Red Hat build of MicroShift 4.16

Configuring

Configuring MicroShift

Last Updated: 2024-06-26

Red Hat build of MicroShift 4.16 Configuring

Configuring MicroShift

Abstract

This document provides instructions for configuring MicroShift.

Table of Contents

CHAPTER 1. HOW CONFIGURATION TOOLS WORK

A YAML file customizes MicroShift instances with your preferences, settings, and parameters.

NOTE

If you want to make configuration changes or deploy applications through the MicroShift API with tools other than **kustomize** manifests, you must wait until the Greenboot health checks have finished. This ensures that your changes are not lost if Greenboot rolls your **rpm-ostree** system back to an earlier state.

1.1. DEFAULT SETTINGS

If you do not create a **config.yaml** file, default values are used. The following example shows the default configuration settings.

- To see the default values, run the following command:

```
$ microshift show-config
```

Default values example output in YAML form

```
apiServer:
  advertiseAddress: 10.44.0.0/32 1
  auditLog:
    maxFileAge: 0 2
    maxFileSize: 200 3
    maxFiles: 10 4
    profile: Default 5
  namedCertificates:
    - certPath: ""
      keyPath: ""
      names:
        - ""
  subjectAltNames: [] 6
debugging:
  logLevel: "Normal" 7
dns:
  baseDomain: microshift.example.com 8
etcd:
  memoryLimitMB: 0 9
ingress:
  listenAddress:
    - "" 10
  ports: 11
    http: 80
    https: 443
  routeAdmissionPolicy:
    namespaceOwnership: InterNamespaceAllowed 12
  status: Managed 13
manifests: 14
  kustomizePaths:
    - /usr/lib/microshift/manifests
    - /usr/lib/microshift/manifests.d/*
    - /etc/microshift/manifests
    - /etc/microshift/manifests.d/*
network:
  clusterNetwork:
    - 10.42.0.0/16 15
  serviceNetwork:
    - 10.43.0.0/16 16
  serviceNodePortRange: 30000-32767 17
node:
  hostnameOverride: "" 18
  nodeIP: "" 19
```

1. A string that specifies the IP address from which the API server is advertised to members of the cluster. The default value is calculated based on the address of the service network.

2. How long log files are kept before automatic deletion. The default value of **0** in the **maxFileAge** parameter means a log file is never deleted based on age. This value can be configured.

3. By default, when the **audit.log** file reaches the **maxFileSize** limit, the **audit.log** file is rotated and MicroShift begins writing to a new **audit.log** file. This value can be configured.

4. The total number of log files kept. By default, MicroShift retains 10 log files. The oldest is deleted when an excess file is created. This value can be configured.

5. Logs only metadata for read and write requests; does not log request bodies except for OAuth access token requests. If you do not specify this field, the **Default** profile is used.

6. Subject Alternative Names for API server certificates.

7. Log verbosity. Valid values for this field are **Normal**, **Debug**, **Trace**, or **TraceAll**.

8. By default, **etcd** uses as much memory as needed to handle the load on the system. However, in memory constrained systems, it might be preferred or necessary to limit the amount of memory **etcd** can to use at a given time.

9. Base domain of the cluster. All managed DNS records are subdomains of this base.

10. The **ingress.listenAddress** value defaults to the entire network of the host. The valid configurable value is a list that can be either a single IP address or NIC name or multiple IP addresses and NIC names.

11. Default ports shown. Configurable. Valid values for both port entries are a single, unique port in the 1-65535 range. The values of the **ports.http** and **ports.https** fields cannot be the same.

12. Describes how hostname claims across namespaces are handled. By default, allows routes to claim different paths of the same hostname across namespaces. Valid values are **Strict** and **InterNamespaceAllowed**. Specifying **Strict** prevents routes in different namespaces from claiming the same hostname. If the value is deleted in a customized MicroShift **config.yaml**, the **InterNamespaceAllowed** value is automatically set.

13. Default router status, can be **Managed** or **Removed**.

14 The locations on the file system to scan for **kustomization** files to use to load manifests. Set to a list of paths to scan only those paths. Set to an empty list to disable loading

15 A block of IP addresses from which pod IP addresses are allocated. This field is immutable after installation.

16 A block of virtual IP addresses for Kubernetes services. IP address pool for services. A single entry is supported. This field is immutable after installation.

17 The port range allowed for Kubernetes services of type **NodePort**. If not specified, the default range of 30000-32767 is used. Services without a **NodePort** specified are automatically allocated one from this range. This parameter can be updated after the cluster is installed.

18 The name of the node. The default value is the hostname. If non-empty, this string is used to identify the node instead of the hostname.

19 The IP address of the node. The default value is the IP address of the default route.

1.2. USING A YAML CONFIGURATION FILE

At start up, MicroShift checks the system-wide **/etc/microshift/** directory for a configuration file named **config.yaml**. If the configuration file does not exist in the directory, the built-in default values are used to start the service.

1.2.1. Custom settings

To create custom configurations, you must create a **config.yaml** file in the **/etc/microshift/** directory, and then change any settings that are expected to override the defaults before starting or restarting MicroShift.

IMPORTANT

Restart MicroShift after changing any configuration settings to have them take effect. The **config.yaml** file is read only when MicroShift starts.

TIP

If you add all of the configurations you need at the same time, you can minimize system restarts.

1.2.2. Configuring the advertise address network flag

The **apiserver.advertiseAddress** flag specifies the IP address on which to advertise the API server to members of the cluster. This address must be reachable by the cluster. You can set a custom IP address here, but you must also add the IP address to a host interface. Customizing this parameter preempts MicroShift from adding a default IP address to the **br-ex** network interface.

IMPORTANT

If you customize the **advertiseAddress** IP address, make sure it is reachable by the cluster when MicroShift starts by adding the IP address to a host interface.

If unset, the default value is set to the next immediate subnet after the service network. For example, when the service network is **10.43.0.0/16**, the **advertiseAddress** is set to **10.44.0.0/32**.

1.2.3. Extending the port range for NodePort services

The **serviceNodePortRange** setting extends the port range available to NodePort services. This option is useful when specific standard ports under the **30000-32767** range need to be exposed. For example, if your device needs to expose the **1883/tcp** MQ Telemetry Transport (MQTT) port on the network because client devices cannot use a different port.

IMPORTANT

NodePorts can overlap with system ports, causing a malfunction of the system or MicroShift.

Consider the following when configuring the NodePort service ranges:

- Do not create any NodePort service without an explicit **nodePort** selection. When an explicit **nodePort** is not specified, the port is assigned randomly by the **kube-apiserver** and cannot be predicted.
- Do not create any NodePort service for any system service port, MicroShift port, or other services you expose on your device **HostNetwork**.
- Table one specifies ports to avoid when extending the port range:

Table 1.1. Ports to avoid.

Port	Description
22/tcp	SSH port
80/tcp	OpenShift Router HTTP endpoint
443/tcp	OpenShift Router HTTPS endpoint
1936/tcp	Metrics service for the openshift-router, not exposed today
2379/tcp	etcd port
2380/tcp	etcd port
6443	kubernetes API
8445/tcp	openshift-route-controller-manager
9537/tcp	cri-o metrics
10250/tcp	kubelet

Port	Description
10248/tcp	kubelet healthz port
10259/tcp	kube scheduler

1.3. ADDITIONAL RESOURCES

- Checking Greenboot status

CHAPTER 2. CLUSTER ACCESS WITH KUBECONFIG

Learn about how **kubeconfig** files are used with MicroShift deployments. CLI tools use **kubeconfig** files to communicate with the API server of a cluster. These files provide cluster details, IP addresses, and other information needed for authentication.

2.1. KUBECONFIG FILES FOR CONFIGURING CLUSTER ACCESS

The two categories of **kubeconfig** files used in MicroShift are local access and remote access. Every time MicroShift starts, a set of **kubeconfig** files for local and remote access to the API server are generated. These files are generated in the **/var/lib/microshift/resources/kubeadmin/** directory using preexisting configuration information.

Each access type requires a different authentication certificate signed by different Certificate Authorities (CAs). The generation of multiple **kubeconfig** files accommodates this need.

You can use the appropriate **kubeconfig** file for the access type needed in each case to provide authentication details. The contents of MicroShift **kubeconfig** files are determined by either default built-in values or a **config.yaml** file.

NOTE

A **kubeconfig** file must exist for the cluster to be accessible. The values are applied from built-in default values or a **config.yaml**, if one was created.

Example contents of the kubeconfig files

```
/var/lib/microshift/resources/kubeadmin/
├── kubeconfig 1
├── alt-name-1 2
│   └── kubeconfig
├── 1.2.3.4 3
│   └── kubeconfig
└── microshift-rhel9 4
    └── kubeconfig
```

1 Local host name. The main IP address of the host is always the default.

2 Subject Alternative Names for API server certificates.

3 DNS name.

4 MicroShift host name.

2.2. LOCAL ACCESS KUBECONFIG FILE

The local access **kubeconfig** file is written to **/var/lib/microshift/resources/kubeadmin/kubeconfig**. This **kubeconfig** file provides access to the API server using **localhost**. Choose this file when you are connecting the cluster locally.

Example contents of kubeconfig for local access

```
clusters:
- cluster:
    certificate-authority-data: <base64 CA>
    server: https://localhost:6443
```

The **localhost kubeconfig** file can only be used from a client connecting to the API server from the same host. The certificates in the file do not work for remote connections.

2.2.1. Accessing the MicroShift cluster locally

Use the following procedure to access the MicroShift cluster locally by using a **kubeconfig** file.

Prerequisites

- You have installed the **oc** binary.

Procedure

1. Optional: to create a **~/.kube/** folder if your RHEL machine does not have one, run the following command:

   ```
   $ mkdir -p ~/.kube/
   ```

2. Copy the generated local access **kubeconfig** file to the **~/.kube/** directory by running the following command:

   ```
   $ sudo cat /var/lib/microshift/resources/kubeadmin/kubeconfig > ~/.kube/config
   ```

3. Update the permissions on your **~/.kube/config** file by running the following command:

   ```
   $ chmod go-r ~/.kube/config
   ```

Verification

- Verify that MicroShift is running by entering the following command:

  ```
  $ oc get all -A
  ```

2.3. REMOTE ACCESS KUBECONFIG FILES

When a MicroShift cluster connects to the API server from an external source, a certificate with all of the alternative names in the SAN field is used for validation. MicroShift generates a default **kubeconfig** for external access using the **hostname** value. The defaults are set in the **<node.hostnameOverride>**, **<node.nodeIP>** and **api.<dns.baseDomain>** parameter values of the default **kubeconfig** file.

The **/var/lib/microshift/resources/kubeadmin/<hostname>/kubeconfig** file uses the **hostname** of the machine, or **node.hostnameOverride** if that option is set, to reach the API server. The CA of the **kubeconfig** file is able to validate certificates when accessed externally.

Example contents of a default kubeconfig file for remote access

```
clusters:
```

```
- cluster:
    certificate-authority-data: <base64 CA>
    server: https://microshift-rhel9:6443
```

2.3.1. Remote access customization

Multiple remote access **kubeconfig** file values can be generated for accessing the cluster with different IP addresses or host names. An additional **kubeconfig** file generates for each entry in the **apiServer.subjectAltNames** parameter. You can copy remote access **kubeconfig** files from the host during times of IP connectivity and then use them to access the API server from other workstations.

2.4. GENERATING ADDITIONAL KUBECONFIG FILES FOR REMOTE ACCESS

You can generate additional **kubeconfig** files to use if you need more host names or IP addresses than the default remote access file provides.

IMPORTANT

You must restart MicroShift for configuration changes to be implemented.

Prerequisites

- You have created a **config.yaml** for MicroShift.

Procedure

1. Optional: You can show the contents of the **config.yaml**. Run the following command:

```
$ cat /etc/microshift/config.yaml
```

2. Optional: You can show the contents of the remote-access **kubeconfig** file. Run the following command:

```
$ cat /var/lib/microshift/resources/kubeadmin/<hostname>/kubeconfig
```

IMPORTANT

Additional remote access **kubeconfig** files must include one of the server names listed in the Red Hat build of MicroShift **config.yaml** file. Additional **kubeconfig** files must also use the same CA for validation.

3. To generate additional **kubeconfig** files for additional DNS names SANs or external IP addresses, add the entries you need to the **apiServer.subjectAltNames** field. In the following example, the DNS name used is **alt-name-1** and the IP address is **1.2.3.4**.

Example config.yaml with additional authentication values

```
dns:
  baseDomain: example.com
node:
  hostnameOverride: "microshift-rhel9" 1
  nodeIP: 10.0.0.1
apiServer:
  subjectAltNames:
  - alt-name-1 (2)
  - 1.2.3.4 (3)
```

1 Hostname

2 DNS name

3 IP address or range

4. Restart MicroShift to apply configuration changes and auto-generate the **kubeconfig** files you need by running the following command:

```
$ sudo systemctl restart microshift
```

5. To check the contents of additional remote-access **kubeconfig** files, insert the name or IP address as listed in the **config.yaml** into the **cat** command. For example, **alt-name-1** is used in the following example command:

```
$ cat /var/lib/microshift/resources/kubeadmin/alt-name-1/kubeconfig
```

6. Choose the **kubeconfig** file to use that contains the SAN or IP address you want to use to connect your cluster. In this example, the **kubeconfig** containing`alt-name-1` in the **cluster.server** field is the correct file.

Example contents of an additional kubeconfig file

```
clusters:
- cluster:
    certificate-authority-data: <base64 CA>
    server: https://alt-name-1:6443 (1)
```

1 The **/var/lib/microshift/resources/kubeadmin/alt-name-1/kubeconfig** file values are from the **apiServer.subjectAltNames** configuration values.

NOTE

All of these parameters are included as common names (CN) and subject alternative names (SAN) in the external serving certificates for the API server.

2.4.1. Opening the firewall for remote access to the MicroShift cluster

Use the following procedure to open the firewall so that a remote user can access the MicroShift cluster. This procedure must be completed before a workstation user can access the cluster remotely.

For this procedure, **user@microshift** is the user on the MicroShift host machine and is responsible for setting up that machine so that it can be accessed by a remote user on a separate workstation.

Prerequisites

- You have installed the **oc** binary.
- Your account has cluster administration privileges.

Procedure

- As **user@microshift** on the MicroShift host, open the firewall port for the Kubernetes API server (**6443/tcp**) by running the following command:

```
[user@microshift]$ sudo firewall-cmd --permanent --zone=public --add-port=6443/tcp &&
sudo firewall-cmd --reload
```

Verification

- As **user@microshift**, verify that MicroShift is running by entering the following command:

```
[user@microshift]$ oc get all -A
```

2.4.2. Accessing the MicroShift cluster remotely

Use the following procedure to access the MicroShift cluster from a remote location by using a **kubeconfig** file.

The **user@workstation** login is used to access the host machine remotely. The **<user>** value in the procedure is the name of the user that **user@workstation** logs in with to the MicroShift host.

Prerequisites

- You have installed the **oc** binary.
- The **user@microshift** has opened the firewall from the local host.

Procedure

1. As **user@workstation**, create a **~/.kube/** folder if your RHEL machine does not have one by running the following command:

```
[user@workstation]$ mkdir -p ~/.kube/
```

2. As **user@workstation**, set a variable for the hostname of your MicroShift host by running the following command:

```
[user@workstation]$ MICROSHIFT_MACHINE=<name or IP address of MicroShift machine>
```

3. As **user@workstation**, copy the generated **kubeconfig** file that contains the host name or IP address you want to connect with from the RHEL machine running MicroShift to your local machine by running the following command:

```
[user@workstation]$ ssh <user>@$MICROSHIFT_MACHINE "sudo cat
/var/lib/microshift/resources/kubeadmin/$MICROSHIFT_MACHINE/kubeconfig" >
~/.kube/config
```

NOTE

To generate the **kubeconfig** files for this step, see Generating additional kubeconfig files for remote access.

4. As **user@workstation**, update the permissions on your **~/.kube/config** file by running the following command:

```
$ chmod go-r ~/.kube/config
```

Verification

- As **user@workstation**, verify that MicroShift is running by entering the following command:

```
[user@workstation]$ oc get all -A
```

CHAPTER 3. CONFIGURING CUSTOM CERTIFICATE AUTHORITIES

You can encrypt connections by using custom certificate authorities (CAs) with the MicroShift service.

3.1. HOW CUSTOM CERTIFICATE AUTHORITIES WORK IN MICROSHIFT

The default API server certificate is issued by an internal MicroShift cluster certificate authority (CA). Clients outside of the cluster cannot verify the API server certificate by default. This certificate can be replaced by a custom server certificate that is issued externally by a custom CA that clients trust. The following steps illustrate the workflow in MicroShift:

1. Copy the certificates and keys to the preferred directory in the host operating system. Ensure that the files are accessible by root only.

2. Update the MicroShift configuration for each custom CA by specifying the certificate names and new fully qualified domain name (FQDN) in the MicroShift **/etc/microshift/config.yaml** configuration file.
   Each certificate configuration can contain the following values:

   - The certificate file location is a required value.

   - A single common name containing the API server DNS and IP address or IP address range.

     TIP

     In most cases, MicroShift generates a new **kubeconfig** for your custom CA that includes the IP address or range that you specify. The exception is when wildcards are specified for the IP address. In this case, MicroShift generates a **kubeconfig** with the public IP address of the server. To use wildcards, you must update the **kubeconfig** file with your specific details.

   - Multiple Subject Alternative Names (SANs) containing the API server DNS and IP addresses or a wildcard certificate.

   - You can provide additional DNS names for each certificate.

3. After the MicroShift service restarts, you must copy the generated **kubeconfig** files to the client.

4. Configure additional CAs on the client system. For example, you can update CA bundles in the Red Hat Enterprise Linux (RHEL) truststore.

5. The certificates and keys are read from the specified file location on the host. Testing and validation of configuration is done from the client.

6. External server certificates are not automatically renewed. You must manually rotate your external certificates.

NOTE

If any validation fails, the MicroShift service skips the custom configuration and uses the default certificate to start. The priority is to continue the service uninterrupted. MicroShift logs errors when the service starts. Common errors include expired certificates, missing files, or incorrect IP addresses.

IMPORTANT

Custom server certificates have to be validated against CA data configured in the trust root of the host operating system. For information, see The system-wide truststore.

3.2. CONFIGURING CUSTOM CERTIFICATE AUTHORITIES

To configure externally generated certificates and domain names using custom certificate authorities (CAs), add them to the MicroShift **/etc/microshift/config.yaml** configuration file. You must also configure the host operating system trust root.

NOTE

Externally generated **kubeconfig** files are created in the **/var/lib/microshift/resources/kubeadmin/<hostname>/kubeconfig** directory. If you need to use **localhost** in addition to externally generated configurations, retain the original **kubeconfig** file in its default location. The **localhost kubeconfig** file uses the self-signed certificate authority.

Prerequisites

- The OpenShift CLI (**oc**) is installed.
- You have access to the cluster as a user with the cluster administration role.
- The certificate authority has issued the custom certificates.
- A MicroShift **/etc/microshift/config.yaml** configuration file exists.

Procedure

1. Copy the custom certificates you want to add to the trust root of the MicroShift host. Ensure that the certificate and private keys are only accessible to MicroShift.
2. For each custom CA that you need, add an **apiServer** section called **namedCertificates** to the **/etc/microshift/config.yaml** MicroShift configuration file by using the following example:

```
apiServer:
  namedCertificates:
  - certPath: ~/certs/api_fqdn_1.crt (1)
    keyPath:  ~/certs/api_fqdn_1.key (2)
  - certPath: ~/certs/api_fqdn_2.crt
    keyPath:  ~/certs/api_fqdn_2.key
    names: (3)
    - api_fqdn_1
    - *.apps.external.com
```

(1) Add the full path to the certificate.

(2) Add the full path to the certificate key.

(3) Optional. Add a list of explicit DNS names. Leading wildcards are allowed. If no names are provided, the implicit names are extracted from the certificates.

3. Restart the {microshift-service} to apply the certificates by running the following command:

```
$ systemctl microshift restart
```

4. Wait a few minutes for the system to restart and apply the custom server. New **kubeconfig** files are generated in the **/var/lib/microshift/resources/kubeadmin/** directory.

5. Copy the **kubeconfig** files to the client. If you specified wildcards for the IP address, update the **kubeconfig** to remove the public IP address of the server and replace that IP address with the specific wildcard range you want to use.

6. From the client, use the following steps:

    a. Specify the **kubeconfig** to use by running the following command:

    ```
    $ export KUBECONFIG=~/custom-kubeconfigs/kubeconfig (1)
    ```

    (1) Use the location of the copied **kubeconfig** file as the path.

    b. Check that the certificates are applied by using the following command:

    ```
    $ oc --certificate-authority ~/certs/ca.ca get node
    ```

    Example output

    ```
    oc get node
    NAME                             STATUS   ROLES                          AGE   VERSION
    dhcp-1-235-195.arm.example.com   Ready    control-plane,master,worker   76m   v1.29.2
    ```

    c. Add the new CA file to the $KUBECONFIG environment variable by running the following command:

    ```
    $ oc config set clusters.microshift.certificate-authority /tmp/certificate-authority-data-new.crt
    ```

    d. Verify that the new **kubeconfig** file contains the new CA by running the following command:

    ```
    $ oc config view --flatten
    ```

    Example externally generated kubeconfig file

    ```
    apiVersion: v1
    clusters:
    - cluster:
        certificate-authority: /tmp/certificate-authority-data-new.crt (1)
        server: https://api.ci-ln-k0gim2b-76ef8.aws-2.ci.openshift.org:6443
      name: ci-ln-k0gim2b-76ef8
    contexts:
    - context:
        cluster: ci-ln-k0gim2b-76ef8
        user:
    ```

```
  name:
current-context:
kind: Config
preferences: {}
```

1. The **certificate-authority-data** section is not present in externally generated **kubeconfig** files. It is added with the **oc config set** command used previously.

e. Verify the **subject** and **issuer** of your customized API server certificate authority by running the following command:

```
$ curl --cacert /tmp/caCert.pem https://${fqdn_name}:6443/healthz -v
```

Example output

```
Server certificate:
  subject: CN=kas-test-cert_server
  start date: Mar 12 11:39:46 2024 GMT
  expire date: Mar 12 11:39:46 2025 GMT
  subjectAltName: host "dhcp-1-235-3.arm.eng.rdu2.redhat.com" matched cert's "dhcp-1-235-3.arm.eng.rdu2.redhat.com"
  issuer: CN=kas-test-cert_ca
  SSL certificate verify ok.
```

IMPORTANT

Either replace the **certificate-authority-data** in the generated **kubeconfig** file with the new **rootCA** or add the **certificate-authority-data** to the trust root of the operating system. Do not use both methods.

f. Configure additional CAs in the trust root of the operating system. For example, in the RHEL Client truststore on the client system. See The system-wide truststore for details.

- Updating the certificate bundle with the configuration that contains the CA is recommended.
- If you do not want to configure your certificate bundles, you can alternately use the **oc login localhost:8443 --certificate-authority=/path/to/cert.crt** command, but this method is not preferred.

3.3. CUSTOM CERTIFICATES RESERVED NAME VALUES

The following certificate problems cause MicroShift to ignore certificates dynamically and log an error:

- The certificate files do not exist on the disk or are not readable.
- The certificate is not parsable.
- The certificate overrides the internal certificates IPAddress/DNSNames in a **SubjectAlternativeNames** (SAN) field. Do not use a reserved name when configuring SANs.

Table 3.1. Reserved Names values

Address	Type	Comment
localhost	DNS	
127.0.0.1	IP Address	
10.42.0.0	IP Address	Cluster Network
10.43.0.0/16,10.44.0.0/16	IP Address	Service Network
169.254.169.2/29	IP Address	br-ex Network
kubernetes.default.svc	DNS	
openshift.default.svc	DNS	
svc.cluster.local	DNS	

3.4. TROUBLESHOOTING CUSTOM CERTIFICATES

To troubleshoot the implementation of custom certificates, you can take the following steps.

Procedure

1. From MicroShift, ensure that the certificate is served by the **kube-apiserver** and verify that the certificate path is appended to the **--tls-sni-cert-key** FLAG by running the following command:

   ```
   $ journalctl -u microshift -b0 | grep tls-sni-cert-key
   ```

   #### Example output

   ```
   Jan 24 14:53:00 localhost.localdomain microshift[45313]: kube-apiserver I0124
   14:53:00.649099   45313 flags.go:64] FLAG: --tls-sni-cert-key="
   [/home/eslutsky/dev/certs/server.crt,/home/eslutsky/dev/certs/server.key;/var/lib/microshift/certs/
   kube-apiserver-external-signer/kube-external-serving/server.crt,/var/lib/microshift/certs/kube-
   apiserver-external-signer/kube-external-serving/server.key;/var/lib/microshift/certs/kube-
   apiserver-localhost-signer/kube-apiserver-localhost-
   serving/server.crt,/var/lib/microshift/certs/kube-apiserver-localhost-signer/kube-apiserver-
   localhost-serving/server.key;/var/lib/microshift/certs/kube-apiserver-service-network-
   signer/kube-apiserver-service-network-serving/server.crt,/var/lib/microshift/certs/kube-
   apiserver-service-network-signer/kube-apiserver-service-network-serving/server.key
   ```

2. From the client, ensure that the **kube-apiserver** is serving the correct certificate by running the following command:

   ```
   $ openssl s_client -connect <SNI_ADDRESS>:6443 -showcerts | openssl x509 -text -noout -
   in - | grep -C 1 "Alternative\|CN"
   ```

3.5. CLEANING UP AND RECREATING THE CUSTOM CERTIFICATES

To stop the MicroShift services, clean up the custom certificates and recreate the custom certificates, use the following steps.

Procedure

1. Stop the MicroShift services and clean up the custom certificates by running the following command:

   ```
   $ sudo microshift-cleanup-data --cert
   ```

   Example output

   ```
   Stopping MicroShift services
   Removing MicroShift certificates
   MicroShift service was stopped
   Cleanup succeeded
   ```

2. Restart the MicroShift services to recreate the custom certificates by running the following command:

   ```
   $ sudo systemctl start microshift
   ```

3.6. ADDITIONAL RESOURCES

- OpenShift: Add an API server named certificate
- RHEL: Creating and managing TLS keys and certificates
- The system-wide truststore
- OpenShift CLI Reference: oc login

CHAPTER 4. CHECKING GREENBOOT SCRIPTS STATUS

To deploy applications or make other changes through the MicroShift API with tools other than **kustomize** manifests, you must wait until the Greenboot health checks have finished. This ensures that your changes are not lost if Greenboot rolls your **rpm-ostree** system back to an earlier state.

The **greenboot-healthcheck** service runs one time and then exits. After Greenboot has exited and the system is in a healthy state, you can proceed with configuration changes and deployments.

4.1. CHECKING THE STATUS OF GREENBOOT HEALTH CHECKS

Check the status of Greenboot health checks before making changes to the system or during troubleshooting. You can use any of the following commands to help you ensure that Greenboot scripts have finished running.

Procedure

- To see a report of health check status, use the following command:

  ```
  $ systemctl show --property=SubState --value greenboot-healthcheck.service
  ```

  - An output of **start** means that Greenboot checks are still running.
  - An output of **exited** means that checks have passed and Greenboot has exited. Greenboot runs the scripts in the **green.d** directory when the system is a healthy state.
  - An output of **failed** means that checks have not passed. Greenboot runs the scripts in **red.d** directory when the system is in this state and might restart the system.

- To see a report showing the numerical exit code of the service where **0** means success and non-zero values mean a failure occurred, use the following command:

  ```
  $ systemctl show --property=ExecMainStatus --value greenboot-healthcheck.service
  ```

- To see a report showing a message about boot status, such as **Boot Status is GREEN - Health Check SUCCESS**, use the following command:

  ```
  $ cat /run/motd.d/boot-status
  ```

CHAPTER 5. CONFIGURING AUDIT LOGGING POLICIES

You can control audit log file rotation and retention by using configuration values.

5.1. ABOUT SETTING LIMITS ON AUDIT LOG FILES

Controlling the rotation and retention of the audit log file by using configuration values helps keep the limited storage capacities of far-edge devices from being exceeded. On such devices, logging data accumulation can limit host system or cluster workloads, potentially causing the device stop working. Setting audit log policies can help ensure that critical processing space is continually available.

The values you set to limit audit logs enable you to enforce the size, number, and age limits of audit log backups. Field values are processed independently of one another and without prioritization.

You can set fields in combination to define a maximum storage limit for retained logs. For example:

- Set both **maxFileSize** and **maxFiles** to create a log storage upper limit.
- Set a **maxFileAge** value to automatically delete files older than the timestamp in the file name, regardless of the **maxFiles** value.

5.1.1. Default audit log values

MicroShift includes the following default audit log rotation values:

Table 5.1. MicroShift default audit log values

Audit log parameter	Default setting	Definition
maxFileAge:	**0**	How long log files are retained before automatic deletion. The default value means that a log file is never deleted based on age. This value can be configured.
maxFiles:	**10**	The total number of log files retained. By default, MicroShift retains 10 log files. The oldest is deleted when an excess file is created. This value can be configured.
maxFileSize:	**200**	By default, when the **audit.log** file reaches the **maxFileSize** limit, the **audit.log** file is rotated and MicroShift begins writing to a new **audit.log** file. This value is in megabytes and can be configured.
profile:	**Default**	The **Default** profile setting only logs metadata for read and write requests; request bodies are not logged except for OAuth access token requests. If you do not specify this field, the **Default** profile is used.

The maximum default storage usage for audit log retention is 2000Mb if there are 10 or fewer files.

If you do not specify a value for a field, the default value is used. If you remove a previously set field value, the default value is restored after the next MicroShift service restart.

5.2. ABOUT AUDIT LOG POLICY PROFILES

Audit log profiles define how to log requests that come to the OpenShift API server and the Kubernetes API server.

MicroShift supports the following predefined audit policy profiles:

Profile	Description
Default	Logs only metadata for read and write requests; does not log request bodies except for OAuth access token requests. This is the default policy.
WriteRequestBodies	In addition to logging metadata for all requests, logs request bodies for every write request to the API servers (**create**, **update**, **patch**, **delete**, **deletecollection**). This profile has more resource overhead than the **Default** profile. [1]
AllRequestBodies	In addition to logging metadata for all requests, logs request bodies for every read and write request to the API servers (**get**, **list**, **create**, **update**, **patch**). This profile has the most resource overhead.[1]
None	No requests are logged, including OAuth access token requests and OAuth authorize token requests. WARNING Do not disable audit logging by using the **None** profile unless you are fully aware of the risks of not logging data that can be beneficial when troubleshooting issues. If you disable audit logging and a support situation arises, you might need to enable audit logging and reproduce the issue to troubleshoot properly.

1. Sensitive resources, such as **Secret**, **Route**, and **OAuthClient** objects, are only logged at the metadata level.

By default, MicroShift uses the **Default** audit log profile. You can use another audit policy profile that also logs request bodies, but be aware of the increased resource usage such as CPU, memory, and I/O.

5.3. CONFIGURING AUDIT LOG VALUES

You can configure audit log settings by using the MicroShift service configuration file.

Procedure

1. Make a copy of the provided **config.yaml.default** file in the **/etc/microshift/** directory, renaming

it **config.yaml**. Keep the new MicroShift **config.yaml** you create in the **/etc/microshift/** directory. The new **config.yaml** is read whenever the MicroShift service starts. After you create it, the **config.yaml** file takes precedence over built-in settings.

2. Replace the default values in the **auditLog** section of the YAML with your desired valid values.

Example default auditLog configuration

```
apiServer:
# ....
  auditLog:
    maxFileAge: 7 (1)
    maxFileSize: 200 (2)
    maxFiles: 1 (3)
    profile: Default (4)
# ....
```

1. Specifies the maximum time in days that log files are kept. Files older than this limit are deleted. In this example, after a log file is more than 7 days old, it is deleted. The files are deleted regardless of whether or not the live log has reached the maximum file size specified in the **maxFileSize** field. File age is determined by the timestamp written in the name of the rotated log file, for example, **audit-2024-05-16T17-03-59.994.log**. When the value is **0**, the limit is disabled.
2. The maximum audit log file size in megabytes. In this example, the file is rotated as soon as the live log reaches the 200 MB limit. When the value is set to **0**, the limit is disabled.
3. The maximum number of rotated audit log files retained. After the limit is reached, the log files are deleted in order from oldest to newest. In this example, the value **1** results in only 1 file of size **maxFileSize** being retained in addition to the current active log. When the value is set to **0**, the limit is disabled.
4. Logs only metadata for read and write requests; does not log request bodies except for OAuth access token requests. If you do not specify this field, the **Default** profile is used.

3. Optional: To specify a new directory for logs, you can stop MicroShift, and then move the **/var/log/kube-apiserver** directory to your desired location:

   a. Stop MicroShift by running the following command:

   ```
   $ sudo systemctl stop microshift
   ```

   b. Move the **/var/log/kube-apiserver** directory to your desired location by running the following command:

   ```
   $ sudo mv /var/log/kube-apiserver <~/kube-apiserver> (1)
   ```

   1. Replace *<~/kube-apiserver>* with the path to the directory that you want to use.

   c. If you specified a new directory for logs, create a symlink to your custom directory at **/var/log/kube-apiserver** by running the following command:

   ```
   $ sudo ln -s <~/kube-apiserver> /var/log/kube-apiserver (1)
   ```

1. Replace *<~/kube-apiserver>* with the path to the directory that you want to use. This enables the collection of logs in sos reports.

4. If you are configuring audit log policies on a running instance, restart MicroShift by entering the following command:

```
$ sudo systemctl restart microsohift
```

5.4. TROUBLESHOOTING AUDIT LOG CONFIGURATION

Use the following steps to troubleshoot custom audit log settings and file locations.

Procedure

- Check the current values that are configured by running the following command:

```
$ sudo microshift show-config --mode effective
```

Example output

```
auditLog:
    maxFileSize: 200
    maxFiles: 1
    maxFileAge: 7
    profile: AllRequestBodies
```

- Check the **audit.log** file permissions by running the following command:

```
$ sudo ls -ltrh /var/log/kube-apiserver/audit.log
```

Example output

```
-rw-------. 1 root root 46M Mar 12 09:52 /var/log/kube-apiserver/audit.log
```

- List the contents of the current log directory by running the following command:

```
$ sudo ls -ltrh /var/log/kube-apiserver/
```

Example output

```
total 6.0M
-rw-------. 1 root root 2.0M Mar 12 10:56 audit-2024-03-12T14-56-16.267.log
-rw-------. 1 root root 2.0M Mar 12 10:56 audit-2024-03-12T14-56-49.444.log
-rw-------. 1 root root 962K Mar 12 10:57 audit.log
```