Installing Thunder Container on Kubernetes

July, 2021

A10

PATENT PROTECTION

A10 Networks, Inc. products are protected by patents in the U.S. and elsewhere. The following website is provided to satisfy the virtual patent marking provisions of various jurisdictions including the virtual patent marking provisions of the America Invents Act. A10 Networks, Inc. products, including all Thunder Series products, are protected by one or more of U.S. patents and patents pending listed at:

a10-virtual-patent-marking.

TRADEMARKS

A10 Networks, Inc. trademarks are listed at: a10-trademarks

CONFIDENTIALITY

This document contains confidential materials proprietary to A10 Networks, Inc.. This document and information and ideas herein may not be disclosed, copied, reproduced or distributed to anyone outside A10 Networks, Inc. without prior written consent of A10 Networks, Inc..

DISCLAIMER

This document does not create any express or implied warranty about A10 Networks, Inc. or about its products or services, including but not limited to fitness for a particular use and non-infringement. A10 Networks, Inc. has made reasonable efforts to verify that the information contained herein is accurate, but A10 Networks, Inc. assumes no responsibility for its use. All information is provided "as-is." The product specifications and features described in this publication are based on the latest information available; however, specifications are subject to change without notice, and certain features may not be available upon initial product release. Contact A10 Networks, Inc. for current information regarding its products or services. A10 Networks, Inc. products and services are subject to A10 Networks, Inc. standard terms and conditions.

ENVIRONMENTAL CONSIDERATIONS

Some electronic components may possibly contain dangerous substances. For information on specific component types, please contact the manufacturer of that component. Always consult local authorities for regulations regarding proper disposal of electronic components in your area.

FURTHER INFORMATION

For additional information about A10 products, terms and conditions of delivery, and pricing, contact your nearest A10 Networks, Inc. location, which can be found by visiting www.a10networks.com.

Table of Contents

Chapter 1: Kubernetes for ACOS Thunder Container

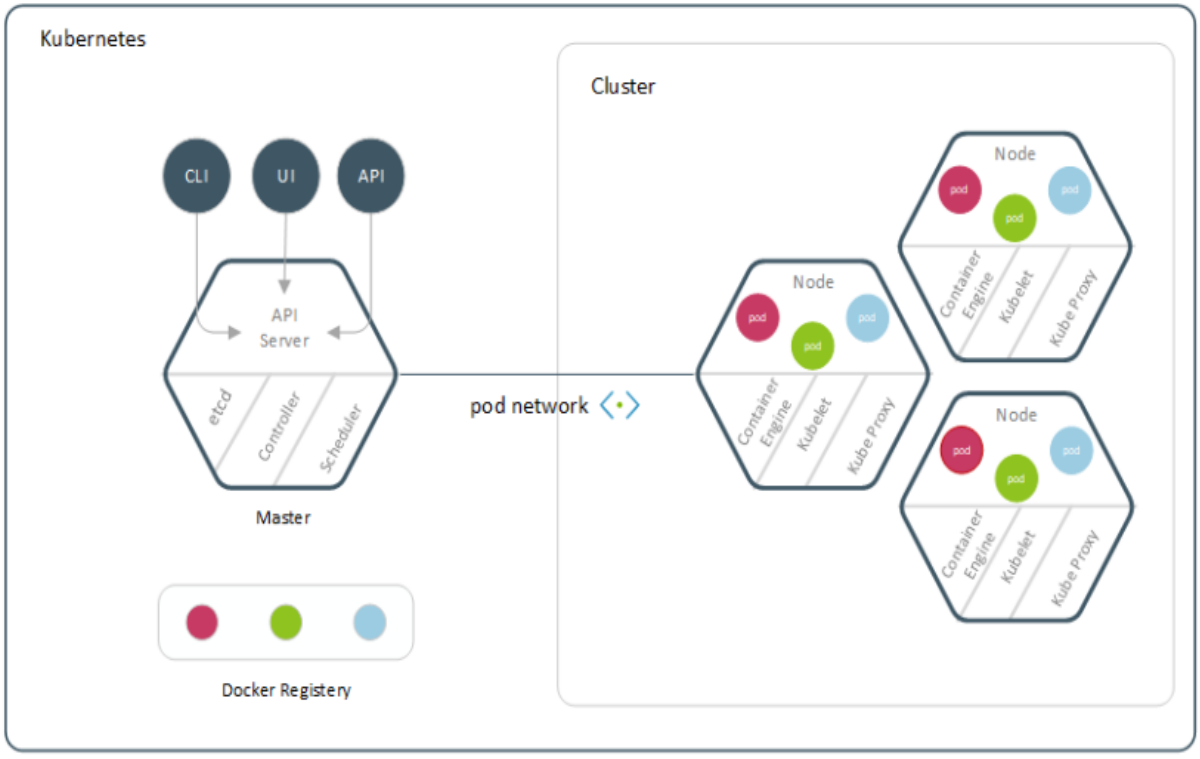
This chapter describes Kubernetes for Thunder Container.

The following topics are covered:

Overview of Kubernetes

Kubernetes is an orchestration tool used for the deployment of containerized applications onto a cluster. In the Kubernetes system, clusters allow containers to be abstracted from its locality. This is done by introducing the concept of a Pod. Pods represent the unit of deployment within the cluster. Each Pod, running on a cluster node, encapsulates one or more containerized applications. This abstraction allows Kubernetes to deploy Pods at will, based on the state of the cluster. For example, if a node fails, all assigned Pods and services can be rescheduled to other nodes within the cluster. This abstraction is further extended at the node level by layering additional software to encapsulate the Pods from the actual hardware. This allows Kubernetes to pool from virtual machines, or bare-metal machines, to participate as a node within the cluster. The termination point of the cluster is the master. Its role is to do the orchestration (deployment, scaling, and management) of applications. The diagram below shows the relationship between the master and nodes.

FIGURE 1-1: Kubernetes Topology



Master Components:

1. **API Server** -- Exposes the Kubernetes API. This API is consumed by a CLI (Kubectl) or UI (web front-end).
2. **Scheduler** - Creates Pods on the cluster node.
3. **Controller** - Manages the state of the deployment with the following:
    a. Node Controller: Responsible for noticing and responding when nodes go down.
    b. Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
    c. Endpoints Controller: Populates the Endpoints object (that is, joins Services & Pods).
    d. Service Account & Token Controllers: Creates default accounts and API access tokens for new namespaces.
    e. Cloud-controller: Interacts with the underlying cloud providers.

Node Components:

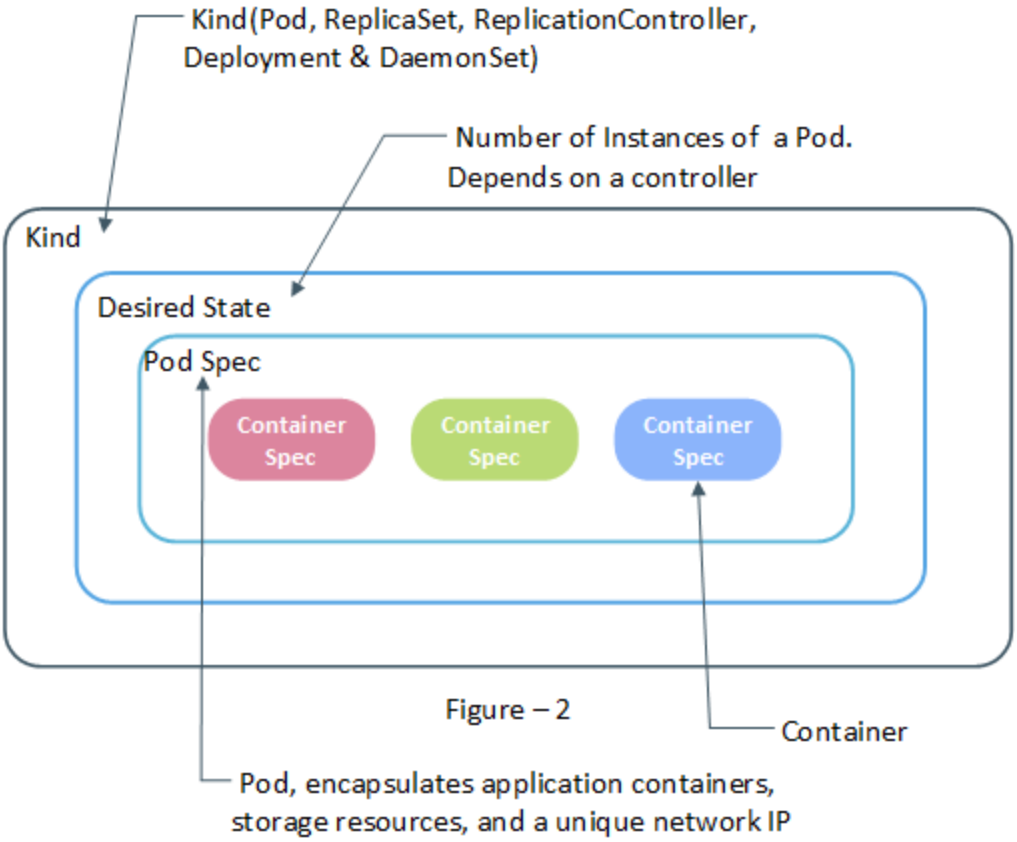
1. Kubelet - An agent that monitors the pods and its containers.
2. Kube-proxy - Handles the network connectivity.
3. Container Engine -- Software that runs the container (Docker, rkt, runc).

Kubernetes Objects

Kubernetes system deals in abstractions so that it can efficiently manage the “state” of the clusters. The state of a cluster describes which persistent entities (objects or services) are currently active in the cluster and on which nodes they are running. This state also includes policies that govern the behavior of these entities--such as restarting, upgrades, and fault-tolerance.

The process of creating an object or service is to define an object-spec (record-on-intent), which tells Kubernetes what the desired state of the incoming entity is within a cluster (clusters-workload). Once an object-spec is handed off to Kubernetes, it continuously works to maintain the actual status within the object's desired state.

FIGURE 1-2: Kubernetes Objects



In contrast, the same Kubernetes object can be created by using the Pod object-spec shown on the right side of the diagram. In this object-spec, we are telling Kubernetes to create the Pod object and run the containerized applications.

Kubernetes Networks

There are three different types of networks used by Kubernetes for managing its process and automation:

1. **Physical Networks:** It is used to connect physical or virtual machines.
2. **Pod Network:** It connects all the Pods within the cluster.
3. **Service Network:** It is a virtual network that is used to expose services internally or externally to the cluster.

Chapter 2: A10 Thunder Container Kubernetes Cluster Installation

This chapter describes how to install Thunder Container on Kubernetes.

The following topics are covered:

Kubernetes Orchestration of Thunder Container

A10 Network's family of Thunder product line is now available to Kubernetes orchestration. Thunder Container is the next generation of ACOS as a container image (executable software package), which has been included in the ACOS 5.0.0 version and onwards.

A10 Networks has optimized ACOS to be adaptive in the Kubernetes ecosystem. Thunder Container is designed to run with limited resources and can also be throttled for utilization of maximum resources provided by a cluster node. This adaptiveness is extended further into more in-depth integration with the Kubernetes CNI networking plug-in. ACOS can operate using virtual Ethernet interfaces provided by any CNI. Additionally, ACOS can also utilize performance-based networking interface controller (NIC) cards that support SRIOV functionality to provide network acceleration for running ACOS application within the node/pod.

Before deploying Thunder Container, the end-user can tailor it on the basis of deployment requirements and cluster node resources. For example, the end-user can specify the memory footprint and the number of network interfaces to be bound to the container. Given the flexibility of our container, the A10 Networks' Thunder Container product line has not been compromised by how a container is created. In the Thunder Container product line, the end-user can expect the same feature sets that are available in the ACOS hardware and the vThunder product line.

Thunder Containers are supported from ACOS 5.0.0 version onwards.

System Requirements

The following are the system requirements for configuring Thunder Container on Kubernetes.

For Software,

- Kubernetes version, with the following feature gates supported:
  - CPU Manager (supported in Kubernetes version 1.10)
  - Huge Pages
    - Alpha (from 1.8 to 1.9)
    - Beta (from 1.10 to 1.13)

    - General Availability (starting from 1.14)
- Kubernetes CNI Plugins:
    - Multus (version 3.3)
    - SRIOV Network Device (latest version)
    - SRIOV CNI (release-version 1)
- Docker Community Edition (version 18.09)

For Node (Host Platform),

- SRIOV Technology Support
    - Intel Virtualization Technology (Intel VT-x) enabled
    - Intel Virtualization Technology for Directed I/O (Intel VT-d) enabled

NOTE: For optimum performance, disable the Intel Hyper-Threading Technology (Intel HT) and improve the performance in L7 and SSL traffic.

- Networking Interfaces
    - Intel 82576 GbE Network Adapter
    - Intel 82599 10 GbE Network Adapter
    - Intel Network Adapter X710 (10 GbE)
    - Intel Network Adapter XL710 (40GbE)
    - Intel Network Adapter XXV710 (25GBE)
- Local Storage: A minimum of 120 GB

NOTE: Additional local storage prevents the eviction of the pod due to disk pressure.

For Thunder Container,

- Dual-stack IPv6 migration support,
- ACOS Products:
    - Server Load Balancer (SLB)
    - Application Delivery Controller (ADC)

    - SSL Insight (SSLi)
    - Convergent Firewall (CFW)
    - Carrier-Grade Network (CGN)
- Minimum system requirements:
    - One CPU
    - Memory: 4 GB
    - One data interface
- Optimum system requirements:
    - Eight CPUs or more
    - Memory: 8 GB or more
    - One management interface (vEth)
    - One or more data interfaces
        - vEth Interface
        - SRIOV Interface - Virtual Function (VF) or Physical Function (PF)
- Huge Pages feature gate supported
- Intel HT Technology disabled

NOTE: The Thunder Container uses the vEth provided by the Pod Network and adds multiple vEth interfaces by using the Multus plugin. The SRIOV interface for Physical Function is required for Thunder Container configuration parameters only.

Installing the Cluster

This guide shows the installation of an on-premise cluster, which consists of a two-node cluster. For more information on creating on-premise clusters, refer to the Kubernetes documentation.

Pre-pulling Thunder Container Image

Pre-install the Thunder Container image on Docker before orchestrating the container with Kubernetes. To download the licensed Thunder Container image, log in to the GLM account and visit the A10 Networks Downloads repository.

To pre-pull the Thunder Container image, which is provided exclusively by A10 Networks, you can locate the latest image file and install it on your Docker environment with the following steps:

1. Log in to your Docker environment on your host machine

   ```
   # docker login
   ```

2. Verify the local availability of licensed Thunder Container images

   ```
   # docker images
   REPOSITORY TAG IMAGE ID CREATED SIZE
   cth-51-91 latest fce289e99eb9 4 months ago 2.263GB
   ```

3. Use the correct version name and load the image on Docker

   For this example, “cth-51-91” is the name of image file (TGZ)

   ```
   # docker load -i cth-51-91.tgz
   028f7796282d: Loading layer [==================================>]
   2.263GB/2.263GB
   7b118ec6bbc0: Loading layer [==================================>]
   319.7MB/319.7MB
   Loaded image: cth-51-91:latest
   ```

Please contact the A10 Networks Support team for additional assistance on pre-pulling the Thunder Container image.

Running Thunder Container with SRIOV Data Interface

ACOS uses the Intel SRIOV Network Device plugin, the SRIOV CNI plugin, and the Kubernetes Multus plugin, and attaches an SRIOV-supported NIC card to the container. Here, the Multus plugin works as an arbiter between Kubernetes and any Software Defined Network (SDN) being used as the underlying network for the newly-created data interfaces. The Multus also

invokes respective SDNs along with Custom Resource Definition (CRD) for instantiating an SRIOV data interface to a Thunder Container.

Preparing Kubernetes Nodes

While preparing the Kubernetes node to the cluster, an adequate number of VFs and an ample amount of memory has to be set up on the node for allowing one or more instances of Thunder Container to run on the node. For this, the Thunder Container utilizes huge pages of 2Mi size or 1Gi size.

Allocating Virtual Functions

Allocate the number of VFs that the deployment needs. In this example, the end-user uses an IXGBE kernel driver, which supports the Intel 82599 gigabyte Ethernet controller that has a maximum allocation limit of 64 VFs.The following configuration shows how an end-user can allocate 30 VFs for each root device.

```
#modprobe ixgbe
#ifconfig enp3s0f0 up
#ifconfig enp3s0f1 up
#echo 30 > /sys/class/net/enp3s0f0/device/sriov_numvfs
#echo 30 > /sys/class/net/enp3s0f1/device/sriov_numvfs
```

NOTE: Specify the names of your respective root devices in the above configuration. Replace `enp3s0f0` and `enp3s0f1` with the names of your root devices in the commands shown above.

Allocating Huge Pages

Depending on the number of Thunder Container instances needed, configure the target node with huge page tables to handle the memory load for each Thunder Container. In this example, you can use 16Gi memory from the 2Mi huge page table.

```
#echo $number_of_pages_needed_in_hugepages >/sys/devices/sys-
tem/node/node0/hugepages/ hugepages-2048kB/nr_hugepages
```

```
#echo $number_of_pages_needed_in_hugepages >/sys/devices/sys-
tem/node/node1/hugepages/ hugepages-2048kB/nr_hugepages

#mkdir /dev/hugepages2M

#mount -t hugetlbfs -o pagesize=2M none /dev/hugepages2M
```

Master Setup Configuration

Each node in the cluster, that runs on the Thunder Container with SRIOV, needs to be configured with the following plugin binaries. You can manually install these binaries by using their respective daemon-set to automate this configuration. Download the plugin source code from their respective Github repositories linked below:

- Multus
- SRIOV Network Device
- SRIOV CNI

These daemon-sets enable the installation of these plugin binaries in Steps 3, 5, and 7 below.

The following steps take you through the creation of Kubernetes cluster for the deployment of Thunder Container with SRIOV:

1. Creating the Cluster

   Run the following `kubeadm` commands to initiate the cluster.

   ```
   $sudo kubeadm reset

   $sudo kubeadm init --pod-network-cidr=10.244.0.0/16
   ```

2. Creating the Pod Network

   Use the Flannel CNI to create the pod network.

   ```
   $kubectl apply -f https://raw.-
   git-
   hubusercontent.com/coreos/flannel/2140ac876ef134e0ed5af15c65e414cf26827915/
   Documentation/kube-flannel.yml
   ```

3. Deploying the Multus Plug-in

   Deploy the Multus daemon set as shown below.

```
$kubectl create -f multus-cni/images/multus-daemonset.yml
$kubectl get pods -o wide --all-namespaces
NAMESPACE NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
kube-system kube-multus-ds-amd64-8d8vh 1/1 Running 0 148m 192.168.29.112 mas-ternode <none> <none>
```

4. Defining the Network Resources

   Use the Kubernetes configMap object to define the network resource available to the Thunder Container. A resource describes the NIC by the vendor ID and device ID that will be used to publish its available VFs to the cluster. In the following configuration, you configure two SRIOV network resources, namely “north” and “south.”

```
apiVersion: v1
kind: ConfigMap
metadata:
name: sriovdp-config
namespace: kube-system
data:
config.json: |
{
"resourceList": [{
"resourceName": "north_traffic",
"selectors": {
"vendors": ["8086"],
"devices": ["154d","10ed"],
"drivers": ["ixgbevf"]
}
},
{
"resourceName": "south_traffic",
"selectors": {
```

```
"vendors": ["8086"],
"devices": ["154d","10ed"],
"drivers": ["ixgbevf"]
}
}
]
}
```

To define the node role, deploy the configMap file as follows:

```
$kubectl create -f acos/acos-node-interface-resource-list-configmap.yaml
```

5. Deploying the SRIOV Network Device Plugin

   Run the following `kubectl` commands to deploy the SRIOV Network Device plugin:

```
$kubectl create -f sriov-network-device-plugin/images/sriovdp-daemonset.yaml
$kubectl get pods -o wide --all-namespaces
NAMESPACE NAME READY STATUS
RESTARTS AGE IP NODE NOMINATED  NODE READINESS  GATES
default sriov-device-plugin 1/1 Running 0
143m 192.168.29.112 masternode <none> <none>
```

6. Checking the Node Resource Status

   Check the status of allocated VFs and huge pages for the target node.

```
kubectl get node masternode -o json | jq '.status.allocatable'
{
"cpu": "40",
"ephemeral-storage": "332122473127",
"hugepages-1Gi": "0",
"hugepages-2Mi": "16Gi",
"intel.com/north_traffic": "20",
"intel.com/south_traffic": "20",
"memory": "114730480Ki",
```

```
"pods": "110"
}
```

7. Deploying the SRIOV Custom Resource Definition

   Deploy the SRIOV CRD to save the northbound traffic configuration to the following address: acos/north-bound-traffic.yaml.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
name: north-network
annotations:
k8s.v1.cni.cncf.io/resourceName: intel.com/north_traffic
spec:
config: '{
"type": "sriov",
"name": "north-bound-network",
"master": "enp3s0f0",
}'
```

   For southbound traffic, save the configuration to the following address: acos/south-bound-traffic.yaml

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
name: south-network
annotations:
k8s.v1.cni.cncf.io/resourceName: intel.com/south_traffic
spec:
config: '{
"type": "sriov",
"name": "south-bound-network",
```

```
"master": "enp3s0f1",
}'
```

NOTE: The above configuration uses DPDK for binding. For more information, please refer to SRIOV Network Device source code.

Then, deploy both, south-bound and north-bound, network resource definition objects as follows:

```
kubectl create -f acos/north-bound-traffic.yaml
kubectl create -f acos/south-bound-traffic.yaml
```

8. Deploying the Thunder Container

   Before deploying the Thunder Container, the user needs to add a network annotation and update the Thunder Container YAML file to use the SRIOV network.

```
metadata:
labels:
run: Thunder-Container
annotations:
k8s.v1.cni.cncf.io/networks: north-network,south-network
```

   Define the number of network interfaces to be attached to Thunder Container.

```
resources:
requests:
memory: 4Gi
cpu: "4"
hugepages-2Mi: 4Gi
intel.com/north_traffic: '1'
intel.com/south_traffic: '1'
limits:
memory: 4Gi
cpu: "4"
hugepages-2Mi: 4Gi
```

```
intel.com/north_traffic: '1'
intel.com/south_traffic: '1'
```

NOTE: In the above example, two network interfaces are used for Thunder Container deployment. In addition to this, Kubernetes has been configured to limit the memory usage to 4Gi and the CPU usage to 4 CPUs.

Define the huge page reservation for Thunder Container as follows:

```
spec:
containers:
- env:
- name: ACOS_CTH_DEVICES_OPTS
value: 1G@0=2,1G@1=2,mount
```

NOTE: In the above example, you are reserving two 1G huge pages on node 0 and two 1G huge pages on node 1.

TABLE 2-1 : Recommended Huge Pages Reservation for Thunder Container

Memory	***CPU***	***Huge pages size***	***Reserving 1G huge pages from node 0 and node 1***	***Reserving 2M huge pages from node 0 and node 1***
4G	4	1G	1G@0=1 or 1G@1=1	2M@0=256,2M@1=256
8G	8	2G	1G@0=1,1G@1=1	2M@0=512,2M@1=512
16G	16	4G	1G@0=2,1G@1=2	2M@0=1024,2M@1=1024
32G	48	8G	1G@0=4,1G@1=4	2M@0=2048,2M@1=2048
64G	48	16G	1G@0=8,1G@1=8	2M@0=4096,2M@1=4096
128G	48	32G	1G@0=16,1G@1=16	2M@0=8129,2M@1=8129

NOTE: For optimum performance, A10 recommends to use 1G huge pages.

Deploy the Thunder Container as follows:.

```
$kubectl create -f Thunder-Container-one-port-vfio.yaml
```

Configuring Thunder Container for One-Arm Server Load Balancing

You can also configure Thunder Container for a one-arm deployment in the SLB mode. With this configuration, the Multus plugin can provide multi-homed interface support to the Thunder Container by enabling the transparent mode on the SLB. In one-arm deployment topology, the Kubernetes cluster consists of two nodes and a master. You also need to use a Flannel plugin with a classless inter-domain routing (CIDR) block of the assigned IP address for configuring the pod network.

The following steps take you through the configuration of Thunder Container for one-arm SLB deployment.

1. Starting the HTTP Server

   Use the `kubectl` command to deploy the HTTP web server (NGINX) as follows:

   ```
   $kubectl create -f nginx-web-farm.yaml
   deployment.apps/nginx-web-farm created
   ```

2. Checking the Deployment State

   Use the `kubectl get deployment` command to send a query for the state of web server deployment.

   ```
   $kubectl get deployments
   NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE
   web-farm-deployment 1 1 1 0 3s
   ```

3. Checking the Assigned IP

   Send a query for the assigned IP addresses with the `kubectl get pods` command

   ```
   $kubectl get pods -o wide
   NAME READY STATUS RESTARTS AGE IP NODE
   NOMINATED NODE
   web-farm-deployment-9f9f854c6-k9cd2 1/1 Running 0 20s 10.244.1.139 node1-
   vmware-1 <none>
   ```

4. Initiating the SLB Server Configuration

   Use the web server IP address obtained from the above step to initiate the SLB server configuration for the web-server, the service group, and the virtual server.

```
!
ip route 0.0.0.0 /0 10.244.1.1
!
slb server wsrv-1 10.244.1.139
port 80 tcp
health-check-disable
!
slb service-group nginx-sg tcp
member wsrv-1 80
!
slb virtual-server nginx-farm use-if-ip ethernet 1
port 80 tcp
source-nat auto
service-group nginx-sg
!
```

5. Creating a Service

   Create a service that makes NGINX web server accessible from outside the cluster

```
apiVersion: v1
kind: Service
metadata:
name: acos-slb-deployment
labels:
app: acos-slb-one-arm
spec:
ports:
- port: 8080
targetPort: 80
protocol: TCP
selector:
```

```
app: acos-slb-one-arm
externalIPs:
- 192.168.207.166
```

NOTE: The above example is of a local machine, where you specify the service external IP address and re-assign it as the master node's 'host' IP address for routing the external traffic into Thunder Container. The above YAML file shows how an external port 8080 is mapped to the Thunder Container's internal port 80, allowing Kubernetes to route all the port 8080-bound traffic into Thunder Container. For a hosted Kubernetes cluster, this external IP address is automatically assigned to the cluster.

Save the above configuration in an 'acos_service' YAML file and use the `kubectl create` command to create the service.

```
$kubectl create -f acos_service.yaml
service/acos-svc created
```

View the service details with `kubectl get services` command.

```
$kubectl get services -o wide
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE SELECTOR
acos-slb-deployment ClusterIP 10.109.55.253 192.168.207.166 8080/TCP 13s app-p=acos-slb-one-arm
kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 111m <none>
```

6. Deploying the Thunder Container and Saving the Object-Spec

   Use the saved YAML file to start the Thunder Container deployment.

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: acos-slb-deployment
spec:
selector:
```

```
matchLabels:
app: acos-slb-one-arm
replicas: 1
template:
metadata:
labels:
app: acos-slb-one-arm
tier: frontend
spec:
containers:
- name: Thunder-Container
image: acos_non_fta_5_0_0-p1_69
imagePullPolicy: Never
env:
- name: ACOS_CTH_SUPPORT_MGMT
value: "n"
- name: ACOS_CTH_SUPPORT_VETH
value: "y"
resources:
requests:
memory: 4Gi
cpu: "4"
limits:
memory: 4Gi
cpu: "4"
ports:
- containerPort: 80
securityContext:
capabilities:
```

```
add: ["SYS_ADMIN", "NET_ADMIN","IPC_LOCK"]
```

Save the above object-spec in an 'acos_slb_one_arm' YAML file and create the one-arm SLB deployment of Thunder Container with the `kubectl create` command

```
$kubectl create -f acos_slb_one_arm.yaml
deployment.apps/acos-slb-deployment created
```

7. Checking the State of the Deployment and the Pod

   Use the `kubectl get deployments` command to view the deployment status.

```
$kubectl get deployments
NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE
acos-slb-deployment 1 1 1 0 10s
web-farm-deployment 1 1 1 1 35m
```

   Use the `kubectl get pods` command to view the status of the pod.

```
$kubectl get pods -o wide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE
acos-slb-deployment-79cfc74b46-8gmfd 1/1 Running 4 20s 10.244.1.145 node1
<none>
web-farm-deployment-9f9f854c6-k9cd2 1/1 Running 0 32m 10.244.1.139 node1
<none>
```

8. Updating the SLB One-Arm Deployment Configuration

   After the deployment, you can update the running configuration of Thunder Container with the **a10login** from the host terminal shell. To access the utility, run the following `kubectl exec` command:

```
$kubectl exec -ti acos-slb-deployment-79cfc74b46-8gmfd – a10login
cThunder login: admin
Password:
Last login: Thu May 6 05:10:52 IST 2021 on pts/0
Last login: Thu May 6 05:11:53 on pts/0
System is ready now.
[type ? for help]
```

```
cThunder(NOLICENSE)>
```

Visibility and Analytics Monitoring

Thunder Container users with critical infrastructure can monitor network resources through visibility and analytics. Thunder Container supports a logging system to monitor resources like system interface statistics, virtual server, remote server, and virtual port.

All ACOS 5.2.0 platforms, ACOS Thunder, vThunder, and Thunder Container have native support for Prometheus. A Prometheus server can query various analytics as specified in its configuration.

Functionalities

Users and systems can now use the following functionalities:

- Create and view dashboards to communicate with the Prometheus server using a Visualization and Analytics tool, like Grafana.
- Query any API statistics configured in the Prometheus server's YAML file.
- View the default metrics logged, when no filters are specified:
  - All Interface Metrics
  - CPU Usage
  - Memory Usage

Configuration Example

For example, to monitor a particular object or class of objects, add that object or class of objects to the parameters (params) in the Prometheus YAML file as follows.

Sample prometheus.yml Configuration Snippet

```
global:
scrape_interval: 15s
evaluation_interval: 15s
```

```
- job_name: 'acos-scraper-job'
metrics_path: '/metrics'
params:
host_ip: ["10.43.12.122:9443"]
api_endpoint: ["/slb/dns", "/slb/virtual-server/10.10.10.2/port/80+tcp",
"/slb/fix"]
api_name: ["_slb_dns", "_slb_virtual_server_10.10.10.2_port_80_tcp", "_slb_fix"]
partition: ["P1"]
```

The descriptions for the parameters are as follows:

Parameter	***Description***
scrape_interval	Time intervals for querying the statistics fields.
target	Hostname and port that the Exporter is running on and port must be the same as the port number of the webserver on ACOS Prometheus client.
api_endpoint	URI endpoint that the Exporter intercepts to invoke the appropriate aXAPI. (A comma-separated list of APIs can be provided here for a single host..)
api_name	API name, or any name used to identify the API endpoint, which is the unique identifier for a job. (Comma-separated list of API names that must be in sync with the API endpoint list.)
partition	Name of the partition. (This is an optional parameter. If not specified, the system will use the shared partition.)

In this scenario, once the Prometheus server is up and running, it invokes the query every 15 seconds, as specified in the “scraping interval.api_endpoint”. The API names are passed to them as parameters. The ACOS Prometheus client creates the gauge metrics for each statistics field; and exposes the metrics to the Prometheus server.

NOTE: To enable Prometheus support for previous versions of ACOS, refer to the Prometheus Exporter procedure, click here.

Additional Resources

For more information on Kubernetes and plugins, refer to the Multus, SRIOV Network Device, and SRIOV CNI. For more information, visit A10 Networks Support and browse the ACOS products to select Thunder Container documents.