

# IBM

# Logging and Log Forwarding to Splunk in OpenShift Environment

IBM Hybrid Cloud Integration Test team  
Philip Chan ([chanphil@us.ibm.com](mailto:chanphil@us.ibm.com))

March 31, 2023

## Introduction

---

IBM Hybrid Cloud Integration Test (HCIT) is an effort within IBM Systems to build and maintain a customer like software environment running multiple application stacks on the LinuxONE server platform. The objective is to find defects across IBM and open-source software as well as LinuxONE and IBM Storage firmware, which can only be found when running a complex software configuration for a long stretch of time. This now includes software stacks running on top of RedHat OpenShift Container Platform (OCP) and some potential Independent Software Vendor products (ISVs).

## Purpose

---

Red Hat OpenShift Container Platform (OCP) provides a centralized logging solution that offers high availability, scalability and security. The default logging subsystem for OCP consists of three main components – Vector, Elasticsearch and Kibana. The logging subsystem aggregates these logs throughout the cluster logs and stores them in the default log store. The logs are separated into the following types – application, infrastructure(infra) and audit logs.

Vector is a log collector that resides on each OCP node collecting logs from various sources, such as containers, nodes, and applications. The logs are sent to Elasticsearch which stores and indexes the logs. Kibana is a Web UI, accessed by the browser to view the Elasticsearch data.

More details on the OCP default logging subsystem and terms can be found here:  
<https://docs.openshift.com/container-platform/4.11/logging/cluster-logging.html#openshift-logging-common-terms> cluster-logging

In addition to the default centralized logging subsystem, where logs are stored to the internal log store (Elasticsearch), logs can also be sent to external third-party log aggregators by use of the OpenShift Container Platform Cluster Log Forwarder API. For our test, we forwarded the logs to the Splunk Enterprise server using HTTP and HTTPS. This is achieved through the Cluster Log Forwarder API, which enables to send container, infrastructure and audit logs to the Splunk instance endpoint that resides outside the immediate OCP cluster.

More details on the OCP Log Forwarder can be found here:

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.11/html/logging/cluster-logging-external](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.11/html/logging/cluster-logging-external)

To configure OpenShift Container Platform Logging and Log Forwarding, we created and customized instances in the **ClusterLogging** and **ClusterLogForwarder** custom resources. We performed our testing using OCP version 4.11. We installed and configured Logging 5.6.3. As of Logging 5.6, Fluentd is deprecated and is planned to be removed in a future release. We used Vector as our logging collector for both centralized logging subsystem and log forwarding tests. Note, Vector is not enabled by default.

The purpose of this document is to outline and provide information about the following:

- Install and Configure default logging subsystem
- Install and Configure Splunk Enterprise Server
- Configure Log Forwarding to Splunk Enterprise Server
  - HEC Collector using HTTP
  - HEC Collector using HTTPS

## Disconnected OpenShift Cluster Testing Environment Overview

---

### Requirements:

- OCP Cluster running on System z
  - OpenShift Elasticsearch Operator
  - Red Hat OpenShift Logging Operator
- OCP Cluster running on x86
  - Splunk Operator

### Test Environment:

- OCP version 4.11 on System z
- OCP version 4.10 on x86
- zVM Hypervisor
- RedHat Enterprise Linux 8
- OCP CLI
- Splunk Enterprise Server (standalone instance)

## Test Experience

To install the logging subsystem for OCP, you will need to deploy both the OpenShift Elasticsearch and Red Hat OpenShift Logging Operators. The install and configuration details can be found here:

<https://docs.openshift.com/container-platform/4.11/logging/cluster-logging-deploying.html>

We installed version 5.6.3 of both Operators using the OCP web console:

Name	Managed Namespaces	Status	Last updated	Provided APIs
 <b>Red Hat OpenShift Logging</b> 5.6.3 provided by Red Hat, Inc.	NS openshift-logging	 Succeeded Up to date	Mar 8, 2023, 9:36 AM	<a href="#">Cluster Log Forwarder</a> <a href="#">Cluster Logging</a>
 <b>OpenShift Elasticsearch Operator</b> 5.6.3 provided by Red Hat	All Namespaces	 Succeeded Up to date	Mar 8, 2023, 9:30 AM	<a href="#">Elasticsearch</a> <a href="#">Kibana</a>

Create the OpenShift Logging Instance under the ClusterLogging Custom Resource. This is the YAML that we provided:

```

apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogging"
metadata:
  name: "instance"
  namespace: "openshift-logging"
spec:
  managementState: "Managed"
  logStore:
    type: "elasticsearch"
    retentionPolicy:
      application:
        maxAge: 1d
      infra:
        maxAge: 7d
      audit:
        maxAge: 7d
    elasticsearch:
      nodeCount: 3
    storage:
      storageClassName: "xiv-block-storageclass"
      size: 200G
    resources:
      limits:

```

```

        memory: "16Gi"
      requests:
        memory: "16Gi"
    proxy:
      resources:
        limits:
          memory: 256Mi
        requests:
          memory: 256Mi
      redundancyPolicy: "SingleRedundancy"
    visualization:
      type: "kibana"
      kibana:
        replicas: 1
    collection:
      logs:
        type: application
      type: vector

```

*Note: the storage class "xiv-block-storageclass" already existed*

Verify the openshift-logging is configured correctly by checking on the status of several pods which include OpenShift Logging, Elasticsearch, Kibana and Vector:

```

$ oc -n openshift-logging get pods -w
NAME                                                    READY   STATUS    RESTARTS   AGE
cluster-logging-operator-67456bd999-45nqx             1/1     Running   0           10d
collector-2slng                                         2/2     Running   0           10d
collector-4zwmg                                         2/2     Running   0           10d
collector-5fhf2                                         2/2     Running   0           10d
collector-6ptwv                                         2/2     Running   0           10d
collector-985bw                                         2/2     Running   0           10d
collector-c2dmb                                         2/2     Running   0           10d
collector-ds827                                         2/2     Running   0           10d
collector-j278d                                         2/2     Running   0           10d
collector-ztmw2                                         2/2     Running   0           10d
elasticsearch-cdm-nivp2rof-1-7c875556dc-8rgml         2/2     Running   0           10d
elasticsearch-cdm-nivp2rof-2-764dc9888b-8vf68         2/2     Running   0           10d
elasticsearch-cdm-nivp2rof-3-644f5fdfb5-8mbg6         2/2     Running   0           10d
elasticsearch-im-app-27986595-lkp4d                   0/1     Completed 0           12m
elasticsearch-im-audit-27986595-jmlp4                 0/1     Completed 0           12m
elasticsearch-im-infra-27986595-ksqkl                 0/1     Completed 0           12m
kibana-67d4b46ff8-mqfvq                               2/2     Running   0           10d

```

The Kibana web console can now be used for visualizing the collected log data. By using Kibana, you can search, filter and analyze the logs. To access the Kibana web console, while logged into the OCP console, select the Applications Launcher and select Logging. Before we can view any log data, an index pattern which defines the Elasticsearch indices must be created first.

To create an index pattern, go to Management -> Index Patterns -> Create index pattern.

For Step 1, entered the example "infra-000001" that was provided.

## Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

Include system indices

### Step 1 of 2: Define index pattern

#### Index pattern

infra-000001

You can use a \* as a wildcard in your index pattern.  
You can't use spaces or the characters \, /, ?, ", <, >, |.

> Next step

✓ **Success!** Your index pattern matches **1 index**.

infra-000001

Rows per page: 10 ▾

For Step 2, selected @timestamp for the Time Filter field name:

## Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

Include system indices

### Step 2 of 2: Configure settings

You've defined **infra-000001** as your index pattern. Now you can specify some settings before we create it.

#### Time Filter field name

Refresh

@timestamp ▾

The Time Filter will use this field to filter your data by time.  
You can choose not to have a time field, but you will not be able to narrow down your data by a time range.

> Show advanced options

< Back

Create index pattern

Select Create index pattern which is successfully created with these fields:

Create index pattern

★ infra-000001

★ infra-000001

Time Filter field name: @timestamp

This page lists every field in the **infra-000001** index and the field's associated core type as recorded by Elasticsearch. To change a field type, use the Elasticsearch [Mapping API](#).

Fields (330)    Scripted fields (0)    Source filters (0)

Filter:  All field types

Name	Type	Format	Searchable	Aggregatable	Excluded
@timestamp	date		●	●	
@timestamp.raw	string		●	●	
CPU_USAGE_NSEC	string		●		
CPU_USAGE_NSEC.raw	string			●	
INVOCATION_ID	string		●		
INVOCATION_ID.raw	string			●	
NM_DEVICE	string		●		
NM_DEVICE.raw	string			●	
NM_LOG_DOMAINS	string		●		
NM_LOG_DOMAINS.raw	string			●	

Rows per page: 10

< 1 2 3 4 5 ... 33 >

Still from the Kibana console, go to Discover to view the “infra-000001” index created above:

For more details on Kibana:

<https://docs.openshift.com/container-platform/4.11/logging/cluster-logging-visualizer.html>

Before testing the Log Forwarding, we need to install and setup a Splunk Operator and create a Splunk Enterprise Standalone Server instance. The Splunk Operator is available on OCP 4.10 running on x86. From the OCP web console, selected and installed Splunk Operator v2.2.0:

splunk: **Splunk Operator** All Namespaces ✔ Succeeded Up to date 🕒 Mar 19, 2023, 8:04 AM Cluster Manager  
Indexer Cluster  
License Manager  
Monitoring Console v4  
[View 4 more...](#)

2.2.0 provided by Splunk Inc.

To create a Splunk Enterprise instance, go to the Custom Resource “Standalone” -> Instances -> Create Standalone. This is the YAML file I supplied:

```
apiVersion: enterprise.splunk.com/v4
kind: Standalone
metadata:
  name: example
  finalizers:
  - enterprise.splunk.com/delete-pvc
spec:
  etcVolumeStorageConfig:
    ephemeralStorage: true
  varVolumeStorageConfig:
    ephemeralStorage: true
```

The instance takes approximately a few minutes to start:

```
$ oc get pods -o wide --all-namespaces|grep splunk
default
1/1      Running          0           98s      172.16.6.11      splunk-example-standalone-0
worker3.xocp1.fpet.pokprv.stglabs.ibm.com    <none>      <none>
```

You can now access the Splunk Enterprise Web UI by using the URL of the service created by the instance creation.

<http://splunk-example-standalone-service-default.apps.xocp1.fpet.pokprv.stglabs.ibm.com>

Let’s review some key ports and routing information with regards to the Splunk Service that was used in our testing:

## Service routing

### Hostname

splunk-example-standalone-service.default.svc.cluster.local

Accessible within the cluster only

### Service address

Type	Location
Cluster IP	172.30.60.30
Accessible within the cluster only	

### Service port mapping

Name	Port	Protocol	Pod port or name
http-splunkweb	 8000	TCP	 8000
http-hec	 8088	TCP	 8088
https-splunkd	 8089	TCP	 8089
tcp-s2s	 9997	TCP	 9997

The Splunk Web UI or http-splunkweb access is mapped to port 8000.

The HTTP Event Collector or http-hec is mapped to port 8088.

The default Splunk management service uses port 8089.

Logon to the Splunk Web UI using a default username of 'admin'. The password can be retrieved using the following command:

```
$ oc get secret splunk-example-standalone-secret-v1 -o  
jsonpath='{.data.password}' | base64 --decode
```

### **Preparing Splunk Server to receive the forwarded logs from OCP over HTTP**

The following is the setup we performed to use an insecure connection between the Vector log collectors in the OpenShift cluster and Splunk Enterprise Server.

Create a new index called 'openshift' from the Splunk Web UI. Go to Settings -> Indexes -> select New Index. Provide the index name 'openshift' and use the default values.

## New Index



### General Settings

Index Name	<input type="text" value="openshift"/>
	<small>Set index name (e.g., INDEX_NAME). Search using index=INDEX_NAME.</small>
Index Data Type	<input checked="" type="radio"/> Events <input type="radio"/> Metrics
	<small>The type of data to store (event-based or metrics).</small>
Home Path	<input type="text" value="optional"/>
	<small>Hot/warm db path. Leave blank for default (\$SPLUNK_DB/INDEX_NAME/db).</small>
Cold Path	<input type="text" value="optional"/>
	<small>Cold db path. Leave blank for default (\$SPLUNK_DB/INDEX_NAME/coldb).</small>
Thawed Path	<input type="text" value="optional"/>
	<small>Thawed/resurrected db path. Leave blank for default (\$SPLUNK_DB/INDEX_NAME/thaweddb).</small>
Data Integrity Check	<input type="radio"/> Enable <input checked="" type="radio"/> Disable
	<small>Enable this if you want Splunk to compute hashes on every slice of your data for the purpose of data integrity.</small>
Max Size of Entire Index	<input type="text" value="500"/> <input type="text" value="GB"/>
	<small>Maximum target size of entire index.</small>
Max Size of Hot/Warm/Cold Bucket	<input type="text" value="auto"/> <input type="text" value="GB"/>
	<small>Maximum target size of buckets. Enter 'auto_high_volume' for high-volume indexes.</small>
Frozen Path	<input type="text" value="optional"/>
	<small>Frozen bucket archive path. Set this if you want Splunk to automatically archive frozen buckets.</small>
App	<input type="text" value="Search &amp; Reporting"/>

### Storage Optimization

Tsidx Retention Policy	<input type="radio"/> Enable Reduction <input checked="" type="radio"/> Disable Reduction
	<small>Warning: Do not enable reduction without understanding the full implications. It is extremely difficult to rebuild reduced buckets. <a href="#">Learn More</a></small>
Reduce tsidx files older than	<input type="text" value=""/> <input type="text" value="Days"/>
	<small>Age is determined by the latest event in a bucket.</small>

Save

Cancel

View the HTTP Event Collector token. Go to Settings -> Data Inputs and select HTTP Event Collector. Record the 'Token Value' for later use when configuring the Log Forwarding on OCP, it is the HEC token that allows the LogForwarding to communicate with Splunk.

Name ^	Actions	Token Value ⇅	Source Type ⇅	Index ⇅	Status ⇅
splunk_hec_token	Edit Disable Delete	1697061D-9419-0C2A-0477-A1CB1E2721C		Default	Enabled

Modify the hec\_token. Select the 'splunk\_hec\_token'. In Edit, select 'opensearch' under Available indexes so that it is moved to Selected indexes:

### Edit Token: splunk\_hec\_token ✕

Description

Source

Set Source Type

Source Type

Select Allowed Indexes (optional)

Available indexes add all »

- history
- main
- opensearch
- summary

Selected indexes « remove all

- opensearch

Select indexes that clients will be able to select from.

Default Index

Output Group (optional)

Enable indexer acknowledgement

While in the HTTP Event Collector setting, select "Global Settings" in the upper right corner. Deselect "Enable SSL", so that verification of SSL certificates is disabled. We will test HEC with HTTP enabled first.

### Edit Global Settings ✕

All Tokens	<div style="display: flex; justify-content: space-between; border: 1px solid #ccc; padding: 2px;"> <span style="background-color: #d3d3d3; padding: 2px 10px;">Enabled</span> <span style="background-color: #f0f0f0; padding: 2px 10px;">Disabled</span> </div>
Default Source Type	<div style="border: 1px solid #ccc; padding: 2px; display: flex; justify-content: space-between;"> <span>Select Source Type</span> <span>▼</span> </div>
Default Index	<div style="border: 1px solid #ccc; padding: 2px; display: flex; justify-content: space-between;"> <span>Default</span> <span>▼</span> </div>
Default Output Group	<div style="border: 1px solid #ccc; padding: 2px; display: flex; justify-content: space-between;"> <span>None</span> <span>▼</span> </div>
Use Deployment Server	<input type="checkbox"/>
Enable SSL	<input checked="" type="checkbox"/>
HTTP Port Number ?	<input style="width: 90%;" type="text" value="8088"/>

Cancel
Save

The Global Settings change require the Splunk Server to be restarted. Go to **Settings -> Server controls** and select Restart Splunk. Restarting the server takes a few minutes.

As noted earlier above, the service routing for the Splunk service shows that the Cluster IP is only accessible within the cluster only. We added an external private IP address for the service:

```
$ oc patch svc splunk-example-standalone-service -p '{"spec":{"externalIPs":["10.xxx.xxx.xxx"]}}'
service/splunk-example-standalone-service patched
```

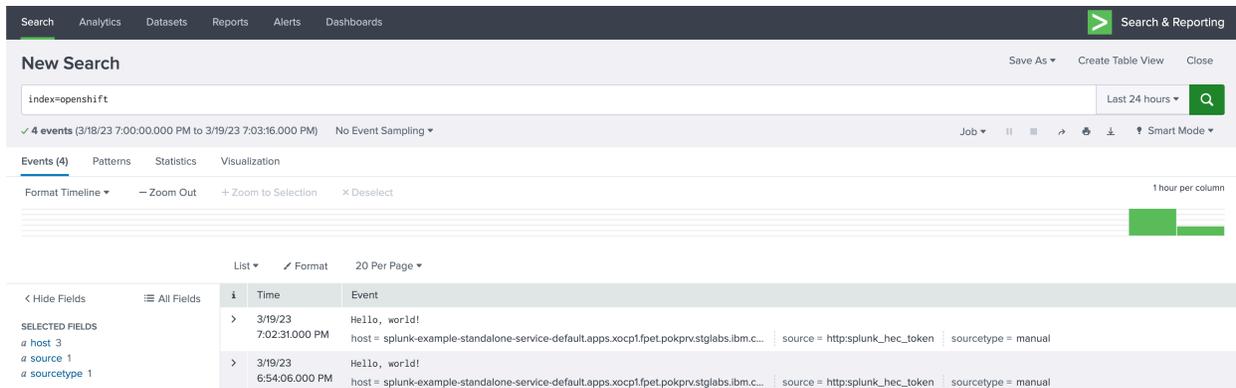
We can now run curl commands to verify the HTTP Event Collector using port 8088.

```
$ curl "http://splunk-example-standalone-service-
default.apps.xocpl.fpet.pokprv.stglabs.ibm.com:8088/services/collector" -H
"Authorization: Splunk 1697061D-9419-0C2A-0477-A1CB11E2721C" -d '{"event": "Hello,
world!", "sourcetype": "manual"}'
```

The output should return this:

```
{"text": "Success", "code": 0}
```

We can now view the event added to the index. Go back to the Splunk Web UI, select **Search and Reporting** -> under Search, enter "index=openshift":



The Splunk Server is now prepared for Log Forwarding.

### Prepare OpenShift for Log Forwarding

From the OCP Console, create a Secret object named “openshift-logforwarding-splunk” under **Workloads -> Secrets -> Select Create – Key/value** secret with the following values – supplying the recorded HEC token value from above:

Project: openshift-logging ▾

## Create key/value secret

Key/value secrets let you inject sensitive data into your application as files or environment variables.

**Secret name \***

openshift-logforwarding-splunk

Unique name of the new secret.

**Key \***

hecToken

**Value**

Browse...

Drag and drop file with your value here or browse to upload it.

1697061D-9419-0C2A-0477-A1CB11E2721C

+ Add key/value

Create Cancel

To create a Log Forwarding instance, go to the Custom Resource “ClusterLogForwarder” -> Instances -> Create ClusterLogForwarder. This is the YAML file I supplied:

```

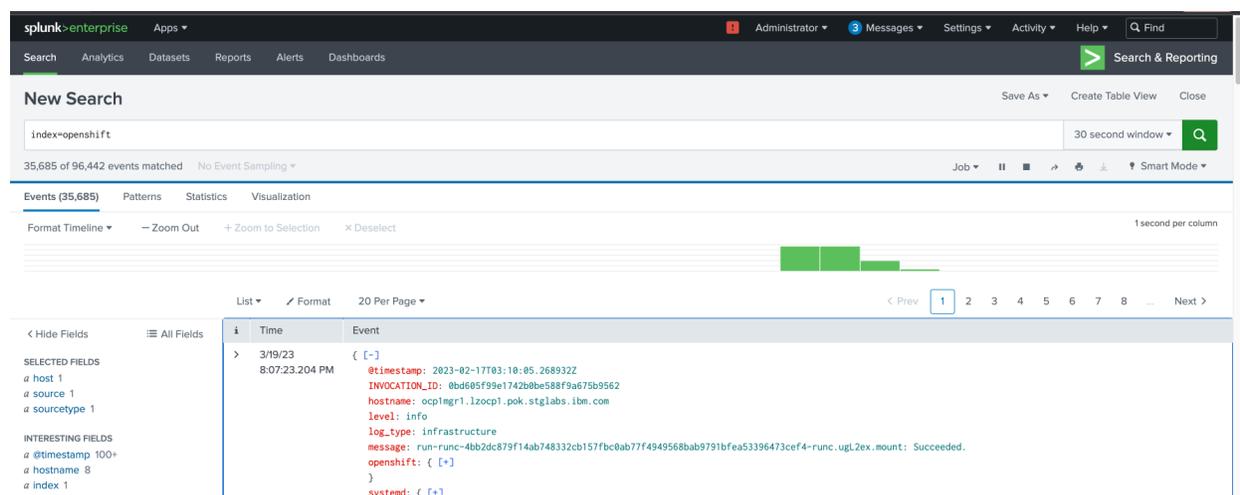
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging
spec:
  outputs:
    - name: remotesplunk
      secret:
        name: openshift-logforwarding-splunk
      splunk: {}
      type: splunk
      url: 'http://my-nginx-proxy-logforwarding-splunk:8088'
  pipelines:
    - inputRefs:
      - application
      - infrastructure
      name: application-infra-logs
      outputRefs:
      - remotesplunk

```

In the example above, there is a single output defined forwarding to the external Splunk Standalone instance. A pipeline is defined for the application and infrastructure log types which use this newly defined output called remotesplunk.

A very important note about the URL we supplied. Our two OCP clusters do not reside on the same network. The Splunk Server is deployed on a OCP cluster that is on our private 10.dot network. The LogForwarding is deployed on a OCP cluster that is on our public 9.dot network. If we were to use the URL <http://splunk-example-standalone-service-default.apps.xocp1.fpet.pokprv.stglabs.ibm.com>, the 10.dot targeted packets would not be reached from our public OCP cluster. We configured a NGINX server **my-nginx-proxy-logforwarding-splunk** that forwards the 10.dot traffic at port 8088 to the private OCP cluster.

The collector pods will restart once the Log Forwarding Instance is successfully created. From the Splunk Web UI, search “index=openshift” and you will see events populate the index for example:



The above completes our testing of OCP log forwarding to Splunk using an insecure connection.

## Preparing Splunk Server to receive the forwarded logs from OCP over HTTPS

The following is the setup we performed to secure the connection between our OpenShift cluster and Splunk Enterprise Server by configuring certificates in our Splunk deployment.

Splunk certificate management can be very confusing, and even though they provide detailed documentation, there is always some dependency or ‘gotcha’ that you encounter when trying to communicate with a secure Splunk Server. I’ll provide some guidance on some of the issues we encountered in our test.

Splunk comes bundled with an internal root certificate authority (CA), which is located at \$ SPLUNK\_HOME/etc/auth/ca.pem.default. Even though this can be used to secure Splunk, you probably should not use this in your production environment to secure communications. It would be best to enable TLS on all your Splunk communications with a private or third party signed CA. There are four main certificates to be aware of when enabling TLS on Splunk:

1. A web certificate is used on port 8000 when you access the Splunk Web UI over port 8000.
2. A Splunkd server certificate is used for all Splunk to Splunk communication on port 8089.
3. An index certificate is used for incoming data on port 9997.
4. The HTTP endpoint collector (HEC) certificate is used here for incoming data on port 8088.

Our focus will be enabling the HEC port 8088 to use HTTPS. Currently, from the previous section, we had disabled SSL in the Global Settings under Data Inputs. Go back to the Global Settings under Settings -> Data Inputs -> Select Global Settings – Check “Enable SSL” and select “openshift” index as the Default Index and Save:

**Edit Global Settings** ✕

All Tokens  Enabled  Disabled

Default Source Type

Default Index

Default Output Group

Use Deployment Server

Enable SSL

HTTP Port Number

This requires the Splunk Server to restart. Go to **Settings -> Server controls** and select Restart Splunk. Restarting the server takes a few minutes.

## Splunk Certificate Setup

These steps need to be performed directly on the Splunk Server instance that we have running on the x86 OCP cluster. Connect to the pod using the following command:

```
$ oc rsh splunk-example-standalone-0
```

Note: The environment variable `$SPLUNK_HOME` is `/opt/splunk` when inside the container.

Details regarding the creation of the root certificate authority and server certificate:

<https://docs.splunk.com/Documentation/Splunk/9.0.3/Security/Howtoself-signcertificates>

We use `openssl` to create the certificates inside the Splunk container. As we described above in our network traffic workflow from our OCP log forwarding cluster to Splunk server, this goes through a number of different machines with various hostnames. Due to that fact that each machine has a different hostname, we need to create Subject Alternative Name (SAN) certificates in order to have multiple DNS names that the certificate can protect. If you do not use SAN certificates, you will encounter an SSL error where the certificate subject name does not match the target host name. This issue would still occur even if you were to define a Common Name (CN) for a wildcard certificate. The host names of these intermediate machines must also be included (e.g. nginx proxy server, load balancer).

Using a wildcard SSL certificate where `CN=*.ibm.com` was not sufficient. This would protect `a.ibm.com`, `b.ibm.com`, `c.ibm.com` and so on. But our hostnames have second, third and other sublevel domains where a wildcard will not work. This is why we created sublevel domains that are added in the SAN. We created our own custom SSL configuration file with the required parameters - `/opt/splunk/openssl/openssl-san.cnf`

```
[req]
default_bits          = 2048
distinguished_name    = req_distinguished_name
req_extensions        = req_ext

[req_distinguished_name]
countryName           = Country Name (2 letter code)
stateOrProvinceName  = State or Province Name (full name)
localityName          = Locality Name (eg, city)
organizationName      = Organization Name (eg, company)
commonName            = Common Name (e.g. server FQDN or YOUR name)

# Optionally, specify some defaults.
countryName_default   = [Country]
stateOrProvinceName_default = [State]
localityName_default  = [City]
organizationName_default = [Organization]
organizationalUnitName_default = [Organization unit]
emailAddress_default  = [Email]

[req_ext]
subjectAltName = @alt_names

[alt_names]
IP.1 = <ip address of nginx proxy server>
```

IP.2 = <ip address of our load balancer>  
DNS.1 = <FQDN of our nginx proxy server>  
DNS.2 = <FQDN of our load balancer>

### Create the root CA certificate:

1. Prepare new directory for the certificates  
`mkdir $SPLUNK_HOME/etc/auth/mycerts`
2. Create a private key for the root CA certificate  
`$SPLUNK_HOME/bin/splunk cmd openssl genrsa -aes256 -out myCertAuthPrivateKey.key 2048`
3. Generate a Certificate Sign Request for SAN certificate using the custom conf file  
`$SPLUNK_HOME/bin/splunk cmd openssl req -new -key myCertAuthPrivateKey.key -out myCertAuthCertificate.csr -config /opt/splunk/openssl/openssl-san.cnf -extensions 'req_ext'`
4. Verify the SAN value in CSR under the “Requested Extensions”  
`$SPLUNK_HOME/bin/splunk cmd openssl req -noout -text -in myCertAuthCertificate.csr | grep -A 1 "Subject Alternative Name"`
5. Sign the CSR with private key created above using the custom conf file  
`$SPLUNK_HOME/bin/splunk cmd openssl x509 -req -in myCertAuthCertificate.csr -sha512 -signkey myCertAuthPrivateKey.key -CAcreateserial -out myCertAuthCertificate.pem -days 1095 -extensions req_ext -extfile /opt/splunk/openssl/openssl-san.cnf`
6. Verify the SAN value in signed CA certificate  
`$SPLUNK_HOME/bin/splunk cmd openssl x509 -text -noout -in myCertAuthCertificate.pem | grep -A 1 "Subject Alternative Name"`

### Create the server certificate and sign them with the root CA certificate

1. Create a private key for the server certificate  
`$SPLUNK_HOME/bin/splunk cmd openssl genrsa -aes256 -out myServerPrivateKey.key 2048`
2. Generate a Certificate Sign Request for SAN certificate using the custom conf file  
`$SPLUNK_HOME/bin/splunk cmd openssl req -new -key myServerPrivateKey.key -out myServerCertificate.csr -config /opt/splunk/openssl/openssl-san.cnf -extensions 'req_ext'`
3. Verify the SAN value in CSR under the “Requested Extensions”  
`$SPLUNK_HOME/bin/splunk cmd openssl req -noout -text -in myServerCertificate.csr | grep -A 1 "Subject Alternative Name"`
4. Sign the CSR with the private CA key and CA cert created in previous CA steps  
`$SPLUNK_HOME/bin/splunk cmd openssl x509 -req -in myServerCertificate.csr -SHA256 -CA myCertAuthCertificate.pem -CAkey`

```
myCertAuthPrivateKey.key -CAcreateserial -out myServerCertificate.pem -
days 1095 -extensions req_ext -extfile /opt/splunk/openssl/openssl-
san.cnf
```

#### 5. Verify the San value in signed server certificate

```
$(SPLUNK_HOME)/bin/splunk cmd openssl x509 -text -noout -in
myServerCertificate.pem | grep -A 1 "Subject Alternative Name"
```

The certificate authority certificate and server certificate are now created. We now need to prepare the certificates for Splunk. Splunk expects the certificates to be in a certain order and format. As stated in their documentation, a certificate or key alone does not work even if it is configured correctly. All certs and keys must be concatenated or chained so that the Splunk platform can use them.

We need to create two chained certificates:

#### 1. A new certificate that we call 'ca\_chain\_cert.pem' that consists of the following certificates:

- The server certificate file
- The server private key file
- The certificate authority certificate file

For Example:

```
$ cat myServerCertificate.pem myServerPrivateKey.key
myCertAuthCertificate.pem > ca_chain_cert.pem
```

#### 2. A new certificate that we call 'server\_chain\_with\_key.pem' that consists of the following certificates:

- The server certificate file
- The server private key file

For Example:

```
$ cat myServerCertificate.pem myServerPrivateKey.key >
server_chain_with_key.pem
```

Now that we've prepared the certificates for Splunk, we need to define them under the Splunk configuration files. Splunk has a unique approach with their config files, where many copies of the same configuration file are typically layered in directories. I had a difficult time determining which config file affected either the user, app, or system. There is also a hierarchy with configuration file precedence. Use this link to learn about what priority is given for a configuration file depending on its location:

[https://docs.splunk.com/Documentation/Splunk/8.2.6/Admin/Wheretofindtheconfigurationfile?s?\\_ga=2.194185198.290357988.1679316408-1675238498.1679316408&\\_gl=1\\*1onlbgh\\*\\_ga\\*MTY3NTIzODQ5OC4xNjc5MzE2NDA4\\*\\_ga\\_5EP M2P39FV\\*MTY3OTM0OTgxOS43LjAuMTY3OTM0OTgyNC41NS4wLjA](https://docs.splunk.com/Documentation/Splunk/8.2.6/Admin/Wheretofindtheconfigurationfile?s?_ga=2.194185198.290357988.1679316408-1675238498.1679316408&_gl=1*1onlbgh*_ga*MTY3NTIzODQ5OC4xNjc5MzE2NDA4*_ga_5EP M2P39FV*MTY3OTM0OTgxOS43LjAuMTY3OTM0OTgyNC41NS4wLjA)

For the purposes of our test regarding the certificates that were generated, we are going to modify the inputs.conf file. Edit the `$SPLUNK_HOME/system/local/inputs.conf` file and add the following under the `[http]` stanza:

```
enableSSL = 1
sslPassword = <keyfile passphrase>
serverCert = /opt/splunk/etc/auth/mycerts/server_chain_with_key.pem
caCertFile = /opt/splunk/etc/auth/mycerts/ca_chain_cert.pem
```

These are all the changes we need to make on the Splunk Server side, we can now restart the Splunk Server:

```
$SPLUNK_HOME/bin/splunk restart splunkd
```

Once Splunk Server is back up, you can verify if the `https://` protocol on the HEC port 8088 is working by running a curl command and supplying the `ca_chain_cert.pem` certificate file:

```
$ curl "https://ltickvmd.pok.stglabs.ibm.com:8088/services/collector" --cacert /mydirectory/ca_chain_cert.pem -H "Authorization: Splunk 1697061D-9419-0C2A-0477-A1CB11E2721C" -d '{"event": "Hello, world!", "sourcetype": "manual"}'
```

The output should return this:

```
{"text": "Success", "code": 0}
```

You may also go to the Splunk Web UI, select **Search and Reporting** -> under Search, enter “`index=openshift`” to view if a new event was added. As you can see it is posting this over the `https` protocol.

## OCP Log Forwarding Setup to use HTTPS

Now that SSL is enabled on our Splunk Server. We need to first create a new secret object that contains all the required certificates to establish a secure communication with Splunk.

We require the following certificates created earlier – the private key for the server certificate, the signed server certificate and the ‘`ca_chain_cert`’ chain cert that consists of both the server private key, server certificate the ca certificate. They are copied to a local machine where we can open a browser to OCP Console, so that the files can be uploaded:

For Example:

```
myServerPrivateKey.key
myServerCertificate.pem
```

ca\_chain\_cert.pem

Create the new secret named “openshift-logforwarding-splunk-https” that will consist of the HEC Token, certificate and key files. From the OCP Console, go to **Workloads -> Secrets -> Select Create – Key/value** secret with the following values(use add key/value for additional keys):

```
Secret name: openshift-logforwarding-splunk-https
Key: hecToken
Value: 1697061D-9419-0C2A-0477-A1CB11E2721C
Key: ca-bundle.crt
Value: ca_chain_cert.pem
Key: passphrase
Value: <keyfile passphrase>
Key: tls.key
Value: myServerPrivateKey.key
Key: tls.crt
Value: myServerCertificate.pem
```

**Note:** Key names for the certificates and key files must be called **ca-bundle.crt**, **tls.key**, and **tls.crt**. Any variation will cause a “certificate verify failed error”.

For more details regarding creating a secret with certificates and files:

<https://docs.openshift.com/container-platform/4.11/logging/cluster-logging-external.html#creating-a-secret>

The Log Forwarding instance can now be modified. Using the one created earlier, go to the Custom Resource “ClusterLogForwarder” -> Instances -> select “instance”. Under the YAML tab, supply the name of the new secret and change the url to use ‘https’.

For Example:

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging
spec:
  outputs:
    - name: remotesplunk
      secret:
        name: openshift-logforwarding-splunk-https
      splunk: {}
      type: splunk
      url: 'https://my-nginx-proxy-logforwarding-splunk:8088'
  pipelines:
    - inputRefs:
        - application
        - infrastructure
      name: application-infra-logs
      outputRefs:
```

- remotesplunk

The collector pods will restart once the Log Forwarding Instance is saved. From the Splunk Web UI, search “index=openshift” and you will see new events populate the index again.

## Suggestions

---

When first testing the LogForwarding instance, the collector pods would continually Terminate and try again to restart. This had nothing to do with the YAML file we were supplying, but a problem reaching the Splunk Server.

As of Logging 5.6, Vector Collector now replaces Fluentd as the default log collector. Splunk is now an available output option for log forwarding.

Logging 5.6 release is only available on OperatorHub for OpenShift 4.11 and 4.12.

Splunk certificate management expects the certificate files to follow a specific format under the configuration.

There are numerous Splunk configuration files and it is important to identify the ones you need to modify correctly. Otherwise, if you make a mistake, the Splunk Server fails to restart, and in our case when using the Splunk Operator, when it fails to restart, we cannot connect back into the container to undo any mistake. We then need to create a new Splunk standalone instance.