Zettaset

Zettaset XCrypt Container Encryption for Docker Enterprise by Mirantis

Deployment and Maintenance Guide

Version 8.5.0

Table of Contents

Overview

Zettaset XCrypt Container Encryption for Docker Enterprise offers high-performance transparent and granular data-at-rest encryption for Docker containers.

The product consists of the following components:

- Service Containers
  - Certificate Authority – issues and manages certificates used for secure communication between product components
  - Virtual Key Manager – KMIP-compliant software key manager that is responsible for creating, storing, and managing various keys used by the product
  - License Server – responsible for product license management
- Host Components
  - Cryptographic Module – Linux kernel module that provides data-at-rest encryption and decryption services
  - Zettaset Volume Driver – proprietary volume driver developed to Docker Volume Driver API specification. The Volume Driver is responsible for transparent allocation and management of encrypted volumes.

Why XCrypt for Container Encryption?

XCrypt Container is a first-of-its-kind encryption solution built specifically for containers. Because it's software-only, it overcomes all the typical adoption barriers for encryption – offering a negligible effect on performance and adding no additional complexity to your container environments. XCrypt Container Encryption makes container encryption so simple, it's hard ***not*** to do it.

Zettaset's software-defined solution provides high performance encryption that transparently protects containers from data theft in any physical or virtual environment. Its ease of use and intelligence enables pervasive and on-demand encryption that's scalable across any deployment.

- **Transparent integration** into major container software via Zettaset volume driver
- **All required services run in containers** – key manager, certificate authority, and license server
- **Management of host storage is completely automated**
- **A dedicated volume group** is allocated for each container volume
- **Container volumes are cryptographically tied** to containers

Kubernetes protects your "secrets." XCrypt protects your data

XCrypt Kubernetes Encryption is a software-only solution built specifically to protect data-at-rest in Kubernetes environments. It provides a transparent, high performance layer of security for your entire DevOps pipeline that simplifies data protection while also acting as a 'last-line-of-defense' against all the new attack vectors that exist in today's highly-distributed Kubernetes environments. It integrates directly with

Kubernetes' storage layer and can be deployed without changes to existing processes to enable an efficient, elegant transformation to DevSecOps.

- **Direct integration** into Kubernetes' storage layer
- **Software-only** for simple deployment
- **Negligible impact on performance**
- **Container storage and data separation**
- **Unique encryption key per each container volume**
- **Automated encryption policy management**
- **Secure erase of volumes** rather than partitions

Use Cases

- **Transitioning from DevOps to DevSecOps**
- **Ensuring data protection** in highly-distributed Kubernetes environments
- **Achieving compliance** in highly regulated industries like healthcare and finance

Part I: Deployment

This part contains information on the deployment of your XCrypt software and includes the following sections:

- Section 1: Installation Prerequisites
- Section 2: Preparing Installation
- Section 3: Building and Installing Service Containers
- Section 4: Installing Host Components
- Section 5: (Optional) Using XCrypt Container Encryption with an External Key Manager

Section 1: Installation Prerequisites

1 Confirm that the OS is either CentOS or RHEL 7.2 or above:

```
$ cat /etc/issue.net
CentOS Linux release 7.3.1611 (Core)
```

2 Make sure that installation user can run privileged commands via *sudo*.

3 Confirm that Java 8 OpenJDK is installed.

```
$ java -version
openjdk version "1.8.0_222"
OpenJDK Runtime Environment (build 1.8.0_222-b10)
OpenJDK 64-Bit Server VM (build 25.222-b10, mixed mode)
```

4 Confirm that Docker Enterprise version 3.1, Docker version 19, and Kuberetes version 1.17 (for Kubernetes deployments only) are installed.

Section 2: Preparing Installation

1 Install deployment RPM

```
$ sudo rpm –i zts-xcrypt-docker-package*.rpm
```

2 Create *ztsnet* network. Depending on the type of the installation you may need to create *bridge* or *macvlan* network.

```
$ docker network create ztsnet
```

Use the correct IP addresses that have not been assigned to other resources when specifying service IP addresses in the following steps. You can use the following command to determine IP address information for the network created using the commands above.

```
$ docker network inspect ztsnet
```

3 Place license file provided by Zettaset into a file called *zts_license.xml* in */usr/share/zts/deploy* directory. The file must be readable by the user that will be running the installation. The file name and path must be exactly as specified; otherwise, the installation will abort.

```
$ sudo cp zts license.xml /usr/share/zts/deploy
```

Section 3: Building and Installing Service Containers

1 Navigate to deployment directory

```
$ cd/usr/share/zts/deploy
```

2 Run the installer

```
$ ./install_xcrypt_docker.sh -c ca_ip_address -p ca_password \
  -l license_server_ip_address -k kmip_server_ip_address
```

For example:

```
$ ./install_xcrypt_docker.sh -c 172.18.0.2 -p 'asdf$#1234!' -l 172.18.0.3 \
  -k 172.18.0.4
```

Section 4: Installing Host Components

1 Run the host components installer

```
$ ./configure_docker_host.sh -c ca_ip_address -p ca_password \
   -l license_server_ip_address -k kmip_server_ip_address
```

For example:

```
$ ./configure_docker_host.sh -c 172.18.0.2 -p 'asdf$#1234!' -l 172.18.0.3
   -k 172.18.0.4
```

Note that you must specify the same IP addresses and password that you used when installing service containers.

The Certificate Authority service container is stopped following this step, since all required certificates are provisioned at this point.

Section 5: (Optional) Using XCrypt Container Encryption with an External Key Manager

Zettaset XCrypt Container Encryption may be used with external Key Manager that supports Key Management Interoperability Protocol (KMIP) version 1.1.

To use external key manager to manage encryption keys created by Zettaset Container Encryption, install Zettaset Container Encryption product as described in the Installation section above. Following successful installation, stop the internal Key Manager by running the following command:

```
$ docker stop kmipserver
```

You will need the following information from the external key manager. Refer to the documentation for your external key manager on how to obtain this information.

A. Key Manager CA file (in PEM format)
B. Key Manager client certificate keystore (JKS in PKCS#12 format)
C. Key Manager IP address
D. Key Manager Port
E. Key Manager client keystore password

Modify */usr/share/zts/config/kmip-client.properties* file as follows:

1. Set *KmipCafile* property to value A above.

2. Set *KmipP12* property to value B above.
3. Set *KmipServer* property to value C above.
4. Set *KmipPort* property to value D above.
5. Set *KmipPasswd* property to value E above.
6. Set *KmipClient* property to the IP address or host name of the docker host.

Note that if required, connection to the external Key Manager must be configured immediately following successful installation and before using the product for the first time.

Part II: Maintenance

This part contains information on the maintenance of your Zettaset deployment and includes the following sections:

- Section 1: Adding Host Volumes
- Section 2: Starting Zettaset Volume Driver
- Section 3: Usage Example
- Section 4: Restarting the Host
- Section 5: Secure Host Removal

Configuring the Host

Section 1: Adding Host Volumes

In order for the product to provision encrypted volumes for containers, one or more disks on the host must be made available to the Zettaset Volume Driver. These disks must not contain any other applications or data.

To make a new disk available to Zettaset Volume Driver, execute the following command:

```
$ sudo /usr/share/zts/bin/zts_xcrypt_docker_add_device.sh device
```

For example:

```
$ sudo /usr/share/zts/bin/zts_xcrypt_docker_add_device.sh /dev/sdb
```

This command will partition the device and add make it available to Zettaset Volume Driver. Do not attempt to change partition structure of the device; this will cause data loss.

Section 2: Starting Zettaset Volume Driver

After successful installation, use the following command to start Zettaset Volume Driver service to enable encrypted volume provisioning.

```
$ sudo systemctl start zts-volume-driver
```

Monitor the Volume Driver startup using the following command:

```
$ sudo systemctl status zts-volume-driver -l
```

Volume Driver startup is complete when you see the following log line:

```
Started Application in ... seconds ...
```

Section 3: Usage Example

After confirming successful Volume Driver startup, you can create encrypted volumes using standard Docker volume management commands.

```
$ docker volume create -d ztsvolume --name zts-test-vol --opt size=500mb
```

Note the two required options when provisioning encrypted volumes:

- `-d ztsvolume –` used to specify Zettaset Volume Driver. The option argument should always specify *ztsvolume*.

- `--opt size=500mb` – used to specify encrypted volume size. Depending on the sizes of drives available to Zettaset Volume Driver and previous volume allocation requests, the actual size of the encrypted volume may be greater than requested size.

Once the encrypted volume is provisioned, you can run a container and attach the encrypted volume to it using standard Docker container management commands.

```
$ docker run -it --name zts-container -v zts-test-vol:/opt/ztsvol centos:7.3.1611
```

This command will place you inside the running container. You can access the encrypted volume by navigating to */opt/ztsvol* directory, as specified in the example above. Note that the data on the encrypted volume will persist even when the container exits. When the container is not running, the actual host volume is not mounted and the container data is therefore not accessible.

Host Maintenance Tasks

Section 4: Restarting the Host

After restarting the host, you must run the following command before to starting any containers using encrypted storage.

```
$ /usr/share/zts/bin/zts_xcrypt_docker_start.sh
```

This command will ensure that all required service containers are up and running and that the Zettaset Volume Driver is ready to provision and manage encrypted volumes.

Section 5: Secure Host Removal

In an event when Docker host is compromised, follow the steps listed in this section to securely remove the host and prevent access to any data on this host.

Removing the host does not remove any data that the containers running on this host have generated.

When a host is removed, its certificate used to gain access to cryptographic keys is revoked. Consequently, the containers running on the host will not be able to access any data it generated on any of the encrypted volumes.

Since the key requests take place when encrypted volumes are mounted, containers that are running prior to the host removal will still be able to access encrypted data until they are stopped. It is recommended to terminate all containers on the removed host or to restart the host after executing the removal commands.

1 Start Certificate Authority service container

```
$ docker start caserver
```

2 Check list of authorized hosts and make sure that the host you are about to revoke appears in this list

```
$ docker exec caserver /usr/share/zts/bin/zts-ca.sh list-hosts
```

3 Revoke host access to cryptographic keys

```
$ sudo docker exec caserver /usr/share/zts/bin/zts-ca.sh \
      revoke-host docker_host_ip_address
```

For example:

```
$ sudo docker exec caserver /usr/share/zts/bin/zts-ca.sh revoke-host 172.16.1.51
```

4 Instruct the Key Manager to retrieve the updated Certificate Revocation List (CRL).

```
$ sudo docker exec kmipserver /usr/share/zts/bin/crlgrabber.sh
```

5 Restart the Key Manager

```
$ sudo docker exec kmipserver service zts-kmip restart
```

6 Confirm successful host removal by executing the following command on the removed host, if you have access to it.

```
$ sudo /usr/share/zts/bin/kmipclient get abc
```

If the host is successfully removed, the command above will report an error:

```
get operation failed: I/O failed
```

7 Stop Certificate Authority service container

```
$ docker stop caserver
```

Part III: Deploying Encryption Services for Kubernetes

Section 1: Initial Configuration

1 Connect to Red Hat Container Registry

```
$ docker login registry.connect.redhat.com
```

2 Encode credentials

```
$ cat ~/.docker/config.json | base64 -w0
```

3 Create Kubernetes secret that will be used to pull container images

```
# xcrypt-rhconnect-registrykey.yaml
apiVersion: v1
kind: Secret
metadata:
  name: zts-redhat-token
data:
  .dockerconfigjson: <insert output of the previous command here>
type: kubernetes.io/dockerconfigjson
```

4 Apply the secret to your cluster

```
$ kubectl apply -f xcrypt-rhconnect-registrykey.yaml
```

Alternatively, you can create the image pull secret from the command line as follows

```
$ kubectl create secret zts-redhat-token regcred \
       --docker-server=registry.connect.redhat.com \
       --docker-username=<your-rhconnect-username> \
       --docker-password=<your-rhconnect-pword> \
       --docker-email=<your-rhconnect-email>
```

5 Create deployment configuration

```
# xcrypt-install-config.yaml
apiVersion: v1
kind: Secret
metadata:
  name: zts-install-config
type: Opaque
stringData:
  zts.conf: |-
   ca_password=<specify Certificate Authority password here>
   os_type=rh7
```

6 Apply deployment configuration

```
$ kubectl apply -f xcrypt-install-config.yaml
```

Section 2: Deployment

1 Login to your Kubernetes cluster

```
$ kubectl login -u kubeadmin -p <password>
```

2 Create service discovery entries

```
$ kubectl apply -f xcrypt-caserver-service.yaml
$ kubectl apply -f xcrypt-license-server-service.yaml
$ kubectl apply -f xcrypt-kmip-server-service.yaml
```

3 Deploy services

```
$ kubectl apply -f xcrypt-caserver-pod.yaml
$ kubectl apply -f xcrypt-license-server-pod.yaml
$ kubectl apply -f xcrypt-kmip-server-pod.yaml
$ kubectl apply -f xcrypt-host-manager-pod.yaml
```

4 Provision licenses for all worker nodes

```
$ kubectl exec -it zts-license-server-pod -- /usr/share/zts/bin/edit_nodes.sh \
      -a <public-IP-of-worker-node>
```

5 Determine IP addresses of the Key Manager server and the Host Manager

```
$ kubectl describe service zts-kmipserver-service | grep 'IP:'
$ kubectl describe pod zts-host-manager-pod | grep 'IP:'
```

6 Connect to the Host Manager

```
$ kubectl exec -it zts-host-manager-pod /bin/bash
$ vi /etc/hosts
```

Add host configuration entries to /etc/hosts file

```
<kmip-service-ip>            zts-kmip-server
<host-manager-public-ip>     zts-host-manager
```

Section 3: Examples

1 Login to your Kubernetes cluster

```
$ kubectl login -u kubeadmin -p <password>
```

2 Define Storage Class object

```
# storageclass.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: zts-sc
provisioner: zts.csi.zettaset.com
volumeBindingMode: WaitForFirstConsumer
```

```
$ kubectl apply -f storageclass.yaml
```

3 Create Volume Claim object requesting 3GB storage volume

```
# claim-3gi.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: zts-3gi-claim
spec:
  storageClassName: zts-sc
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

```
$ kubectl apply -f claim-3gi.yaml
```

4 Create sample pod that uses encrypted volume claim

```
# pod-3gi.yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-3gi-test
spec:
  nodeSelector:
    zts: zts-master
  containers:
  - name: app-3gi-test
    image: nginx
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo $(date -u) >> /mnt/data/out.txt; sleep 5;
done"]
    volumeMounts:
    - name: encryptedvolume
      mountPath: "/mnt/data"
  volumes:
  - name: encryptedvolume
    persistentVolumeClaim:
      claimName: zts-3gi-claim
```

```
$ kubectl apply –f pod-3gi.yaml
```

5 Cleanup

```
$ kubectl delete pod app-3gi-test
$ kubectl delete pvc zts-3gi-claim
```

Contacting Zettaset

Zettaset Corporate Headquarters
5050 El Camino Real, Suite 220
Los Altos, California 94022
USA

Toll Free: +1-888-511-3736
Fax: +1-650-314-7950

Customer Support
Email: support@zettaset.com
Phone: +1-866-561-4981

Resources

Company Website:	https://www.zettaset.com/
Data sheets:	https://www.zettaset.com/info-center/data-sheets/
Product info:	https://www.zettaset.com/products/xcrypt-container/
	https://www.zettaset.com/products/xcrypt-kubernetes-encryption/