Contents

1. Implementing Egress and Ingress Scenarios Using Selected CNI Plugins 2

1.1. Tools and automation 2

1.2. Egress scenario implementation 8

1.2.1. Antrea 8

1.2.2. Cilium 8

1.3. Inress scenario implementation 8

1.3.1. Cloud deployment 8

1.3.2. Local deployment 8

1. Implementing Egress and Ingress Scenarios Using Selected CNI Plugins

This chapter presents the implementation of egress and ingress scenarios. The egress scenario will be executed locally, while the ingress scenario will be deployed both on local infrastructure (personal laptop) and on the public cloud (Azure). The tools used in this implementation with some example configurations will be described. The Kubernetes cluster will run on a local laptop with the following specifications:

- CPU: AMD Ryzen 8CPUs
- RAM: 20 GB
- Storage: 256 GB SSD
- Operating System: Fedora 40

Cloud infrastrucure consist of two AKS nodes, Azure Standard_A2_v2 virtual machines. Each VM has the following specifications:

- CPU: 2 vCPUs
- RAM: 4 GB
- Storage: Standard SSD
- Operating System: Ubuntu 22.04

1.1. Tools and automation

In this section, the tools used to provision the egress and ingress implementations will be described. A Kubernetes cluster will be created to simulate the scenarios, and Infrastructure as

Code (IaC) tool Terraform will be used to provision and interact with the cluster. Additionally, Ansible will be used for creating, configuring cluster setups along with running terraform and performance tools.

Ansible

Ansible is an opensource tool which is able to automate provisioning and configuring infrastructure. Configuration in Ansible is written in playbooks, which are YAML files as blueprints that contain a set of instructions to be executed. Each playbook consists of one or more plays, and each play describes a set of tasks to be performed on a group of desired hosts [1] [2].

```yaml
- name: Create openstack instance and assign floating ip
  hosts: "{{ openstack_pool | default('localhost') }}"
  var_files:
    - ./vars/auth.yml
  become: yes

  tasks:
    - name: Create the OpenStack instance
      openstack.cloud.server:
        state: present
        name: " {{ inventory_hostname }}"
        key_name: "{{ key_name }}"
        network: "{{ network_name }}"
        auth:
          auth_url: "{{ auth_url }}"
          username: "{{ username }}"
          password: "{{ password }}"
          project_name: "{{ project_name }}"

  roles:
    - assign_floating_ip

```

Listing 1: Example ansible playbook [3].

```
[openstack_pool]
instance-1.example.com key_name=ansible_key network_name=my-network ansible_host=10.10.10.10
instance-2.example.com key_name=ansible_key network_name=my-network ansible_host=10.10.10.20
```

Listing 2: Example ansible inventory [4] [3] [5]

Listing 1 shows example ansible playbook configuration. Hosts field define group of objects on which configuration script is executed. In this case instances specified in group named openstack_pool in ansible inventory showed on listing 2 will be created when using the playbook. Using var_files is possible to attach file containg, for example authentication variables required to access openstack cloud. Become is used to execute script as root user. Tasks and roles is the place, where actual script is defined. It can be defined directly in tasks, or specified by a roles, in this case will use script from ./roles/assign_floating_ip/tasks/main.yaml [2].

Iperf3

Iperf3 is a tool capable of measure networking metrics. It supports TCP, UDP and SCTP protocols in IPv4 or IPv6 networks. The tool works in client-server architecture. Iperf3 will be used to evalutate throughput and round trip time in egress scenario. Listings 3 and 4 shows how to run iperf three seconds measurement within localhost interface [6].

```
$ iperf3 --server
-----------------------------------------------------------
Server listening on 5201 (test #1)
-----------------------------------------------------------
Accepted connection from 127.0.0.1, port 40496
[  5] local 127.0.0.1 port 5201 connected to 127.0.0.1 port 40502
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec  5.55 GBytes  47.6 Gbits/sec
[  5]   1.00-2.00   sec  5.97 GBytes  51.3 Gbits/sec
[  5]   2.00-3.00   sec  5.50 GBytes  47.3 Gbits/sec
[  5]   3.00-3.00   sec  4.50 MBytes  38.7 Gbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-3.00   sec  17.0 GBytes  48.7 Gbits/sec                  receiver
```

Listing 3: Running iperf3 server command [7].

```
$ iperf3 --client 127.0.0.1 --time 3
Connecting to host 127.0.0.1, port 5201
[  5] local 127.0.0.1 port 40502 connected to 127.0.0.1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  5.55 GBytes  47.6 Gbits/sec    0   1.31 MBytes
[  5]   1.00-2.00   sec  5.97 GBytes  51.3 Gbits/sec    0   1.50 MBytes
[  5]   2.00-3.00   sec  5.50 GBytes  47.2 Gbits/sec    0   1.50 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-3.00   sec  17.0 GBytes  48.7 Gbits/sec    0             sender
[  5]   0.00-3.00   sec  17.0 GBytes  48.7 Gbits/sec                  receiver

iperf Done.
```

Listing 4: Running iperf3 client command [7].

Kind

Kind is a tool used for creating local Kuberentes cluster. This tool can simulate real communication between nodes within one machine. It creates control plane and worker nodes using

Docker containers to enable node to node communication. The important thing is that it does not provide any load balancer for assigning external IP addresses for Kubernetes services. In ingress scenario Gateway API requires outside cluster routable IP address, on local infrastructure MetalLB will be installed and configured to provide IPs [8]. Listing 5 shows kind configuration used in both egress and ingress scenation on local infrastructure. One control plane node and two worker nodes are created. It also disables default CNI plugin which is essentail in this case.

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
networking:
    disableDefaultCNI: true
nodes:
    - role: control-plane
      extraPortMappings:
        - containerPort: 80
          hostPort: 80
        - containerPort: 443
          hostPort: 443
    - role: worker
    - role: worker
```

Listing 5: Kind config used in both scenarios [9].

K6

Grafana K6 is open-source tool designed for load testing by simulating virtual users accessing specified endpoints. The testing configuration is written in a JavaScript file using the k6 library. Listing 6 demonstrates k6 script used in the ingres scenario. It is written as an Ansible template. During the ingress test Ansible injects variables into the script, allowing k6 to probe the Gateway API [10].

```
import http from 'k6/http';

export const options = {
  vus: "{{ number_of_vusers }}",
  duration: "{{ test_duration }}s",
};

export default function () {
    const timestamp = new Date().toISOString();

    const res = http.get('http://{{ gateway_api_ip.stdout }}/echo');

    const hostnameMatch = res.body.match(/Hostname:\s*(\S+)/);
    const hostname = hostnameMatch ? hostnameMatch[1] : 'Hostname not found';

    console.log(`[${timestamp}] Hostname: ${hostname}`);
}
```

Listing 6: Grafana k6 script used in the infress scenario [11].

MetalLB

MetalLB is an implementation of load balancer for bare metal Kubernetes. Kind does not provide implementation of load balancer. Without external tool like this, load balancer type service will persist in "pending" state. It is mandatory in ingres scenario to create loadbalancing serbice for Gateway API [12].

Node Exporter

A tool that exports the current system's metrics, such as CPU usage, memory utilization, disk I/O, and network statistics. The metrics in OpenMetrics format are exposed on /metrics enpoind [13].

Prometheus

Prometheus is a monitoring and alerting tool, which stores data in time series, any data value is associated with time when it was collected. Stored data can be retrieved using PromQL (query language). The tool colects data by pulling from specified enpoints in configuration [14].

Terraform

Terraform is an open-source infrastructure as code (IaC) tool. It allows provision, and manage infrastructure resources, cloud infrastructure, kubernetes cluster, virtual machines, docker

containers, storage and also SaaS features. Configuration files are written in a declarative language called HashiCorp Configuration Language (HCL) shown on listing 7 [15]. The script is responsible for creating Kubernetes cluster in Azure services. It is important to choose appropriate location for cluster, define default node pool (virtual machine type and node count) and to get rid of default CNI by setting network_plugin in network_profile to "none" if different networking plugin is preffered [16].

The Terraform workflow is made up of three stages [15]:

1. Write – define in configuration file resources to be created

2. Plan – shows the actual resources that will be created based on the provided configuration and checks for any errors in the code

3. Apply – provision resources or applys changes defined in write stage to the infrastructure

```
resource "azurerm_resource_group" "rg" {
  location = var.resource_group_location
  name     = "rg${var.common_infix}"
}

resource "azurerm_kubernetes_cluster" "k8s" {
  location            = azurerm_resource_group.rg.location
  name                = "cluster${var.common_infix}"
  resource_group_name = azurerm_resource_group.rg.name
  dns_prefix          = "dns${var.common_infix}"

  identity {
    type = "SystemAssigned"
  }

  default_node_pool {
    name       = "agentpool"
    vm_size    = var.vm_type
    node_count = var.node_count
  }

  linux_profile {
    admin_username = var.username

    ssh_key {
      key_data = azapi_resource_action.ssh_public_key_gen.output.publicKey
    }
  }

  network_profile {
    network_plugin    = "none"
    load_balancer_sku = "standard"
  }
}
```

Listing 7: Terraform Azure Kubernetes Service creation script [16].

1.2. Egress scenario implementation

1.2.1. Antrea

1.2.2. Cilium

1.3. Inress scenario implementation

1.3.1. Cloud deployment

1.3.2. Local deployment

Bibliography

[1] Inc. Red Hat. Introduction to Ansible. https://docs.ansible.com/ansible/latest/getting_started/introduction.html. Accessed, 17-Dec-2024.

[2] Inc. Red Hat. Creating a playbook. https://docs.ansible.com/ansible/latest/getting_started/get_started_playbook.html. Accessed, 17-Dec-2024.

[3] Inc. Red Hat. openstack.cloud.server module – Create/Delete Compute Instances from OpenStack. https://docs.ansible.com/ansible/latest/collections/openstack/cloud/server_module.html. Accessed, 17-Dec-2024.

[4] Inc. Red Hat. How to build your inventory. https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html. Accessed, 17-Dec-2024.

[5] Inc. Red Hat. Special Variables. https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html. Accessed, 17-Dec-2024.

[6] Jon Dugan, Seth Elliott, Bruce A. Mah, Jeff Poskanzer, and Kaustubh Prabhu. What is iPerf / iPerf3 ? https://iperf.fr/. Accessed, 17-Dec-2024.

[7] Jon Dugan, Seth Elliott, Bruce A. Mah, Jeff Poskanzer, and Kaustubh Prabhu. iPerf 3 user documentation. https://iperf.fr/iperf-doc.php. Accessed, 17-Dec-2024.

[8] The Kubernetes Authors. Docs. https://kind.sigs.k8s.io/. Accessed, 17-Dec-2024.

[9] The Kubernetes Authors. Docs. https://kind.sigs.k8s.io/docs/user/configuration/. Accessed, 17-Dec-2024.

[10] Inc. Red Hat. Templating (Jinja2). https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_templating.html. Accessed, 18-Dec-2024.

[11] Grafana Labs. k6/http. https://grafana.com/docs/k6/latest/javascript-api/k6-http/. Accessed, 18-Dec-2024.

[12] Cloud Native Computing Foundation. MetalLB. https://metallb.io/. Accessed, 18-Dec-2024.

[13] Cloud Native Computing Foundation. Node exporter. https://github.com/prometheus/node_exporter. Accessed, 18-Dec-2024.

[14] Cloud Native Computing Foundation. Node exporter. https://prometheus.io/docs/introduction/overview/. Accessed, 18-Dec-2024.

[15] HashiCorp. What is Terraform? https://developer.hashicorp.com/terraform/intro. Accessed, 18-Dec-2024.

[16] Microsoft. Quickstart: Deploy an Azure Kubernetes Service (AKS) cluster using Terraform. https://learn.microsoft.com/en-us/azure/aks/learn/quick-kubernetes-deploy-terraform. Accessed, 18-Dec-2024.