **Network instances**.
3. Use the search box to find the network instance.
4. On the **Deployments** tab, choose the network operation.

AWS CLI

To view a network operation using the AWS CLI

1. Use the list-sol-network-operations command to list all network operations.

```
aws tnb list-sol-network-operations
```

2. Use the get-sol-network-operation command to view details about a network operation.

```
aws tnb get-sol-network-operation --ns-lcm-op-occ-id ^no-[a-f0-9]{17}$
```

Cancel a AWS TNB network operation

Learn how to cancel a network operation.

Console

To cancel a network operation using the console

1. Open the AWS TNB console at https://console.aws.amazon.com/tnb/.
2. In the navigation pane, choose **Networks**.
3. Select the ID of the network to open its details page.
4. On the **Deployments** tab, choose the Network Operation.
5. Choose **Cancel operation**.

AWS CLI

To cancel a network operation using the AWS CLI

Use the cancel-sol-network-operation command to cancel a network operation.

```
aws tnb cancel-sol-network-operation --ns-lcm-op-occ-id ^no-[a-f0-9]{17}$
```

TOSCA reference for AWS TNB

Topology and Orchestration Specification for Cloud Applications (TOSCA) is a declarative syntax that CSPs use to describe a topology of cloud based web services, their components, relationships, and the processes that manage them. CSPs describe the connection points, the logical links between the connection points, and the policies such as affinity and security in a TOSCA template. CSPs then upload the template to AWS TNB which synthesizes the resources needed to establish a functioning 5G network across AWS Availability Zones.

Contents

- VNFD template
- Network service descriptor template
- Common nodes

VNFD template

Defines a virtual network function descriptor (VNFD) template.

Syntax

```
tosca_definitions_version: tnb_simple_yaml_1_0

topology_template:

  inputs:
    SampleInputParameter:
      type: String
      description: "Sample parameter description"
      default: "DefaultSampleValue"

  node_templates:
    SampleNode1: tosca.nodes.AWS.VNF
```

Topology template

`node_templates`

The TOSCA AWS Nodes. The possible nodes are:

- AWS.VNF
- AWS.Artifacts.Helm

AWS.VNF

Defines an AWS virtual network function (VNF) node.

Syntax

```
tosca.nodes.AWS.VNF:
  properties:
    descriptor_id: String
    descriptor_version: String
    descriptor_name: String
    provider: String
  requirements:
    helm: String
```

Properties

`descriptor_id`

The UUID of the descriptor.

Required: Yes

Type: String

Pattern: `[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}`

`descriptor_version`

The version of the VNFD.

Required: Yes

Type: String

Pattern: `^[0-9]{1,5}\\.[0-9]{1,5}\\.[0-9]{1,5}.*`

`descriptor_name`

The name of the descriptor.

Required: Yes

Type: String

`provider`

The author of the VNFD.

Required: Yes

Type: String

Requirements

`helm`

The Helm directory defining container artifacts. This is a reference to AWS.Artifacts.Helm.

Required: Yes

Type: String

Example

```
SampleVNF:
  type: tosca.nodes.AWS.VNF
  properties:
    descriptor_id: "6a792e0c-be2a-45fa-989e-5f89d94ca898"
    descriptor_version: "1.0.0"
    descriptor_name: "Test VNF Template"
    provider: "Operator"
  requirements:
    helm: SampleHelm
```

AWS.Artifacts.Helm

Defines an AWS Helm Node.

Syntax

```
tosca.nodes.AWS.Artifacts.Helm:
```

```
  properties:
    implementation: String
```

Properties

`implementation`

The local directory that contains the Helm chart within the CSAR package.

Required: Yes

Type: String

Example

```
SampleHelm:
  type: tosca.nodes.AWS.Artifacts.Helm
  properties:
    implementation: "./vnf-helm"
```

Network service descriptor template

Defines a network service descriptor (NSD) template.

Syntax

```
tosca_definitions_version: tnb_simple_yaml_1_0

vnfds:
  - descriptor_id: String
    namespace: String

topology_template:

  inputs:
    SampleInputParameter:
      type: String
      description: "Sample parameter description"
      default: "DefaultSampleValue"
```

```
  node_templates:
    SampleNode1: tosca.nodes.AWS.NS
```

Using defined parameters

When you want to dynamically pass a parameter, such as the CIDR block for the VPC node, you can use the `{ get_input: input-parameter-name }` syntax and define the parameters in the NSD template. Then reuse the parameter across the same NSD template.

The following example shows how to define and use parameters:

```
tosca_definitions_version: tnb_simple_yaml_1_0

topology_template:

  inputs:
    cidr_block:
      type: String
      description: "CIDR Block for VPC"
      default: "10.0.0.0/24"

  node_templates:
    ExampleSingleClusterNS:
      type: tosca.nodes.AWS.NS
      properties:
        descriptor_id: "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
         .....

    ExampleVPC:
      type: tosca.nodes.AWS.Networking.VPC
      properties:
        cidr_block: { get_input: cidr_block }
```

VNFD import

`descriptor_id`

The UUID of the descriptor.

Required: Yes

Type: String

Pattern: `[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}`

`namespace`

The unique name.

Required: Yes

Type: String

Topology template

`node_templates`

The possible TOSCA AWS nodes are:

- AWS.NS
- AWS.Compute.EKS
- AWS.Compute.EKS.AuthRole
- AWS.Compute.EKSManagedNode
- AWS.Compute.EKSSelfManagedNode
- AWS.Compute.PlacementGroup
- AWS.Compute.UserData
- AWS.Networking.SecurityGroup
- AWS.Networking.SecurityGroupEgressRule
- AWS.Networking.SecurityGroupIngressRule
- AWS.Resource.Import
- AWS.Networking.ENI
- AWS.HookExecution
- AWS.Networking.InternetGateway
- AWS.Networking.RouteTable
- AWS.Networking.Subnet
- AWS.Deployment.VNFDeployment

- AWS.Networking.VPC
- AWS.Networking.NATGateway
- AWS.Networking.Route

AWS.NS

Defines an AWS network service (NS) node.

Syntax

```
tosca.nodes.AWS.NS:
  properties:
    descriptor_id: String
    descriptor_version: String
    descriptor_name: String
```

Properties

`descriptor_id`

The UUID of the descriptor.

Required: Yes

Type: String

Pattern: `[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}`

`descriptor_version`

The version of the NSD.

Required: Yes

Type: String

Pattern: `^[0-9]{1,5}\\.[0-9]{1,5}\\.[0-9]{1,5}.*`

`descriptor_name`

The name of the descriptor.

Required: Yes

Type: String

Example

```
SampleNS:
  type: tosca.nodes.AWS.NS
  properties:
    descriptor_id: "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    descriptor_version: "1.0.0"
    descriptor_name: "Test NS Template"
```

AWS.Compute.EKS

Provide the name of the cluster, the desired Kubernetes version, and a role that allows the Kubernetes control plane to manage the AWS resources required for your NFs. Multus container network interface (CNI) plugins are enabled. You can attach multiple network interfaces and apply advanced network configuration to the Kubernetes-based network functions. You also specify the cluster endpoint access and the subnets for your cluster.

Syntax

```
tosca.nodes.AWS.Compute.EKS:
  capabilities:
    multus:
      properties:
        enabled: Boolean
        multus_role: String
    ebs_csi:
      properties:
        enabled: Boolean
        version: String
  properties:
    version: String
    access: String
    cluster_role: String
    tags: List
    ip_family: String
  requirements:
```

```
    subnets: List
```

Capabilities

`multus`

Optional. Properties that define the Multus container network interface (CNI) usage.

If you include `multus`, specify the `enabled` and `multus_role` properties.

`enabled`

Indicates whether the default Multus capability is enabled.

Required: Yes

Type: Boolean

`multus_role`

The role for Multus network interface management.

Required: Yes

Type: String

`ebs_csi`

Properties that define the Amazon EBS Container Storage Interface (CSI) driver installed in the Amazon EKS cluster.

Enable this plugin to use Amazon EKS self-managed nodes on AWS Outposts, AWS Local Zones, or AWS Regions. For more information, see Amazon Elastic Block Store CSI driver in the Amazon EKS User Guide.

`enabled`

Indicates whether the default Amazon EBS CSI driver is installed.

Required: No

Type: Boolean

`version`

The version of the Amazon EBS CSI driver add-on. The version must match one of the versions returned by the *DescribeAddonVersions* action. For more information, see DescribeAddonVersions in the *Amazon EKS API Reference*

Required: No

Type: String

Properties

`version`

The Kubernetes version for the cluster. AWS Telco Network Builder supports Kubernetes versions 1.23 through 1.30.

Required: Yes

Type: String

Possible values: 1.23 | 1.24 | 1.25 | 1.26 | 1.27 | 1.28 | 1.29 | 1.30

`access`

The cluster endpoint access.

Required: Yes

Type: String

Possible values: `PRIVATE` | `PUBLIC` | `ALL`

`cluster_role`

The role for cluster management.

Required: Yes

Type: String

`tags`

Tags to be attached to the resource.

Required: No

Type: List

`ip_family`

Indicates the IP family for service and pod addresses in the cluster.

Allowed value: `IPv4`, `IPv6`

Default value: `IPv4`

Required: No

Type: String

Requirements

`subnets`

An AWS.Networking.Subnet node.

Required: Yes

Type: List

Example

```
SampleEKS:
  type: tosca.nodes.AWS.Compute.EKS
  properties:
    version: "1.23"
    access: "ALL"
    cluster_role: "arn:aws:iam::${AWS::TNB::AccountId}:role/SampleRole"
    ip_family: "IPv6"
    tags:
      - "Name=SampleVPC"
      - "Environment=Testing"
  capabilities:
    multus:
      properties:
        enabled: true
```

```
        multus_role: "arn:aws:iam::${AWS::TNB::AccountId}:role/MultusRole"
    ebs_csi:
      properties:
        enabled: true
        version: "v1.16.0-eksbuild.1"
  requirements:
    subnets:
    - SampleSubnet01
    - SampleSubnet02
```

AWS.Compute.EKS.AuthRole

An AuthRole allows you to add IAM roles to the Amazon EKS cluster `aws-auth ConfigMap` so that users can access the Amazon EKS cluster using an IAM role.

Syntax

```
tosca.nodes.AWS.Compute.EKS.AuthRole:
  properties:
    role_mappings: List
      arn: String
      groups: List
  requirements:
    clusters: List
```

Properties

`role_mappings`

List of mappings that define IAM roles that need to be added to the Amazon EKS cluster `aws-auth ConfigMap`.

`arn`

The ARN of the IAM role.

Required: Yes

Type: String

`groups`

Kubernetes groups to assign to the role defined in `arn`.

Required: No

Type: List

Requirements

`clusters`

An AWS.Compute.EKS node.

Required: Yes

Type: List

Example

```
EKSAuthMapRoles:
    type: tosca.nodes.AWS.Compute.EKS.AuthRole
    properties:
        role_mappings:
        - arn: arn:aws:iam::${AWS::TNB::AccountId}:role/TNBHookRole1
          groups:
          - system:nodes
          - system:bootstrappers
        - arn: arn:aws:iam::${AWS::TNB::AccountId}:role/TNBHookRole2
          groups:
          - system:nodes
          - system:bootstrappers
    requirements:
         clusters:
         - Free5GCEKS1
         - Free5GCEKS2
```

AWS.Compute.EKSManagedNode

AWS TNB supports EKS Managed Node groups to automate the provisioning and lifecycle management of nodes (Amazon EC2 instances) for Amazon EKS Kubernetes clusters. To create an EKS Node group, do the following:

- Choose the Amazon Machine Images (AMI) for your cluster workers nodes by providing either the ID of the AMI or the AMI type.

- Provide an Amazon EC2 key pair for SSH access and the scaling properties for your node group.
- Ensure that your node group is associated with an Amazon EKS cluster.
- Provide the subnets for the worker nodes.
- Optionally, attach security groups, node labels, and a placement group to your node group.

Syntax

```
tosca.nodes.AWS.Compute.EKSManagedNode:
  capabilities:
    compute:
      properties:
        ami_type: String
        ami_id: String
        instance_types: List
        key_pair: String
        root_volume_encryption: Boolean
        root_volume_encryption_key_arn: String
    scaling:
      properties:
        desired_size: Integer
        min_size: Integer
        max_size: Integer
  properties:
    node_role: String
    tags: List
  requirements:
    cluster: String
    subnets: List
    network_interfaces: List
    security_groups: List
    placement_group: String
    user_data: String
    labels: List
```

Capabilities

`compute`

Properties that define the computing parameters for the Amazon EKS managed node group, such as, Amazon EC2 instance types and Amazon EC2 instance AMIs.

`ami_type`

The Amazon EKS-supported AMI type.

Required: Yes

Type: String

Possible values: `AL2_x86_64` | `AL2_x86_64_GPU` | `AL2_ARM_64` | `CUSTOM` | `BOTTLEROCKET_ARM_64` | `BOTTLEROCKET_x86_64` | `BOTTLEROCKET_ARM_64_NVIDIA` | `BOTTLEROCKET_x86_64_NVIDIA`

`ami_id`

The ID of the AMI.

Required: No

Type: String

> **Note**
>
> If both `ami_type` and `ami_id` are specified in the template, AWS TNB will use only the `ami_id` value to create `EKSManagedNode`.

`instance_types`

The instance size.

Required: Yes

Type: List

`key_pair`

The EC2 Key pair to enable SSH access.

Required: Yes

Type: String

`root_volume_encryption`

Enables Amazon EBS encryption for the Amazon EBS root volume. If this property is not provided, AWS TNB encrypts Amazon EBS root volumes by default.

Required: No

Default: true

Type: Boolean

`root_volume_encryption_key_arn`

The ARN of the AWS KMS key. AWS TNB supports regular key ARN, multi-region key ARN and alias ARN.

Required: No

Type: String

> **Note**
>
> - If `root_volume_encryption` is false, do not include `root_volume_encryption_key_arn`.
> - AWS TNB supports root volume encryption of Amazon EBS-backed AMI's.
> - If the AMI's root volume is already encrypted, you must include the `root_volume_encryption_key_arn` for AWS TNB to re-encrypt the root volume.
> - If the AMI's root volume is not encrypted, AWS TNB uses the `root_volume_encryption_key_arn` to encrypt the root volume.
>
> If you do not include `root_volume_encryption_key_arn`, AWS TNB uses the default key provided by AWS Key Management Service to encrypt the root volume.
> - AWS TNB does not decrypt an encrypted AMI.

`scaling`

Properties that define the scaling parameters for the Amazon EKS managed node group, such as, the desired number of Amazon EC2 instances, and minimum and maximum number of Amazon EC2 instances in the node group.

`desired_size`

The number of instances in this NodeGroup.

Required: Yes

Type: Integer

`min_size`

The minimum number of instances in this NodeGroup.

Required: Yes

Type: Integer

`max_size`

The maximum number of instances in this NodeGroup.

Required: Yes

Type: Integer

Properties

`node_role`

The ARN of the IAM role that is attached to the Amazon EC2 instance.

Required: Yes

Type: String

`tags`

The tags to be attached to the resource.

Required: No

Type: List

Requirements

`cluster`

An AWS.Compute.EKS node.

Required: Yes

Type: String

`subnets`

An AWS.Networking.Subnet node.

Required: Yes

Type: List

`network_interfaces`

An AWS.Networking.ENI node. Ensure that the network interfaces and subnets are set to the same Availability Zone or instantiation will fail.

When you set `network_interfaces`, AWS TNB obtains the permission related to ENIs from the `multus_role` property if you included the `multus` property in the AWS.Compute.EKS node. Otherwise, AWS TNB obtains the permission related to ENIs from the node_role property.

Required: No

Type: List

`security_groups`

An AWS.Networking.SecurityGroup node.

Required: No

Type: List

`placement_group`

A tosca.nodes.AWS.Compute.PlacementGroup node.

Required: No

Type: String

`user_data`

A tosca.nodes.AWS.Compute.UserData node reference. A user data script is passed to the Amazon EC2 instances launched by the managed node group. Add the permissions required to run custom user data to the node_role passed to the node group.

Required: No

Type: String

`labels`

A list of node labels. A node label must have a name and a value. Create a label using the following criteria:

- The name and value must be separated by =.
- The name and value can each be up to 63 characters in length.
- The label can include letters (A-Z, a-z,), numbers (0-9) and the following characters: `[-, _, ., *, ?]`
- The name and value must start and end with an alphanumeric, ?, or * character.

For example, `myLabelName1=*NodeLabelValue1`

Required: No

Type: List

Example

```
SampleEKSManagedNode:
  type: tosca.nodes.AWS.Compute.EKSManagedNode
  capabilities:
    compute:
      properties:
        ami_type: "AL2_x86_64"
        instance_types:
          - "t3.xlarge"
        key_pair: "SampleKeyPair"
        root_volume_encryption: true
        root_volume_encryption_key_arn: "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    scaling:
      properties:
        desired_size: 1
        min_size: 1
        max_size: 1
  properties:
    node_role: "arn:aws:iam::${AWS::TNB::AccountId}:role/SampleRole"
    tags:
      - "Name=SampleVPC"
```

```
      - "Environment=Testing"
  requirements:
    cluster: SampleEKS
    subnets:
      - SampleSubnet
    network_interfaces:
      - SampleENI01
      - SampleENI02
    security_groups:
      - SampleSecurityGroup01
      - SampleSecurityGroup02
    placement_group: SamplePlacementGroup
    user_data: CustomUserData
    labels:
      - "sampleLabelName001=sampleLabelValue001"
      - "sampleLabelName002=sampleLabelValue002"
```

AWS.Compute.EKSSelfManagedNode

AWS TNB supports Amazon EKS self-managed nodes to automate the provisioning and lifecycle management of nodes (Amazon EC2 instances) for Amazon EKS Kubernetes clusters. To create an Amazon EKS node group, do the following:

- Choose the Amazon Machine Images (AMI) for your cluster workers nodes by providing either the ID of the AMI.
- Provide an Amazon EC2 key pair for SSH access.
- Ensure that your node group is associated with an Amazon EKS cluster.
- Provide the instance type and desired, minimum, and maximum sizes.
- Provide the subnets for the worker nodes.
- Optionally, attach security groups, node labels, and a placement group to your node group.

Syntax

```
tosca.nodes.AWS.Compute.EKSSelfManagedNode:
  capabilities:
    compute:
      properties:
        ami_id: String
        instance_type: String
```

```
        key_pair: String
        root_volume_encryption: Boolean
        root_volume_encryption_key_arn: String
    scaling:
      properties:
        desired_size: Integer
        min_size: Integer
        max_size: Integer
  properties:
    node_role: String
    tags: List
  requirements:
    cluster: String
    subnets: List
    network_interfaces: List
    security_groups: List
    placement_group: String
    user_data: String
    labels: List
```

Capabilities

compute

Properties that define the computing parameters for the Amazon EKS self-managed nodes, such as, Amazon EC2 instance types and Amazon EC2 instance AMIs.

`ami_id`

The AMI ID used to launch the instance. AWS TNB supports instances that leverage IMDSv2. For more information, see IMDS version.

Required: Yes

Type: String

`instance_type`

The instance size.

Required: Yes

Type: String

`key_pair`

The Amazon EC2 key pair to enable SSH access.

Required: Yes

Type: String

`root_volume_encryption`

Enables Amazon EBS encryption for the Amazon EBS root volume. If this property is not provided, AWS TNB encrypts Amazon EBS root volumes by default.

Required: No

Default: true

Type: Boolean

`root_volume_encryption_key_arn`

The ARN of the AWS KMS key. AWS TNB supports regular key ARN, multi-region key ARN and alias ARN.

Required: No

Type: String

Note

- If `root_volume_encryption` is false, do not include `root_volume_encryption_key_arn`.
- AWS TNB supports root volume encryption of Amazon EBS-backed AMI's.
- If the AMI's root volume is already encrypted, you must include the `root_volume_encryption_key_arn` for AWS TNB to re-encrypt the root volume.
- If the AMI's root volume is not encrypted, AWS TNB uses the `root_volume_encryption_key_arn` to encrypt the root volume.

  If you do not include `root_volume_encryption_key_arn`, AWS TNB uses AWS Managed Services to encrypt the root volume.
- AWS TNB does not decrypt an encrypted AMI.

scaling

Properties that define the scaling parameters for the Amazon EKS self-managed nodes, such as, the desired number of Amazon EC2 instances, and minimum and maximum number of Amazon EC2 instances in the node group.

`desired_size`

The number of instances in this NodeGroup.

Required: Yes

Type: Integer

`min_size`

The minimum number of instances in this NodeGroup.

Required: Yes

Type: Integer

`max_size`

The maximum number of instances in this NodeGroup.

Required: Yes

Type: Integer

Properties

`node_role`

The ARN of the IAM role that is attached to the Amazon EC2 instance.

Required: Yes

Type: String

`tags`

The tags to be attached to the resource. Tags will be propagated to the instances created by the resource.

Required: No

Type: List

Requirements

`cluster`

An AWS.Compute.EKS node.

Required: Yes

Type: String

`subnets`

An AWS.Networking.Subnet node.

Required: Yes

Type: List

`network_interfaces`

An AWS.Networking.ENI node. Ensure that the network interfaces and subnets are set to the same Availability Zone or instantiation will fail.

When you set `network_interfaces`, AWS TNB obtains the permission related to ENIs from the `multus_role` property if you included the `multus` property in the AWS.Compute.EKS node. Otherwise, AWS TNB obtains the permission related to ENIs from the node_role property.

Required: No

Type: List

`security_groups`

An AWS.Networking.SecurityGroup node.

Required: No

Type: List

`placement_group`

A tosca.nodes.AWS.Compute.PlacementGroup node.

Required: No

Type: String

`user_data`

A tosca.nodes.AWS.Compute.UserData node reference. A user data script is passed to the Amazon EC2 instances launched by the self-managed node group. Add the permissions required for executing custom user data to the node_role passed to the node group.

Required: No

Type: String

`labels`

A list of node labels. A node label must have a name and a value. Create a label using the following criteria:

- The name and value must be separated by =.
- The name and value can each be up to 63 characters in length.
- The label can include letters (A-Z, a-z,), numbers (0-9), and the following characters: `[-, _, ., *, ?]`
- The name and value must start and end with an alphanumeric, ?, or * character.

For example, `myLabelName1=*NodeLabelValue1`

Required: No

Type: List

Example

```
SampleEKSSelfManagedNode:
  type: tosca.nodes.AWS.Compute.EKSSelfManagedNode
  capabilities:
    compute:
      properties:
        ami_id: "ami-123123EXAMPLE"
        instance_type: "c5.large"
        key_pair: "SampleKeyPair"
        root_volume_encryption: true
```

```
        root_volume_encryption_key_arn: "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    scaling:
      properties:
        desired_size: 1
        min_size: 1
        max_size: 1
  properties:
    node_role: "arn:aws:iam::${AWS::TNB::AccountId}:role/SampleNodeRole"
    tags:
      - "Name=SampleVPC"
      - "Environment=Testing"
  requirements:
    cluster: SampleEKSCluster
    subnets:
      - SampleSubnet
    network_interfaces:
      - SampleNetworkInterface01
      - SampleNetworkInterface02
    security_groups:
      - SampleSecurityGroup01
      - SampleSecurityGroup02
    placement_group: SamplePlacementGroup
    user_data: CustomUserData
    labels:
      - "sampleLabelName001=sampleLabelValue001"
      - "sampleLabelName002=sampleLabelValue002"
```

AWS.Compute.PlacementGroup

A PlacementGroup node supports different strategies to place Amazon EC2 instances.

When you launch a new Amazon EC2instance, the Amazon EC2 service attempts to place the instance in such a way that all of your instances are spread out across underlying hardware to minimize correlated failures. You can use placement groups to influence the placement of a group of interdependent instances to meet the needs of your workload.

Syntax

```
tosca.nodes.AWS.Compute.PlacementGroup
  properties:
    strategy: String
```

```
    partition_count: Integer
    tags: List
```

Properties

`strategy`

The strategy to use to place Amazon EC2 instances.

Required: Yes

Type: String

Possible values: CLUSTER | PARTITION | SPREAD_HOST | SPREAD_RACK

- **CLUSTER** – packs instances close together inside an Availability Zone. This strategy enables workloads to achieve the low-latency network performance necessary for tightly-coupled node-to-node communication that is typical of high-performance computing (HPC) applications.
- **PARTITION** – spreads your instances across logical partitions such that groups of instances in one partition do not share the underlying hardware with groups of instances in different partitions. This strategy is typically used by large distributed and replicated workloads, such as Hadoop, Cassandra, and Kafka.
- **SPREAD_RACK** – places a small group of instances across distinct underlying hardware to reduce correlated failures.
- **SPREAD_HOST** – used only with Outpost placement groups. Places a small group of instances across distinct underlying hardware to reduce correlated failures.

`partition_count`

The number of partitions.

Required: Required only when `strategy` is set to `PARTITION`.

Type: Integer

Possible values: 1 | 2 |3 | 4 | 5 | 6 | 7

`tags`

The tags that you can attach to the placement group resource.

Required: No

Type: List

Example

```
ExamplePlacementGroup:
  type: tosca.nodes.AWS.Compute.PlacementGroup
  properties:
    strategy: "PARTITION"
    partition_count: 5
    tags:
      - tag_key=tag_value
```

AWS.Compute.UserData

AWS TNB supports launching Amazon EC2 instances with custom user data, through the UserData node in Network Service Descriptor (NSD). For more information about custom user data, see User data and shell scripts in the *Amazon EC2 User Guide*.

During network instantiation, AWS TNB provides the Amazon EC2 instance registration to the cluster through a user-data script. When custom user data is also provided, AWS TNB merges both scripts and passes them on as a multimime script to Amazon EC2. The custom user-data script is run prior to the Amazon EKS registration script.

To use custom variables in the user-data script, add an exclamation mark `!` after the open curly brace `{`. For example, to use `MyVariable` in the script, enter: `{!MyVariable}`

Note

- AWS TNB supports user-data scripts up to 7 KB in size.
- Because AWS TNB uses AWS CloudFormation to process and render the `multimime` user-data script, ensure that the script adheres to all AWS CloudFormation rules.

Syntax

```
tosca.nodes.AWS.Compute.UserData:
  properties:
    implementation: String
```

```
    content_type: String
```

Properties

`implementation`

The relative path to the user data script definition. The format must be: `./scripts/script_name.sh`

Required: Yes

Type: String

`content_type`

Content type of the user data script.

Required: Yes

Type: String

Possible values: `x-shellscript`

Example

```
ExampleUserData:
  type: tosca.nodes.AWS.Compute.UserData
  properties:
    content_type: "text/x-shellscript"
    implementation: "./scripts/customUserData.sh"
```

AWS.Networking.SecurityGroup

AWS TNB supports security groups to automate the provisioning of Amazon EC2 Security Groups which you can attach to Amazon EKS Kubernetes cluster node groups.

Syntax

```
tosca.nodes.AWS.Networking.SecurityGroup
  properties:
    description: String
    name: String
```

```
    tags: List
  requirements:
    vpc: String
```

Properties

`description`

The description of the security group. You can use up to 255 characters to describe the group. You can include only letters (A-Z and a-z), numbers (0-9), spaces, and the following special characters: ._-:/()#,@[]+=&;{}!$*

Required: Yes

Type: String

`name`

A name for the security group. You can use up to 255 characters for the name. You can include only letters (A-Z and a-z), numbers (0-9), spaces, and the following special characters: ._-:/()#,@[]+=&;{}!$*

Required: Yes

Type: String

`tags`

The tags that you can attach to the security group resource.

Required: No

Type: List

Requirements

`vpc`

An AWS.Networking.VPC node.

Required: Yes

Type: String

Example

```
SampleSecurityGroup001:
  type: tosca.nodes.AWS.Networking.SecurityGroup
  properties:
        description: "Sample Security Group for Testing"
        name: "SampleSecurityGroup"
        tags:
          - "Name=SecurityGroup"
          - "Environment=Testing"
      requirements:
        vpc: SampleVPC
```

AWS.Networking.SecurityGroupEgressRule

AWS TNB supports security group egress rules to automate the provisioning of Amazon EC2 Security Group Egress Rules which can be attached to AWS.Networking.SecurityGroup. Note that you must provide a cidr_ip/destination_security_group/destination_prefix_list as the destination for egress traffic.

Syntax

```
AWS.Networking.SecurityGroupEgressRule
  properties:
    ip_protocol: String
    from_port: Integer
    to_port: Integer
    description: String
    destination_prefix_list: String
    cidr_ip: String
    cidr_ipv6: String
  requirements:
    security_group: String
    destination_security_group: String
```

Properties

`cidr_ip`

The IPv4 address range in CIDR format. You must specify a CIDR range that allows egress traffic.

Required: No

Type: String

`cidr_ipv6`

The IPv6 address range in CIDR format, for egress traffic. You must specify a destination security group (`destination_security_group` or `destination_prefix_list`) or a CIDR range (`cidr_ip` or `cidr_ipv6`).

Required: No

Type: String

`description`

The description of an egress (outbound) security group rule. You can use up to 255 characters to describe the rule.

Required: No

Type: String

`destination_prefix_list`

The prefix list ID of an existing Amazon VPC managed prefix list. This is the destination from node group instances associated with the security group. For more information on managed prefix lists, see Managed prefix lists in the *Amazon VPC User Guide*.

Required: No

Type: String

`from_port`

If the protocol is TCP or UDP, this is the start of the port range. If the protocol is ICMP or ICMPv6, this is the type number. A value of -1 indicates all ICMP/ICMPv6 types. If you specify all ICMP/ICMPv6 types, you must specify all ICMP/ICMPv6 codes.

Required: No

Type: Integer

`ip_protocol`

The IP protocol name (tcp, udp, icmp, icmpv6) or protocol number. Use -1 to specify all protocols. When authorizing security group rules, specifying -1 or a protocol number other than

tcp, udp, icmp, or icmpv6 allows traffic on all ports, regardless of any port range you specify. For tcp, udp, and icmp, you must specify a port range. For icmpv6, the port range is optional; if you omit the port range, traffic for all types and codes is allowed.

Required: Yes

Type: String

`to_port`

If the protocol is TCP or UDP, this is the end of the port range. If the protocol is ICMP or ICMPv6, this is the code. A value of -1 indicates all ICMP/ICMPv6 codes. If you specify all ICMP/ICMPv6 types, you must specify all ICMP/ICMPv6 codes.

Required: No

Type: Integer

Requirements

`security_group`

The ID of the security group to which this rule is to be added.

Required: Yes

Type: String

`destination_security_group`

The ID or TOSCA reference of the destination security group to which egress traffic is allowed.

Required: No

Type: String

Example

```
SampleSecurityGroupEgressRule:
      type: tosca.nodes.AWS.Networking.SecurityGroupEgressRule
      properties:
```

```
        ip_protocol: "tcp"
        from_port: 8000
        to_port: 9000
        description: "Egress Rule for sample security group"
        cidr_ipv6: "2600:1f14:3758:ca00::/64"
      requirements:
        security_group: SampleSecurityGroup001
        destination_security_group: SampleSecurityGroup002
```

AWS.Networking.SecurityGroupIngressRule

AWS TNB supports security group ingress rules to automate the provisioning of Amazon EC2 Security Group Ingress Rules which can be attached to AWS.Networking.SecurityGroup. Note that you must provide a cidr_ip/source_security_group/source_prefix_list as the source for ingress traffic.

Syntax

```
AWS.Networking.SecurityGroupIngressRule
  properties:
    ip_protocol: String
    from_port: Integer
    to_port: Integer
    description: String
    source_prefix_list: String
    cidr_ip: String
    cidr_ipv6: String
  requirements:
    security_group: String
    source_security_group: String
```

Properties

`cidr_ip`

The IPv4 address range in CIDR format. You must specify a CIDR range that allows ingress traffic.

Required: No

Type: String

`cidr_ipv6`

The IPv6 address range in CIDR format, for ingress traffic. You must specify a source security group (`source_security_group` or `source_prefix_list`) or a CIDR range (`cidr_ip` or `cidr_ipv6`).

Required: No

Type: String

`description`

The description of an ingress (inbound) security group rule. You can use up to 255 characters to describe the rule.

Required: No

Type: String

`source_prefix_list`

The prefix list ID of an existing Amazon VPC managed prefix list. This is the source from which node group instances associated with the security group will be allowed to receive traffic from. For more information on managed prefix lists, see Managed prefix lists in the *Amazon VPC User Guide*.

Required: No

Type: String

`from_port`

If the protocol is TCP or UDP, this is the start of the port range. If the protocol is ICMP or ICMPv6, this is the type number. A value of -1 indicates all ICMP/ICMPv6 types. If you specify all ICMP/ICMPv6 types, you must specify all ICMP/ICMPv6 codes.

Required: No

Type: Integer

`ip_protocol`

The IP protocol name (tcp, udp, icmp, icmpv6) or protocol number. Use -1 to specify all protocols. When authorizing security group rules, specifying -1 or a protocol number other than tcp, udp, icmp, or icmpv6 allows traffic on all ports, regardless of any port range you specify.

For tcp, udp, and icmp, you must specify a port range. For icmpv6, the port range is optional; if you omit the port range, traffic for all types and codes is allowed.

Required: Yes

Type: String

`to_port`

If the protocol is TCP or UDP, this is the end of the port range. If the protocol is ICMP or ICMPv6, this is the code. A value of -1 indicates all ICMP/ICMPv6 codes. If you specify all ICMP/ICMPv6 types, you must specify all ICMP/ICMPv6 codes.

Required: No

Type: Integer

Requirements

`security_group`

The ID of the security group to which this rule is to be added.

Required: Yes

Type: String

`source_security_group`

The ID or TOSCA reference of the source security group from which ingress traffic is to be allowed.

Required: No

Type: String

Example

```
SampleSecurityGroupIngressRule:
      type: tosca.nodes.AWS.Networking.SecurityGroupIngressRule
      properties:
        ip_protocol: "tcp"
```

```
        from_port: 8000
        to_port: 9000
        description: "Ingress Rule for free5GC cluster on IPv6"
        cidr_ipv6: "2600:1f14:3758:ca00::/64"
      requirements:
        security_group: SampleSecurityGroup1
        source_security_group: SampleSecurityGroup2
```

AWS.Resource.Import

You can import the following AWS resources into AWS TNB:

- VPC
- Subnet
- Route Table
- Internet Gateway
- Security Group

Syntax

```
tosca.nodes.AWS.Resource.Import
  properties:
    resource_type: String
    resource_id: String
```

Properties

`resource_type`

The resource type that is imported to AWS TNB.

Required: No

Type: List

`resource_id`

The resource ID that is imported to AWS TNB.

Required: No

Type: List

Example

```
SampleImportedVPC
  type: tosca.nodes.AWS.Resource.Import
  properties:
    resource_type: "tosca.nodes.AWS.Networking.VPC"
    resource_id: "vpc-123456"
```

AWS.Networking.ENI

A network interface is a logical networking component in a VPC that represents a virtual network card. A network interface is assigned an IP address either automatically or manually based on its subnet. After you deploy an Amazon EC2 instance in a subnet, you can attach a network interface to it, or detach a network interface from that Amazon EC2 instance and reattach to another Amazon EC2 instance in that subnet. The device index identifies the position in the attachment order.

Syntax

```
tosca.nodes.AWS.Networking.ENI:
  properties:
    device_index: Integer
    source_dest_check: Boolean
    tags: List
  requirements:
    subnet: String
    security_groups: List
```

Properties

`device_index`

The device index must be greater than zero.

Required: Yes

Type: Integer

`source_dest_check`

Indicates whether the network interface performs source/destination checking. A value of `true` means that checking is enabled, and `false` means that checking is disabled.

Allowed value: true, false

Default: true

Required: No

Type: Boolean

`tags`

The tags to be attached to the resource.

Required: No

Type: List

Requirements

`subnet`

An AWS.Networking.Subnet node.

Required: Yes

Type: String

`security_groups`

An AWS.Networking.SecurityGroup node.

Required: No

Type: String

Example

```
SampleENI:
  type: tosca.nodes.AWS.Networking.ENI
  properties:
```

```
    device_index: 5
    source_dest_check: true
    tags:
      - "Name=SampleVPC"
      - "Environment=Testing"
  requirements:
    subnet: SampleSubnet
    security_groups:
      - SampleSecurityGroup01
      - SampleSecurityGroup02
```

AWS.HookExecution

A lifecycle hook provides you with the ability to run your own scripts as part of your infrastructure and network instantiation.

Syntax

```
tosca.nodes.AWS.HookExecution:
  capabilities:
    execution:
      properties:
        type: String
  requirements:
    definition: String
    vpc: String
```

Capabilities

`execution`

Properties for the hook execution engine that runs the hook scripts.

`type`

The hook execution engine type.

Required: No

Type: String

Possible values: `CODE_BUILD`

Requirements

`definition`

An AWS.HookDefinition.Bash node.

Required: Yes

Type: String

`vpc`

An AWS.Networking.VPC node.

Required: Yes

Type: String

Example

```
SampleHookExecution:
  type: tosca.nodes.AWS.HookExecution
  requirements:
    definition: SampleHookScript
    vpc: SampleVPC
```

AWS.Networking.InternetGateway

Defines an AWS Internet Gateway Node.

Syntax

```
tosca.nodes.AWS.Networking.InternetGateway:
  capabilities:
    routing:
      properties:
        dest_cidr: String
        ipv6_dest_cidr: String
  properties:
    tags: List
    egress_only: Boolean
  requirements:
    vpc: String
```

```
    route_table: String
```

Capabilities

routing

Properties that define the routing connection within the VPC. You must include either the `dest_cidr` or `ipv6_dest_cidr` property.

`dest_cidr`

The IPv4 CIDR block used for the destination match. This property is used to create a route in `RouteTable` and its value is used as the `DestinationCidrBlock`.

Required: No if you included the `ipv6_dest_cidr` property.

Type: String

`ipv6_dest_cidr`

The IPv6 CIDR block used for the destination match.

Required: No if you included the `dest_cidr` property.

Type: String

Properties

`tags`

The tags to be attached to the resource.

Required: No

Type: List

`egress_only`

An IPv6-specific property. Indicates if the internet gateway is only for egress communication or not. When `egress_only` is true, you must define the `ipv6_dest_cidr` property.

Required: No

Type: Boolean

Requirements

`vpc`

An AWS.Networking.VPC node.

Required: Yes

Type: String

`route_table`

An AWS.Networking.RouteTable node.

Required: Yes

Type: String

Example

```
Free5GCIGW:
  type: tosca.nodes.AWS.Networking.InternetGateway
  properties:
    egress_only: false
  capabilities:
    routing:
      properties:
        dest_cidr: "0.0.0.0/0"
        ipv6_dest_cidr: "::/0"
  requirements:
    route_table: Free5GCRouteTable
    vpc: Free5GCVPC
Free5GCEGW:
  type: tosca.nodes.AWS.Networking.InternetGateway
  properties:
    egress_only: true
  capabilities:
    routing:
      properties:
        ipv6_dest_cidr: "::/0"
  requirements:
    route_table: Free5GCPrivateRouteTable
    vpc: Free5GCVPC
```

AWS.Networking.RouteTable

A route table contains a set of rules, called routes, that determine where network traffic from subnets within your VPC or gateway is directed. You must associate a route table with a VPC.

Syntax

```
tosca.nodes.AWS.Networking.RouteTable:
  properties:
    tags: List
  requirements:
    vpc: String
```

Properties

`tags`

Tags to be attached to the resource.

Required: No

Type: List

Requirements

`vpc`

An AWS.Networking.VPC node.

Required: Yes

Type: String

Example

```
SampleRouteTable:
  type: tosca.nodes.AWS.Networking.RouteTable
  properties:
    tags:
      - "Name=SampleVPC"
```

```
      - "Environment=Testing"
  requirements:
    vpc: SampleVPC
```

AWS.Networking.Subnet

A subnet is a range of IP addresses in your VPC, and it must reside entirely within one Availability Zone. You must specify a VPC, a CIDR block, Availability Zone, and a route table for your subnet. You must also define whether your subnet is private or public.

Syntax

```
tosca.nodes.AWS.Networking.Subnet:
  properties:
    type: String
    availability_zone: String
    cidr_block: String
    ipv6_cidr_block: String
    ipv6_cidr_block_suffix: String
    outpost_arn: String
    tags: List
  requirements:
    vpc: String
    route_table: String
```

Properties

`type`

Indicates whether instances launched in this subnet receive a public IPv4 address.

Required: Yes

Type: String

Possible values: `PUBLIC` | `PRIVATE`

`availability_zone`

The Availability Zone for the subnet. This field supports AWS Availability Zones within a AWS Region, for example `us-west-2` (US West (Oregon)). It also supports AWS Local Zones within the Availability Zone, for example `us-west-2-lax-1a`.

Required: Yes

Type: String

`cidr_block`

The CIDR block for the subnet.

Required: No

Type: String

`ipv6_cidr_block`

The CIDR block used to create the IPv6 subnet. If you include this property, do not include `ipv6_cidr_block_suffix`.

Required: No

Type: String

`ipv6_cidr_block_suffix`

The 2-digit hexadecimal suffix of the IPv6 CIDR block for the subnet created over Amazon VPC. Use the following format: *`2-digit hexadecimal`*`::/`*`subnetMask`*

If you include this property, do not include `ipv6_cidr_block`.

Required: No

Type: String

`outpost_arn`

The ARN of AWS Outposts that the subnet will be created in. Add this property to the NSD template if you want to launch Amazon EKS self-managed nodes on AWS Outposts. For more information, see Amazon EKS on AWS Outposts in the *Amazon EKS User Guide*.

If you add this property to the NSD template, you must set the value for the `availability_zone` property to the Availability Zone of the AWS Outposts.

Required: No

Type: String

`tags`

The tags to be attached to the resource.

Required: No

Type: List

Requirements

`vpc`

An AWS.Networking.VPC node.

Required: Yes

Type: String

`route_table`

An AWS.Networking.RouteTable node.

Required: Yes

Type: String

Example

```
SampleSubnet01:
  type: tosca.nodes.AWS.Networking.Subnet
  properties:
    type: "PUBLIC"
    availability_zone: "us-east-1a"
    cidr_block: "10.100.50.0/24"
    ipv6_cidr_block_suffix: "aa::/64"
    outpost_arn: "arn:aws:outposts:region:accountId:outpost/op-11223344EXAMPLE"
    tags:
      - "Name=SampleVPC"
      - "Environment=Testing"
  requirements:
    vpc: SampleVPC
    route_table: SampleRouteTable
```

```
SampleSubnet02:
  type: tosca.nodes.AWS.Networking.Subnet
  properties:
    type: "PUBLIC"
    availability_zone: "us-west-2b"
    cidr_block: "10.100.50.0/24"
    ipv6_cidr_block: "2600:1f14:3758:ca00::/64"
  requirements:
    route_table: SampleRouteTable
    vpc: SampleVPC
```

AWS.Deployment.VNFDeployment

NF deployments are modeled by providing the infrastructure and the application associated to it. The cluster attribute specifies the EKS cluster to host your NFs. The vnfs attribute specifies the network functions for your deployment. You can also provide optional lifecycle hooks operations of type pre_create and post_create to run instructions specific to your deployment, such as calling an Inventory Management system API.

Syntax

```
tosca.nodes.AWS.Deployment.VNFDeployment:
  requirements:
    deployment: String
    cluster: String
    vnfs: List
  interfaces:
    Hook:
      pre_create: String
      post_create: String
```

Requirements

`deployment`

An AWS.Deployment.VNFDeployment node.

Required: No

Type: String

`cluster`

An AWS.Compute.EKS node.

Required: Yes

Type: String

`vnfs`

An AWS.VNF node.

Required: Yes

Type: String

Interfaces

Hooks

Defines the stage when lifecycle hooks are run.

`pre_create`

An AWS.HookExecution node. This hook is run before the `VNFDeployment` node deploys.

Required: No

Type: String

`post_create`

An AWS.HookExecution node. This hook is run after the `VNFDeployment` node deploys.

Required: No

Type: String

Example

```
SampleHelmDeploy:
  type: tosca.nodes.AWS.Deployment.VNFDeployment
  requirements:
```

```
    deployment: SampleHelmDeploy2
    cluster: SampleEKS
    vnfs:
      - vnf.SampleVNF
  interfaces:
    Hook:
      pre_create: SampleHook
```

AWS.Networking.VPC

You must specify a CIDR block for your virtual private cloud (VPC).

Syntax

```
tosca.nodes.AWS.Networking.VPC:
  properties:
    cidr_block: String
    ipv6_cidr_block: String
    dns_support: String
    tags: List
```

Properties

`cidr_block`

The IPv4 network range for the VPC, in CIDR notation.

Required: Yes

Type: String

`ipv6_cidr_block`

The IPv6 CIDR block used to create the VPC.

Allowed value: `AMAZON_PROVIDED`

Required: No

Type: String

`dns_support`

Indicates whether the instances launched in the VPC get DNS hostnames.

Required: No

Type: Boolean

Default: `false`

`tags`

Tags to be attached to the resource.

Required: No

Type: List

Example

```
SampleVPC:
  type: tosca.nodes.AWS.Networking.VPC
  properties:
    cidr_block: "10.100.0.0/16"
    ipv6_cidr_block: "AMAZON_PROVIDED"
    dns_support: true
    tags:
      - "Name=SampleVPC"
      - "Environment=Testing"
```

AWS.Networking.NATGateway

You can define a public or private NAT Gateway node over a subnet. For a public gateway, if you do not provide an Elastic IP allocation id, AWS TNB will allocate an Elastic IP for your account and associate that to the gateway.

Syntax

```
tosca.nodes.AWS.Networking.NATGateway:
  requirements:
    subnet: String
    internet_gateway: String
  properties:
    type: String
    eip_allocation_id: String
```

```
    tags: List
```

Properties

`subnet`

The AWS.Networking.Subnet node reference.

Required: Yes

Type: String

`internet_gateway`

The AWS.Networking.InternetGateway node reference.

Required: Yes

Type: String

Properties

`type`

Indicates if the gateway is public or private.

Allowed value: `PUBLIC`, `PRIVATE`

Required: Yes

Type: String

`eip_allocation_id`

The ID that represents the allocation of the Elastic IP address.

Required: No

Type: String

`tags`

Tags to be attached to the resource.

Required: No

Type: List

Example

```
Free5GCNatGateway01:
  type: tosca.nodes.AWS.Networking.NATGateway
    requirements:
       subnet: Free5GCSubnet01
       internet_gateway: Free5GCIGW
    properties:
       type: PUBLIC
       eip_allocation_id: eipalloc-12345
```

AWS.Networking.Route

You can define a route node that associates the destination route to the NAT Gateway as the target resource, and adds the route to the associated route table.

Syntax

```
tosca.nodes.AWS.Networking.Route:
  properties:
    dest_cidr_blocks: List
  requirements:
    nat_gateway: String
    route_table: String
```

Properties

`dest_cidr_blocks`

The list of destination IPv4 routes to the target resource.

Required: Yes

Type: List

Member type: String

Properties

`nat_gateway`

The AWS.Networking.NATGateway node reference.

Required: Yes

Type: String

`route_table`

The AWS.Networking.RouteTable node reference.

Required: Yes

Type: String

Example

```
Free5GCRoute:
  type: tosca.nodes.AWS.Networking.Route
  properties:
    dest_cidr_blocks:
      - 0.0.0.0/0
      - 10.0.0.0/28
  requirements:
    nat_gateway: Free5GCNatGateway01
    route_table: Free5GCRouteTable
```

Common nodes

Define nodes for the NSD and VNFD.

- AWS.HookDefinition.Bash

AWS.HookDefinition.Bash

Defines an AWS HookDefinition in bash.

Syntax

```
tosca.nodes.AWS.HookDefinition.Bash:
  properties:
    implementation: String
    environment_variables: List
    execution_role: String
```

Properties

`implementation`

The relative path to the hook definition. The format must be: `./hooks/`*`script_name`*`.sh`

Required: Yes

Type: String

`environment_variables`

The environment variables for the hook bash script. Use the following format: **`envName`**`=`**`envValue`** with the following regex: `^[a-zA-Z0-9]+[a-zA-Z0-9\-\_]*[a-zA-Z0-9]+=[a-zA-Z0-9]+[a-zA-Z0-9\-\_]*[a-zA-Z0-9]+$`

Ensure that the **`envName`**`=`**`envValue`** value meets the following criteria:

- Do not use spaces.
- Start **`envName`** with a letter (A-Z or a-z) or number (0-9).
- Do not start the environment variable name with the following AWS TNB reserved keywords (case insensitive):
  - CODEBUILD
  - TNB
  - HOME
  - AWS
- You can use any number of letters (A-Z or a-z), numbers (0-9), and special characters `-` and `_` for **`envName`** and **`envValue`**.

Example: `A123-45xYz=Example_789`

Required: No

Type: List

`execution_role`

The role for hook execution.

Required: Yes

Type: String

Example

```
SampleHookScript:
  type: tosca.nodes.AWS.HookDefinition.Bash
  properties:
    implementation: "./hooks/myhook.sh"
    environment_variables:
      - "variable01=value01"
      - "variable02=value02"
    execution_role: "arn:aws:iam::${AWS::TNB::AccountId}:role/SampleHookPermission"
```

Security in AWS TNB

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS Telco Network Builder, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using AWS TNB. The following topics show you how to configure AWS TNB to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS TNB resources.

Contents

- Data protection in AWS TNB
- Identity and access management for AWS TNB
- Compliance validation for AWS TNB
- Resilience in AWS TNB
- Infrastructure security in AWS TNB
- IMDS version

Data protection in AWS TNB

The AWS shared responsibility model applies to data protection in AWS Telco Network Builder. As described in this model, AWS is responsible for protecting the global infrastructure that runs all

of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with AWS TNB or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Data handling

When you close your AWS account, AWS TNB marks your data for deletion and removes it from any use. If you reactivate your AWS account within 90 days, AWS TNB restores your data. After 120 days, AWS TNB permanently deletes your data. AWS TNB also terminates your networks and deletes your function packages and your network packages.

Encryption at rest

AWS TNB always encrypts all data stored in the service at rest without requiring any additional configuration. This encryption is automatic through AWS Key Management Service.

Encryption in transit

AWS TNB secures all data in transit using Transport Layer Security (TLS) 1.2.

It is your responsibility to encrypt data between your simulation agents and their clients.

Inter-network traffic privacy

AWS TNB compute resources reside in a virtual private cloud (VPC) shared by all customers. All internal AWS TNB traffic stayed within the AWS network and doesn't traverse the internet. Connections between your simulation agents and their clients are routed over the internet.

Identity and access management for AWS TNB

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AWS TNB resources. IAM is an AWS service that you can use with no additional charge.

Contents

- Audience
- Authenticating with identities
- Managing access using policies
- How AWS TNB works with IAM
- Identity-based policy examples for AWS Telco Network Builder
- Troubleshooting AWS Telco Network Builder identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS TNB.

Service user – If you use the AWS TNB service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS TNB features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS TNB, see Troubleshooting AWS Telco Network Builder identity and access.

Service administrator – If you're in charge of AWS TNB resources at your company, you probably have full access to AWS TNB. It's your job to determine which AWS TNB features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS TNB, see How AWS TNB works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS TNB. To view example AWS TNB identity-based policies that you can use in IAM, see Identity-based policy examples for AWS Telco Network Builder.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and AWS Multi-factor authentication in IAM in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. To temporarily assume an IAM role in the AWS Management Console, you can switch from a user to an IAM role (console). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Methods to assume a role in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or

store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

- **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
- **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of AWS services that support RCPs, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's

permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS TNB works with IAM

Before you use IAM to manage access to AWS TNB, learn what IAM features are available to use with AWS TNB.

IAM features you can use with AWS Telco Network Builder

IAM feature	AWS TNB support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	No

To get a high-level view of how AWS TNB and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for AWS TNB

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for AWS TNB

To view examples of AWS TNB identity-based policies, see Identity-based policy examples for AWS Telco Network Builder.

Resource-based policies within AWS TNB

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by

attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for AWS TNB

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of AWS TNB actions, see Actions defined by AWS Telco Network Builder in the *Service Authorization Reference*.

Policy actions in AWS TNB use the following prefix before the action:

```
tnb
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "tnb:CreateSolFunctionPackage",
      "tnb:DeleteSolFunctionPackage"
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `List`, include the following action:

```
"Action": "tnb:List*"
```

To view examples of AWS TNB identity-based policies, see Identity-based policy examples for AWS Telco Network Builder.

Policy resources for AWS TNB

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of AWS TNB resource types and their ARNs, see Resources defined by AWS Telco Network Builder in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by AWS Telco Network Builder.

To view examples of AWS TNB identity-based policies, see Identity-based policy examples for AWS Telco Network Builder.

Policy condition keys for AWS TNB

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of AWS TNB condition keys, see Condition keys for AWS Telco Network Builder in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by AWS Telco Network Builder.

To view examples of AWS TNB identity-based policies, see Identity-based policy examples for AWS Telco Network Builder.

ACLs in AWS TNB

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with AWS TNB

Supports ABAC (tags in policies): Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with AWS TNB

Supports temporary credentials: Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switch from a user to an IAM role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for AWS TNB

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for AWS TNB

Supports service roles: No

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

Service-linked roles for AWS TNB

Supports service-linked roles: No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

Identity-based policy examples for AWS Telco Network Builder

By default, users and roles don't have permission to create or modify AWS TNB resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by AWS TNB, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for AWS Telco Network Builder in the *Service Authorization Reference*.

Contents

- Policy best practices
- Using the AWS TNB console
- Service role policy examples
- Allow users to view their own permissions

Policy best practices

Identity-based policies determine whether someone can create, access, or delete AWS TNB resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the AWS TNB console

To access the AWS Telco Network Builder console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS TNB resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum

required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

Service role policy examples

As an administrator, you own and manage the resources that AWS TNB creates as defined by the environment and service templates. You must attach IAM service roles to your account to permit AWS TNB to create resources for your network life-cycle management.

A IAM service role allows AWS TNB to make calls to resources on your behalf to instantiate and manage your networks. If you specify a service role, AWS TNB uses that role's credential.

You create the service role and its permission policy with the IAM service. For more information about creating a service role, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

AWS TNB service role

As a member of the platform team, you can as an administrator create an AWS TNB service role and provide it to AWS TNB. This role allows AWS TNB to make calls to other services such as Amazon Elastic Kubernetes Service and AWS CloudFormation to provision the required infrastructure for your network and provision network functions as defined in your NSD.

We recommend that you use the following IAM role and trust policy for your AWS TNB service role. When scoping down permission on this policy, keep in mind that AWS TNB may fail with Access Denied errors toward resources descoped from your policy.

The following code shows an AWS TNB service role policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sts:GetCallerIdentity"
            ],
            "Resource": "*",
            "Effect": "Allow",
```

```
            "Sid": "AssumeRole"
        },
        {
            "Action": [
                "tnb:*"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "TNBPolicy"
        },
        {
            "Action": [
                "iam:AddRoleToInstanceProfile",
                "iam:CreateInstanceProfile",
                "iam:DeleteInstanceProfile",
                "iam:GetInstanceProfile",
                "iam:RemoveRoleFromInstanceProfile",
                "iam:TagInstanceProfile",
                "iam:UntagInstanceProfile"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "IAMPolicy"
        },
        {
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": [
                        "eks.amazonaws.com",
                        "eks-nodegroup.amazonaws.com"
                    ]
                }
            },
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "TNBAccessSLRPermissions"
        },
        {
            "Action": [
                "autoscaling:CreateAutoScalingGroup",
                "autoscaling:CreateOrUpdateTags",
```

```
                "autoscaling:DeleteAutoScalingGroup",
                "autoscaling:DeleteTags",
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribeAutoScalingInstances",
                "autoscaling:DescribeScalingActivities",
                "autoscaling:DescribeTags",
                "autoscaling:UpdateAutoScalingGroup",
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:CreateLaunchTemplate",
                "ec2:CreateLaunchTemplateVersion",
                "ec2:CreateSecurityGroup",
                "ec2:DeleteLaunchTemplateVersions",
                "ec2:DescribeLaunchTemplates",
                "ec2:DescribeLaunchTemplateVersions",
                "ec2:DeleteLaunchTemplate",
                "ec2:DeleteSecurityGroup",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeTags",
                "ec2:GetLaunchTemplateData",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:RevokeSecurityGroupIngress",
                "ec2:RunInstances",
                "ec2:AssociateRouteTable",
                "ec2:AttachInternetGateway",
                "ec2:CreateInternetGateway",
                "ec2:CreateNetworkInterface",
                "ec2:CreateRoute",
                "ec2:CreateRouteTable",
                "ec2:CreateSubnet",
                "ec2:CreateTags",
                "ec2:CreateVpc",
                "ec2:DeleteInternetGateway",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteRoute",
                "ec2:DeleteRouteTable",
                "ec2:DeleteSubnet",
                "ec2:DeleteTags",
                "ec2:DeleteVpc",
                "ec2:DetachNetworkInterface",
                "ec2:DescribeInstances",
                "ec2:DescribeInternetGateways",
                "ec2:DescribeKeyPairs",
                "ec2:DescribeNetworkInterfaces",
```

```
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroupRules",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:DetachInternetGateway",
                "ec2:DisassociateRouteTable",
                "ec2:ModifySecurityGroupRules",
                "ec2:ModifySubnetAttribute",
                "ec2:ModifyVpcAttribute",
                "ec2:AllocateAddress",
                "ec2:AssignIpv6Addresses",
                "ec2:AssociateAddress",
                "ec2:AssociateNatGatewayAddress",
                "ec2:AssociateVpcCidrBlock",
                "ec2:CreateEgressOnlyInternetGateway",
                "ec2:CreateNatGateway",
                "ec2:DeleteEgressOnlyInternetGateway",
                "ec2:DeleteNatGateway",
                "ec2:DescribeAddresses",
                "ec2:DescribeEgressOnlyInternetGateways",
                "ec2:DescribeNatGateways",
                "ec2:DisassociateAddress",
                "ec2:DisassociateNatGatewayAddress",
                "ec2:DisassociateVpcCidrBlock",
                "ec2:ReleaseAddress",
                "ec2:UnassignIpv6Addresses",
                "ec2:DescribeImages",
                "eks:CreateCluster",
                "eks:ListClusters",
                "eks:RegisterCluster",
                "eks:TagResource",
                "eks:DescribeAddonVersions",
                "events:DescribeRule",
                "iam:GetRole",
                "iam:ListAttachedRolePolicies",
                "iam:PassRole"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "TNBAccessComputePerms"
        },
        {
            "Action": [
                "codebuild:BatchDeleteBuilds",
```

```
                "codebuild:BatchGetBuilds",
                "codebuild:CreateProject",
                "codebuild:DeleteProject",
                "codebuild:ListBuildsForProject",
                "codebuild:StartBuild",
                "codebuild:StopBuild",
                "events:DeleteRule",
                "events:PutRule",
                "events:PutTargets",
                "events:RemoveTargets",
                "s3:CreateBucket",
                "s3:GetBucketAcl",
                "s3:GetObject",
                "eks:DescribeNodegroup",
                "eks:DeleteNodegroup",
                "eks:AssociateIdentityProviderConfig",
                "eks:CreateNodegroup",
                "eks:DeleteCluster",
                "eks:DeregisterCluster",
                "eks:UpdateAddon",
                "eks:UpdateClusterVersion",
                "eks:UpdateNodegroupConfig",
                "eks:UpdateNodegroupVersion",
                "eks:DescribeUpdate",
                "eks:UntagResource",
                "eks:DescribeCluster",
                "eks:ListNodegroups",
                "eks:CreateAddon",
                "eks:DeleteAddon",
                "eks:DescribeAddon",
                "eks:DescribeAddonVersions",
                "s3:PutObject",
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
                "cloudformation:DescribeStackResources",
                "cloudformation:DescribeStacks",
                "cloudformation:UpdateStack",
                "cloudformation:UpdateTerminationProtection"
            ],
            "Resource": [
                "arn:aws:events:*:*:rule/tnb*",
                "arn:aws:codebuild:*:*:project/tnb*",
                "arn:aws:logs:*:*:log-group:/aws/tnb*",
                "arn:aws:s3:::tnb*",
```

```
                "arn:aws:eks:*:*:addon/tnb*/*/*",
                "arn:aws:eks:*:*:cluster/tnb*",
                "arn:aws:eks:*:*:nodegroup/tnb*/tnb*/*",
                "arn:aws:cloudformation:*:*:stack/tnb*"
            ],
            "Effect": "Allow",
            "Sid": "TNBAccessInfraResourcePerms"
        },
        {
            "Sid": "CFNTemplatePerms",
            "Effect": "Allow",
            "Action": [
              "cloudformation:GetTemplateSummary"
            ],
            "Resource": "*"
         },
        {
            "Sid": "ImageAMISSMPerms",
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": [
                "arn:aws:ssm:*::parameter/aws/service/eks/optimized-ami/*",
                "arn:aws:ssm:*::parameter/aws/service/bottlerocket/*"
            ]
        },
        {
            "Action": [
                "tag:GetResources"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "TaggingPolicy"
        },
        {
            "Action": [
                "outposts:GetOutpost"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "OutpostPolicy"
        }
    ]
```

```
}
```

The following code shows the AWS TNB service trust policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "events.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "codebuild.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "tnb.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
```

```
}
```

AWS TNB service role for Amazon EKS cluster

When you create an Amazon EKS resources in your NSD, you provide the `cluster_role` attribute to specify which role will be used to create your Amazon EKS cluster.

The following example shows a AWS CloudFormation template that creates a AWS TNB service role for the Amazon EKS cluster policy.

```
AWSTemplateFormatVersion: "2010-09-09"
Resources:
  TNBEKSClusterRole:
    Type: "AWS::IAM::Role"
    Properties:
      RoleName: "TNBEKSClusterRole"
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - eks.amazonaws.com
            Action:
              - "sts:AssumeRole"
      Path: /
      ManagedPolicyArns:
        - !Sub "arn:${AWS::Partition}:iam::aws:policy/AmazonEKSClusterPolicy"
```

For more information about IAM roles using AWS CloudFormation template, see the following sections in the *AWS CloudFormation User Guide*:

- AWS::IAM::Role
- Selecting a stack template

AWS TNB service role for Amazon EKS node group

When you create an Amazon EKS node group resources in your NSD, you provide the `node_role` attribute to specify which role will be used to create your Amazon EKS node group.

The following example shows a AWS CloudFormation template that creates a AWS TNB service role for the Amazon EKS node group policy.

```
AWSTemplateFormatVersion: "2010-09-09"
Resources:
  TNBEKSNodeRole:
    Type: "AWS::IAM::Role"
    Properties:
      RoleName: "TNBEKSNodeRole"
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - ec2.amazonaws.com
            Action:
              - "sts:AssumeRole"
      Path: /
      ManagedPolicyArns:
        - !Sub "arn:${AWS::Partition}:iam::aws:policy/AmazonEKSWorkerNodePolicy"
        - !Sub "arn:${AWS::Partition}:iam::aws:policy/AmazonEKS_CNI_Policy"
        - !Sub "arn:${AWS::Partition}:iam::aws:policy/
AmazonEC2ContainerRegistryReadOnly"
        - !Sub "arn:${AWS::Partition}:iam::aws:policy/service-role/
AmazonEBSCSIDriverPolicy"
      Policies:
        - PolicyName: EKSNodeRoleInlinePolicy
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: Allow
                Action:
                  - "logs:DescribeLogStreams"
                  - "logs:PutLogEvents"
                  - "logs:CreateLogGroup"
                  - "logs:CreateLogStream"
                Resource: "arn:aws:logs:*:*:log-group:/aws/tnb/tnb*"
        - PolicyName: EKSNodeRoleIpv6CNIPolicy
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: Allow
```

```
                Action:
                  - "ec2:AssignIpv6Addresses"
                Resource: "arn:aws:ec2:*:*:network-interface/*"
```

For more information about IAM roles using AWS CloudFormation template, see the following sections in the *AWS CloudFormation User Guide*:

- AWS::IAM::Role
- Selecting a stack template

AWS TNB service role for Multus

When you create an Amazon EKS resource in your NSD and you want to manage Multus as part of your deployment template, you must provide the `multus_role` attribute to specify which role will be used for managing Multus.

The following example shows a AWS CloudFormation template that creates a AWS TNB service role for a Multus policy.

```
AWSTemplateFormatVersion: "2010-09-09"
Resources:
  TNBMultusRole:
    Type: "AWS::IAM::Role"
    Properties:
      RoleName: "TNBMultusRole"
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action:
              - "sts:AssumeRole"
          - Effect: Allow
            Principal:
              Service:
                - codebuild.amazonaws.com
            Action:
              - "sts:AssumeRole"
      Path: /
```

```
      Policies:
        - PolicyName: MultusRoleInlinePolicy
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: Allow
                Action:
                  - "codebuild:StartBuild"
                  - "logs:DescribeLogStreams"
                  - "logs:PutLogEvents"
                  - "logs:CreateLogGroup"
                  - "logs:CreateLogStream"
                Resource:
                  - "arn:aws:codebuild:*:*:project/tnb*"
                  - "arn:aws:logs:*:*:log-group:/aws/tnb/*"
              - Effect: Allow
                Action:
                  - "ec2:CreateNetworkInterface"
                  - "ec2:ModifyNetworkInterfaceAttribute"
                  - "ec2:AttachNetworkInterface"
                  - "ec2:DeleteNetworkInterface"
                  - "ec2:CreateTags"
                  - "ec2:DetachNetworkInterface"
                Resource: "*"
```

For more information about IAM roles using AWS CloudFormation template, see the following sections in the *AWS CloudFormation User Guide*:

- AWS::IAM::Role
- Selecting a stack template

AWS TNB service role for a life-cycle hook policy

When your NSD or network function package uses a life-cycle hook, you need a service role to allow you to create an environment for execution of your life-cycle hooks.

> **Note**
>
> Your life-cycle hook policy should be based on what your life-cycle hook is attempting to do.

The following example shows a AWS CloudFormation template that creates a AWS TNB service role for a life-cycle hook policy.

```
AWSTemplateFormatVersion: "2010-09-09"
Resources:
  TNBHookRole:
    Type: "AWS::IAM::Role"
    Properties:
      RoleName: "TNBHookRole"
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - codebuild.amazonaws.com
            Action:
              - "sts:AssumeRole"
      Path: /
      ManagedPolicyArns:
        - !Sub "arn:${AWS::Partition}:iam::aws:policy/AdministratorAccess"
```

For more information about IAM roles using AWS CloudFormation template, see the following sections in the *AWS CloudFormation User Guide*:

- AWS::IAM::Role
- Selecting a stack template

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
```

```
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Troubleshooting AWS Telco Network Builder identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS TNB and IAM.

Issues

- I am not authorized to perform an action in AWS TNB
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my AWS TNB resources

I am not authorized to perform an action in AWS TNB

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but does not have the fictional `tnb:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 tnb:GetWidget on resource: my-example-widget
```

In this case, Mateo's policy must be updated to allow him to access the *`my-example-widget`* resource using the `tnb:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AWS TNB.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS TNB. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AWS TNB resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support

resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS TNB supports these features, see How AWS TNB works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Compliance validation for AWS TNB

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security Compliance & Governance – These solution implementation guides discuss architectural considerations and provide steps for deploying security and compliance features.
- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

> **Note**
>
> Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in AWS TNB

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

AWS TNB runs your Network Service on EKS clusters in a virtual private cloud (VPC) in the AWS Region that you choose.

Infrastructure security in AWS TNB

As a managed service, AWS Telco Network Builder is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access AWS TNB through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Here are some examples of shared responsibilities:

- AWS is responsible for securing components that support AWS TNB, including:
  - Compute instances (also known as *workers*)
  - Internal databases
  - Network communications between internal components
  - The AWS TNB application programming interface (API)
  - AWS Software Development Kits (SDK)
- You are responsible for securing your access to your AWS resources and your workload components, including (but not limited to):
  - IAM users, groups, roles, and policies
  - S3 buckets that you use to store your data for AWS TNB
  - Other AWS services and resources that you use to support the network service that you provisioned through AWS TNB
  - Your application code

- Connections between the network service that you provisioned through AWS TNB and its clients

> **Important**
>
> You are responsible for implementing a disaster recovery plan that can effectively recover a network service that you provisioned through AWS TNB.

Network connectivity security model

The network services that you provision through AWS TNB, run on compute instances within a virtual private cloud (VPC) located in an AWS Region that you select. A VPC is a virtual network in the AWS Cloud, which isolates infrastructure by workload or organizational entity. Communication between compute instances within VPCs stay within the AWS network and don't travel over the internet. Some internal service communication crosses the internet, and is encrypted. Network services provisioned through AWS TNB for all customers running in the same Region share the same VPC. Network services provisioned through AWS TNB for different customers use separate compute instances within the same VPC.

Communications between your network service clients and your network service in AWS TNB traverse the internet. AWS TNB does not manage these connections. It is your responsibility to secure your client connections.

Your connections to AWS TNB through the AWS Management Console, AWS Command Line Interface (AWS CLI), and AWS SDKs are encrypted.

IMDS version

AWS TNB supports instances that leverage Instance Metadata Service version 2 (IMDSv2), a session-oriented method. IMDSv2 includes higher security than IMDSV1. For more information, see Add defense in depth against open firewalls, reverse proxies, and SSRF vulnerabilities with enhancements to the Amazon EC2 Instance Metadata Service.

When launching your instance, you must use IMDSv2. For more information on IMDSv2, see Use IMDSv2 in the *Amazon EC2 User Guide*.

Monitoring AWS TNB

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS TNB and your other AWS solutions. AWS provides AWS CloudTrail to watch AWS TNB, report when something is wrong, and take automatic actions when appropriate.

Use CloudTrail to capture detailed information about the calls made to AWS APIs. You can store these calls as log files in Amazon S3. You can use these CloudTrail logs to determine such information as which call was made, the source IP address where the call came from, who made the call, and when the call was made.

The CloudTrail logs contain information about the calls to API actions for AWS TNB. They also contain information for calls to API actions from services such as Amazon EC2 and Amazon EBS.

Logging AWS Telco Network Builder API calls using AWS CloudTrail

AWS Telco Network Builder is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service. CloudTrail captures all API calls for AWS TNB as events. The calls captured include calls from the AWS TNB console and code calls to the AWS TNB API operations. Using the information collected by CloudTrail, you can determine the request that was made to AWS TNB, the IP address from which the request was made, when it was made, and additional details.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root user or user credentials.
- Whether the request was made on behalf of an IAM Identity Center user.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

CloudTrail is active in your AWS account when you create the account and you automatically have access to the CloudTrail **Event history**. The CloudTrail **Event history** provides a viewable, searchable, downloadable, and immutable record of the past 90 days of recorded management

events in an AWS Region. For more information, see Working with CloudTrail Event history in the *AWS CloudTrail User Guide*. There are no CloudTrail charges for viewing the **Event history**.

For an ongoing record of events in your AWS account past 90 days, create a trail or a CloudTrail Lake event data store.

CloudTrail trails

A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. All trails created using the AWS Management Console are multi-Region. You can create a single-Region or a multi-Region trail by using the AWS CLI. Creating a multi-Region trail is recommended because you capture activity in all AWS Regions in your account. If you create a single-Region trail, you can view only the events logged in the trail's AWS Region. For more information about trails, see Creating a trail for your AWS account and Creating a trail for an organization in the *AWS CloudTrail User Guide*.

You can deliver one copy of your ongoing management events to your Amazon S3 bucket at no charge from CloudTrail by creating a trail, however, there are Amazon S3 storage charges. For more information about CloudTrail pricing, see AWS CloudTrail Pricing. For information about Amazon S3 pricing, see Amazon S3 Pricing.

CloudTrail Lake event data stores

CloudTrail Lake lets you run SQL-based queries on your events. CloudTrail Lake converts existing events in row-based JSON format to Apache ORC format. ORC is a columnar storage format that is optimized for fast retrieval of data. Events are aggregated into *event data stores*, which are immutable collections of events based on criteria that you select by applying advanced event selectors. The selectors that you apply to an event data store control which events persist and are available for you to query. For more information about CloudTrail Lake, see Working with AWS CloudTrail Lake in the *AWS CloudTrail User Guide*.

CloudTrail Lake event data stores and queries incur costs. When you create an event data store, you choose the pricing option you want to use for the event data store. The pricing option determines the cost for ingesting and storing events, and the default and maximum retention period for the event data store. For more information about CloudTrail pricing, see AWS CloudTrail Pricing.

AWS TNB event examples

An event represents a single request from any source and includes information about the requested API operation, the date and time of the operation, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so events don't appear in any specific order.

The following example shows a CloudTrail event that demonstrates the `CreateSolFunctionPackage` operation.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:example",
        "arn": "arn:aws:sts::111222333444:assumed-role/example/user",
        "accountId": "111222333444",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::111222333444:role/example",
                "accountId": "111222333444",
                "userName": "example"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-02-02T01:42:39Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2023-02-02T01:43:17Z",
    "eventSource": "tnb.amazonaws.com",
    "eventName": "CreateSolFunctionPackage",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "XXX.XXX.XXX.XXX",
    "userAgent": "userAgent",
    "requestParameters": null,
    "responseElements": {
        "vnfPkgArn": "arn:aws:tnb:us-east-1:111222333444:function-package/
fp-12345678abcEXAMPLE",
```

```
        "id": "fp-12345678abcEXAMPLE",
        "operationalState": "DISABLED",
        "usageState": "NOT_IN_USE",
        "onboardingState": "CREATED"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111222333444",
    "eventCategory": "Management"
}
```


For information about CloudTrail record contents, see CloudTrail record contents in the *AWS CloudTrail User Guide*.

AWS TNB deployment tasks

Understand the deployment tasks to effectively monitor deployments and take action faster.

The following table lists the AWS TNB deployment tasks:

Task name for deployments started before March 7, 2024	Task name for deployments started on and after March 7, 2024	Task description
AppInstallation	ClusterPluginInstall	Installs the Multus plugin on the Amazon EKS cluster.
AppUpdate	*no change in name*	Updates the network functions that are already installed in a network instance.
-	ClusterPluginUninstall	Uninstalls the plugins on the Amazon EKS cluster.
ClusterStorageClassConfiguration	*no change in name*	Configures the storage class (CSI driver) on an Amazon EKS cluster.

Task name for deployments started before March 7, 2024	Task name for deployments started on and after March 7, 2024	Task description
FunctionDeletion	*no change in name*	Deletes network functions from AWS TNB resources.
FunctionInstantiation	FunctionInstall	Deploys network functions using HELM.
FunctionUninstallation	FunctionUninstall	Uninstalls the network function from an Amazon EKS cluster.
HookExecution	*no change in name*	Executes lifecycle hooks as defined in the NSD.
InfrastructureCancellation	*no change in name*	Cancels a network service.
InfrastructureInstantation	*no change in name*	Provisions AWS resources on behalf of the user.
InfrastructureTermination	*no change in name*	Deprovisions AWS resources invoked through AWS TNB.
-	InfrastructureUpdate	Updates the AWS resources provisioned on behalf of the user.
InventoryDeregistration	*no change in name*	Deregisters AWS resources from AWS TNB.
-	InventoryRegistration	Registers the AWS resources in AWS TNB.
KubernetesClusterConfiguration	ClusterConfiguration	Configures the Kubernetes cluster and adds additional IAM roles to the Amazon EKS AuthMap as defined in the NSD.
NetworkServiceFinalization	*no change in name*	Finalizes the network service and provides a success or failure status update.

Task name for deployments started before March 7, 2024	Task name for deployments started on and after March 7, 2024	Task description
NetworkServiceInstantiation	*no change in name*	Initializes the network service.
SelfManagedNodesConfiguration	*no change in name*	Bootstraps self-managed nodes with Amazon EKS and Kubernetes control plane.
-	ValidateNetworkServiceUpdate	Runs the validations before updating a network instance.

Service quotas for AWS TNB

Service quotas, also referred to as limits, are the maximum number of service resources or operations for your AWS account. For more information, see AWS service quotas in the *Amazon Web Services General Reference*.

The following are the service quotas for AWS TNB.

Name	Default	Adjustable	Description
Concurrent ongoing network service operations	Each supported Region: 40	Yes	The maximum number of concurrent ongoing network service operations in one Region.
Function packages	Each supported Region: 200	Yes	The maximum number of function packages in one region.
Network packages	Each supported Region: 40	Yes	The maximum number of network packages in one region.
Network service instances	Each supported Region: 800	Yes	The maximum number of network service instances in one Region.

Document history for the AWS TNB user guide

The following table describes the documentation releases for AWS TNB.

Change	Description	Date
Kubernetes version for cluster	AWS TNB now supports Kubernetes version 1.30 to create Amazon EKS clusters.	August 19, 2024

AWS TNB supports an additional operation to manage the network lifecycle.	You can update an instantiated or previously updated network instance with a new network package and parameter values. See:
	• Lifecycle operations
	• Update a network instance
	• AWS TNB service role example:
	  • Add these Amazon EKS actions: `eks:UpdateAddon`, `eks:UpdateClusterVersion`, `eks:UpdateNodegroupConfig`, `eks:UpdateNodegroupVersion`, `eks:DescribeUpdate`
	  • Add this AWS CloudFormation action: `cloudformation:UpdateStack`
	• New Deployment tasks: `InfrastructureUpdate`, `InventoryRegistration`,
		July 30, 2024

	`ValidateNetworkServiceUpdate` • API updates: GetSolNetworkOperation, ListSolNetworkOperations , and UpdateSolNetworkInstance	
New task and new task names for existing tasks	A new task is available. As of March 7, 2024, some existing tasks have new names for clarity.	May 7, 2024
Kubernetes version for cluster	AWS TNB now supports Kubernetes version 1.29 to create Amazon EKS clusters.	April 10, 2024
Support for network interface `security_groups`	You can attach security groups to the AWS.Networking.ENI node.	April 2, 2024
Support for Amazon EBS root volume encryption	You can enable Amazon EBS encryption for the Amazon EBS root volume. To enable, add the properties in the AWS.Compute.EKSManagedNode or AWS.Compute.EKSSelfManagedNode node.	April 2, 2024
Support for node `labels`	You can attach node labels to your node group in the AWS.Compute.EKSManagedNode or AWS.Compute.EKSSelfManagedNode node.	March 19, 2024

Support for network interface `source_dest_check`	You can indicate whether you want to enable or disable the network interface source/destination check through the AWS.Networking.ENI node.	January 25, 2024
Support for Amazon EC2 instances with custom user data	You can launch Amazon EC2 instances with custom user data through the AWS.Compute.UserData node.	January 16, 2024
Support for Security Group	AWS TNB allows you to import the Security Group AWS resource.	January 8, 2024
Updated description of `network_interfaces`	When the `network_interfaces` property is included in the AWS.Compute.EKSManagedNode or AWS.Compute.EKSSelfManagedNode node, AWS TNB gets the permission related to ENIs from the `multus_role` property if available, or from the `node_role` property.	December 18, 2023
Support for private cluster	AWS TNB now supports private clusters. To indicate a private cluster, set the `access` property to `PRIVATE`.	December 11, 2023
Kubernetes version for cluster	AWS TNB now supports Kubernetes version 1.28 to create Amazon EKS clusters.	December 11, 2023

AWS TNB supports placement group	Added placement group for the `AWS.Compute.EKSManagedNode` and `AWS.Compute.EKSSelfManagedNode` node definitions.	December 11, 2023
AWS TNB adds support for IPv6	AWS TNB now supports creating network instances with IPv6 infrastructure. Check the nodes AWS.Networking.VPC, AWS.Networking.Subnet, AWS.Networking.InternetGateway, AWS.Networking.SecurityGroupIngressRule, AWS.Networking.SecurityGroupEgressRule, and AWS.Compute.EKS for IPv6 configurations. We also added the nodes AWS.Networking.NATGateway and AWS.Networking.Route for NAT64 configuration. We updated the AWS TNB service role and the AWS TNB service role for Amazon EKS node group for IPv6 permissions. See Service role policy examples.	November 16, 2023
Added permissions to the AWS TNB service role policy	We added permissions to the AWS TNB service role policy for Amazon S3 and AWS CloudFormation to enable infrastructure instantiation.	October 23, 2023

AWS TNB launched in more Regions	AWS TNB is now available in the Asia Pacific (Seoul), Canada (Central), Europe (Spain), Europe (Stockholm), and South America (São Paulo) Regions.	September 27, 2023
Tags for AWS.Compute.EKSSelfManagedNode	AWS TNB now supports tags for the `AWS.Compute.EKSSelfManagedNode` node definition.	August 22, 2023
AWS TNB supports instances that leverage IMDSv2	When launching your instance, you must use IMDSv2.	August 14, 2023
Updated permissions for the MultusRoleInlinePolicy	The MultusRoleInlinePolicy now includes the `ec2:DeleteNetworkInterface` permission.	August 7, 2023
Kubernetes version for cluster	AWS TNB now supports Kubernetes versions 1.27 to create Amazon EKS clusters.	July 25, 2023
AWS.Compute.EKS.AuthRole	AWS TNB supports AuthRole that allows you to add IAM roles to the Amazon EKS cluster `aws-auth ConfigMap` so that users can access the Amazon EKS cluster using an IAM role.	July 19, 2023

AWS TNB supports security groups.	Added the AWS.Networking.SecurityGroup, AWS.Networking.SecurityGroupEgressRule, and AWS.Networking.SecurityGroupIngressRule to the NSD template.	July 18, 2023
Kubernetes version for cluster	AWS TNB supports Kubernetes versions 1.22 through 1.26 to create Amazon EKS clusters. AWS TNB no longer supports Kubernetes versions 1.21.	May 11, 2023
AWS.Compute.EKSSelfManagedNode	You can create self-managed worker nodes on in-region, AWS Local Zones, and AWS Outposts.	March 29, 2023
Initial release	This is the first release of the AWS TNB User Guide.	February 21, 2023