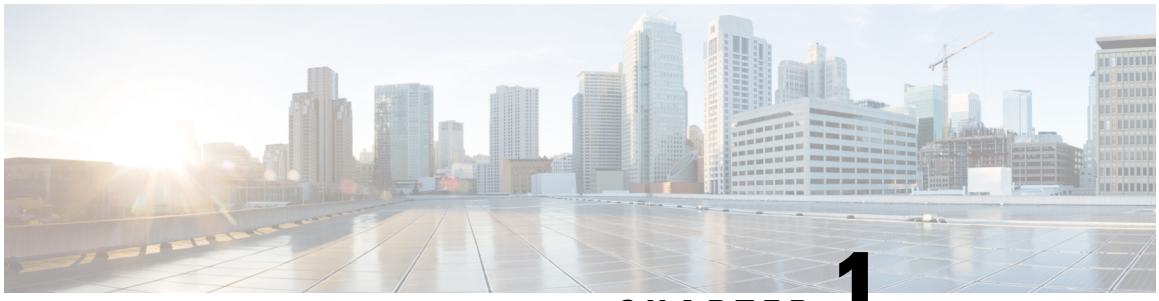


Cisco Secure Firewall Threat Defense Container Getting Started Guide

First Published: 2026-04-22

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883



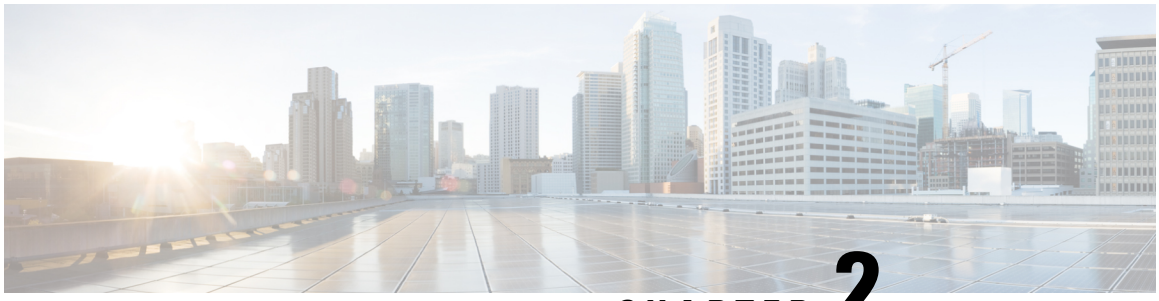
CHAPTER 1

Overview

• Overview of the Firewall Threat Defense Container, on page 1

Overview of the Firewall Threat Defense Container

From Secure Firewall version 10.0, you can deploy the Firewall Threat Defense container (FTDc) in open source Docker, Kubernetes, and Amazon Elastic Kubernetes Service (EKS) environments. A container is a software package that bundles up code and associated requirements such as system libraries, system tools, default settings, and runtime, to ensure that the application runs successfully in a computing environment. In this solution, the FTDc is integrated with the Container Network Interface (CNI) and is deployed as an Infrastructure-as-Code (IaC) solution. The integration with CNI provides improved flexibility in deployment of network infrastructure.



CHAPTER 2

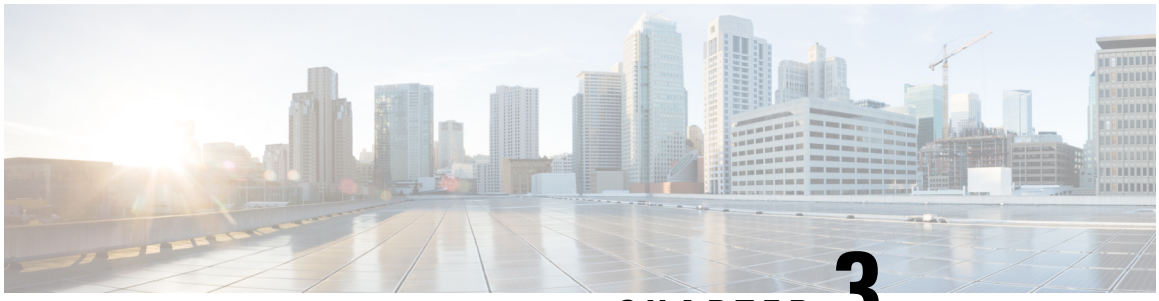
Licensing

- Licenses to Deploy Firewall Threat Defense Container in Docker and Kubernetes Environments, on page 3

Licenses to Deploy Firewall Threat Defense Container in Docker and Kubernetes Environments

Use one of the following licenses to enable deployment of Firewall Threat defense container:

- FTDv5 - 4 vCPU, 8 GB RAM, and 100 Mbps rate limit
- FTDv10 - 4 vCPU, 8 GB RAM, and 1 Gbps rate limit
- FTDv20 - 4 vCPU, 8 GB RAM, and 3 Gbps rate limit
- FTDv30 - 8 vCPU, 16 GB RAM, and 5 Gbps rate limit
- FTDv50 - 12 vCPU, 24 GB RAM, and 10 Gbps rate limit
- FTDv100 - 16 vCPU, 32 GB RAM, and 16 Gbps rate limit



CHAPTER 3

Deploy Firewall Threat Defense Container in Docker Environment

- Prerequisites to Deploy Firewall Threat Defense Container in Docker Environment, on page 5
- Sample Topology to Deploy Firewall Threat Defense Container in Docker Environment, on page 6
- Deploy Firewall Threat Defense Container in Docker Environment, on page 6
- Verify and Access Firewall Threat Defense Container in Docker Environment, on page 10

Prerequisites to Deploy Firewall Threat Defense Container in Docker Environment

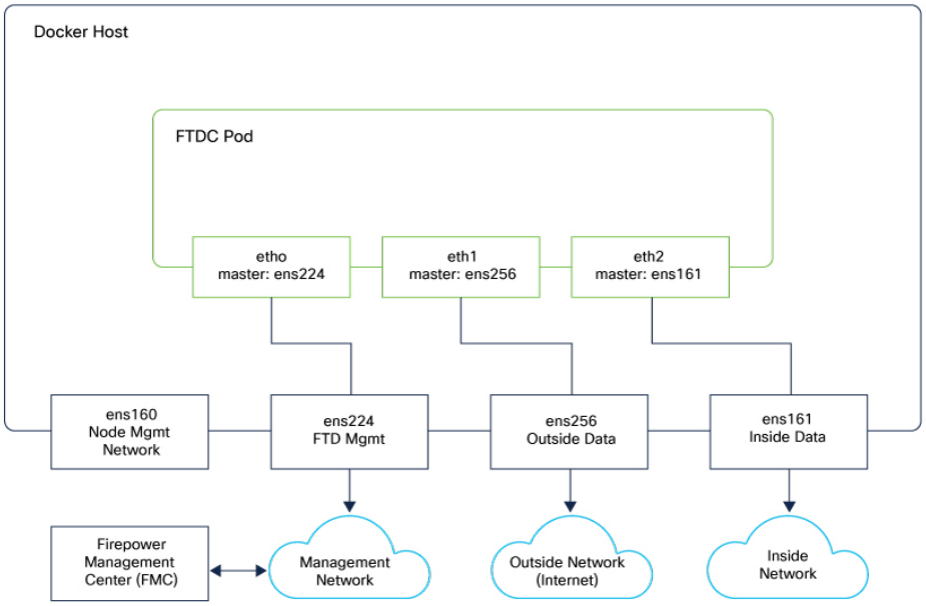
Following are the prerequisites to deploy Firewall Threat Defense container in a Docker environment:

- **Operating System**: Ubuntu, Version - 20.04 LTS (minimum), 22.04 LTS (maximum)
- **Docker Version**: 26.1.3 or later
- **Storage Requirements**: Use high-performance storage (SSD/NVMe) for the Docker host.

  Note: HDD-based storage is not supported, as slower disks may cause lina process failures during startup, particularly when using SR-IOV.
- **Network Configuration**: Configure a Docker network with macvlan driver for container deployment.
- **Network Interfaces**: Allocate at least three virtual interfaces on the Docker host for FTD container operations.
- **Host Management Access**: Configure a management interface on the Docker host to allow SSH access.
- **Memory Configuration**: Enable Hugepages on the Docker host. Allocate minimum of 2048 hugepages on the host.

For more information on general Docker operations mentioned in these prerequisites, refer to Docker documentation.

Sample Topology to Deploy Firewall Threat Defense Container in Docker Environment



The diagram shows an FTD container deployed on an Ubuntu-based Docker host, connected to multiple network segments.

- The management interface is used to communicate with the Management Center.
- The inside interface connects to the inside endpoint network.
- The outside interface connects to the outside endpoint network.

Deploy Firewall Threat Defense Container in Docker Environment

Perform this procedure to deploy Firewall Threat Defense container (FTDc) in Docker environment.

Procedure

Step 1 Set up the requirements mentioned in the Prerequisites.

Step 2 Use the **ifconfig** command to verify the network interface configuration. In this example, ens160 is the node's management interface. The interfaces ens192, ens224, and ens256, are mapped to the FTDc interfaces.

Note

The outputs shown are sample outputs.

Make sure that one interface is used for management and a minimum of two interfaces are available for data traffic.

```
$ ifconfig
ens160: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet6 fe80::250:56ff:fe9d:6125 prefixlen 64 scopeid 0x20<link>
        ether 00:50:56:9d:61:25 txqueuelen 1000 (Ethernet)
        RX packets 317297807 bytes 447854277676 (447.8 GB)
        RX errors 0 dropped 2100 overruns 0 frame 0
        TX packets 5517880 bytes 378756756 (378.7 MB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens192: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.10.3.4 netmask 255.255.255.224 broadcast 10.10.3.31
        inet6 fe80::250:56ff:fe9d:fa1c prefixlen 64 scopeid 0x20<link> ether 00:50:56:9d:fa:1c
txqueuelen 1000 (Ethernet)
        RX packets 70324790 bytes 30189381762 (30.1 GB)
        RX errors 0 dropped 2437 overruns 0 frame 0
        TX packets 60676399 bytes 16108954006 (16.1 GB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens224: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet6 fe80::250:56ff:fe9d:2cbe prefixlen 64 scopeid 0x20<link>
        ether 00:50:56:9d:2c:be txqueuelen 1000 (Ethernet)
        RX packets 489699 bytes 41669463 (41.6 MB)
        RX errors 0 dropped 1969 overruns 0 frame 0
        TX packets 285031 bytes 23421780 (23.4 MB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens256: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet6 fe80::250:56ff:fe9d:92ba prefixlen 64 scopeid 0x20<link>
        ether 00:50:56:9d:92:ba txqueuelen 1000 (Ethernet)
        RX packets 7023252 bytes 8223100366 (8.2 GB)
        RX errors 0 dropped 2145 overruns 0 frame 0
        TX packets 31481074 bytes 44913129384 (44.9 GB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Step 3 Use the **cat** command to verify that hugepages are configured correctly.

```
$ cat /proc/meminfo | grep -E 'HugePages_Total|HugePages_Free'
```

Expected output (example):

```
HugePages_Total: 2048
HugePages_Free: 2048
```

Ensure HugePages_Total is equal to the value configured (for example: 2048).

Ensure HugePages_Free is less than or equal to HugePages_Total, as per your configuration.

Step 4 Download the Threat Defense docker tar bundle that includes the Threat Defense container image from software.cisco.com.

Step 5 Load the docker tar bundle on the host. Verify that the FTDc image is successfully loaded.

```
$ docker load -i ftdc-<version>.tar
$ docker images
```

Step 6 Download the templates and other files from the Docker folder in the FTDc GitHub repository.

Step 7 Copy the **deploy-FTDc** folder to the Docker host.

Edit interface mapping: `ftdc_shared/interface-config`

- Edit `ftdc_shared/interface-config` for SR-IOV mode.

Set `iface_id` to the PCI ID of the desired VF and `uio_driver` to `vfio-pci`:

```
[interface0]
iface_id = 0b:0a.0;
uio_driver = vfio-pci;
[interface1]
iface_id = 0b:0a.1;
uio_driver = vfio-pci;
```

- Edit `ftdc_shared/interface-config` for macvlan mode.

```
[interface0]
iface_id = eth1;
uio_driver = afpacket;
[interface1]
iface_id = eth2;
uio_driver = afpacket
```

Step 8 Configure Day-0 parameters.

`ftdc_shared/day0.json`

The `day0.json` file provides initial bootstrap configuration for FTDc.

Default Configuration (day0.json):

```
{
"EULA": "accept",
"Hostname": "cisco-ftdc",
"AdminPassword": "<PASSWORD>",
"FirewallMode": "routed",
"DNS1": "208.67.222.222",
"Diagnostic": "OFF",
"IPv4Mode": "DHCP",
"IPv4Addr": "",
"IPv4Mask": "",
"IPv4Gw": "",
"FmcIp": "DONTRESOLVE",
"FmcRegKey": "docker",
"FmcNatId": "docker"
}
```

Step 9 Deploy FTDc.

FTDc supports two deployment modes:

- Deploy using host network.

  Ensure eth0 (management) and at least two additional interfaces for data are available in the host.

  Use the following command:

  ```
  ./deploy_ftdc_host_network.sh --version <version> [--cpus <num>] [--memory <mb>]
  ```

  Or

  ```
  ./deploy_ftdc_host_network.sh -v <version> [-C <num>] [-M <mb>]
  ```

  Examples:

  Use defaults (4 CPUs, 8192 MB)

  ```
  ./deploy_ftdc_host_network.sh --version latest
  ```

  ```
  ./deploy_ftdc_host_network.sh -v latest
  ```

Custom resources (8 CPUs, 16 GB)

```
./deploy_ftdc_host_network.sh --version latest --cpus 8 --memory 16384
```

```
./deploy_ftdc_host_network.sh -v latest -C 8 -M 16384
```

- Deploy using Docker network
  - Create Docker networks

    Management network (mandatory)

    ```
    docker network create -d macvlan \
    --subnet=10.10.4.0/24 \
    -o parent=eth0 \
    mgmt0
    ```

    Data networks (minimum 2 interfaces required)

    ```
    docker network create -d macvlan \
      --subnet=10.10.10.0/24 \
      --ipv4=false \
      -o parent=ens192 \
      data0
    docker network create -d macvlan \
      --subnet=10.10.20.0/24 \
      --ipv4=false \
      -o parent=ens224 \
      data1
    ```

    Note
    - Minimum: 2 data networks
    - Maximum: 9 data networks
    - Ensure MTU 9000 is configured on all parent interfaces before creating Docker networks.
    - Data interfaces do not receive IP addresses from Docker. IP addresses are assigned through Management Center after deployment.

  - Deploy container

    ```
    ./deploy_ftdc_docker_network.sh \
    --version <version> \
    --mgmt-net <network> \
    --mgmt-ip <ip> \
    --data-net <network> \
    --data-net <network> \
    [--cpus <num>] \
    [--memory <mb>]
    ```

    Example:

    Minimum networks (2 data), default resources (4 CPUs, 8192 MB)

    ```
    ./deploy_ftdc_docker_network.sh --version latest --mgmt-net mgmt0 --mgmt-ip 192.168.1.100
    --data-net data0 --data-net data1
    ```

    ```
    ./deploy_ftdc_docker_network.sh -v latest -m mgmt0 -i 192.168.1.100 -d data0 -d data1
    ```

    Maximum networks (9 data), custom resources (8 CPUs, 16 GB)

    ```
    ./deploy_ftdc_docker_network.sh --version latest --mgmt-net mgmt0 --mgmt-ip 192.168.1.100
    --data-net data0 --data-net data1 [--data-net ... up to 7 more] --cpus 8 --memory 16384
    ```

```
./deploy_ftdc_docker_network.sh -v latest -m mgmt0 -i 192.168.1.100 -d data0 -d data1 [-d
... up to 7 more] -C 8 -M 16384
```

Note
Default resources: 4 CPUs, 8192 MB RAM

Minimum number of interfaces: 1 management interface, minimum of 2 data interfaces

Maximum number of interfaces: 1 management and 9 data interfaces

- Deploy using SR-IOV network

  Edit `ftdc_shared/interface-config` for SR-IOV mode.

  Set `iface_id` to the PCI ID of the desired VF and `uio_driver` to `vfio-pci`:

```
[interface0]
iface_id = 0b:0a.0;
uio_driver = vfio-pci;
[interface1]
iface_id = 0b:0a.1;
uio_driver = vfio-pci;
```

  Use the following command to deploy FTDc with SR-IOV interfaces.

```
./deploy_ftdc_docker_network.sh \
--version <version> \
--mgmt-net <network> \
--mgmt-ip <ip> \
[--cpus <num>] \
[--memory <mb>] \
-sriov
```

  Example:

```
./deploy_ftdc_docker_network.sh --version 10.0.0-14 -C 8 -M 16384 --mgmt-net ftdc-mgmt
--mgmt-ip 192.168.123.100 -sriov
```

Verify and Access Firewall Threat Defense Container in Docker Environment

Verify deployment status:

Verify and access the FTD container.

Verify that the FTD container is successfully deployed and running by using the following command:

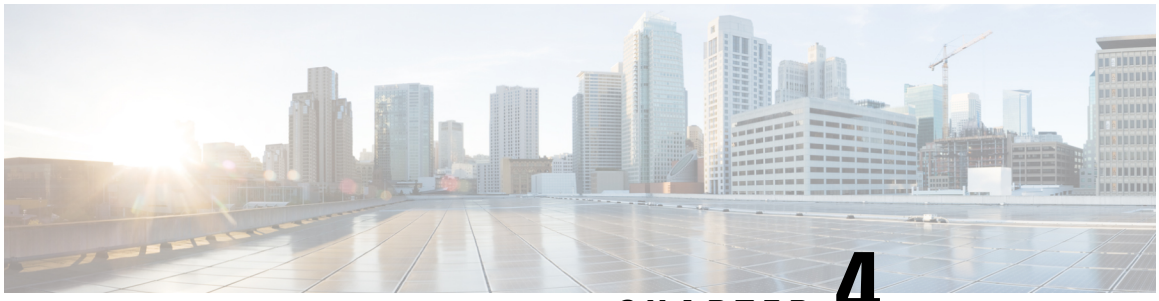
```
$ docker ps -a
```

Ensure that the FTD container is in the running state.

Access the FTD container using the following command:

```
docker attach ftdc
```

This will provide you with access to the FTDc CLI (clish).



CHAPTER 4

Deploy Firewall Threat Defense Container in Kubernetes Environment

- Prerequisites to Deploy Firewall Threat Defense Container in Kubernetes Environment, on page 11
- Sample Topology to Deploy Firewall Threat Defense Container in Kubernetes Environment, on page 12
- Deploy Firewall Threat Defense Container in Kubernetes Environment, on page 13
- Verify and Access Firewall Threat Defense Container in Kubernetes Environment, on page 20

Prerequisites to Deploy Firewall Threat Defense Container in Kubernetes Environment

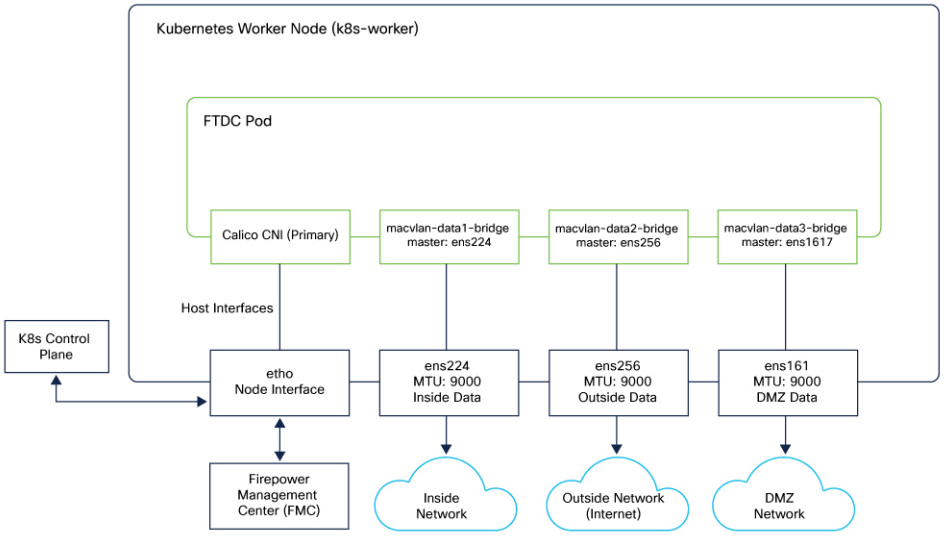
Following are the prerequisites to deploy Firewall Threat Defense container in a Kubernetes environment:

- **Operating System**: Ubuntu, Version - 20.04 LTS (minimum), 22.04 LTS (maximum)
- **Kubernetes Environment**:
  - Kubernetes cluster version - 1.29.15 (minimum), 1.31.14 (maximum)
  - Cluster must include master and worker nodes.
  - Multus CNI installed for multi-network support
  - MetalLB installed for LoadBalancer services
- **Kubernetes CNI**
  - POD management CNI - Calico
  - FTDc data network CNI - Multus macvlan
  - FTDc data network CNI - Multus SR-IOV
- **Worker Node Requirements:**
  - Hugepages configured (2Mi pages)
  - Use SSD/NVMe storage on worker nodes. HDD-based storage is not supported.

  - For Macvlan: Host interfaces available for data traffic
  - For SR-IOV: SR-IOV Network Operator installed and VFs configured
- **Helm**: Helm version minimum 3.8 or later for deploying FTD using Helm charts (YAML-based Infrastructure-as-Code)
- Firewall Threat Defense Container (FTDc) image available in an accessible registry
- Management Center accessible from the cluster (if using ftd-manager)

For more information on general Kubernetes operations mentioned in these prerequisites, see Kubernetes documentation.

Sample Topology to Deploy Firewall Threat Defense Container in Kubernetes Environment



The diagram shows an FTD pod deployed on a Kubernetes worker node, connected to multiple network segments using Calico and macvlan CNI.

- The primary interface (eth0) is managed by the Calico CNI and is used for pod communication with the K8s control plane and the Management Center.
- The inside interface connects to the inside endpoint network.
- The outside interface connects to the outside endpoint network.

Deploy Firewall Threat Defense Container in Kubernetes Environment

Follow the procedure to deploy Firewall Threat Defense container (FTDc) in Kubernetes environment.

Procedure

Step 1 Set up the requirements mentioned in the Prerequisites.

Step 2 Clone the FTDc repository to the local folder.

```
git clone <repository-URL>
```

Step 3 Navigate to the Helm chart directory.

```
cd Helm_charts/ftdc-helm
```

Step 4 Deploy FTDc.

You can deploy FTDc using either the default configuration or a custom values file.

- Using Helm with default configuration:

  ```
  helm install ftdc . -n <namespace>
  ```

- Using Helm with a custom values file:

  ```
  helm install ftdc . -f my-values.yaml -n <namespace>
  ```

Note
Namespace should be created before using it in the helm install command.

Step 5 Deploy FTDc using Helm with additional parameters.

```
helm install ftdc . \
  --set ftdc.repository=myregistry.example.com/ftdc:7.4 \
  --set ftdc.app_name=ftdc \
  --set ftdc.cpus=8 \
  --set ftdc.memory=16384 \
  --set ftdc.shmSize=4Gi \
  --set ftdc.hugepages=2Gi \
  --set ftdc_manager.enabled=true \
  --set ftdc_manager.repository=myregistry.example.com/ftd-manager:v1 \
  --set fmcv.ip=10.0.0.50 \
  --set fmcv.user=admin \
  --set fmcv.password=<your-password> \
  --set fmcv.access_policy=Production-Policy \
  --set fmcv.inside_zone=inside \
  --set fmcv.outside_zone=outside \
  --set fmcv.license_caps=BASE \
  --set fmcv.performance_tier=FTDv30 \
  --set worker_nodes.cni.type=macvlan \
  --set worker_nodes.persistence.lina.enabled=true \
  --set worker_nodes.persistence.lina.storageClass=lina-storage \
  --set worker_nodes.persistence.lina.size=1Gi \
  --set worker_nodes.persistence.lina.hostPath=/data/ftdc/lina \
  --set worker_nodes.persistence.lina.accessMode=ReadWriteMany \
```

```
--set worker_node1.name=k8s-worker1 \
--set worker_node1.ip=10.10.3.4 \
-n default
```

Step 6 Update the configuration parameters in the Helm chart.

See the configuration parameters in the following tables:

Table 1: FTDc container settings

Parameter	Description	Default Value	Required
ftdc.repository	FTDc container image repository	localhost:5000/ftdc_10.0.0.143	Yes
ftdc.app_name	Application name for deployment	ftdc	Yes
ftdc.cpus	Number of CPUs allocated to FTDc	4	Yes
ftdc.memory	Memory in MB allocated to FTDc	8192	Yes
ftdc.shmSize	Shared memory (/dev/shm) size	2Gi	No
ftdc.hugepages	Hugepages allocation	auto	No
replicas	Number of pod replicas	1	No
namespace	FTD container name space for deployment	default	No

Supported CPU and memory configurations:

The following CPU and memory configurations are supported for FTDc deployment in Kubernetes environments. These configurations align with supported performance tiers.

- FTDv5 – 4 vCPU, 8 GB RAM
- FTDv10 – 4 vCPU, 8 GB RAM
- FTDv20 – 4 vCPU, 8 GB RAM
- FTDv30 – 8 vCPU, 16 GB RAM
- FTDv50 – 12 vCPU, 24 GB RAM
- FTDv100 – 16 vCPU, 32 GB RAM

Hugepages configuration:

The ftdc.hugepages parameter supports two modes:

a. Auto-calculation ("auto"): Calculates hugepages based on CPUs and interfaces:

Formula: (CPUs*128) + (numInterfaces*64) Mi, rounded upto nearest 256 Mi.

Example: 4 CPUs + 3 interfaces = (4×128) + (3×64) = 704Mi → rounded to 768Mi

b. Fixed value: Specify exact size like "512Mi", "1Gi", "2Gi"

FTD manager settings:

The FTD manager is an optional component that automates FTDc registration with Management Center.

Table 2: FTD manager settings

Parameter	Description	Default Value	Required
ftdc_manager.enabled	Enable/disable FTD Managerdeployment	false	No
ftdc_manager.repository	FTD Manager container image	localhost:5000/ftd-manager:v1	If enabled

When enabled, FTD manager creates the following resources:

- Secret for Management Center credentials
- ConfigMap for day0 configuration
- Deployment with RBAC (ServiceAccount, ClusterRole, ClusterRoleBinding)

Management Center configuration:

Prerequisites in Management Center for the ftd-manager pod:

- The required user account, access policy, and security zones must already be configured in Management Center.
- Management Center must have a valid license or it should be operating within the evaluation license period.

Table 3: Configuration for Management Center integration

Parameter	Description	Default Value	Required
fmcv.ip	Management Center IP address	10.10.3.9	Yes
fmcv.user	Management Center admin username	admin	Yes
fmcv.password	Management Center admin password	"" (empty)	Yes
fmcv.access_policy	Management Center access policy name	allowall	Yes
fmcv.inside_zone	Inside security zone name	inside	Yes
fmcv.outside_zone	Outside security zone name	outside	Yes
fmcv.license_caps	License capabilities	BASE	Yes
fmcv.performance_tier	Performance tier	FTDv5	Yes

License capabilities:

Deploy Firewall Threat Defense Container in Kubernetes Environment

Valid values for fmcv.license_caps are BASE, THREAT, URL, MALWARE, or Multiple.

- BASE: Base license
- THREAT: Threat protection
- URL: URL filtering
- MALWARE: Malware protection
- Multiple: BASE, THREAT, URL, MALWARE, CARRIER

Performance tiers:

Valid values for fmcv.performance_tier:

- FTDv5: 100 Mbps
- FTDv10: 1 Gbps
- FTDv20: 3 Gbps
- FTDv30: 5 Gbps
- FTDv50: 10 Gbps
- FTDv100: 16 Gbps

Passwords:

Use one of these methods to create passwords:

- Command line override

```
helm install ftdc . --set fmcv.password=<your-password>
```

- Separate secrets file (add to .gitignore)

```
helm install ftdc . -f secrets.yaml
```

Table 4: Worker Node Configuration

Parameter	Description	Default Value	Required
worker_node1.name	Kubernetes hostname of primary worker	k8s-worker1	Yes
worker_node1.ip	IP address for MetalLB LoadBalancer	10.10.3.4	Yes

CNI Configuration:

The Helm chart supports two CNI types for data plane interfaces: Macvlan and SR-IOV.

Table 5: CNI Configuration

Parameter	Description	Default Value
worker_nodes.cni.type	CNI type: macvlan or sriov	macvlan

Macvlan Configuration:

Use this configuration when worker_nodes.cni.type is set to macvlan.

```
worker_nodes:
  cni:
    type: "macvlan"
    macvlan:
  networks:
    - name: "macvlan-data1-bridge" # NetworkAttachmentDefinition name
      interface: "ens224"          # Host interface
    - name: "macvlan-data2-bridge"
      interface: "ens256"
    - name: "macvlan-data3-bridge"
      interface: "ens161"
```

The Helm chart automatically creates *NetworkAttachmentDefinition* resources for each network.

- **Generated interface-config:**

```
[interface0]
  iface_id = net1;
  uio_driver = afpacket;
[interface1]
  iface_id = net2;
  uio_driver = afpacket;
[interface2]
  iface_id = net3;
  uio_driver = afpacket;
```

SR-IOV Configuration:

Use this configuration when worker_nodes.cni.type is set to sriov.

```
worker_nodes:
  cni:
    type: "sriov"
    sriov:
      networks:
        - name: "sriov-net-data1"
          resourceName: "intel.com/intel_sriov_ens7f0"
        - name: "sriov-net-data2"
          resourceName: "intel.com/intel_sriov_ens7f1"
```

Table 6: SR-IOV Configuration

Parameter	Description
name	SriovNetwork CRD name
resourceName	Must match SriovNetworkNodePolicy resourceName

Note

- You can use the same network multiple times to bind to multiple VFs from the same PF.
- SR-IOV requires additional volume mounts for */sys/bus/pci and /dev/vfio*.

- **Generated interface-config:**

```
[interface0]
  iface_id = net1;
  uio_driver = vfio-pci;
```

```
[interface1]
  iface_id = net2;
  uio_driver = vfio-pci;
```

Persistent storage:

Persistent volumes retain FTDc data across restarts.

Lina storage (default):

Table 7: CNI configuration

Parameter	Description	Default Value
worker_nodes.persistence.lina.storageClass	Storage class name	lina-storage
worker_nodes.persistence.lina.size	Volume size	1Gi
worker_nodes.persistence.lina.hostPath	Host path on workernodes	/home/ubuntu/lina-path

NGFW storage (Optional):

Table 8: CNI configuration

Parameter	Description	Default Value
worker_nodes.persistence.ngfw.storageClass	Storage class name	NGFW-storage
worker_nodes.persistence.ngfw.hostPath	Volume size	5Gi
worker_nodes.persistence.ngfw.hostPath	Host path on workernodes	/home/ubuntu/ngfw-path

Step 7 Configure Day-0 parameters in the `day0.json` file.

The day0.json file provides initial bootstrap configuration for FTDc.

Default Configuration (day0.json):

```
{
"EULA": "accept",
"Hostname": "cisco-ftdc",
"AdminPassword": "<PASSWORD>",
"FirewallMode": "routed",
"DNS1": "208.67.222.222",
"Diagnostic": "OFF",
"IPv4Mode": "DHCP",
"IPv4Addr": "",
"IPv4Mask": "",
"IPv4Gw": "",
"FmcIp": "DONTRESOLVE",
"FmcRegKey": "docker",
"FmcNatId": "docker"
}
```

Table 9: Day-0 parameters Configuration

Field	Description
EULA	Accept EULA (accept)
Hostname	FTDc hostname

Field	Description
AdminPassword	Admin password for FTDc
FirewallMode	routed or transparent
DNS1	Primary DNS server
Diagnostic	Diagnostic interface (ON/OFF)
IPv4Mode	DHCP or Manual
IPv4Addr	Static IP (if Manual mode)
IPv4Mask	Subnet mask (if Manual mode)
IPv4Gw	Gateway (if Manual mode)
FmcIp	Management Center IP or DONTRESOLVE
FmcRegKey	Registration key for Management Center
FmcNatId	NAT ID for Management Center registration

Deployment Examples:

Example 1: Basic Deployment with Macvlan

```
# values-basic.yaml
ftdc:
  repository: "myregistry.example.com/ftdc:7.4"
  app_name: "ftdc"
  cpus: 4
  memory: 8192
  shmSize: "2Gi"
  hugepages: "auto"

ftdc_manager:
  enabled: false

worker_nodes:
  cni:
    type: "macvlan"
    macvlan:
       networks:
         - name: "macvlan-inside"
           interface: "eth1"
         - name: "macvlan-outside"
           interface: "eth2"
  persistence:
      lina:
        enabled: true
        storageClass: "lina-storage"
        size: "1Gi"
        hostPath: "/data/ftdc/lina"
        accessMode: "ReadWriteMany"

 worker_node1:
   name: "worker-01"
   ip: "192.168.1.100"

helm install ftdc . -f values-basic.yaml
```

Example 2: Production Deployment with SR-IOV and FTD manager

```
# values-production.yaml
ftdc:
  repository: "myregistry.example.com/ftdc:7.4"
  app_name: "ftdc-prod"
  cpus: 8
  memory: 16384
  shmSize: "4Gi"
  hugepages: "2Gi"

ftdc_manager:
  enabled: true
  repository: "myregistry.example.com/ftd-manager:v1"

fmcv:
  ip: "10.0.0.50"
  user: "admin"
  password: "" # Set via --set or secrets file
  access_policy: "Production-Policy"
  inside_zone: "inside"
  outside_zone: "outside"
  license_caps: "BASE,THREAT,URL"
  performance_tier: "FTDv30"

worker_nodes:
  cni:
    type: "sriov"
    sriov:
       networks:
         - name: "sriov-inside"
           resourceName: "intel.com/intel_sriov_ens3f0"
         - name: "sriov-outside"
           resourceName: "intel.com/intel_sriov_ens3f1"
         - name: "sriov-dmz"
           resourceName: "intel.com/intel_sriov_ens3f0"

  persistence:
      lina:
        enabled: true
        storageClass: "fast-storage"
        size: "5Gi"
        hostPath: "/data/ftdc/lina"
        accessMode: "ReadWriteMany"

  worker_node1:
    name: "prod-worker-01"
    ip: "10.0.1.100"

 helm install ftdc-prod . -f values-production.yaml --set fmcv.password=<password>
```

Verify and Access Firewall Threat Defense Container in Kubernetes Environment

Verify deployment status:

- Check all resources.

  ```
  kubectl get all -l app=FTDC
  ```

- Check pods.

```
kubectl get pods -o wide
```

- Check service.

```
kubectl get svc
```

- Check persistent volumes.

```
kubectl get pv,pvc
```

- Check NetworkAttachmentDefinitions (for macvlan).

```
kubectl get net-attach-def
```

Verify FTDc is running

- Check pod logs.

```
kubectl logs -f <ftdc-pod-name>
```

- Check lina process (readiness probe).

```
kubectl exec -it <ftdc-pod-name> -- pgrep -x lina
```

- Check readiness file.

```
kubectl exec -it <ftdc-pod-name> -- cat /tmp/lina_ready
```

- Access FTDc CLI.

```
kubectl exec -it <ftdc-pod-name> -- /bin/bash
```

Verify MetalLB service

- Check LoadBalancer external IP address.

```
kubectl get svc ftdc-k8s-worker1 -o jsonpath='{.status.loadBalancer.ingress[0].ip}'
```

- Test SSH connectivity (port 3000 maps to 22).

```
ssh -p 3000 admin@<external-ip>
```

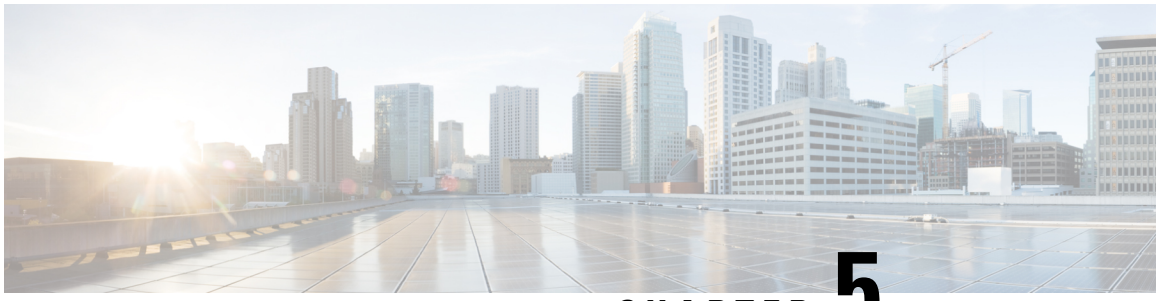
Verify FTD manager (if enabled)

- Check ftd-manager pod.

```
kubectl get pods -l app=ftd-manager
```

- Check logs.

```
kubectl logs -f -l app=ftd-manager
```



CHAPTER 5

Deploy Firewall Threat Defense Container in Amazon EKS Environment

You can deploy the Firewall Threat Defense container (FTDc) in an Amazon EKS environment.

- Prerequisites to Deploy Firewall Threat Defense Container in Amazon EKS Environment, on page 23
- Sample Topology to Deploy Firewall Threat Defense Container in Amazon EKS Environment, on page 26
- Deploy Firewall Threat Defense Container in Amazon EKS Environment, on page 26
- Verify and Access Firewall Threat Defense Container in Amazon EKS Environment, on page 31
- Managing Inter-Namespace Traffic in Amazon EKS Environment, on page 31

Prerequisites to Deploy Firewall Threat Defense Container in Amazon EKS Environment

Following are the prerequisites to deploy Firewall Threat Defense container in an Amazon EKS environment:

- **Operating System**: Ubuntu, Version - 20.04 LTS (minimum), 22.04 LTS (maximum)
- **Kubernetes Environment**:
  - Amazon EKS cluster version - 1.35
  - Managed or self-managed node group with at least one worker node.
  - Multus CNI installed for multi-network support.
- **Kubernetes CNI**
  - POD management CNI - AWS VPC CNI
  - FTDc data network CNI - Multus macvlan (passthru mode)
- **Worker Node Requirements:**
  - Hugepages configured (2Mi pages)
  - Use SSD/NVMe storage. HDD-based storage is not supported.
  - Data ENIs attached to data subnets.

- Data ENIs tagged: `node.k8s.amazonaws.com/no_manage=true`.
- Source/Destination check disabled on all data ENIs.
- Worker node labeled with a custom label (for example, `ftdc-node=worker1`).

EKS Cluster Setup (one-time)

1. **Multus CNI must be installed on the EKS cluster**:

```
kubectl apply -f
https://raw.githubusercontent.com/k8snetworkplumbingwg/multus-cni/master/deployments/multus-daemonset-thick.yml
```

2. **Tune VPC CNI to prevent it from managing data ENIs and pin a stable pod management IP address**:

```
kubectl set env daemonset aws-node -n kube-system \
  WARM_ENI_TARGET=0 \
  WARM_IP_TARGET=1 \
  MINIMUM_IP_TARGET=1
```

3. **Pin a stable management IP address for the FTDc pod**: By default, VPC CNI assigns a random secondary IP address from the primary ENI's subnet on each pod restart. To ensure the pod always gets the same management IP address:

   a. Remove all existing secondary IP addresses from the primary ENI:

```
aws ec2 unassign-private-ip-addresses --network-interface-id <primary-eni-id> \
  --private-ip-addresses <ip1> <ip2> ...
```

   b. Assign a single specific IP address (for example, `10.0.1.10`):

```
aws ec2 assign-private-ip-addresses --network-interface-id <primary-eni-id> \
  --private-ip-addresses 10.0.1.10
```

   Since `10.0.1.10` is the only secondary IP address in the pool, VPC CNI will always assign it to the pod. The IP address stays on the ENI across pod restarts — VPC CNI only manages the pod-to-IP address mapping, not the ENI-level assignment.



Note On node replacement, the new node has a different primary ENI. You must re-assign `10.0.1.10` on the new ENI.

4. **Label the worker nodes**:

```
kubectl label node <node-name> ftdc-node=worker1
```

AWS Infrastructure Requirements

1. **Create the EKS node group in the management subnet (`10.0.1.x`) so the primary ENI is in the mgmt network.**
2. **Attach data ENIs from other subnets (must be same AZ as the node)**:

```
aws ec2 attach-network-interface --instance-id <instance-id> \
  --network-interface-id <eni-id> --device-index <index>
```

3. **Tag data ENIs with `no_manage` to prevent VPC CNI from allocating secondary IP addresses or detaching them:**

```
aws ec2 create-tags --resources <eni-inside> <eni-diag> <eni-outside> \
  --tags Key=node.k8s.amazonaws.com/no_manage,Value=true
```

Without this tag, VPC CNI will treat data ENIs as its own warm ENIs and may detach or delete them during pod lifecycle events.

4. **Disable Source/Destination check on data ENIs (required for FTDc to route/inspect traffic):**

   ```
   aws ec2 modify-network-interface-attribute --network-interface-id <eni-id> \
     --no-source-dest-check
   ```

   This must be done for each data ENI. Without this, AWS will drop any traffic where the source or destination IP address does not match the ENI's assigned IP address.

5. **Assign secondary IP addresses on data ENIs for FTDc pod data interfaces:**

   FTDc data interfaces (configured via DHCP or statically inside FTDc) need IP addresses that are registered on the corresponding ENI to be VPC-routable. Assign a secondary IP address to each data ENI:

   ```
   aws ec2 assign-private-ip-addresses --network-interface-id <eni-diag-id>
   --private-ip-addresses 10.0.2.10
   aws ec2 assign-private-ip-addresses --network-interface-id <eni-inside-id>
   --private-ip-addresses 10.0.3.10
   aws ec2 assign-private-ip-addresses --network-interface-id <eni-outside-id>
   --private-ip-addresses 10.0.4.10
   ```

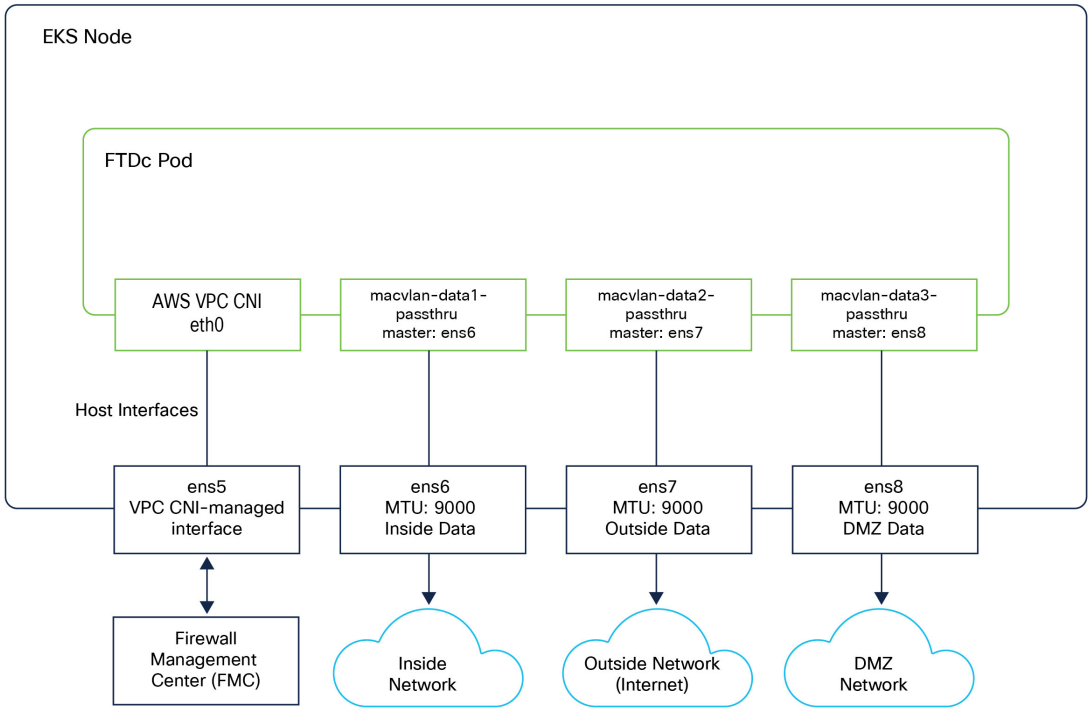
   - DHCP from inside the pod will assign the ENI's primary IP address (for example, `10.0.2.101`)
   - To use a specific IP address (for example, `x.x.x.10`), configure it statically inside FTDc — AWS DHCP only hands out the primary IP address
   - Any IP address not registered on the ENI will be silently dropped by the VPC router

6. **ECR access must be configured via IAM instance profile or node role.**

Additional Requirements

- **Helm:** Version 3.8 or later for deploying FTDc using Helm charts (YAML-based Infrastructure-as-Code)
- **Firewall Threat Defense Container (FTDc)** image available in an accessible registry
- **Management Center** accessible from the cluster

Sample Topology to Deploy Firewall Threat Defense Container in Amazon EKS Environment



The diagram shows an FTDc pod deployed on an EKS node, connected to multiple network segments using AWS VPC and macvlan CNI.

- The primary interface (ens5) is managed by the AWS VPC CNI and is used for pod communication with the Management Center.
- The inside interface connects to the inside endpoint network.
- The outside interface connects to the outside endpoint network.

Deploy Firewall Threat Defense Container in Amazon EKS Environment

Follow the procedure to deploy Firewall Threat Defense container (FTDc) in an Amazon EKS environment.

Procedure

Step 1 Set up the requirements mentioned in the Prerequisites.

Step 2 Clone the FTDc repository to the local folder.

```
git clone <repository-URL>
```

Step 3 Navigate to the Helm chart directory.

```
cd Helm_charts/ftdc-helm
```

Step 4 Deploy FTDc.

You can deploy FTDc using either the default configuration or a custom values file.

- Using Helm with default configuration:

  ```
  helm install ftdc . -n <namespace>
  ```

- Using Helm with a custom values file:

  ```
  helm install ftdc . -f my-values.yaml -n <namespace>
  ```

Note
Namespace should be created before using it in the helm install command.

Step 5 Deploy FTDc using Helm with additional parameters.

```
helm install ftdc . \
  --set ftdc.repository=<account>.dkr.ecr.<region>.amazonaws.com/ftdc:<tag> \
  --set ftdc.app_name=ftdc \
  --set ftdc.cpus=4 \
  --set ftdc.memory=8192 \
  --set ftdc.hugepages=auto \
  --set worker_nodes.cni.type=macvlan \
  --set worker_nodes.persistence.lina.enabled=true \
  --set worker_nodes.persistence.lina.hostPath=/home/ubuntu/lina-path \
  --set worker_nodes.persistence.ngfw.enabled=true \
  --set worker_nodes.persistence.ngfw.hostPath=/home/ubuntu/ngfw-path \
  --set worker_node1.label=worker1 \
  -n default
```

Step 6 Update the configuration parameters in the Helm chart.

See the configuration parameters in the following tables:

Table 10: FTDc container settings

Parameter	Description	Default Value	Required
ftdc.repository	FTDc container image repository	localhost:5000/ftdc_10.0.0.143	Yes
ftdc.app_name	Application name for deployment	ftdc	Yes
ftdc.cpus	Number of CPUs allocated to FTDc	4	Yes
ftdc.memory	Memory in MB allocated to FTDc	8192	Yes
ftdc.shmSize	Shared memory (/dev/shm) size	2Gi	No

Deploy Firewall Threat Defense Container in Amazon EKS Environment

Parameter	Description	Default Value	Required
ftdc.hugepages	Hugepages allocation	auto	No
replicas	Number of pod replicas	1	No
namespace	FTD container name space for deployment	default	No

Supported CPU and memory configurations:

The following CPU and memory configurations are supported for FTDc deployment in Kubernetes environments. These configurations align with supported performance tiers.

- FTDv5 – 4 vCPU, 8 GB RAM
- FTDv10 – 4 vCPU, 8 GB RAM
- FTDv20 – 4 vCPU, 8 GB RAM
- FTDv30 – 8 vCPU, 16 GB RAM
- FTDv50 – 12 vCPU, 24 GB RAM
- FTDv100 – 16 vCPU, 32 GB RAM

Hugepages configuration:

The ftdc.hugepages parameter supports two modes:

a. Auto-calculation ("auto"): Calculates hugepages based on CPUs and interfaces:

Formula: (CPUs*128)* + *(numInterfaces*64) Mi, rounded upto nearest 256 Mi.

Example: 4 CPUs + 3 interfaces = (4×128) + (3×64) = 704Mi → rounded to 768Mi

b. Fixed value: Specify exact size like "512Mi", "1Gi", "2Gi"

Table 11: Worker Node Configuration

Parameter	Description	Default Value	Required
worker_node1.label	Kubernetes label for the worker node	ftdc-node=worker1	Yes

Table 12: CNI Configuration

Parameter	Description	Default Value
worker_nodes.cni.type	CNI type: macvlan	macvlan

Macvlan Configuration:

Use this configuration when worker_nodes.cni.type is set to macvlan.

```
worker_nodes:
  cni:
    type: "macvlan"
    macvlan:
```

```
networks:
  - name: "macvlan-data1-passthrough" # NetworkAttachmentDefinition name
    interface: "ens6"          # Host interface
  - name: "macvlan-data2-passthrough"
    interface: "ens7"
  - name: "macvlan-data3-passthrough"
    interface: "ens8"
```

The Helm chart automatically creates *NetworkAttachmentDefinition* resources for each network.

- **Generated interface-config:**

```
[interface0]
  iface_id = net1;
  uio_driver = afpacket;
[interface1]
  iface_id = net2;
  uio_driver = afpacket;
[interface2]
  iface_id = net3;
  uio_driver = afpacket;
```

Persistent storage:

Persistent volumes retain FTDc data across restarts.

Table 13: Lina storage (default) parameters and values

Parameter	Description	Default Value
worker_nodes.persistence.lina.storageClass	Storage class name	lina-storage
worker_nodes.persistence.lina.size	Volume size	1Gi
worker_nodes.persistence.lina.hostPath	Host path on workernodes	/home/ubuntu/lina-path

Table 14: NGFW storage (optional) parameters and values

Parameter	Description	Default Value
worker_nodes.persistence.ngfw.storageClass	Storage class name	NGFW-storage
worker_nodes.persistence.ngfw.size	Volume size	5Gi
worker_nodes.persistence.ngfw.hostPath	Host path on workernodes	/home/ubuntu/ngfw-path

Step 7 Configure Day-0 parameters in the `day0.json` file.

The day0.json file provides initial bootstrap configuration for FTDc.

Default Configuration (day0.json):

```
{
"EULA": "accept",
"Hostname": "cisco-ftdc",
"AdminPassword": "<PASSWORD>",
"FirewallMode": "routed",
"DNS1": "<enter-ip-address>",
"Diagnostic": "OFF",
"IPv4Mode": "DHCP",
"IPv4Addr": "",
"IPv4Mask": "",
"IPv4Gw": "",
```

```
"FmcIp": "DONTRESOLVE",
"FmcRegKey": "docker",
"FmcNatId": "docker"
}
```

Table 15: Day-0 parameters Configuration

Field	Description
EULA	Accept EULA (accept)
Hostname	FTDc hostname
AdminPassword	Admin password for FTDc
FirewallMode	routed or transparent
DNS1	Primary DNS server
Diagnostic	Diagnostic interface (ON/OFF)
IPv4Mode	DHCP or Manual
IPv4Addr	Static IP (if Manual mode)
IPv4Mask	Subnet mask (if Manual mode)
IPv4Gw	Gateway (if Manual mode)
FmcIp	Management Center IP or DONTRESOLVE
FmcRegKey	Registration key for Management Center
FmcNatId	NAT ID for Management Center registration

Deployment Examples:

Example: Basic Deployment with Macvlan

```
# values-basic.yaml
ftdc:
  repository: "myregistry.example.com/ftdc:7.4"
  app_name: "ftdc"
  cpus: 4
  memory: 8192
  shmSize: "2Gi"
  hugepages: "auto"

worker_nodes:
  cni:
    type: "macvlan"
    macvlan:
      networks:
        - name: "macvlan-inside"
          interface: "eth1"
        - name: "macvlan-outside"
          interface: "eth2"
  persistence:
    lina:
      enabled: true
      storageClass: "lina-storage"
      size: "1Gi"
      hostPath: "/data/ftdc/lina"
      accessMode: "ReadWriteMany"
```

```
worker_node1:
  label: "worker1"

bash
helm install ftdc . -f values-basic.yaml
```

Verify and Access Firewall Threat Defense Container in Amazon EKS Environment

Verify deployment status:

- Check all resources.

  ```
  kubectl get all -l app=FTDC
  ```

- Check pods.

  ```
  kubectl get pods -o wide
  ```

- Check service.

  ```
  kubectl get svc
  ```

- Check persistent volumes.

  ```
  kubectl get pv,pvc
  ```

- Check NetworkAttachmentDefinitions (for macvlan).

  ```
  kubectl get net-attach-def
  ```

Verify FTDc is running

- Check pod logs.

  ```
  kubectl logs -f <ftdc-pod-name>
  ```

- Check lina process (readiness probe).

  ```
  kubectl exec -it <ftdc-pod-name> -- pgrep -x lina
  ```

- Check readiness file.

  ```
  kubectl exec -it <ftdc-pod-name> -- cat /tmp/lina_ready
  ```

- Access FTDc CLI.

  ```
  kubectl exec -it <ftdc-pod-name> -- /bin/bash
  ```

Managing Inter-Namespace Traffic in Amazon EKS Environment

FTDc facilitates secure inter-namespace communication through a combination of dynamic discovery and traffic redirection.

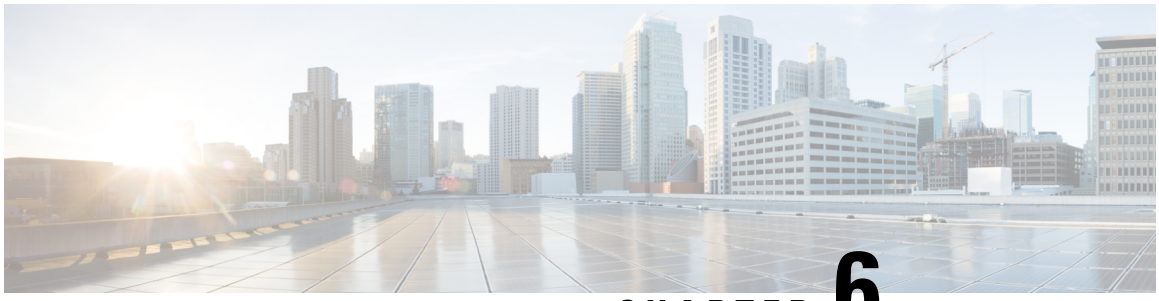
Dynamic discovery and object mapping

FTDc utilizes a controller that monitors the Kubernetes environment for changes in pod status.

- **Dynamic objects**: When a pod is created or terminated in a specific namespace, the controller automatically updates the corresponding dynamic object on the Management Center Virtual.
- **IP address mapping**: The Management Center Virtual continuously tracks the IP addresses associated with these namespaces. Because these objects are dynamic, firewall policies remain consistent regardless of pod scaling, eliminating the need for manual rule updates.

Traffic redirection

- To ensure inspection, network traffic must be directed through the FTDc instance.
  - **Routing/CNI configuration**: The cluster's Container Network Interface (CNI) or routing tables are configured to route inter-namespace traffic through the FTDc.
  - **Transparent inspection**: The FTDc acts as a transparent gateway. Traffic between a frontend pod and a backend pod is intercepted by the FTDc rather than being routed directly.



CHAPTER 6

Uninstall Firewall Threat Defense Container

- Uninstall Firewall Threat Defense Container in Docker Environment, on page 33
- Uninstall Firewall Threat Defense Container in Kubernetes and Amazon EKS Environments, on page 33

Uninstall Firewall Threat Defense Container in Docker Environment

To stop and remove the FTDc container, use the following commands:

```
docker kill <FTDc container name or ID>
```

```
docker rm <FTDc container name or ID>
```

To remove persistent state that stores configuration and management data, use the following command:

```
sudo rm -rf ftdc_shared/lina-path ftdc_shared/ngfw-path
```

Note Removing persistent state is required only if you want to perform a fresh deployment.

Uninstall Firewall Threat Defense Container in Kubernetes and Amazon EKS Environments

The Helm chart includes pre-delete hooks that:

1. Collect runtime information for all FTDc pods.
2. Scale down and delete FTDc deployments gracefully.
3. Clean up persistent storage on worker nodes.

   To uninstall FTDc, use the following command.

   ```
   helm uninstall ftdc
   ```

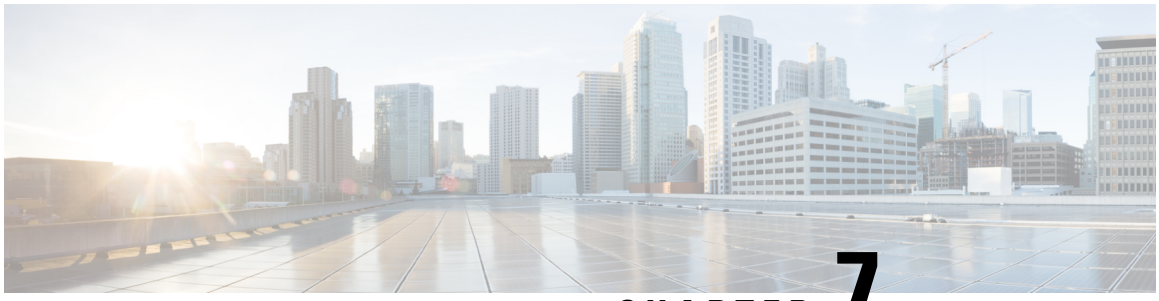
   To verify cleanup, use the following commands.

```
kubectl get pods
kubectl get pv,pvc
```

If required, do a manual cleanup.

To delete any remaining resources, use the following commands.

```
kubectl delete deployment -l tier=ftdvc
kubectl delete pvc lina-pvc ngfw-pvc
kubectl delete pv lina-pv ngfw-pv
```



CHAPTER 7

Limitations

• Limitations for Deployment of Firewall Threat Defense Container, on page 35

Limitations for Deployment of Firewall Threat Defense Container

The Firewall Threat Defense container solution is validated on open-source Docker, Kubernetes, and Amazon EKS environments.

- If a reboot or shutdown is triggered from Management Center, the lina process inside the container may be terminated. This can result in traffic disruption. The container will not be gracefully restarted or recovered by this operation.

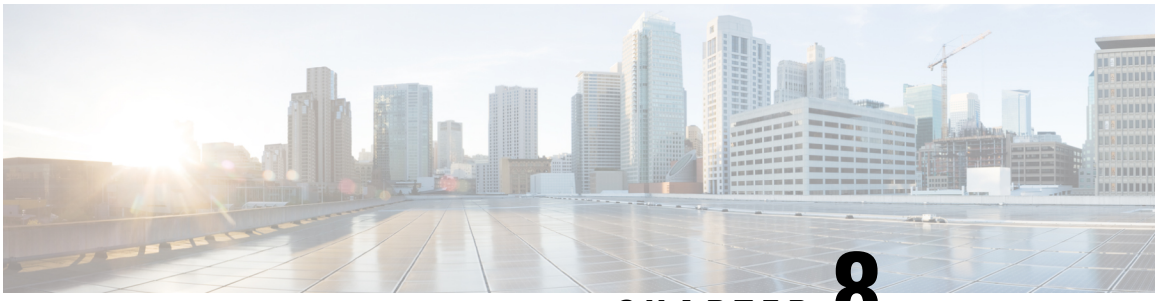
Caution Management Center currently allows reboot/shutdown operations for FTDc devices. These operations must not be used for container-based deployments.

- Docker does not guarantee consistent network interface order (for example, eth0, eth1) when a container with multiple networks is restarted. After a restart, verify and, if required, reconnect the Docker network interfaces for FTDc.
- SNMP polling behavior in Kubernetes:
  - Kubernetes applies Source NAT (SNAT) for external traffic by default.

    SNMP polling must be configured using the Kubernetes node IP instead of original source IP of the SNMP client.
  - As a result, the original SNMP client IP is not visible to the FTDc container.
- The following features are not supported:
  - Clustering
  - High Availability
  - Transparent mode
  - Subinterfaces are not supported only when using macvlan CNI.
  - IPv6 is not supported when using macvlan CNI.

  - Inline mode interface
  - SR-IOV CNI in Amazon EKS

- In afpacket mode, the Maximum Transmission Unit (MTU) must not exceed 8140 bytes, even though Management Center may allow higher values; such configurations are not supported for FTDc deployments.



CHAPTER 8

Troubleshooting

• Troubleshooting Deployment of Firewall Threat Defense Container, on page 37

Troubleshooting Deployment of Firewall Threat Defense Container

- **Issue**: Pod stuck in pending state

  Cause: Insufficient resources (hugepages, CPU, memory, SR-IOV VFs)

  Workaround:

  - Check the events.

    ```
    kubectl describe pod <pod-name>
    ```

  - Verify hugepages on the node.

    ```
    kubectl describe node <node-name> | grep hugepages
    ```

  - Verify SR-IOV resources.

    ```
    kubectl describe node <node-name> | grep intel.com
    ```

- **Issue**: Pod CrashLoopBackOff

  Cause: Configuration issues or insufficient shared memory

  Workaround:

  - Check the logs.

    ```
    kubectl logs --previous
    ```

  - Increase shmSize if necessary.

    ```
    helm upgrade ftdc . --set ftdc.shmSize=4Gi
    ```

- **Issue**: Network interfaces not working

  Cause: CNI misconfiguration

  Workaround:

- Check NetworkAttachmentDefinitions

  ```
  kubectl get net-attach-def -o yaml
  ```

- Verify pod annotations.

  ```
  kubectl get pod <pod-name> -o jsonpath='{.metadata.annotations}'
  ```

- Check interface-config ConfigMap.

  ```
  kubectl get configmap interface-config -o yaml
  ```

- **Issue**: Management Center registration failed

  Cause: Network connectivity or credential issues

  Workaround:

  - Check ftd-manager logs.

    ```
    kubectl logs -l app=ftd-manager
    ```

  - Verify Management Center credentials secret.

    ```
    kubectl get secret fmc-credentials -o yaml
    ```

  - Test Management Center connectivity from pod.

    ```
    kubectl exec -it -- curl -k https://<fmc-ip>
    ```

- **Issue**: MetalLB not Assigning IP address

  Cause: IPAddressPool misconfiguration

  Workaround:

  - Check IPAddressPool.

    ```
    kubectl get ipaddresspool -n metallb-system
    ```

  - Check L2Advertisement.

    ```
    kubectl get l2advertisement -n metallb-system
    ```

  - Check MetalLB controller logs.

    ```
    kubectl logs -n metallb-system -l app=metallb,component=controller
    ```

- **Issue**: SNMP polling not working

  Cause: SNMP requests may fail due to misconfiguration, missing NAT rules, or Kubernetes SNAT behavior.

  Workaround:

  - Verify SNMP configuration.

    Ensure SNMP host configuration is correct:

    ```
    show running-config snmp-server
    ```

    ```
    show snmp-server statistics
    ```

  - Verify snmpd process.

In expert mode, ensure SNMP daemon is running and listening on UDP port 161:

```
$ ps aux | grep snmpd
$ netstat -ulnp | grep 161
```

- Verify iptables rules.

  Ensure SNMP traffic is allowed and NAT rules are present:

  ```
  $ sudo iptables -L INPUT -n | grep 161
  $ sudo iptables -t nat -L -n | grep 161
  ```

- Capture SNMP traffic.

  Capture packets on management interface:

  ```
  $ sudo tcpdump -i eth0 -n udp port 161 -v
  ```

- Identify source IP address behavior (Kubernetes only).

  Compare source IP address in packet capture with configured SNMP host:

  In Docker: Source IP address = SNMP client IP address

  In Kubernetes: Source IP address = Node IP address (due to SNAT)

  If mismatch is observed, update SNMP configuration to use Kubernetes node IP address.