Lars Milland Follow
Jul 20, 2017

Secure architecture for Jaeger with Apache httpd reverse proxy on OpenShift

Introduction

The Jaeger project with its implementation of Open Tracing and Zipkin support of both collecting traces and allowing access to collected traces through its web-based UI is great. But if one wishes for means of securing access to either submitting traces to the Jaeger collector and/or web UI parts, the Jaeger codebase and its inherent functionality will not help you. This makes it difficult to deploy and use Jaeger in strictly governed environments where services such as Jaeger needs to be protected from misuse as well as access to the collected data needs to be protected.

Luckily there are other means of protecting access and use of Jaeger than through functionality the Jaeger system supplies it self. One way is to wrap the endpoints/ports of Jaeger that needs protecting and then add the security in the wrapping component, a typical side-car approach. When deploying Jaeger on either

Kubernetes or OpenShift as a docker container running in a pod context, the Kubernetes/OpenShift platform can let you run multiple containers on the same pod, allowing for unsecure endpoints/ports to be “localhost” scoped, and then other containers adding security letting their secure endpoints/ports to be exposed.

This article shows one such design and deployment configuration of Jaeger deployed in an OpenShift context with locked down endpoints using Apache httpd server as the side-car component adding security with both encryption of data under transport, authentication and authorization of web UI and collector endpoints of Jaeger.

Design

The design consists of the following elements:

- Prevention of access to unsecure Jaeger endpoints—achieved with removed of exposed ports and adding Apache httpd reverse proxy with security features added (the choice of Apache httpd server as the reverse proxy is based only on good experience and knowledge about the Apache httpd server. You can just as well use HAProxy or NGINX if preferred)
- Encryption of transport of data in communication with Jaeger—achieved with Apache httpd configuration setting up TLS over HTTPS, and with TLS over LDAPS towards LDAP directory, and using passthrough routes on OpenShift to ensure end-2-end encryption also when accessing Jaeger outside of OpenShift
- Authentication of access to Jaeger—achieved with Apache httpd server over HTTP protocol with Basic Authentication

with username and password verification forwarded to LDAP directory

- Authorization of access to Jaeger—achieved with Apache httpd server configured to lookup up authenticated users group membership in connected LDAP directory

A full production ready deployment and configuration for Jaeger would need persistent storage for its Cassandra database, provided in a failover resilient manor, but for this design we only look at the security parts of submitting and accessing traces with Jaeger, so please remember to add such features if you consider to roll your own Jaeger installation.

The design is based on the jaeger-all-in-one OpenShift deployment configuration from here:

https://github.com/jaegertracing/jaeger-openshift/blob/master/all-in-one/jaeger-all-in-one-template.yml

The easiest network protocols to work with in OpenShift are TCP/HTTP based ones, especially if one wants to expose access to services outside OpenShift, as they are fully supported with the router component based on HAProxy that comes out-of-the-box with OpenShift. So this design will only look at the two TCP and HTTP Jaeger endpoints, the one for the UI being the Jaeger Query service typically running on http://localhost:16686:

http://jaeger.readthedocs.io/en/latest/deployment/#query-service-ui

and the Jaeger Collector HTTP service typically running on http://localhost:14268:

http://jaeger.readthedocs.io/en/latest/deployment/#collectors

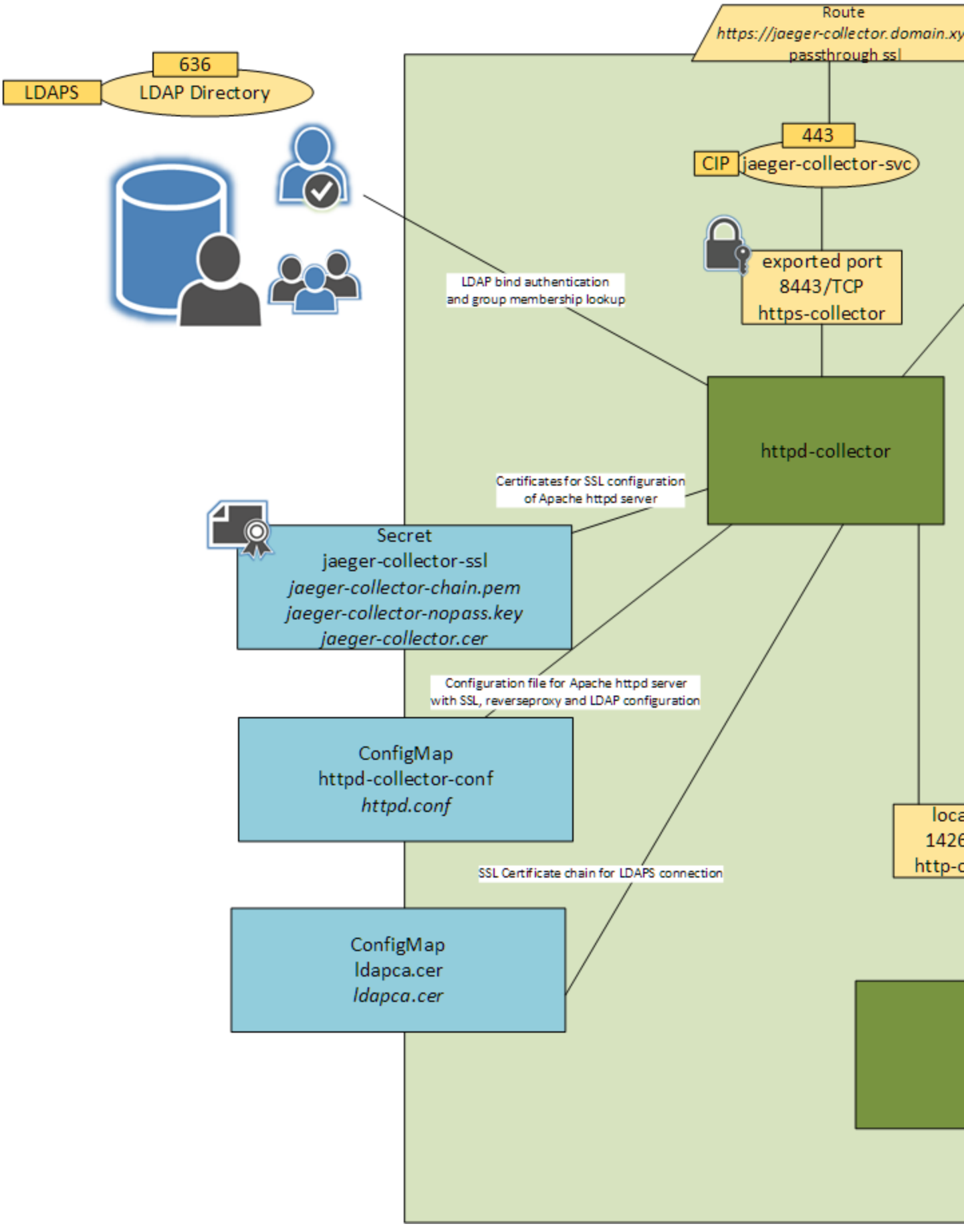
The design is based on the concept of adding extra Docker containers which can wrap the unsecure ports, adding security, and the letting the new secured side-car containers ports be exposed instead of the Jaeger components ones. In this design we will be using the Apache httpd server as the side-car:

https://httpd.apache.org/—in its 2.4 version

In the design an external LDAP directory service is used for authentication and authorization, where users that needs access to UI submits username and password over Basic Authentication http scheme and same applies to systems submitting traces to the collector. Connectivity to LDAP directory is done over LDAPS with TLS encryption to ensure that passwords are not send in clear text over the wire from the Apache httpd server. Same applies when users and systems submits the passwords to the Apache httpd server, that is also done with TLS encryption on https protocol.

This drawing shows all the elements needed to get the security in place. The graphical notation for the drawing is partly based on the layouts presented here:

https://github.com/raffaelespazzoli/kdl


Route
https://jaeger-collector.domain.xy
passthrough ssl
636
LDAPS
LDAP Directory
443
CIP
jaeger-collector-svc
exported port
8443/TCP
https-collector
LDAP bind authentication
and group membership lookup
httpd-collector
Certificates for SSL configuration
of Apache httpd server
Secret
jaeger-collector-ssl
jaeger-collector-chain.pem
jaeger-collector-nopass.key
jaeger-collector.cer
Configuration file for Apache httpd server
with SSL, reverseproxy and LDAP configuration
ConfigMap
httpd-collector-conf
httpd.conf
SSL Certificate chain for LDAPS connection
ConfigMap
ldapca.cer
ldapca.cer

There are quite a few endpoint types on the Jaeger collector component, both several UDP and TCP based ones. In this context, where Jaeger is deployed on Kubernetes/OpenShift, we will restrict the design only consider the TCP/HTTP based endpoints for the UI and Collector. Such TCP/HTTP based endpoints are both very easy to expose with routes on OpenShift through the out-of-the-box provided router functionality that OpenShift provides with HAProxy, but also easy to protect with both transport protocol encryption with TLS over HTTPS, authentication and authorization with a reverse proxy wrapping the unsecure Jaeger components endpoints.

In this configuration we are only considering functionality of how to secure endpoints relevant to this specific usecase, so the Jaeger configuration we are going base us on here is the so called jaeger-all-in-one:

https://github.com/jaegertracing/jaeger-openshift/blob/master/all-in-one/jaeger-all-in-one-template.yml

- which lack production ready configuration of the underlying Cassandra database, and also lacks persistent storage, so please don't forget to add such configurations in any real-life deployments.

So the design is to add extra Docker containers with Apache httpd servers onto the same OpenShift pod where Jaeger is running, in that way they can share the "localhost" network, and the Jaeger ports can be wrapped by the Apache httpd server protecting the access from external locations.

Another disclaimer would also be that the design is based on two individual Apache httpd server instances each running as Docker containers together with the jaeger-all-in-one container. It would

of course be possible to have both endpoints of the Jaeger component that needs securing to be secured by the same Apache httpd server, through different listeners or virtual host configurations in the Apache httpd servers httpd.conf file.

The Jaeger UI's endpoint would typically be on this location:

http://localhost:16686

and the Jaeger components HTTP Collector would typically be here:

http://localhost:14268

So the Apache httpd servers will act as reverse proxy to these two endpoint.

The Apache httpd Docker container that is used for the design would look like this:

Dockerfile part for Apache httpd in OpenShift

```
FROM httpd:2.4.25
COPY run-httpd.sh /opt/run-httpd/
COPY httpd.conf /usr/local/apache2/conf/
RUN chmod 755 /usr/local/apache2/conf/httpd.conf
RUN chmod 755 /opt/run-httpd/run-httpd.sh

# Docker main process requires a numeric, non-0 UID in order to avoid need of elevated privileges in OpenShift.
RUN chown -R 1001:0 /etc && \
chown -R 1001:0 /var && \
chown -R 1001:0 /usr
USER 1001
CMD [ "/opt/run-httpd/run-httpd.sh" ]
```

The start script used in this Apache httpd Docker container is extended to support locating configuration files through environment variables, such that those configuration files can be mounted in with OpenShift ConfigMap and Secret resources as volumes when the Docker container is used in the OpenShift context.

Apache httpd server start script with configuration file support

```
#!/bin/bash
if [ -z "${LOGS_DIR}" ]; then
LOGS_DIR="/var/log/httpd/"
fi
if [[ ! -e "${LOGS_DIR}" ]]; then
mkdir -p $LOGS_DIR
fi
export LOGS_DIR
if [ -z "${ROTATELOGS_ARGS}" ]; then
ROTATELOGS_ARGS="-t $LOGS_DIR/*log* 20M"
fi
if [ -z "${HTTPD_CONF}" ]; then
HTTPD_CONF="/usr/local/apache2/conf/httpd.conf"
fi
# Apache gets grumpy about PID files pre-existing
rm -f ${LOGS_DIR}/httpd.pid
rotatelogs $ROTATELOGS_ARGS &
if [ -z "${HTTPD_CONF}" ]; then
httpd -DFOREGROUND $*
else
httpd -f ${HTTPD_CONF} -DFOREGROUND $*
fi
```

So the httpd.conf file can then be selected by the value of the environment variable: HTTPD_CONF—which again is set when deploying the container on OpenShift.

With the elements of the OpenShift compatible Apache httpd container that can be injected with configuration we are ready to set this up on OpenShift.

There are two types of configuration in play here—one for the Apache part with httpd.conf files with the reverse proxy, TLS and LDAP configuration and another part for getting these elements into OpenShift and deploying the actual components on OpenShift.

Apache httpd Configuration

The Apache httd configuration is almost all done in the one httpd.conf file—here showing all the contents with the most important files high-lighted:

```
LoadModule authn_core_module
/usr/local/apache2/modules/mod_authn_core.so
LoadModule actions_module
/usr/local/apache2/modules/mod_actions.so
LoadModule mime_magic_module
/usr/local/apache2/modules/mod_mime_magic.so
LoadModule log_config_module
/usr/local/apache2/modules/mod_log_config.so
LoadModule unixd_module /usr/local/apache2/modules/mod_unixd.so
LoadModule authz_core_module
/usr/local/apache2/modules/mod_authz_core.so
LoadModule authz_host_module
/usr/local/apache2/modules/mod_authz_host.so
LoadModule authnz_ldap_module
/usr/local/apache2/modules/mod_authnz_ldap.so
LoadModule ldap_module /usr/local/apache2/modules/mod_ldap.so
LoadModule headers_module
/usr/local/apache2/modules/mod_headers.so
LoadModule slotmem_shm_module
/usr/local/apache2/modules/mod_slotmem_shm.so
LoadModule proxy_module /usr/local/apache2/modules/mod_proxy.so
LoadModule proxy_http_module
/usr/local/apache2/modules/mod_proxy_http.so
LoadModule proxy_balancer_module
/usr/local/apache2/modules/mod_proxy_balancer.so
LoadModule lbmethod_bytraffic_module
/usr/local/apache2/modules/mod_lbmethod_bytraffic.so
```

```
LoadModule setenvif_module
/usr/local/apache2/modules/mod_setenvif.so
LoadModule ssl_module /usr/local/apache2/modules/mod_ssl.so
LoadModule auth_basic_module
/usr/local/apache2/modules/mod_auth_basic.so
#opens up port for Apache httpd server to listen for incoming
traffic
listen 9443 http
ServerRoot /usr/local/apache2
#sets the Apache httpd servers own name to match the fully
qualified domain name where the server will be exposed
ServerName Route jaeger-ui.domain.xyz:9443
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog ${LOGS_DIR}/access_log common
ErrorLog ${LOGS_DIR}/error_log
LogLevel debug
LDAPTrustedMode SSL
LDAPVerifyServerCert Off
#file reference to Certificate Chain of LDAP directory server
allowing for trust of LDAPS TLS encryption
LDAPTrustedGlobalCert CA_BASE64 /ldap/ldapca.cer
#enabling TLS encryption on routed servername with private key
and other X-509 certificate part
#generation of certificates is ont show in this guide, but is
basic X-509 certificates for use to identify servers and add TLS
encryption to HTTPS
SSLEngine on
SSLProtocol all -SSLv2 -SSLv3
SSLCertificateKeyFile /ssl/jaeger-nopass.key
SSLCertificateFile /ssl/jaeger.cer
SSLCertificateChainFile /ssl/jaeger-chain.pem
SSLVerifyClient none
SSLVerifyDepth 10
<Directory />
AllowOverride none
Require all denied
</Directory>
<Location />
#LogLevel trace8
AuthName "Jaeger"
#Forces all access to the Apache httpd servers listener to
require Basic Authenticatio directing user
#authentication to LDAP directory and also requiring users to be
members of specific LDAP group
AuthType Basic
AuthBasicProvider ldap
```

```
AuthLDAPURL
ldaps://ldapdirectory.domain.xyz:636/ou=AllUsers,dc=domain,dc=xy
z?uid SSL
Require ldap-group cn=JaegerUsers,ou=Groups,dc=domain,dc=xyz
AuthLDAPGroupAttribute uniqueMember
AuthLDAPGroupAttributeIsDN on
</Location>
#Configures the Apache httpd server to work in reverse proxy
mode forwarding all incoming traffic after authentication and
authorization parts
#are taken care of by the Apache httpd server
#this wraps the Jaeger UI endpoints http traffic protecting it
from access
ProxyPass "/"  "http://localhost:16686/"
ProxyPassReverse "/"  "https://jaeger-ui.domain.xyz/"
ProxyPreserveHost       On
ProxyRequests           Off
AllowEncodedSlashes     NoDecode
<IfModule mod_mime_magic.c>
MIMEMagicFile /usr/local/apache2/conf/magic
</IfModule>
```

The other Apache configuration parts with the certificates and trust stores are not shown here, but contains keys and certificates to add TLS encryption and trust the LDAP directory's certificates.

The configuration for the Apache httpd server wrapping the Jaeger Collector port is completely similar to the configuration shown here, just with other ports, certificates, groups and domain names.

OpenShift configuration

So in order use this Apache configuration on top of the jaeger-all-in-one container we can add container configurations to the DeploymentConfig where the Jaeger container is defined, and mounting in all the resources needed from ConfigMap and Secret resources:

OpenShift DeploymentConfig for Jaeger with Apache side-car containers

```
apiVersion: v1
kind: DeploymentConfig
metadata:
  name: jaeger-secure
  namespace: jaeger-demo
  labels:
    app: jaeger-all-in-one
    jaeger-infra: all-in-one
spec:
  strategy:
    type: Recreate
    recreateParams:
      timeoutSeconds: 600
    resources: {}
    activeDeadlineSeconds: 21600
  triggers:
    - type: ImageChange
      imageChangeParams:
        containerNames:
          - jaeger-http
        from:
          kind: ImageStreamTag
          namespace: openshift
          name: 'httpd:latest'
        lastTriggeredImage: >-

172.30.0.100:5000/openshift/httpd@sha256:46799387c381307142d5ddf
fd8e1ef14b963bc054e25a9fec9f4532e86a3aeef
    - type: ImageChange
      imageChangeParams:
        containerNames:
          - jaeger-collector-http
        from:
          kind: ImageStreamTag
          namespace: openshift
          name: 'httpd:latest'
        lastTriggeredImage: >-

172.30.0.100:5000/openshift/httpd@sha256:46799387c381307142d5ddf
fd8e1ef14b963bc054e25a9fec9f4532e86a3aeef
  replicas: 1
  test: false
```

```
selector:
  name: jaeger-all-in-one
template:
  metadata:
    creationTimestamp: null
    labels:
      name: jaeger-all-in-one
  spec:
    volumes:
      - name: volume-config
        configMap:
          name: httpd.conf
          defaultMode: 420
      - name: volume-ldap
        configMap:
          name: ldapca.cer
          defaultMode: 420
      - name: volume-ssl
        secret:
          secretName: jaeger-ssl
          defaultMode: 420
      - name: volume-collector-config
        configMap:
          name: httpd-collector.conf
          defaultMode: 420
      - name: volume-collector-ssl
        secret:
          secretName: jaeger-collector-ssl
          defaultMode: 420
    containers:
      - name: jaeger-all-in-one
        image: 'jaegertracing/all-in-one:latest'
        ports: {}
        resources: {}
        readinessProbe:
          httpGet:
            path: /
            port: 16686
            scheme: HTTP
          initialDelaySeconds: 5
          timeoutSeconds: 1
          periodSeconds: 10
          successThreshold: 1
          failureThreshold: 3
        terminationMessagePath: /dev/termination-log
        imagePullPolicy: IfNotPresent
      - name: jaeger-http
```

```
          image: >-
172.30.0.100:5000/openshift/httpd@sha256:46799387c381307142d5ddf
fd8e1ef14b963bc054e25a9fec9f4532e86a3aeef
          ports:
            - containerPort: 8443
              protocol: TCP
          env:
            - name: HTTPD_CONF
              value: /config/httpd.conf
          resources: {}
          volumeMounts:
            - name: volume-config
              mountPath: /config
            - name: volume-ldap
              mountPath: /ldap
            - name: volume-ssl
              mountPath: /ssl
          terminationMessagePath: /dev/termination-log
          imagePullPolicy: IfNotPresent
        - name: jaeger-collector-http
          image: >-
172.30.0.100:5000/openshift/httpd@sha256:46799387c381307142d5ddf
fd8e1ef14b963bc054e25a9fec9f4532e86a3aeef
          ports:
            - containerPort: 9443
              protocol: TCP
          env:
            - name: HTTPD_CONF
              value: /config/httpd.conf
          resources: {}
          volumeMounts:
            - name: volume-collector-config
              mountPath: /config
            - name: volume-ldap
              mountPath: /ldap
            - name: volume-collector-ssl
              mountPath: /ssl
          terminationMessagePath: /dev/termination-log
          imagePullPolicy: IfNotPresent
      restartPolicy: Always
      terminationGracePeriodSeconds: 30
      dnsPolicy: ClusterFirst
      securityContext: {}
```

The ConfigMap and Secret parts in its OpenShift YAML formats are very simple, and will not be shown here. They just simply contains the contents of the configuration files that are mounted in through volumes to the different Apache httpd containers.

The Service and Router configuration is interesting as those parts shows how the Apache httpd ports are accessed both as services for internal use inside the OpenShift environment and for external access through the HAProxy routed functionality.

Service configuration shown here is then only exposing the secured ports of the container selected on the jaeger-all-in-one pod, and not other ports, that are not accessible as not present as exposed ports from the Docker containers:
Service definition for secured Jaeger

```
apiVersion: v1
kind: Service
metadata:
  name: jaeger-all-in-one
  namespace: jaeger-demo
  labels:
    app: jaeger-all-in-one
    jaeger-infra: all-in-one
spec:
  ports:
    - name: query-https
      protocol: TCP
      port: 8443
      targetPort: 8443
    - name: collector-https
      protocol: TCP
      port: 9443
      targetPort: 9443
  selector:
    name: jaeger-all-in-one
  clusterIP: 172.30.0.101
  type: ClusterIP
  sessionAffinity: None
```

The Route definition shown here is using the passthrough option of routes, so the encryption and HTTPS protocol parts are passed all through the router layer, the underlying service layer and down to the Apache httpd Docker containers where the HTTPS communication originates from.
The route shown is for the UI part, and for the Collector part it is completely similar just with another hostname and targeting the port on the service matching the secured Collector port on the matching Apache httpd Docker container.
Service definition for secured Jaeger

```
apiVersion: v1
kind: Route
metadata:
  name: jaeger
  namespace: jaeger-demo
  labels:
    app: jaeger-all-in-one
    jaeger-infra: all-in-one
spec:
  host: jaeger-ui.domain.xyz
  to:
    kind: Service
    name: jaeger-all-in-one
    weight: 100
  port:
    targetPort: query-https
  tls:
    termination: passthrough
  wildcardPolicy: None
```

Send traces to secured endpoints

In order to send traces to the secured endpoints one can use any of the http supporting client libraries of Jaeger in either Open Tracing or Zipkin formats. In our case we have used a slightly modified version of the Java Spring Boot setup explained here:

OpenTracing Spring Boot Instrumentation

First let's write a simple web app. Or better, let's generate it! All we have to do is just to select a web dependency...www.hawkular.org

where we have targeted the secured https exposed Zipkin endpoint of the Jaeger collector with code similar to this:

```
@Bean
 public io.opentracing.Tracer zipkinTracer() {
  ZipkinSender sender =
ZipkinSender.create(domain.xyz.URLConnectionSender.builder()
    .endpoint(
      "https://jaeger-
collector.domain.xyz/api/traces?format=zipkin.thrift")
    .compressionEnabled(false).build());
  Metrics metrics = new Metrics(new StatsFactoryImpl(new
NullStatsReporter()));
  Reporter reporter = new RemoteReporter(sender, 1000, 100,
metrics);
  Sampler sampler = new ProbabilisticSampler(1);
  io.opentracing.Tracer tracer = new Tracer.Builder("DEMO",
reporter, sampler).build();
  return tracer;
 }
```

using a slightly changed version of the class zipkin.reporter.urlconnection.URLConnectionSender, where the send method have been changed to add the Basic Authentication support to pass on the username and password needed for the authentication:

```
void send(byte[] body, String mediaType) throws IOException

    HttpURLConnection connection = (HttpURLConnection)
endpoint().openConnection();
    String username="theusername"; // added
    String password="thepassword";  // added
    String basicAuth =
Base64.getEncoder().encodeToString((username+":"+password).getBy
```

```
tes(StandardCharsets.UTF_8)); // added
    connection.setRequestProperty ("Authorization", "Basic
"+basicAuth); // added
```

This article was written to show one example on how to utilise a sidecar deployment technique on OpenShift/Kubernetes platform with Apache httpd server to protect a Jaeger tracing server on both its UI and Collecting http endpoints.

- Openshift
- Opentracing
- Distributed Tracing

Lars Milland

Meghan Daum
10.5K

Top on Medium
The Most Important Skill Nobody Taught You

Zat Rana
198K
Responses

Write a response…

- 5
-
-
-