AVOCADO SECURITY PLATFORM

USAGE INSTRUCTIONS

Table of Contents

Avocado Security Platform (ASP) Deployment Guide

Avocado Security Platform (ASP) is delivered as container images with a Helm 3 chart for deployment on Amazon Elastic Kubernetes Service (Amazon EKS), Amazon EKS Anywhere, or self-managed Kubernetes clusters.

ASP helps security and engineering teams apply AI-assisted threat modeling, contextual risk awareness, automated vulnerability mapping, real-time mitigation guidance, runtime security controls, application security workflows, application-level zero trust, continuous threat modeling, and DevSecOps automation.

Services

The Helm chart deploys the following jobs and services.

Component	Type	Purpose
Avocado MySQL Schema	Job	Creates the required Avocado MySQL database schema on an existing MySQL cluster.
Avocado Elasticsearch Schema	Job	Creates the required Avocado Elasticsearch indices on an existing Elasticsearch cluster.
Avocado Reveal AI	Service	Provides AI suggestions for vulnerabilities found by Avocado Orchestrator.
Avocado Version Info Parser (VIP)	Service	Loads NVD/NIST CVE data into Elasticsearch, keeps CVE data current, and maps product/vendor/version details to CPE and CVE data.
Avocado Orchestrator	Service	Manages Avocado Security plugins, tenants, applications, users, data security policies, application security policies, and security reports.
Avocado UI	Service	Provides the frontend web interface for Avocado Orchestrator.

Prerequisites

Before deploying ASP, ensure that the following prerequisites are available.

1. MySQL Cluster

A MySQL cluster must be running with the required configuration. Alternatively, you can deploy MySQL Operator for Kubernetes.

For MySQL Operator deployment instructions, see the official MySQL Operator documentation:

https://dev.mysql.com/doc/mysql-operator/en/

Required MySQL Configuration

Configure the MySQL cluster with the following settings in `my.cnf/mycnf`:

```
[mysqld]
bind-address=0.0.0.0
default-time-zone='+00:00'
sql-mode=STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION,NO_AUTO_VALUE_ON_ZERO
transaction-isolation=READ-COMMITTED
innodb_buffer_pool_size=4G
innodb_buffer_pool_instances=8
innodb_flush_log_at_trx_commit=2
innodb_flush_log_at_timeout=1800
innodb_log_file_size=1G
key_buffer_size=64M
sort_buffer_size=256M
join_buffer_size=256K
max_allowed_packet=500M
binlog_expire_logs_seconds=604800
```

2. Elasticsearch Cluster

An Elasticsearch cluster must be running.

If Elasticsearch is not already deployed, follow the Elastic Cloud on Kubernetes quickstart documentation:

https://www.elastic.co/docs/deploy-manage/deploy/cloud-on-k8s/install-using-yaml-manifest-quickstart

Deployment Configuration

Configure the following Helm values before deploying ASP.

1. **`global.externalServices.mysql.host`**

Kubernetes DNS name or reachable hostname for the existing MySQL service.

2. **`global.externalServices.elasticsearch.host`**

Kubernetes DNS name or reachable hostname for the existing Elasticsearch HTTP service.

3. **`global.externalSecrets.mysql.name`**

Name of the Kubernetes secret that contains the MySQL admin credentials and Avocado database user password.

Create the `avocado-mysql-secret` secret:

```
kubectl create secret generic avocado-mysql-secret \
  --namespace <YOUR_NAMESPACE> \
  --from-literal=password=<AVOCADO_USER_PASSWORD> \
  --from-literal=rootPassword=<ADMIN_USER_PASSWORD> \
  --from-literal=rootUser=<ADMIN_USERNAME>
```

4. `global.externalSecrets.elasticsearch.name`

Name of the Kubernetes secret that contains the Elasticsearch username and password.

Create the `avocado-es-user` secret:

```
kubectl create secret generic avocado-es-user \
  --namespace <YOUR_NAMESPACE> \
  --from-literal=user=<ELASTICSEARCH_USERNAME> \
  --from-literal=password=<ELASTICSEARCH_PASSWORD>
```

5. `global.externalSecrets.keyStoreTrustStore.name`

Name of the Kubernetes secret that contains keystore and truststore files and their passwords.

For instructions, see Generate SSL Keystore and Truststore.

Create the `avocado-keystore-truststore-secret` secret:

```
kubectl create secret generic avocado-keystore-truststore-secret \
  --namespace <YOUR_NAMESPACE> \
  --from-file=avocado-keystore.p12=<YOUR_KEYSTORE_P12_FILE> \
  --from-file=avocado-truststore.p12=<YOUR_TRUSTSTORE_P12_FILE> \
  --from-literal=keystore-password=<YOUR_KEYSTORE_PASSWORD> \
  --from-literal=truststore-password=<YOUR_TRUSTSTORE_PASSWORD>
```

6. `global.externalSecrets.externalApiKeys.name`

Name of the Kubernetes secret that contains external API keys, such as NVD NIST, IP geolocation, and Gemini API keys.

Create the `avocado-external-api-keys` secret:

```
kubectl create secret generic avocado-external-api-keys \
  --namespace <YOUR_NAMESPACE> \
  --from-literal=nvdNistApiKey=<NVD_NIST_API_KEY> \
  --from-literal=geminiApiKey=<GEMINI_API_KEY> \
  --from-literal=ipGeolocationApiKey=<IP_GEOLOCATION_API_KEY>
```

7. `global.externalSecrets.mysqlTruststore.name`

Name of the Kubernetes secret that contains the MySQL truststore file and password. This is required only when MySQL JDBC SSL verification is enabled.

Create the `mysql-truststore-secret` secret:

2. Get the MySQL cluster CA certificate file.
3. Create the truststore with the MySQL CA certificate:

```
<JAVA_HOME>/bin/keytool -importcert \
```

```
  -trustcacerts \
  -alias mysql-root-ca \
  -file <YOUR_CA_FILE> \
  -keystore mysql-truststore.p12 \
  -storetype PKCS12 \
  -storepass <TRUSTSTORE_PASSWORD> \
  -noprompt
```

4. Create the Kubernetes secret using the generated `mysql-truststore.p12` file:

```
kubectl create secret generic mysql-truststore-secret \
  --namespace <YOUR_NAMESPACE> \
  --from-file=mysql-truststore.p12=./mysql-truststore.p12 \
  --from-literal=truststore-password=<TRUSTSTORE_PASSWORD>
```

8. `global.externalSecrets.cipher.name`

Name of the Kubernetes secret that contains the Orchestrator cipher public and private key files.

Create the `avocado-cipher-secret` secret:

5. Generate a private and public key pair for the Orchestrator cipher encryption configuration:

```
openssl genrsa -out cipher_private_key.pem 2048

openssl pkcs8 -topk8 \
  -inform PEM \
  -outform DER \
  -nocrypt \
  -in cipher_private_key.pem \
  -out cipher_private_key.der

openssl rsa \
  -in cipher_private_key.pem \
  -pubout \
  -outform DER \
  -out cipher_public_key.der
```

6. Create the `avocado-cipher-secret` secret using the generated key pair:

```
kubectl create secret generic avocado-cipher-secret \
  --namespace <YOUR_NAMESPACE> \
  --from-file=cipher-private.key=./cipher_private_key.der \
  --from-file=cipher-public.key=./cipher_public_key.der
```

9. `global.externalSecrets.uiTls.name`

Name of the Kubernetes secret that contains the TLS certificate and key used by the Avocado UI.

Create the `avocado-ui-tls-secret` secret:

7. Get your SSL CA certificate and existing SSL certificate/key files, or generate a certificate and key using your CA certificate.

For instructions, see Generate SSL Certificate and Key.

8. Create the UI TLS secret:

```
kubectl create secret generic avocado-ui-tls-secret \
  --namespace <YOUR_NAMESPACE> \
  --from-file=tls.crt=<YOUR_SSL_CERT_FILE> \
  --from-file=tls.key=<YOUR_SSL_KEY_FILE> \
```

```
  --from-file=ca.crt=<YOUR_SSL_CA_CERT_FILE>
```

10. **`global.externalSecrets.proxy.name`**

Name of the Kubernetes secret that contains the proxy password. This is required only when proxy authentication is enabled.

Create the `avocado-proxy-secret` secret:

```
kubectl create secret generic avocado-proxy-secret \
  --namespace <YOUR_NAMESPACE> \
  --from-literal=password=<YOUR_PROXY_PASSWORD>
```

11. **`global.runtimeConfig.existingConfigMap`**

Name of the ConfigMap that contains runtime Spring Boot property overrides for Orchestrator, VIP, and Reveal AI.

Create the `avocado-runtime-config` ConfigMap using the following sample YAML:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: avocado-runtime-config
data:
  orchestrator-application.properties: |
    use.proxy=false
    proxy.host=
    proxy.port=
    proxy.username=
    proxy.password=${PROXY_PASSWORD:}
    proxy.bypass.hosts=avocado-vip,avocado-reveal-ai
    reveal-ai.api.enabled=true

  vip-application.properties: |
    use.proxy=false
    proxy.host=
    proxy.port=
    proxy.username=
    proxy.password=${PROXY_PASSWORD:}
    proxy.bypass.hosts=

  revealai-application.properties: |
    proxy.enabled=false
    proxy.host=
    proxy.port=
    proxy.type=HTTP
    proxy.usrnm=proxy-user
    proxy.usrcred=${PROXY_USRCRED:}
```

Apply the ConfigMap:

```
kubectl apply -f avocado-runtime-config.yaml --namespace <YOUR_NAMESPACE>
```

12. **`orchestrator.pvc.storageClassName`**

StorageClass used for Orchestrator persistent storage. Use an RWX-capable storage class for multi-replica deployments.

13. `orchestrator.pvc.size`

Requested PVC size for Orchestrator persistent storage, for example `10Gi`.

Helm Deployment

Update the parameter values and run the following command to deploy ASP using the Helm chart:

```
helm upgrade --install avocado-security-platform <CHART_URI> \
  --namespace <YOUR_NAMESPACE> \
  --create-namespace \
  --set global.externalServices.mysql.host=<MYSQL_SERVICE_HOST> \
  --set global.externalServices.elasticsearch.host=<ELASTICSEARCH_SERVICE_HOST> \
  --set global.externalSecrets.mysql.name=<MYSQL_SECRET_NAME> \
  --set global.externalSecrets.elasticsearch.name=<ELASTICSEARCH_SECRET_NAME> \
  --set global.externalSecrets.keyStoreTrustStore.name=<KEYSTORE_TRUSTSTORE_SECRET_NAME> \
  --set global.externalSecrets.externalApiKeys.name=<EXTERNAL_API_KEYS_SECRET_NAME> \
  --set global.externalSecrets.mysqlTruststore.name=<MYSQL_TRUSTSTORE_SECRET_NAME> \
  --set global.externalSecrets.cipher.name=<CIPHER_SECRET_NAME> \
  --set global.externalSecrets.uiTls.name=<UI_TLS_SECRET_NAME> \
  --set global.runtimeConfig.existingConfigMap=<AVOCADO_RUNTIME_CONFIGMAP_NAME> \
  --set orchestrator.pvc.storageClassName=<RWX_STORAGE_CLASS> \
  --set orchestrator.pvc.size=<PVC_SIZE> \
  --wait \
  --timeout 30m
```

Generate SSL Keystore and Truststore

The keystore and truststore are used by Avocado Orchestrator, Avocado Reveal AI, and Avocado VIP services to communicate over TLS.

Generate `keystore.p12` and `truststore.p12` Using Existing CA Files

This procedure assumes the following CA certificate and CA private key files are available:

```
ca.cert.pem
ca.key.pem
```

1. Generate the Server/Application Private Key

```
mkdir -p p12-manual-output
openssl genpkey \
  -algorithm RSA \
  -pkeyopt rsa_keygen_bits:4096 \
  -out p12-manual-output/server.key.pem
```

2. Create the Server CSR Configuration File

The following is a sample certificate configuration file. Update it for your environment.

```
cat > p12-manual-output/server-csr.cnf <<'EOF_CSR'
[ req ]
prompt = no
default_md = sha384
distinguished_name = dn
req_extensions = req_ext
```

```
[ dn ]
C = US
O = Example Org
OU = Security
CN = localhost

[ req_ext ]
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = localhost
IP.1 = 127.0.0.1
EOF_CSR
```

3. Generate the Server/Application CSR

```
openssl req -new \
  -key p12-manual-output/server.key.pem \
  -out p12-manual-output/server.csr.pem \
  -config p12-manual-output/server-csr.cnf
```

4. Create the Server Certificate Extension File

The following is a sample certificate extension file. Update it for your environment.

```
cat > p12-manual-output/server-cert.ext <<'EOF_EXT'
basicConstraints = critical, CA:false
keyUsage = critical, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth, clientAuth
subjectAltName = @alt_names
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer

[alt_names]
DNS.1 = localhost
IP.1 = 127.0.0.1
EOF_EXT
```

5. Sign the Server/Application Certificate Using the CA

```
openssl x509 -req \
  -sha384 \
  -days 825 \
  -in p12-manual-output/server.csr.pem \
  -CA ca.cert.pem \
  -CAkey ca.key.pem \
  -CAcreateserial \
  -out p12-manual-output/server.cert.pem \
  -extfile p12-manual-output/server-cert.ext
```

6. Create **`truststore.p12`** Using the Existing CA Certificate

```
keytool -importcert \
  -alias root-ca \
  -file ca.cert.pem \
  -keystore truststore.p12 \
  -storetype PKCS12 \
  -storepass <TRUSTSTORE_PASSWORD> \
  -noprompt
```

7. Create `keystore.p12` Using the Generated Server Key and Certificate

```
openssl pkcs12 -export \
  -out keystore.p12 \
  -inkey p12-manual-output/server.key.pem \
  -in p12-manual-output/server.cert.pem \
  -certfile ca.cert.pem \
  -name localhost \
  -passout pass:<KEYSTORE_PASSWORD> \
  -keypbe AES-256-CBC \
  -certpbe AES-256-CBC \
  -macalg SHA256
```

Generate SSL Certificate and Key

Use this procedure to generate the SSL certificate and key used by the Avocado UI.

1. Generate the Apache HTTPD Private Key

Use RSA 4096 bits:

```
openssl genpkey \
  -algorithm RSA \
  -pkeyopt rsa_keygen_bits:4096 \
  -out apache-httpd.key.pem
```

2. Create the CSR Configuration File

The following is a sample certificate configuration file. Update it for your environment.

```
cat > apache-httpd-csr.cnf <<'EOF_CSR'
[ req ]
prompt = no
default_md = sha384
distinguished_name = dn
req_extensions = req_ext

[ dn ]
C = US
O = Example Org
OU = IT
CN = myserver.example.com

[ req_ext ]
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = myserver.example.com
DNS.2 = www.myserver.example.com
IP.1 = 127.0.0.1
EOF_CSR
```

3. Generate the Certificate Signing Request

```
openssl req -new \
  -key apache-httpd.key.pem \
  -out apache-httpd.csr.pem \
  -config apache-httpd-csr.cnf
```

4. Create the Server Certificate Extension File

The following is a sample certificate extension file. Update it for your environment.

```
cat > apache-httpd-cert.ext <<'EOF_EXT'
basicConstraints = critical, CA:false
keyUsage = critical, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer

[alt_names]
DNS.1 = myserver.example.com
DNS.2 = www.myserver.example.com
IP.1 = 127.0.0.1
EOF_EXT
```

5. Sign the Apache HTTPD Certificate Using the Existing CA

```
openssl x509 -req \
  -sha384 \
  -days 825 \
  -in apache-httpd.csr.pem \
  -CA ca.cert.pem \
  -CAkey ca.key.pem \
  -CAcreateserial \
  -out apache-httpd.cert.pem \
  -extfile apache-httpd-cert.ext
```

This command creates the following certificate file:

```
apache-httpd.cert.pem
```