IBM Hyper Protect Virtual Servers User's Guide - Version 1.2.x (2020-03-24)

Contents

About this documentation

This documentation describes how to use the IBM Hyper Protect Virtual Servers to deploy and manage Docker-based workloads on IBM Z and LinuxONE servers in your environment.

The documentation is structured based on the following major workflow:

- How to configure and start a Secure Service Container partition.
- How to install Hyper Protect hosting appliance by using the Secure Service Container user interface.
- How to configure and start IBM Hyper Protect Virtual Servers on IBM Z and LinuxONE servers in your cloud environment.
- How to securely build and deploy your containerized workload into the Hyper Protect Virtual Server containers.

Note that IBM Hyper Protect Virtual Servers contains components to support the following deployment:

- Secure Service Container for IBM Cloud Private. For more information about this component, see Working with Secure Service Container for IBM Cloud Private.
- IBM Hyper Protect Virtual Servers and its modules. For more information about this component, continue with this documentation.

This documentation describes the version of IBM Hyper Protect Virtual Servers that is available for deployment with:

- Hardware Management Console (HMC) / Support Element (SE) Version 2.14.0 (z14, z14 ZR1, LinuxONE Emperor II or LinuxONE Rockhopper II). For more information about Secure Service Container, see Secure Service Container User's Guide, SC28-6978-02a.
- Hardware Management Console (HMC) / Support Element (SE) Version 2.15.0 (z15, LinuxONE III). For more information about Secure Service Container, see Secure Service Container User's Guide, SC28-7005-00b.

Figures that are included in this document illustrate concepts and are not necessarily accurate in content, appearance, or specific behavior.

Important: The PDF version of this product document is a snapshot of the content on IBM Knowledge Center on 2020-03-24. To read the up-to-date documentation about IBM Hyper Protect Virtual Servers, see IBM Knowledge Center.

Intended audience

The primary audience for this documentation is developers wanting to securely build their applications, and administrators who are responsible for installing, and managing containerized applications in the secured cloud environment. Those containerized applications can be hosted within Hyper Protect Virtual Server containers on an IBM Z or LinuxONE server.

This documentation distinguishes the following types of user roles:

- Cloud admin
- Appliance admin
- System admin
- ISV or App developer

The different tasks that are described in this documentation are associated to one of these user roles. A single user can have a single role or multiple roles. For more information about the roles, see User roles.

Prerequisite and related information

To deploy and manage containerized workloads within Hyper Protect Virtual Server containers on IBM Z or LinuxONE servers, in addition to this documentation, system administrators also need to access the following reference to accomplish specific tasks.

- For more information about IBM Secure Service Container deployment to z14, z14 ZR1, LinuxONE Emperor II or LinuxONE Rockhopper II, see Secure Service Container User's Guide, SC28-6978-02a.
- For more information about IBM Secure Service Container deployment to z15 or LinuxONE III, see Secure Service Container User's Guide, SC28-7005-00b.

Release notes

- What's new
- Known issues and limitations
- Accessibility features

What's new in version 1.2.0

Get a quick overview of what's added, changed, improved, or deprecated in this release.

IBM Hyper Protect Virtual Servers Version 1.2.0 introduces the following new features and enhancements:

Support of IBM z15 and LinuxONE III

You can run IBM Hyper Protect Virtual Servers on the latest IBM z15 and IBM LinuxONE III.

Support of configuring storage on the Secure Service Container partition by using commands

When configuring the storage pool on the Secure Service Container partition, you can use the `disk` commands to manage and add disks into the storage pool.

And you can also use the `quotagroup update` command to configure the `appliance_data` quotagroup, which is used by the Hyper Protect hosting appliance.

For more information, see Configuring the storage on the Secure Service Container partition.

Support of creating networks for Secure Build Container on-the-fly

When configuring the network for a Secure Build container, you can have a list of options:

- Use the port mapping on the partition
- Use the dedicated IP address for external access
- Use an existing network on the partition
- Create a network on the partition when the Secure Build container is created

You can also control whether to delete the network when the Secure Build container is deleted by using the `delete_network_on_container_delete` parameter in the `securebuild.yaml` file.

For more information, see Configuring the network for Secure Build containers.

Support of creating the repository registration file on-the-fly

When creating the container image by using the Secure Build, you can retrieve the cleartext repository registration file in either python or JSON format.

The JSON repository registration file can be used as direct input to generate the encrypted repository definition file.

For more information, see Building the container image for your application by using the Secure Build.

Support of generating the signing key pair when creating and encrypting the repository registration files

When encrypting the repository registration file, you can use the private and public key pair generated by using IBM Hyper Protect Virtual Servers CLI tool, or generate your own key pair for the encryption.

For more information, see Creating your own public and private key pair to encrypt the repository definition file and Creating and encrypting the repository definition file.

Support of updating the Secure Build container and its base image

When updating the Secure Build container by using a different configuration or a newer version, you can use the `securebuild update` command to complete the tasks.

For more information, see Updating the base image of a Secure Build container and Updating the configuration of a running Secure Build Container.

Support of using personal certificate for the Secure Build container

Before creating the Secure Build container, you can specify your own certificate to override the default settings such as the certificate's Subject Name, duration, key length, signature algorithm, and so on. You can use your own certificate by using the `cert_name` key in the `securebuild.yaml` file.

For more information, see Using your own certificate for the Secure Build container.

Support of specifying the Dockerfile name and location for the Secure Build container

When building your application with the Secure Build, you can use the `dockerfile_path` key in the `securebuild.yaml` to specify the relative path of the Dockerfile in your github repository. The default value is the root directory of your github repository.

Also, the `build:script` and `build:dir` keys are not supported in this release.

For more information, see Building the container image for your application by using the Secure Build.

Support of monitoring

With the monitoring infrastructure provided in IBM Hyper Protect Virtual Servers, you can collect monitoring metrics from Hyper Protect hosting appliance and Secure Service Container partition.

For more information, see Monitoring IBM Hyper Protect Virtual Servers and Commands for monitoring

Support of Enterprise PKCS #11 (EP11) integration

With the Enterprise PKCS #11 over gRPC (GREP11) containers provided in IBM Hyper Protect Virtual Servers, you can integrate your application with the asymmetric (public and private) key pairs generated by the Hardware Security Modules (HSM) on the IBM z or LinuxONE servers.

For more information, see Integrating with the EP11 library.

More configuration example files

Additional configuration example files are available under the `/config` directory of each CLI command module, and you can refer to them when configuring your own environment for IBM Hyper Protect Virtual Servers.

Those example files covers the following network topologies for Hyper Protect Virtual Server and Secure Build containers:

- Layer 2 and Layer 3 network based on Ethernet type connections
- Layer 2 network based on VLAN type connections
- Layer 3 network based on VLAN type connections

For more information about the layer 2 and layer 3 network, see this blog.

For more information about VLAN or Ethernet type connections, download Secure Service Container User's Guide from the About topic.

Federal Information Processing Standards (FIPS)

Federal Information Processing Standards (FIPS) are information technology standards that are developed by the United States federal government. See IBM Hyper Protect Virtual Servers platform considerations for FIPS compliance for information about FIPS compliance in IBM Hyper Protect Virtual Servers.

Known issues and limitations

IBM Hyper Protect Virtual Servers V1.2.0 has the following limitations:

- The snapshots of a Hyper Protect Virtual Server container can only be created on the same Secure Service Container partition that the server instance resides.
- Logging is not supported.
- The Secure Service Container for IBM Cloud Private feature and IBM Hyper Protect Virtual Servers feature cannot co-exist on the same Secure Service Container partitions or Linux master/management servers. Each feature must use separate, dedicated Secure Service Container partitions and Linux master/management servers.
- You can only use Ubuntu 16.04 LTS and 18.04 LTS on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.
- You can only use IBM Hyper Protect Virtual Servers with Docker Hub or IBM Container Registry.
- You must restart the Hyper Protect Virtual Server container after you revert a snapshot of the Hyper Protect Virtual Server container.
- A `/newroot` mount point is initiated by default to bootstrap the Hyper Protect Virtual Server container.
- The actual root quotagroup size available for a Hyper Protect Virtual Server container is always 1 GB less than the assigned size when creating the root quotagroup for the container.
- Secure Build does not support Git Large File Storage (LFS).
- Secure Build requires that the private key, used to secure access to the source Github repository, does not have a passphrase.
- IBM Cloud Object Storage service is only supported for archiving application manifest files.
- Backup and restore of encrypted credentials used by the Secure Build container can only be supported by using Hosting Appliance snapshots.
- The monitoring infrastructure only collects metrics from the Hyper Protect hosting appliance and Secure Service Container partition.
- You cannot run the `monitoring` commands against a subset of Secure Service Container partitions if the VS`/monitoring-cli/config/monitoring.yaml` includes the configuration of multiple partitions. To run the `monitoring` commands against an individual Secure Service Container partition,

or multiple Secure Service Container partitions at the same time, only include the target Secure Service Container LPAR's stanza(s) for the intended `monitoring` command in the `VS/monitoring-cli/config/monitoring.yaml` configuration file.

- When using ep11 for text file encryption, the text file size architectural limit is 4MB.
- For the known issues and limitation about the IBM Secure Service Container for IBM Cloud Private, see Known issues of Secure Service Container for IBM Cloud Private.

Accessibility features for IBM Hyper Protect Virtual Servers

Accessibility features assist users who have a disability, such as restricted mobility or limited vision, to use information technology content successfully.

Overview

IBM Hyper Protect Virtual Servers includes the following major accessibility features:

- Keyboard-only operations
- Screen reader operations
- Command line interface (CLI) to configure the IBM Hyper Protect Virtual Servers offering

IBM Hyper Protect Virtual Servers uses the latest W3C Standard, WAI-ARIA 1.0, to ensure compliance with Section 508 Standards for Electronic and Information Technology and Web Content Accessibility Guidelines (WCAG) 2.0. To take advantage of accessibility features, use the latest release of your screen reader and the latest web browser that is supported by IBM Hyper Protect Virtual Servers.

The IBM Hyper Protect Virtual Servers online product documentation in IBM Knowledge Center is enabled for accessibility. The accessibility features of IBM Knowledge Center are described in the Accessibility section of the IBM Knowledge Center release notes. For general accessibility information, see Accessibility in IBM.

Keyboard navigation

IBM Hyper Protect Virtual Servers uses standard navigation keys.

IBM Hyper Protect Virtual Servers uses the following keyboard shortcuts.

Action	Shortcut for Internet Explorer	Shortcut for Firefox
Move to the Contents View frame	Alt+C, then press Enter and Shift +F6	Shift+Alt+C and Shift+F6

Table 1. Keyboard shortcuts in IBM Hyper Protect Virtual Servers documentation

Vendor software

IBM Cloud Private includes certain vendor software that is not covered under the IBM license agreement. IBM makes no representation about the accessibility features of these products. Contact the vendor for accessibility information about its products.

Related accessibility information

In addition to standard IBM help desk and support websites, IBM has a TTY telephone service for use by deaf or hard of hearing customers to access sales and support services:

TTY service
800-IBM-3383 (800-426-3383)
(within North America)

For more information about the commitment that IBM has to accessibility, see IBM Accessibility.

Introduction

To understand how IBM Hyper Protect Virtual Servers works, you can start with the following topics.

- Overview
- Advantages
- Technology at a glance
- Architecture overview
- Components
- System requirements
- User roles
- FAQs

Overview

Many technologies need to protect applications in production, leveraging encryption technologies; however, security threats can also surface during the development, pre-production phases. Additionally, during deployment and production, insiders who manage the infrastructure that hosts critical applications, may pose a threat given their super-user credentials and level of access to secrets or encryption keys.

Organizations need to incorporate secure design practices in their development operations and embrace DevSecOps to ensure the protection of their applications from the vulnerabilities and threat vectors that can compromise their data and potentially threaten their business.

IBM Hyper Protect Virtual Servers, the evolution of the IBM Secure Service Container for IBM Cloud Private offering, protects Linux workloads on IBM Z and LinuxONE throughout their lifecycle build management and deployment. This solution delivers the security needed to protect mission critical applications in hybrid multi-cloud deployments.

IBM Hyper Protect Virtual Servers enables:

- Developers to securely build their applications in a trusted environment with integrity.
- IT infrastructure providers to manage the servers and virtualized environment where the applications are deployed without having access to those applications or their sensitive data
- Application users to validate that those securely built applications originate from a trusted source by integrating this validation into their own auditing processes.
- Chief Information Security Officers (CISOs) to be confident that their data is both protected and private from internal and external threats.

IBM Hyper Protect Virtual Servers solutions delivers security measures to address threat vectors that appear at different phases of an application's lifecycle: build, deployment, and management. It is designed to uniquely protect such workloads that are deployed on IBM Z and LinuxONE servers in hybrid, multi-cloud environments.

Advantages

Running Hyper Protect Virtual Server containers on the Secure Service Container partitions provides you the following advantages in terms of security and integrity.

- System administrators do not need the access to the application data, memory, logs, secrets, applications or the operating system in the Hyper Protect Virtual Server containers.

- Application developers do not need the secret to the production environment, and managing the Hyper Protect Virtual Server containers does not require access to the application secrets.
- The containerized application images are signed with GPG keys when publishing, and verified again when being deployed. The signing keys are generated within the Secure Build process and your private keys are never revealed. Only the images generated by using the Secure Build procedure can be uploaded to your docker repository and installed to the Secure Service Container partitions.
- The Secure Build generates a signed manifest indicating the origin of the image for future audit. The manifest contains a copy of the Github project that was cloned by the Secure Build server container, and a copy of the build log (build.log) and overall build status result (build.json). The manifest tar ball is signed with the manifest private key. The user can download the manifest public key and use it to verify a manifest. You can optionally store the manifest in the IBM Cloud Object Storage (COS) by using the Secure Build.
- You can integrate IBM Hyper Protect Virtual Servers into your own Continuous Integration and Continuous Delivery (CICD) pipeline to fully adopt the security advantages provided by the offering.

Technology at a glance

IBM Hyper Protect Virtual Servers is a software solution built on the IBM Secure Service Container framework, which enables users to run containerized Linux workloads in the secure virtual server containers on IBM Z and LinuxONE.

IBM Hyper Protect Virtual Servers provides an encrypted environment (data at rest, data in flight), with peer to peer and peer to host isolation protecting container applications from access via privileged credentials, whether access is accidental or malicious, internal or external to an organization.

IBM Hyper Protect Virtual Servers ensures your applications can be deployed and managed from trusted sources without the infrastructure team being able to access the data, secrets or application.

IBM Secure Service Container

IBM Secure Service Container is a software appliance infrastructure that combines an operating system, middleware, and application components into a single software image. Software images deployed to a Secure Service Container partition can exploit the underlying security capabilities of the IBM Z and LinuxONE infrastructure.

By focusing on ease of management, ease of deployment, and security, the Secure Service Container is delivered in a virtual software appliance form factor, which can also isolate the running workload and deliver protections around the access of the environment.

In the Secure Service Container, a specialized Docker runtime environment called `runq` is used to spawn a dedicated qemu Virtual Server (VS) instance for each Docker image, including a guest operating system (OS) kernel for each qemu virtual server, and during deployment, a runtime environment of the workloads.

All these components are packaged together as the hosting appliance, and can be deployed on a partition of an IBM Z and LinuxONE server in a single step.

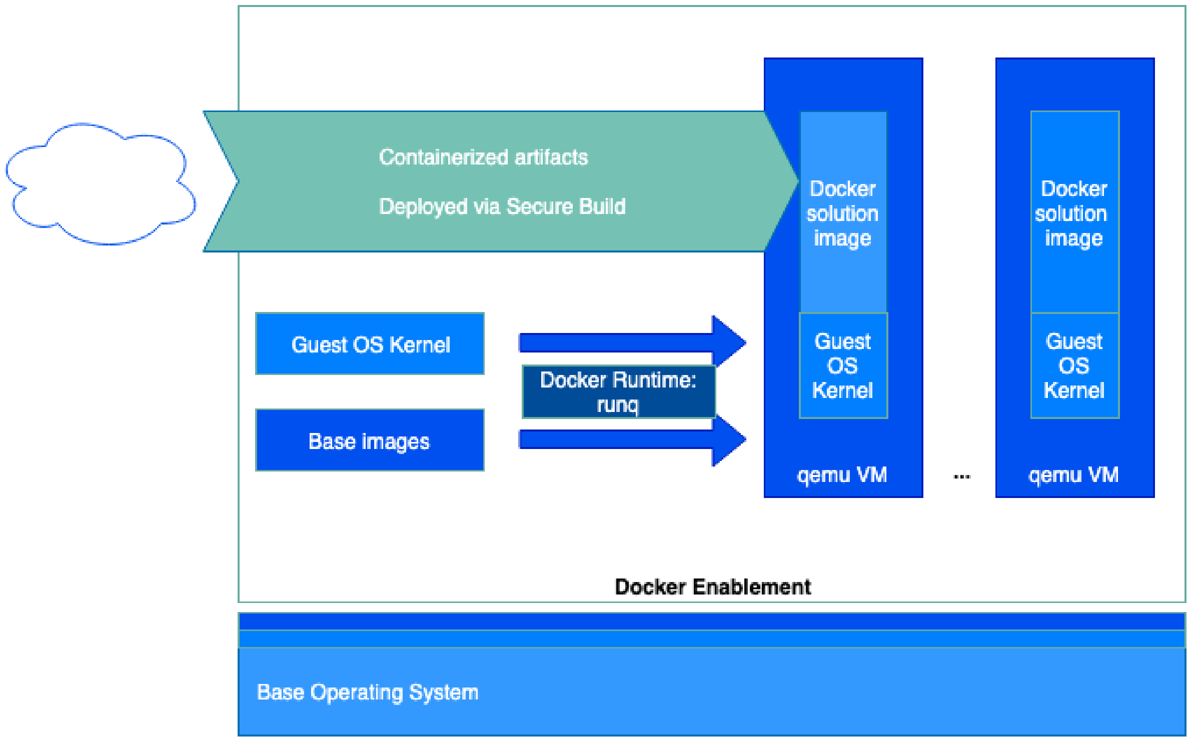


Figure 1. IBM Secure Service Container framework - Docker Enablement

Architecture overview

When using IBM Hyper Protect Virtual Servers, you need to prepare a management server (x86 or Linux on IBM Z or LinuxONE, for example, s390x) to run the commands and manage the components of the offering.

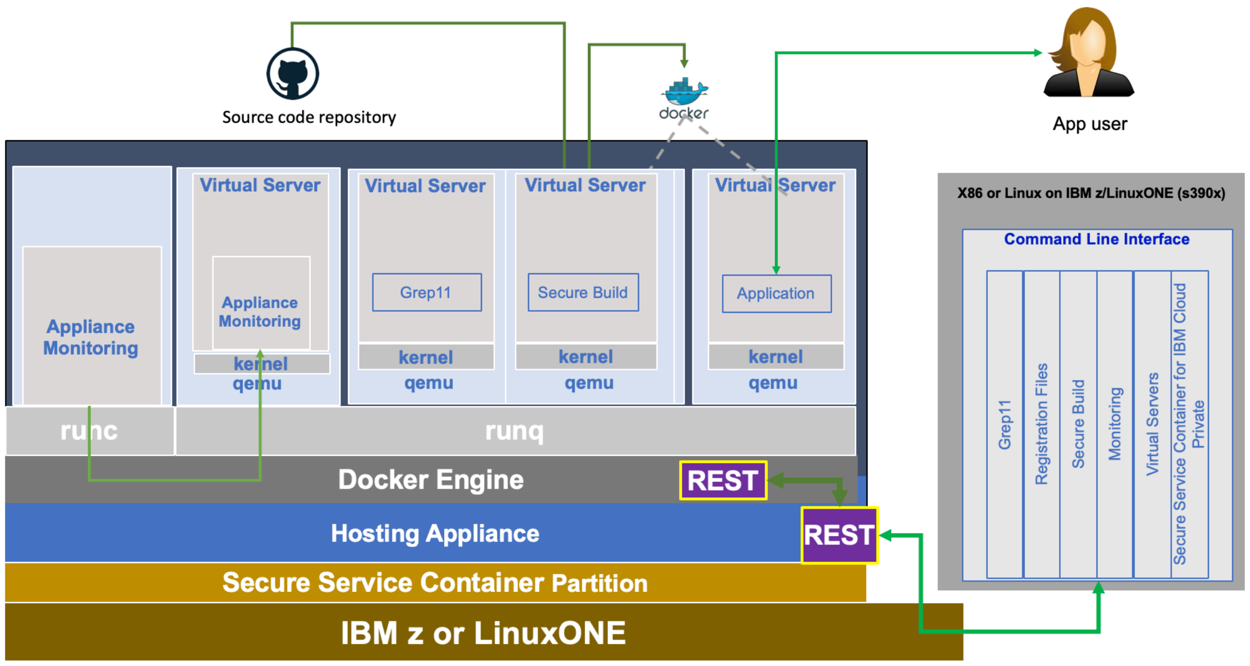


Figure 2. IBM Hyper Protect Virtual Servers - Architecture

The IBM Hyper Protect Virtual Servers offering provides a list of commands with the following capabilities across the application lifecycle phases:

- Build
  - Build user-provided source code (located in a git repository) into Linux on IBM Z / LinuxONE (i.e. s390x) compatible workloads
  - Create Hyper Protect Virtual Server containers on the Secure Service Container partition based on images in the git repository
- Register
  - Download a repository definition file template from the hosting appliance
  - Encrypt a repository definition file with security keys
- Deploy
  - Deploy workloads into Hyper Protect Virtual Server containers on the Secure Service Container partitions
  - Install IBM Cloud Private into the Secure Service Container partitions. For more details, see Working with Secure Service Container for IBM Cloud Private).
- Manage
  - Manage Hyper Protect Virtual Server container images
- Monitor
  - Monitor IBM Hyper Protect appliance health such as the usage of CPU, memory, disk, and uptime.
- Crypto
  - Provide Enterprise PKCS #11 (EP11) interfaces for crypto operations such as key generation, encryption, decryption, data wrapping and unwrapping in EP11 over gRPC (grep11) client applications.

IBM Hyper Protect Virtual Servers also leverages Docker Content Trust (DCT), which uses digital signatures for data sent to and received from remote Docker registries on the Secure Service Container partitions. For more information about the DCT, see Content trust in Docker.

By using IBM Hyper Protect Virtual Servers, your repository and containerized images are protected with different keys on different stages.

Key Name	**Originator / Owner**	**Location**	**Function**	**Lifecycle Phase**
IBM Key Pair	IBM	• Public key or certificate: CLI tool • Private key: Hosting appliance	• Public key or certificate: Encrypt repository definition files • Private key: Decrypt repository definition files	• Public key or certificate: Application registration • Private key: Application deployment
Repository signing key pair	IBM	• Public key or certificate: Remote Docker repository • Private key: Hosting appliance	• Public key or certificate: Verify images built by Secure Build • Private key: Sign images built by Secure Build	Application build (First time)

Key Name	**Originator / Owner**	**Location**	**Function**	**Lifecycle Phase**
Image signing key pair	ISV or app developer	• Public key or certificate: Sent to cloud admin(dev) • Private key: Hosting appliance	• Public key or certificate: Cloud admin verifies signature of the repository definition file and images built by the Secure Build • Private key: Sign the repository definition file and images built by Secure Build	• Public key or certificate: Application registration • Private key: Application Registration
Secure Build initialization key pair	• ISV or app developer • Cloud admin	• Public key or certificate: Sent to cloud admin(dev) • Private key: Local file system	• Public key or certificate: Creates the Secure Build container on the Secure Service Container partition • Private key: initialize the Secure Build container so that the Secure Build container only accepts the API calls encrypted with this private key.	• Public key or certificate: Secure Build initialization • Private key: Secure Build invocation
Secure Build manifest key pair	Secure Build container	• Public key or certificate: Sent to audit(dev) • Private key: Hosting appliance	• Public key or certificate: can be retrieved from the Secure Build container to audit the manifest • Private key: Sign the manifest during the Secure Build	• Public key or certificate: Manifest audit • Private key: Manifest signature

Key Name	Originator / Owner	Location	Function	Lifecycle Phase
Monitoring infrastructure (server-side) key pair	Cloud admin	• Public key or certificate: Local file system • Private key: Local file system	• Public key or certificate: Enable TLS encryption for monitoring infrastructure • Private key: Enable TLS encryption for monitoring infrastructure	• Public key or certificate: Collecting monitoring metrics • Private key: N/A
Monitoring client key pair	Cloud admin	• Public key or certificate: Local file system • Private key: Local file system	• Public key or certificate: Enable mutual TLS communication • Private key: Enable TLS encryption for the client tool	• Public key or certificate: Collecting monitoring metrics only if client authentication is enabled • Private key: N/A
GREP11 container key pair	Cloud admin	• Public key or certificate: Hosting appliance • Private key: Local file system	• Public key or certificate: Authenticate secure communication between GREP11 container and client apps • Private key: Encrypt the requests from GREP11 client apps	• Public key or certificate: Invoking GREP11 calls • Private key: N/A

Components

IBM Hyper Protect Virtual Servers consists of the following components:

- A hosting appliance that is based on the IBM Secure Service Container framework, which can host containerized workloads with focus on superior data security in the cloud and on-premise.
- An isolated VM image that is used to host IBM Cloud Private proxy and worker nodes, and delivers VM level isolation to containerized applications.
- Base images of Hyper Protect Virtual Server container (`hpvsop-base` and `hpvsop-base-ssh`), which can be used to host your application code. The `hpvsop-base-ssh` base image provides additional SSH daemon for debugging and testing.
- A base image of the Secure Build container (`secure-docker-build`), which can be provisioned on the Secure Service Container partition and bound to build your application code exclusively.
- Base images of the monitoring infrastructure (`collectd-host` and `monitoring-host`), which can be used to collect metrics from Secure Service Container framework.

- A base image of the Enterprise PKCS #11 (EP11) over gRPC (Grep11) container (`grep11-container`), which can communicate with Hardware Security Module (HSM) and generates asymmetric (public and private) key pairs.
- A set of command line tools that are used to:
  - Automate the base infrastructure of the IBM Cloud Private worker and proxy nodes by using the isolated VM image
  - Create and manage the Hyper Protect Virtual Server containers
  - Securely build and publish your applications as containerized workloads
  - Deploy your containerized workloads to the Secure Service Container framework
  - Monitor IBM Hyper Protect appliance health such as the usage of CPU, memory, disk, and uptime.
  - Provide Enterprise PKCS #11 (EP11) interfaces for crypto operations such as key generation, encryption, decryption, data wrapping and unwrapping in EP11 over gRPC (grep11) client applications.
- A set of files that you can use to install the Secure Service Container for IBM Cloud Private. For more information about this component, see Working with Secure Service Container for IBM Cloud Private.

Table 1. IBM Hyper Protect Virtual Servers components

Component	**Version**
Hosting appliance	1.1.18
Isolated VM	1.2.0
`hpvsop-base` and `hpvsop-base-ssh`	1.2.0
`secure-docker-build`	1.2.0
`collectd-host` and `monitoring-host`	1.2.0
`grep11-container`	1.2.0
Command line tool	1.2.0
IBM Secure Service Container for IBM Cloud Private	1.1.0.3

The command line tool is composed of the following modules:

- Repository commands
- Secure Build commands
- Registration files commands
- Hyper Protect Virtual Server containers commands
- Images commands
- quotagroup commands
- Monitoring commands
- Grep11 commands
- Snapshot commands
- Secure Service Container for IBM Cloud Private commands

See Downloading the installation package for the information on how to get these components.

System requirements

Software, hardware, and system configuration settings that are required for setting up a Hyper Protect Virtual Server offering.

For the system requirements to install IBM Cloud Private, see System requirements for IBM Secure Service Container for IBM Cloud Private.

Hardware requirements for the Linux management server

The x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server is used to download the Hyper Protect Virtual Server installation binary, install IBM Hyper Protect Virtual Servers CLI tool, and using this tool, perform and manage configuration tasks for IBM Hyper Protect Virtual Servers , including on the Hyper Protect hosting appliance or partition.

Table 1. 64-bit x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server requirements

Minimal requirement
4 or more x86 Linux cores with at least 2.4 GHz, or 1 Integrated Facility (IFL) on mainframe
8 GB RAM
150 GB disk space

Hardware requirements for Secure Service Container partition

You can configure Secure Service Container partitions on the following IBM Z and LinuxONE systems.

- IBM z15 (z15)
- IBM z14 (z14) (machine type 3906 or 3907)
- IBM LinuxONE III (LinuxONE III)
- IBM LinuxONE Emperor II (Emperor II), or IBM LinuxONE Rockhopper II (Rockhopper II)

The suggested practice is to use the latest available firmware for Secure Service Container, which is identified by the engineering changes (ECs) in the following table. To find the latest available EC microcode control levels (MCLs) for Secure Service Container, use the instructions for hardware updates in "Prerequisites for using Secure Service Container" after you download Secure Service Container User's Guide from the About topic.

Table 2. Engineering changes by machine type

Machine Type	Version / Driver	Bundle	Engineering Changes
8561	Version 2.15.0Driver 41	S10 or later	• SE-BCBASE P46639 • SE-BCBOOT P46640 • SE-BCINST P46655
3906 or 3907	Version 2.14.1 Driver 36	S12 or later	• SE-BCBASE P41453 • SE-BCBOOT P41454 • SE-BCINST P41467
3906 or 3907	Version 2.14.0 Driver 32	S53 or later	• SE-BCBASE P42638 • SE-BCBOOT P42639 • SE-BCINST P42652

The following table shows the minimal requirement for one Secure Service Container partition, which hosts only one Hyper Protect Virtual Server container.

Table 3. Secure Service Container partition requirements

Minimal (one Hyper Protect Virtual Server container + one Secure Build container)
2 IFLs
12 GB RAM
190 GB storage (50 GB for the hosting appliance, 100 GB in the storage pool for one Hyper Protect Virtual Server container, and 40 GB for one Secure Build container)

Note:

- The actual resources required on the Secure Service Container partition depends on the resource consumption of your workload to be deployed into the Hyper Protect Virtual Server container.
- If you plan to have multiple Hyper Protect Virtual Server containers or Secure Build containers communicating with each other on the Secure Service Container partition, and assign IP addresses to each of them, you need to use at least 1 Open System Adapter (OSA) card to create multiple virtual devices for data traffic.
- You must use different Secure Service Container partitions for Secure Service Container for IBM Cloud Private, or IBM Hyper Protect Virtual Servers. For more information about Secure Service Container for IBM Cloud Private, see Working with Secure Service Container for IBM Cloud Private.
- If you want to use Enterprise PKCS #11 over gRPC (GREP11) containers in IBM Hyper Protect Virtual Servers, you must prepare a Trusted Key Entry (TKE) workstation and Crypto Express cards, such as IBM Crypto Express6s (CEX6S) and IBM Crypto Express7s (CEX7S). The Crypto Express will differ by machine generation (CEX6S for the z14 generation, CEX7S for the z15 generation).

Software requirements

- IBM PCIe Cryptographic Coprocessor Version 3 (PCIeCC3) software, which includes IBM Common Cryptographic Architecture (CCA) and Enterprise PKCS #11 (EP11), and be ordered from Cryptocards software-package selection page.

Supported operating systems and platforms

The operating system for running the containers on the Secure Service Container partitions is Ubuntu 18.04, which is provided by the hosting appliance.

However, you must configure the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server with the supported operating system in the following table.

Table 4. Supported operating system and platform

Platform	Operating system
Linux 64-bit	Ubuntu 16.04 LTS and 18.04 LTS

Note:

- Redhat, SUSE Linux, or other Linux distributions are compatible operating systems for the management server, but they have not been tested with IBM Hyper Protect Virtual Servers. Use one of those operating systems at your own risks.
- Linux Unified Key Setup (LUKS) hardware encryption on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server can protect the hardware from faulty access. When installing the Ubuntu onto the x86 or Linux on Z server, select the **Encrypt the new Ubuntu installation for Security** option to encrypt the hard disk.

Networking

IBM Hyper Protect Virtual Servers requires two levels of network to work properly.

- Network among Hyper Protect Virtual Server containers by using the internal IP addresses

- Network for external requests to the services inside the workload deployed in the Hyper Protect Virtual Server container

Table 5. Supported network interfaces on the Secure Service Container partitions

Interface	Layer 2 network	Layer 3 network
Ethernet	Yes	Yes
VLAN	Yes	Yes

On the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, network connection must be available to the Secure Service Container partition by using its IP address or host name.

Note:

- The default network driver is `bridge` and sufficient for communication among Hyper Protect Virtual Server containers.
- If you plan to access Hyper Protect Virtual Server containers from your underly network or being accessed by external workload, use the network driver `macvlan` and assign IP addresses to those containers, or configure the port mapping for the container on the Secure Service Container partition.
- If you plan to access Hyper Protect Virtual Server containers on another Secure Service Container partition, use the network driver `macvlan` and assign IP address to the containers on both partitions.
- For more information, see Networking overview for Docker containers.

Supported Docker versions

You must install the supported Docker version on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.

- For the x86 management server, the minimum Docker version required by IBM Hyper Protect Virtual Servers is V19.03.2 or above.
- For the Linux on IBM Z/LinuxONE (such as s390x architecture), the minimum Docker version required by IBM Hyper Protect Virtual Servers is v18.06.3-ce or above.

Note: You can only use IBM Hyper Protect Virtual Servers with Docker Hub or IBM Container Registry.

Required ports

If you use port mapping for Secure Build container, Monitoring infrastructure, and GREP11 container, ensure that the following ports are available on the Secure Service Container partition. Otherwise, You need to request IP address for each container on the Secure Service Container partition.

Table 6. Required ports on the Secure Service Container partition

Port No.	Required by Module
443	Secure Build container
8443	Monitoring infrastructure
9876	GREP11 container

User roles

IBM Hyper Protect Virtual Servers distinguishes between different types of administrators and users to perform tasks:

- Application developer or ISV

The application developer or independent software vendor (ISV) is responsible for developing and publishing containerized applications or solutions to the private cloud environment, and to ensure the security, integrity, and audit requirements of the software.

- Private cloud operations administrator

  The private cloud operations administrator (cloud admin) is responsible for infrastructure, security, and management of the on-premises, shared, and multi-tenant private cloud environment for containerized applications.
- Appliance administrator

  The appliance administrator (appliance admin) is responsible for deploying and managing the hosting appliances, that runs in a logical partition (LPAR) on an IBM Z or IBM LinuxONE server.
- IBM Z or LinuxONE system administrator

  The IBM Z or LinuxONE system admin (Z system admin) is responsible for creating and managing Secure Service Container partitions by performing tasks on the HMC of the IBM Z or IBM LinuxONE server.

Those roles might also interact with:

- IBM Z or LinuxONE storage admin
- Network admin
- Solution admin
- Containerized application user

FAQs

What is IBM Hyper Protect Virtual Servers?

IBM Hyper Protect Virtual Servers provides a secure virtualized infrastructure for private cloud deployments - protecting the entire lifecycle of critical Linux workloads during their build, deployment and management.

As an application developer or ISV, how can I benefit from IBM Hyper Protect Virtual Servers?

Application developers and ISVs can securely build applications with integrity.

As a cloud administrator or system administrator, how can I benefit from IBM Hyper Protect Virtual Servers?

Cloud administrators or system administrators can help manage their layer of the IT infrastructure without having access to the higher level applications and sensitive data.

As a solution end-user, how can I benefit from IBM Hyper Protect Virtual Servers?

Solution end-users can ensure the provenance of the applications being deployed by validating that applications originate from a trusted source.

What is Secure Service Container Framework?

Secure Service Container framework provides the base infrastructure for an integration of operating system, middleware, and software components into an appliance with extra security. In addition to extra security based on the `runq` container environment, the host operating system itself is also extremely secure. The Secure Service Container framework works autonomously and provides core services and infrastructure focusing on consumability and security.

What is a hosting appliance?

A hosting appliance is a software appliance built with the Secure Service Container Framework, and adds the capability to securely run containerized workloads.

What is a software appliance?

A software appliance is an integrated software containing an operating system, libraries, and so on to fulfill a single purpose, which can be installed as an appliance image on IBM Z or LinuxONE servers.

Can I deploy an application as is or is containerizing my application required to use IBM Hyper Protect Virtual Servers?

As long as your applications are developed based on Open Container Initiative (OCI) specification, you can use them in IBM Hyper Protect Virtual Servers.

Can I use my own private key to sign the images for the Docker Content Trust?

Yes. You can either use the `docker trust key generate` command to generate the signing key pair, or use the `docker trust key load` command to load an existing key for signing. However, passing in an existing key pair would invalidate the Secure Build concept as the private trust key exists(existed) outside of the Secure Build, and therefore someone else could use that key to push a bad image to the same docker repo.

What happens when I run the `docker push` command against a DCT-enabled repository?

The `docker push` command establishes trust at the time the first push to the docker repo is done. The command uses DOCKER_CONTENT_TRUST environment variables to determine where to establish the trust with.

Where is the Secure Build container?

The Secure Build container is created on the hosting appliance when you run the `securebuild create` command.

When is the docker repo key pair generated?

The docker repo key pair is generated on the first build when the `docker push` command is executed.

What are manifest signing keys?

The manifest signing keys are generated inside the Secure Build server container on first creation of a manifest by a Secure Build instance. It then uses gpg to sign the manifest tar file and will optionally push that signed tar to an external Cloud Object Store. The manifest and public key to validate the signature can also be retrieved from the Secure Build using the cli.

Can the Secure Build server container be used to build an existing docker image on the Docker Hub?

Yes. The newly built image must have a new name so that DCT can be established by the Secure Build server container.

Setting up the environment

The following topics shows how to prepare the environment when using IBM Hyper Protect Virtual Servers .

1. Planning for the environment
2. Downloading the installation package
3. Creating the Secure Service Container Partition
4. Installing the Hyper Protect hosting appliance
5. Configuring the storage on the Secure Service Container partition
6. Configuring the network on the Secure Service Container partition
7. Installing the IBM Hyper Protect Virtual Servers CLI tool

Planning for the environment

You can use a PLANNING FOR YOUR IBM HYPER PROTECT VIRTUAL SERVERS WORKSHEET or the tables listed on this topic to get an overall understanding of what information you will need to run the offering, and where to get such information.

Before you begin

- Ensure that you have the required hardware, softwares, network devices, and ports ready as listed on the System requirements.

Management server

The following table shows the required information for the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.

Table 1. Management server checklist

#	Resource	The actual value	Example	Where to get
1	Architecture	x86 or s390x Linux	s390x	Cloud administrator
2	Host name		`management_server`	`hostname`
3	Primary Network interface Controller (NIC)		eth0	`ifconfig -a`
4	External IP address for NIC		10.152.151.100	`ifconfig -a` (inet addr parameter in the result)
5	Password for the user `root`		`root_user_password`	System administrator
6	Internal IP address		`192.168.40.251`	Network administrator
7	NIC for internal network		`eth1`	Network administrator

#	Resource	The actual value	Example	Where to get
8	Subnet mask for internal IP		192.168.40.0/24	Network administrator
9	Gateway for internal IP		192.168.40.1	Network administrator

To configure multiple aliases to one network interface controller (NIC) on the management server, see IP-Aliasing.

Secure Service Container partitions

The following table shows the required information you will need when configuring Secure Service Container storage.

Table 2. Secure Service Container partition checklist

#	Resource	The actual value	Example	Where to get
1	Partition IP address		`10.152.151.105`	System administrator
2	Master ID		`ssc_master_user`	System administrator
3	Master password		`ssc_master_password`	System administrator
4	Storage disks for quotagroups resizing		`3600507630affc427000000000002000` (FCP) or `0.0.78CA` (FICON DASD)	System administrator

Note: If you plan to use multiple Secure Service Container partitions, make sure you have a checklist for each partition.

Secure Build containers

The following table shows the required information you will need to create a Secure Build container on the Secure Service Container partition.

Table 3. A Secure Build container checklist

#	Resource	The actual value	Example	Where to get
1	Partition IP address		`10.152.151.105`	System administrator
2	Secure Build container name		`securebuild1`	Cloud administrator
3	CPU thread number (vcpu)		`2`	System administrator
4	Memory (MB)		`2048`	System administrator
5	Storage for the Secure Build container application (GB)		`10`	System administrator

#	Resource	The actual value	Example	Where to get
6	Storage for the Docker images built by Secure Build (GB)		16	System administrator
7	Storage for logs configuration data for the Secure Build Container (GB)		2	System administrator
8	Quotagroup of Secure Build container		`myquotagroup`	Cloud administrator
9	Connection method (port-mapping/IP)		`IP`	System administrator
10	Internal network name		`encf900_internal_network`	Cloud administrator
11	Internal IP address (Only needed if an internal network is being used.)		`192.168.40.6`	Cloud administrator
12	External IP address (Only needed if an external network is being used.)		`164.23.2.77`	System administrator
13	Forward port for external (Only needed if an external IP address is not assigned.)		`10433`	System administrator
14	Repository ID of the Secure Build container image		`SecureDockerBuild`	Cloud administrator
15	Tag of the Secure Build container image		`latest`	Cloud administrator
16	Repository ID for your apps		`MyDockerApp`	Cloud administrator
17	Source code repository URL		`github.com:MyOrg/my-docker-app.git`	App developer or ISV
18	Source code branch		`dev`	App developer or ISV
19	Private key for Source code repository			App developer or ISV

#	Resource	The actual value	Example	Where to get
20	Remote docker registry server		`docker.io`	Cloud administrator
21	Remote docker repository name for built images		`docker_base_user/MyDockerApp`	Cloud administrator
22	Remote docker registry user name to register the base images		`docker_base_user`	Cloud administrator
23	Remote docker registry user password to register the base images		`passw0rd`	Cloud administrator
24	Remote docker registry user name to push the images		`docker_writable_user`	Cloud administrator
25	Remote docker registry user password to push the images		`passw0rd`	Cloud administrator
26	Cloud Object Storage service API key (Optional)		`0viPH...kliJ`	Cloud administrator
27	Cloud Object Storage service bucket (Optional)		`my-cos-bucket1`	Cloud administrator
28	Cloud Object Storage service resource crn (Optional)		`crn:v1...::1`	Cloud administrator
29	Cloud Object Storage service auth_endpoint (Optional)		`iam.cloud.ibm.com`	Cloud administrator
30	Cloud Object Storage service end_point (Optional)		`s3.....cloud`	Cloud administrator

For more information, see Building the container image for your application by using the Secure Build.

Repository definition files

The following table shows the required information you will need to encrypt a repository definition file.

Table 4. A repository definition file checklist

#	Resource	The actual value	Example	Where to get
1	Repository name		`docker.io/docker_base_user/MyDockerApp`	Cloud administrator
2	Readonly Docker Hub user ID		`docker_readonly_user`	Cloud administrator
3	Docker Hub User password		`docker_password`	Cloud administrator
4	The public key		`isv_user.pub`	App developer or ISV
5	The private key		`isv_user.private`	App developer or ISV

For more information, see Creating and encrypting the repository definition file.

Hyper Protect Virtual Server containers

The following table shows the required information you will need to create a Hyper Protect Virtual Server container on the Secure Service Container Partition.

Table 5. A Hyper Protect Virtual Server container checklist

#	Resource	The actual value	Example	Where to get
1	Partition IP address		`10.152.151.105`	System administrator
2	External network name		`encf900_network`	Cloud administrator
3	Container external IP address		`164.20.5.78`	cloud administrator
4	Internal network name		`encf900_internal_network`	Cloud administrator
5	Internal IP address		`192.168.40.188`	Cloud administrator
6	Parent device		`encf900`	Appliance administrator
7	Gateway		`192.168.40.1`	Cloud administrator
8	Subnet		`192.168.40.0/24`	Cloud administrator
9	Repository name		`MyDockerApp`	Cloud administrator
10	Image tag		`latest` or `1.2.0-abcedfg`	Cloud administrator
11	CPU threads number (vcpu)		2	Cloud administrator
12	Memory size (MB)		2048	Cloud administrator

#	Resource	The actual value	Example	Where to get
13	Quotagroup size (GB)		`100G`	Cloud administrator

For more information, see Deploying your application into the Hyper Protect Virtual Server container.

Monitoring

The following table shows the required information you will need to set up the monitoring infrastructure for IBM Hyper Protect Virtual Servers.

Table 6. Monitoring infrastructure checklist

#	Resource	The actual value	Example	Where to get
1	Partition IP address		`10.152.151.105`	System administrator
2	Domain suffix		`first`	System administrator
3	DNS name		`example.com`	System administrator
4	Connection method (port-mapping/IP)		`8443`	System administrator
5	Private key for the monitoring infrastructure		`server.key`	`openssl` utility
6	Certificate for the monitoring infrastructure		`server-certificate.pem`	`openssl` utility
7	Certificates for the monitoring client		`client-certificate.pem`	`openssl` utility

For more information, see Monitoring IBM Hyper Protect Virtual Servers.

Grep11

The following table shows the required information you will need to set up the GREP11 container for IBM Hyper Protect Virtual Servers.

Table 7. A GREP11 container checklist

#	Resource	The actual value	Example	Where to get
1	Partition IP address		`10.152.151.105`	System administrator
2	Crypto domain name		`09.000b`	System administrator
3	Domain suffix		`grep11`	System administrator
4	DNS name		`example.com`	System administrator

#	Resource	The actual value	Example	Where to get
5	Connection method (port-mapping/IP)		IP	System administrator
6	Internal network name		`my-private-network-name`	System administrator
7	Ip address		`192.168.10.106`	System administrator
8	TLS key and certificate		`key.pem`, `server-key.pem`	`openssl` utility
9	CA certificate for mutual_TLS (Optional)		`ca.pem`	`openssl` utility

For more information, see Integrating with the EP11 library.

Next

You can download the IBM Hyper Protect Virtual Servers installation package by following the instructions on the Downloading the installation package topic.

Downloading the installation package

You can get IBM Hyper Protect Virtual Servers software package from the IBM Passport Advantage.

This procedure is intended for users with the role *Cloud administrator*.

Before you begin

- Ensure you have the management server ready with one of the supported architectures as required in the **Hardware requirements for management server** section.
- Ensure that you install the OpenSSL or similar tool on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.

Procedure

On your x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, complete the following steps with root user authority.

1. Log in to IBM Passport Advantage website by using your IBM account ID and password. Contact your sales representative if you do not have one.
2. Go to **My Programs**, and then select the **IBM Hyper Protect Virtual Servers** program.
3. Download IBM Hyper Protect Virtual Servers image `CC37UEN.tar.gz` to your x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.
4. On the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server with root user authority, create an installation directory to store IBM Hyper Protect Virtual Servers image and configuration files.

```
mkdir /opt/<installation_directory>
```

5. Change to the installation directory, and extract the compressed file on the x86 on Linux on Z server.

```
cd /opt/<installation_directory>
tar -zxvf CC37UEN.tar.gz
```

You will get the following files in the current directory:

- `CC37UEN.tar.gz`, the offering image tar file.
- `CC37UEN.sig`, the signature file for the offering image.
- `CC37UEN.`pub, the public key issued by IBM for the offering image.

6. To verify the integrity of IBM Hyper Protect Virtual Servers image tar file, run the following example command by using the signature file with the `.sig` suffix and the public key issued by IBM with the suffix `.`pub along with the image tar file.

   ```
   openssl dgst -sha256 -verify CC37UEN.pub -signature CC37UEN.sig CC37UEN.tar.gz
   ```

7. Extract the compressed tar file on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.

   ```
   cd /opt/<installation_directory>
   gunzip CC37UEN.tar.gz
   tar -xvf CC37UEN.tar
   ```

Result

After extracting the `CC37UEN.tar` file, you can see the similar layout of files and directories under the `<installation_directory>` directory.

```
|- CC37UEN.tar
|- hpvs-cli-installer.docker-image.tar
|- License
|- readme.txt
|- secure-service-container-for-icp.appliance.1.1.18.img.gz
|- SSC4ICP
|    |-config
|    |    |- ICPIsolatedvm.tar.gz
|    |- readme.txt
|- version
|- VS
    |- grep11-cli
    |    |- config
    |         |-grep11-config.yaml.example
    |         |-hosts.example
    |         |-hpcsKpGrep11_runq.tar.gz
    |- hpvs-cli
    |    |- config
    |         |- hosts.example
    |         |- hpvs-config.yaml.example
    |         |- hpvs-config.yaml.example.VLAN.L2
    |         |- hpvs-config.yaml.example.VLAN.L3
    |         |- HpvsopBaseSSH.tar.gz
    |         |- HpvsopBase.tar.gz
    |- monitoring-cli
    |   |-config
    |         |- CollectdHost.tar.gz
    |         |- hosts.example
    |         |- monitoring-configuration-delete.yaml.example
    |         |- monitoring-configuration-service.yaml.example
    |         |- monitoring-configuration.yaml.example
    |         |- Monitoring.tar.gz
    |         |- monitoring.yaml.example
    |- readme.txt
    |- regfile-cli
    |    |- config
    |- securebuild-cli
         |- config
            |- hosts.example
            |- hosts.readme
            |- securebuild.yaml.example
            |- securebuild.yaml.example.VLAN.L2
            |- securebuild.yaml.example.VLAN.L3
            |- securebuild.yaml.readme
            |- SecureDockerBuild.tar.gz
```

Note:

- `readme.txt`, which is the general README file for IBM Hyper Protect Virtual Servers .

- `License`, a directory that contains the license files of IBM Hyper Protect Virtual Servers .
- `version`, which states the current version of IBM Hyper Protect Virtual Servers .
- `hpvs-cli-installer.docker-image.tar`, which contains the command line tools required to set up IBM Hyper Protect Virtual Servers environment, or set up the Secure Service Container for IBM Cloud Private. For the details about the Secure Service Container for IBM Cloud Private, see Working with Secure Service Container for IBM Cloud Private.
- `secure-service-container-for-icp.appliance.1.1.18.img.gz`, which is the hosting appliance to be installed on the IBM Z or LinuxONE system. Note that this hosting appliance image is identical with Hyper Protect hosting appliance version v3.7.7.
- `./VS/grep11-cli/config`, a directory that contains configuration example of `hosts` and `grep11-config.yaml` files.
- `./VS/grep11-cli/config/hpcsKpGrep11_runq.tar.gz`, which is the base image of the GREP11 container and must be under the `./VS/grep11-cli/config`.
- `./VS/hpvs-cli/config/HpvsopBase.tar.gz`, which is the base image of a Hyper Protect Virtual Server container without the secure shell (SSH) access.
- `./VS/hpvs-cli/config/HpvsopBaseSSH.tar.gz`, which is the base image of a Hyper Protect Virtual Server container with the secure shell (SSH) access.
- `./VS/hpvs-cli/config`, a directory that contains configuration example of `hosts` and `hpvs-config.yaml` files for Hyper Protect Virtual Server containers.
- `./VS/monitoring-cli/config`, a directory that contains configuration example of `hosts` and `monitoring.yaml` files for the monitoring infrastructure.
- `./VS/monitoring-cli/config/CollectdHost.tar.gz`, which is the base image of collectd-host container of the monitoring infrastructure. The file must be under the `./VS/monitoring-cli/config` directory.
- `./VS/monitoring-cli/config/Monitoring.tar.gz`, which is the base image of monitoring-host container of the monitoring infrastructure. The file must be under the `./VS/monitoring-cli/config` directory.
- `./VS/regfile-cli/config`, which is an empty directory upon initial extract, and can be used to contain configuration files for repository definition files.
- `./VS/securebuild-cli/config/SecureDockerBuild.tar.gz`, which is the docker image of the Secure Build container.
- `./VS/securebuild-cli/config`, a directory that contains configuration example files (with the `.example` extension) and readme files (with the `.readme` extension) of the `hosts` and `securebuild.yaml` files for Secure Build containers.
- `./SSC4ICP/config`, a directory that contains configuration example files for IBM Secure Service Container for IBM Cloud Private.
- `./SSC4ICP/config/ICPIsolatedvm.tar.gz`, which is the isolated VM image for hosting proxy and worker nodes of Secure Service Container for IBM Cloud Private. For more details, see Working with Secure Service Container for IBM Cloud Private.

Next

You can create and start the Secure Service Container partition as instructed in the Creating the Secure Service Container partition topic.

Creating the Secure Service Container partition

Use this procedure to configure a Secure Service Container partition on a host system and afterwards install IBM Hyper Protect Virtual Servers on that partition.

This procedure is intended for users with the role *system administrator*.

Before you begin

- Ensure that the hosting system is one of supported servers as required in the **Hardware requirements for Secure Service Container partition** section.
- Check that you download Secure Service Container User's Guide from the About topic.
- Refer to the checklist that you prepared on this topic Planning for the environment.

Procedure

To create and manage Secure Service Container partitions, you can use specific tasks on the Hardware Management Console (HMC) for a host system running either in standard mode (that is, with Processor Resource/System Manager or PR/SM), or with Dynamic Partition Manager (DPM) enabled. For more information about the host system, see the appropriate overview on the IBM® Redbooks® website at http://www.redbooks.ibm.com/. For example, for the z15, see the IBM z15 Technical Introduction, SG24-8850.

- For a host system (CPC) running in standard mode
  1. Open the **Customize/Delete Activation Profiles** task, and then select **SSC** mode on the **Customize Image Profiles** page.
  2. Configure the processor requirements on the **Processor** page, specify partition security options on the **Security** page, and specify the amount of storage required on the **Storage** page.
  3. Provide or modify any cryptographic controls as appropriate on the **Crypot** page.
  4. On the **SSC** page, ensure the **Secure Service Container installer** option is selected under **Boot selection** if you create the partition for the first time, and then provide values for the default master user ID, password, and IP address of the network adapter for the Secure Service Container partition.
  5. Click **Save** to save the changes and wait for the partition to be created.
  6. Select the image of the Secure Service Container partition, and start the Secure Service Container partition by using the **Activate** task.
- For a host system with DPM enabled
  1. Open the HMC **New Partition** task, and then select **Secure Service Container** as the **Partition Type** from the list.
  2. Provide the values for the master user ID and password as prompted.
  3. Define the number of virtual processors for the partition on the **Processor** page, define the initial and maximum amounts of memory to be assigned to the partition on the **Memory** page. The minimal initial amount of a Secure Service Container partition is 4 GB.
  4. Define all of the network interface cards (NICs) for the partition on the **Network** page. For a Secure Service Container partition, you must also specify at least one NIC for communication with the Secure Service Container web interface.
  5. Attach storagegroups or create host bus adapters (HBAs) for the partition on the **Storage** page.
  6. Configure the cryptographic features on the **Cryptos** page as needed.
  7. On the **Boot** page, note that option set in the "Boot from" menu is **Secure Service Container**. This boot option cannot be changed unless you first change the partition type.
  8. Click **Finish** to save the partition definition, and then wait for DPM creating the partition.
  9. Select the image of the Secure Service Container partition, and activate the partition by selecting **Yes** on the **Start** task page.

Note:

- Write down the following values specified in the image profile (standard mode system) or the partition definition (DPM-enabled system) for the Secure Service Container partition when you configure the Secure Service Container. You will need them when configuring the appliance network and creating cluster nodes.
  – Master user ID

    - Master password
    - IP address
- For more detailed information on how to create and start the Secure Service Container partition, see Secure Service Container User's Guide from the About topic.

Next

You can install the hosting appliance onto the Secure Service Container partition by following the instructions on Installing the Hyper Protect hosting appliance.

Installing the Hyper Protect hosting appliance

Use this procedure to install and start the Hyper Protect hosting appliance in a {{site.date.keyword.ssc}} partition on the IBM z or LinuxONE server.

This procedure is intended for users with the role *appliance administrator*.

Note:

- The Hyper Protect hosting appliance is an enhanced version of the IBM Secure Service Container software appliance.
- The Hyper Protect hosting appliance displays with the name **IBM Secure Service Container** on the Secure Service Container user interface.
- The Hyper Protect hosting appliance uses all of the IBM Secure Service Container documentation and techniques to install, administer, and maintain.
- The Hyper Protect hosting appliance version numbering scheme is unique to the Hyper Protect hosting appliance, as opposed to the general Secure Service Container verion numbering scheme.
- Only one appliance can be installed and run in a Secure Service Container partition at any given time; this type of partition does not support running multiple appliances simultaneously. You can define more than one Secure Service Container partitions on the same system, and run instances of the same appliance in each one. In this case, each partition must use separate storage devices.

Before you begin

- Check that you have the appliance image `secure-service-container-for-icp.appliance.<version_number>.img.gz` in the installation directory. For instructions, see Downloading the installation package
- Check that you have the Secure Service Container partition created to install the hosting appliance as instructed in the Creating the Secure Service Container partition topic.
- Check that you download Secure Service Container User's Guide from the About topic.

Procedure

Complete the following tasks through the browser of your choice.

1. Log in to the Secure Service Container installer by using the master user ID and password in your browser. For example, `https://<secure_service_container_partition_ip_address>`.
2. On the main page, click the plus (+) icon to install image files from local disk. The page display changes to the **Install Software Appliance** page.
3. On the **Install Software Appliance** page, select the **Upload image to target disk** option, and then locate the appliance image file on your local disk under the **Local Installation Image** section.
4. Under **Target Disk on Server**, select the device type **FICON DASD** or **FCP**, and then click **Apply** to upload the appliance image to the target disk on the server. **Note:**
    - You can only specify one type of disk (either DASD or FCP) during the appliance installation stage.

    - Target FCP disks must be large enough to fit the uncompressed appliance, with an additional 2 GB for the Secure Service Container installer to use.
5. Click **Reboot** on the confirmation diaglog to have the installer automatically reactivate the partition. The Secure Service Container installer uploads the appliance image to the target disk, and prepares the partition to load the appliance after the next reboot. a. When the reboot process begins, the installer displays the Reboot window. b. If an IP address type other than DHCP is in use for the appliance page, the Secure Service Container installer redirects the browser to the software appliance page.
6. On the appliance page, accept the self-signed certificate for the SSL connection, and log in to the Secure Service Container user interface by using the master user ID and password.

For more detailed instructions, see the following topic after you download Secure Service Container User's Guide from the About topic.

- Chapter 13 - Installing a new software appliance in a Secure Service Container partition

Next

You can configure the storage on the Secure Service Container partition as instructed in the Configuring the storage on the Secure Service Container partition topic.

Configuring the storage on the Secure Service Container partition

Use this procedure to make resources like storage devices assigned to the Secure Service Container partition available in IBM Hyper Protect Virtual Servers. These resources can then be utilized by the containerized applications running on the Secure Service Container partition.

This procedure is intended for users with the role *appliance administrator*.

Before you begin

- Check with the cloud administrator the list of requirements of the containerized application to assign sufficient resources (disk space, network adapters) to IBM Hyper Protect Virtual Servers.
- Check with the IBM Z or LinuxONE system administrator that sufficient resources are assigned to the Secure Service Container partition to fit the requirements of the containerized application.
- Check with the IBM Z or LinuxONE system administrator to get the disk IDs that can be used for the Secure Service Container partition.
- Check that you download Secure Service Container User's Guide from the About topic.
- Refer to the checklist that you prepared on this topic Planning for the environment.

Procedure

Complete the following steps under the `<installation_directory>/VS/hpvs-cli/config` directory with root user authority.

1. Add the storage disks to the storage pool on the Secure Service Container partition. Choose one of the following options to add the disks.
    - Use the Secure Service Container user interface to manage the storage resources. For the instructions, see *Viewing and managing storage resources* section after you download Secure Service Container User's Guide from the About topic.
    - Add the disks into the storage pool by using the `disk add` command.

    ```
      docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
    disk add --disk-id 0.0.78CA --lpar [LPARIPORADDRESS]
    ```

    Note:

- The default storage disk type is `FICON DASD`. You can use the `--sub-resource fcp-disks` parameter for the FCP storage type.
- To add multiple disks into the storage pool, use whitespace as the separator for the value of the `--disk-id` parameter. For example, `--disk-id 0.0.78CA 0.0.80AC`.
- To verify the current status and completion status of the `disk add` command, run the `disk get` command because the completion of the `disk add` command might require several minutes or more.

```
  docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
disk get --lpar [LPARIPORADDRESS]
```

- For a full list of supported actions and parameters in the `disk` commands, see Commands for disks.

2. Configure the quotagroup size for the hosting appliance on the Secure Service Container partition by using the `quotagroup` commands.

   a. Get the quotagroup size by using the `quotagroup get` command.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
quotagroup get --all --lpar [LPARIPORADDRESS]
```

   b. Update the quotagroup size for the hosting appliance by using the `quotagroup update` command. The unit of the `--size` paratmer is GB.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
quotagroup update --name appliance_data --size 50 --lpar [LPARIPORADDRESS]
```

   For a full list of supported actions and parameters in the `quotagroup` commands, see Commands for quotagroups.

Note: The quotagroups for Hyper Protect Virtual Server containers and Secure Build containers can be created and allocated from the storage pool when those containers are created on the Secure Service Container partition.

Next

You can configure the network devices by following the instructions on Configuring the network on the Secure Service Container partition.

Configuring the network on the Secure Service Container partition

You can configure the network devices for the hosting appliance by using the Secure Service Container user interface. The containers on the Secure Service Container partitions communicate through the Ethernet-type or VLAN-type connections over the network devices bound to Open Systems Adapter-Express (OSA-Express) devices.

If you want the Hyper Protect Virtual Server container on the Secure Service Container partition to be accessed by external services, you must configure two network devices with one for internal communication, and another for external access. You can configure one network device to each of the OSA-Express devices on the Secure Service Container partitions, or multiple network devices on one OSA-Express device.

This procedure is intended for users with the role *appliance administrator*.

Before you begin

- Check that you have the connection information to each Secure Service Container partition. For more information, see Creating Secure Service Container partitions.
- Check that you install the hosting appliance by following the instructions on Installing the Hyper Protect hosting appliance.

- Refer to the checklist that you prepared on this topic Planning for the environment.

Procedure

Complete the following steps to configure the network devices.

1. Connect to the Secure Service Container partition through the browser of your choice. For example, `https://<secure_service_container_partition_ip_address>`.
2. On the **Login** page, enter the master use ID and password values that you supplied in the image profile (standard mode system) or the partition definition (DPM-enabled system), and click **Login**.
3. In the navigation pane, click the **Network** icon to display the network connections page.
4. Select one of the network devices to get the channel path identifier (CHPID) of the OSA-Express device. For example, `encf900_network` is the network device name, and AA is the CHPID. The network device can only be used for the external communication for the Hyper Protect Virtual Server container.
5. Configure another network device on the Secure Service Container partition.

- For an ethernet-type connection:
  1. Click the plus (+) icon to add a new connection, and then select **Ethernet** as the connection type.
  2. Select a new network device from the drop-down list. Ensure that the CHPID in the Device Details section is different from the one in step 4. For example, the network device name is `encf900_internal_network`, and the CHPID is AB. This network device can only be used for the internal communication for the Hyper Protect Virtual Server container.
  3. Use the default value for the **Port** field, and set the connection state to **Active**.
  4. Use **Automatic** for both IPV4 and IPV6 addresses fields.
- For a VLAN-type connection, ensure that your OSA device is tagged with an VLAN ID (for example, `1121`) and the OSA device is connected with the trunk port of the switch.
  1. Click the plus (+) icon to add a new connection, and then select **VLAN** as the connection type.
  2. Select a parent device (also known as a tagged OSA device) from the drop-down list. If the parent device is not available, click the plus (+) icon to create a parent device. For example, the parent device name is `encf300`.
  3. Enter the VLAN ID by which the OSA device is tagged. For example, `1121`.
  4. Use the auto-generated connection name. For example, `vxlan0f300.1121`.
  5. If the DHCP is not configured in your network, select the **Manual** checkbox on the **IPv4** tab and assign an appropriate IP address according to your network.
  6. Set the connection state on the **General** tab to **Active**.
  7. Click the **ADD** button to save the changes.

Note: The Secure Service Container partition requires configuration of the necessary DNS entries if you plan to explore the following features in IBM Hyper Protect Virtual Servers.

- Configure appropriate DNS entry or entries for Secure Build containers on the IBM Hyper Protect Virtual Servers partition, so that the Secure Build containers can access the github source code URLs. This DNS configuration is performed on the Hardware Management Console (HMC) as part of the Secure Service Container LPAR profile's network configuration.
- Configure a DNS entry for the monitoring infrastructure, so that the monitoring client tools can access the monitoring infrastructure on the IBM Hyper Protect Virtual Servers partition.
- Configure a DNS entry for the GREP11 container, so that the client application code can access the GREP11 container on the IBM Hyper Protect Virtual Servers partition.

For more information on how to configure DNS entries on the Secure Service Container partition, see the following topic after you download IBM z Secure Service Container User's Guide from the About page.

- Chapter 14, "Using the Secure Service Container user interface", section "Viewing and managing network connections"
- Chapter 3 or 7, "Configuring a Secure Service Container partition"

Next

You can install the IBM Hyper Protect Virtual Servers CLI as instructed in the Installing the IBM Hyper Protect Virtual Servers CLI tool topic.

Installing the IBM Hyper Protect Virtual Servers CLI tool

IBM Hyper Protect Virtual Servers command line interface (CLI) tool provides commands to work with all the components in the offering. This includes management of Hyper Protect Virtual Server containers, securely building applications, generating signed and encrypted registration files, and automation to setup the IBM Cloud Private infrastructure. The IBM Cloud Private infrastructure automation works by creating all the necessary cluster nodes or isolated VMs, and provisions them with appropriate network, storage, CPU, memory resources.

This procedure is intended for users with the role *cloud administrator*.

Note: When setting up an instance of IBM Hyper Protect Virtual Servers CLI tool, you must be cautious of the file system that stores the CLI tool configuration because the file system contains sensitive information such as credentials to a Hosting Appliance or a Secure Build server container. The CLI tool can be run on an x86 or s390 Linux management server.

- It is recommended that the CLI tool is run on individual users machines to simplify the security setup and protect credentials against users without a need to know having access to them.
- If a shared management server setup is required, then you need to ensure that the management server is configured in such a way that only users that have a need to know those credentials have access to the directories where the configuration is stored. By doing so, you can protect against other Linux management server administrators using their admin privileges from exploiting those credentials.
- The `hosts` file under the `/config` directory of each CLI command contains the master password of the Secure Service Container partition in clear text as the `rest_password` parameter.
- Also some other parameters in the `securebuild.yaml` file, such as `github:key`, `manifest_cos:api_key`, `docker:password`, and `docker:base_password`, must be kept private and not shared.

Before you begin

- Check that you have IBM Hyper Protect Virtual Servers installation binary from the IBM Passport Advantage website retrieved on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.
- Check that the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server has the supported Docker environment installed. See Docker versions for more details.

Procedure

Complete the following steps under the `<installation_directory>` directory as the root user.

1. Go to the installation directory where you extract IBM Hyper Protect Virtual Servers archive file, and then install the docker images of the command line tool.

```
cd /opt/<installation-directory>
docker load -i hpvs-cli-installer.docker-image.tar
```

2. Run the `docker images` command to verify the CLI tool is installed correctly. You can see the similar output as the following example on your screen once the CLI tool installation completes.

```
REPOSITORY                          TAG                     IMAGE ID
CREATED            SIZE
ibmzcontainers/hpvs-cli-installer   1.2.0.s390x             97c5b4b59bd7        2 weeks
ago          447MB
ibmzcontainers/hpvs-cli-installer   1.2.0                   481cf705fc57        2 weeks
ago          475MB
```

   Where

   - `ibmzcontainers/hpvs-cli-installer` with the image tag `1.2.0`, which is the CLI docker image for the x86 Linux management server.
   - `ibmzcontainers/hpvs-cli-installer` with the image tag `1.2.0.s390x`, which is the CLI docker image for the s390x Linux management server.

You can run the commands for different modules in IBM Hyper Protect Virtual Servers . Those modules include:

- `hpvs` - The command line tool to manage Hyper Protect Virtual Server containers.
- `crypto` - The command line tool to check crypto domains on the Hardware Security Modules (HSM).
- `disk` - The command line tool to manage disks on the Secure Service Container partition.
- `grep11` - The command line tool to manage GREP11 containers for IBM Hyper Protect Virtual Servers .
- `image` - The command line tool to load container images into the Secure Service Container partitions.
- `monitoring` - The command line tool to manage monitoring infrastructure for IBM Hyper Protect Virtual Servers .
- `securebuild` - The command line tool to manage the Secure Build containers.
- `regfile` - The command line tool to manage the repository registration files for your applications.
- `repository` - The command line tool to manage the repositories on the Secure Service Container partitions.
- `quotagroup` - The command line to manage the quotagroups on the Secure Service Container partition.
- `snapshot` - The command line tool to manage the snapshots of Hyper Protect Virtual Server containers.

Next

- To deploy your workload with IBM Hyper Protect Virtual Servers, follow the instructions in Deploying your workloads with IBM Hyper Protect Virtual Serverse.
- To configure the environment for Secure Service Container for IBM Cloud Private, follow the instructions in Configuring the Secure Service Container for IBM Cloud Private CLI tool.

Deploying your workloads with IBM Hyper Protect Virtual Servers

To bring your own container images by using IBM Hyper Protect Virtual Servers , check out the following topics.

1. Registering base images in the remote registry server
2. Loading the Secure Build image into the Secure Service Container partition
3. Building the container image for your application by using the Secure Build
   - Configuring the network for Secure Build containers
   - Using your own certificate for Secure Build containers

    - Archiving the application manifest files in the Cloud Object storage
    - Updating the configuration of a running Secure Build container
    - Updating the base image of a Secure Build container
    - Rolling keys used in a Secure Build container
4. Creating and encrypting the repository definition file
    - Creating your own key pair to encrypt the repository definition file
    - Creating repository definition files for your existing images
5. Registering the repository on the Secure Service Container
6. Deploying your application into the Hyper Protect Virtual Server container
7. Debugging your application in the Hyper Protect Virtual Server container
8. Updating the Hyper Protect Virtual Server container resources
9. Refreshing registered repositories with a new signing key pair

Registering base images in the remote registry server

You must register the base images in the remote docker repository by using your ID and password. The remote docker repository can be Docker Hub or IBM Container Registry.

Note that the following context uses Docker Hub for demonstration. You can use the equivalent values or settings if you choose to use IBM Container Registry.

The base images are the default Hyper Protect Virtual Server container images that can be used to host your application code, and include two different types of container images for your development and production environments.

- `HpvsopBaseSSH`, which packages the SSH daemon into the default Hyper Protect Virtual Server container image, so that you can log in to the Hyper Protect Virtual Server by using the secure shell and your private key for debugging and development.
- `HpvsopBase`, which excludes the SSH daemon on the default Hyper Protect Virtual Server container image, and can be used in the production environment.

This procedure is intended for users with the role *cloud administrator* or *Application developer or ISV*.

Before you begin

- Check that you have the account ID and password on the remote docker registry server to create repositories for base images. For example, `docker_base_user` is your user ID on the remote docker registry server.
- Check that you have installed the GPG command line tool on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server. For more information, see GNU Privacy Guard.
- Check that you enable Docker Content Trust (DCT) for your remote docker registry server. For more information, see Content trust in Docker.

```
export DOCKER_CONTENT_TRUST=1
```

- Refer to the checklist that you prepared for the Hyper Protect Virtual Server on this topic Planning for the environment.

Procedure

Complete the following steps under the `<installation_directory>/VS/hpvs-cli/config` directory with root user authority.

1. Install the Hyper Protect Virtual Server base images to your x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.

a. Extract the base images into different folders.

```
mkdir <destination-folder-HpvsopBase>
mkdir <destination-folder-HpvsopBaseSSH>
tar -xvf HpvsopBase.tar.gz -C <destination-folder-HpvsopBase>
tar -xvf HpvsopBaseSSH.tar.gz -C <destination-folder-HpvsopBaseSSH>
```

b. Create the docker loadable binary under the same directory. Note that the warning message `gpg: Can't check signature: No public key` can be safely ignored when running the following gpg commands.

```
gpg <destination-folder-HpvsopBase>/HpvsopBase.tar.gz.sig
gpg <destination-folder-HpvsopBaseSSH>HpvsopBaseSSH.tar.gz.sig
```

c. Install the base images by using the `docker load` commands.

```
docker load -i <destination-folder-HpvsopBase>/HpvsopBase.tar.gz
docker load -i <destination-folder-HpvsopBaseSSH>/HpvsopBaseSSH.tar.gz
```

d. Run the `docker images` command to check whether the base images are loaded into the local registry successfully.

```
REPOSITORY                              TAG                 IMAGE ID
CREATED             SIZE
...
ibmzcontainers/hpvsop-base              1.2.0-abcedfg       027361ea9438        2 hours
ago         666MB
ibmzcontainers/hpvsop-base-ssh          1.2.0-abcedfb       88b071d7b794        2 hours
ago         597MB
...
```

2. Create two repositories in your namespace for both the `hpvsop-base` image and the `hpvsop-base-ssh` image on the Docker Hub. For example, `docker_base_user/hpvsop-base` and `docker_base_user/hpvsop-base-ssh`. Note that the repository name must match the image name.
3. Use the `docker tag` command to tag base images with the same ID used by the CLI tool. For example, `1.2.0` is the tag ID of the CLI tool that you can get by running the `docker images` command. Run the following commands to tag both base images.

```
docker tag ibmzcontainers/hpvsop-base:1.2.0-abcedfg docker_base_user/hpvsop-base:1.2.0-
abcedfg
docker tag ibmzcontainers/hpvsop-base-ssh:1.2.0-abcedfg docker_base_user/hpvsop-base-
ssh:1.2.0-abcedfg
```

4. Run the `docker images` command to check whether the tags for the base images are expected.

```
REPOSITORY                              TAG                       IMAGE ID
CREATED             SIZE
...
ibmzcontainers/hpvsop-base              1.2.0-abcedfg       027361ea9438        2 hours
ago         666MB
docker_base_user/hpvsop-base            1.2.0-abcedfg       027361ea9438        2 hours
ago         666MB
ibmzcontainers/hpvsop-base-ssh          1.2.0-abcedfb       88b071d7b794        2 hours
ago         597MB
docker_base_user/hpvsop-base-ssh        1.2.0-abcedfb       88b071d7b794        2 hours
ago         597MB
...
```

5. Push the base images to your remote docker repositories. For example:

```
docker login
docker push docker_base_user/hpvsop-base
docker push docker_base_user/hpvsop-base-ssh
```

6. Write down the following information to be used when building your app with the Secure Build container.

- Your Docker Hub ID account used to register the base images. For example, `docker_base_user`

- Your Docker Hub ID password. For example, `passw0rd`

Loading the Secure Build base image into the Secure Service Container partition

Use this procedure to load the base image of the Secure Build container into the Secure Service Container partition.

The Secure Build image on the management server acts as the runtime environment of the `securebuild` commands, and the Secure Build image on the Secure Service Container partition is to refactor your application into s390x compatible image and then push the image to the remote docker repository configured in the `securebuild.yaml` file. For more information, see Building the container image for your application by using the Secure Build.

This procedure is intended for users with the role *cloud administrator*.

Before you begin

- Check that you download Secure Service Container User's Guide from the About topic.
- Check that you have installed the cli tool on your x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server as instructed in the Installing the IBM Hyper Protect Virtual Servers CLI tool.
- Check that you have the IP address or name of the Secure Service Container partition, and pass the firewall if the Secure Service Container partition is behind a firewall or in a private network.
- Refer to the checklist that you prepared for the Hyper Protect Virtual Server on this topic Planning for the environment.

Procedure

Complete the following steps under the `<installation_directory>/VS/hpvs-cli/config` directory with root user authority.

1. Create a `hosts` file with connection information for the underlying RESTful API calls to the Secure Service Container partition. You can use the `hosts.example` file as the reference when configuring the `hosts` file.

   The following `hosts` example file contains the Master user ID and password of the Secure Service Container partition with IP address `10.152.151.105`.

   ```
   [lpars]
   10.152.151.105 rest_user="someuser" rest_pass="somepassword"
   ```

2. Create a `hpvs-config.yaml` file to configure the Hyper Protect Virtual Server container on the Secure Service Container partition with required CPU, memory, port range, quotagroup, and network specifications by using a specified template. You can use the `hpvs-config.yaml.example` file as the reference when configuring the yaml file.

   The following `hpvs-config.yaml` example file shows a Hyper Protect Virtual Server container with the internal IP address `192.168.40.188` and external IP address `164.20.5.79`. The container is bound to the repository `MyDockerApp` on the Secure Service Container partition `10.152.151.105` with name `ZBCSOR10`.

   ```
   cluster:
     name: vs
   LPARS:
     -
       ipaddress: 10.152.151.105
       name: SSC_LPAR_NAME
       containers:
         -
           template: template1
           count: 1
   ```

```
        networks:
          -
            network_name: encf900_network
            ip_address:
              - 164.20.5.78
          -
            network_name: encf900_internal_network
            ip_address:
              - 192.168.40.188
template1:
  name: hpvs1
  repoid: MyDockerApp
  imagetag: latest
  env_file: hpvs-env.json
  cpu: 2
  memory: 2048
  vs_index: 10000
  quotagroup_storage: 100G
  mount: /data_pool
  filesystem: ext4
  quotagroup_keep: false
  networks:
    encf900_network:
      subnet: 164.20.5.0/22
      gateway: 164.20.5.1
      parent: encf900
      driver: macvlan
    encf900_internal_network:
      subnet: 192.168.40.0/24
      gateway: 192.168.40.1
      parent: encf900
```

Note:

- `template1` is a customized template name for a Hyper Protect Virtual Server container, which defines the resources to be allocated to one Hyper Protect Virtual Server container. For example, container `hpvs1` will be created with resources defined under the `template1` section. You can define one resource template for multiple containers, or different resource templates for each container.
- `networks` section under the LPARS section defines the internal and external network configuration for the Hyper Protect Virtual Server container. For example, `network_name: 'encf900_network'` and `ip_address: ['164.20.5.79']` define the external network alias and IP address for the Hyper Protect Virtual Server container, and match the network configuration under the `networks` section in the template.
- `network_name` defines the internal and external network interface name shared by Hyper Protect Virtual Server containers on the Secure Service Container partition.
- `ip_address` defines a list of internal or external IP addresses to be assigned to the Hyper Protect Virtual Server container, which share the same parent network interface on the Secure Service Container partition. For multiple containers with different IP address, use an array such as for the `ip_address` parameter.
- `count` defines the number of containers that will be created on the partition, and each of container will be created by using the same resource template as defined with a customized resource template name. Note that the value of count is an integer and can not be enclosed by using the quotation marks.
- `repoid` defines the repository name on the Secure Service Container partition to retrieve the containerized images for your application.
- `imagetag` defines the image tag to retrieve the container image from.
- `env_file` defines a `json` file with the relative path to the `hpvs-cli/config` directory , which lists environment variables can be passed into the application container hosted on the Hyper Protect Virtual Server container. For example, one of environment variables can be JAVA_PATH for your application container. This parameter is optional. For more information about the json file, see hpvs-env.json.
- `cpu` defines the number of CPU threads to be assigned for the Hyper Protect Virtual Server container.

- memory defines the memory size (in MB) to be assigned for the Hyper Protect Virtual Server container.
- vs_index defines the initial index for the Hyper Protect Virtual Server containers if multiple server containers are created based on the same template. The name format of each Hyper Protect Virtual Server container is ${vs_name}_${vs_index}. For example, the Hyper Protect Virtual Server name is created with the name hpvs1, then the index of the container is hpvs1_10001.
- quotagroup_storage defines the quotagroup size (in GB) used by the Hyper Protect Virtual Server container. If you specify 0 to this parameter, a default 12 GB quotagroup is created for the container. You can use the quotagroup commands to manage the quotagroup size after the Hyper Protect Virtual Server container is created. For more information about the quotagroup commands, see Commands for quotagroups.
- mount defines the mount point inside the Hyper Protect Virtual Server container where the quotagroup is mounted. By default, a quotagroup mounted to /newroot is initiated to bootstrap the Hyper Protect Virtual Server container. If you want to use a dedicated quotagroup for your application data, specify the mount: line with a value in the hpvs-config.yaml file. For example, mount: /data_pool.
- filesystem defines the files system used for the Hyper Protect Virtual Server container. The supported value can be brtfs, ext4, or xfs.
- quotagroup_keep defines whether the quotagroup is removed when the Hyper Protect Virtual Server container is created. If the value is false, the qutogroup is removed upon the server container deletion.
- networks under the template section defines external and internal network configuration of the Hyper Protect Virtual Server container with its subnet, gateway, and parent network interface. For internal network configuration, the default network driver is docker bridge.
- driver:macvlan defines the docker network configuration that the network is for external communication of the Hyper Protect Virtual Server container.
- For a complete list of parameters supported in the hpvs-config.yaml file, see hpvs-config.yaml.

3. Copy the Secure Build docker image file into the <installation_directory>/VS/hpvs-cli/config directory.

```
cp -p <installation_directory>/VS/securebuild-cli/config/SecureDockerBuild.tar.gz
<installation_directory>/VS/hpvs-cli/config
```

4. Load the Secure Build base image on the Secure Service Container partition by running the image load command. For a full list of actions of the image command, see Commands for images.

   The following command example loads the Secure Build container docker image file, and then register the repository ibmzcontainers/secure-docker-build under the repository ID SecureDockerBuild on the Secure Service Container partition. The ibmzcontainers/secure-docker-build repository will be used to provision the Secure Build containers when you build your applications on the Secure Service Container partition.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 image
load -i SecureDockerBuild.tar.gz --lpar [LPARNAMEORADDRESS]
```

5. Verify the repository ibmzcontainers/secure-docker-build is created on the Secure Service Container successfully by running the repository list command. For a full list of actions of the repository command, see Commands for repositories.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
repository list --lpar [LPARNAMEORADDRESS]
```

Next

You can create a Secure Build container and securely build your application image into the Hyper Protect Virtual Server container by following the instructions on Building the container image for your application by using the Secure Build.

Building the container image for your application by using the Secure Build

You can use the Secure Build feature to build a s390x compatible container image for your application.

This procedure is intended for users with the role *Application developer or ISV*.

Note: You must provide the following credentials to use the Secure Build container.

- Credentials to your docker repository
- Credentials to your source code repository
- Credentials to the cloud object store to optionally store build manifests

These credentials are stored in the Secure Build container after the container is created on the Secure Service Container partition, and will be encrypted in transit and at rest.

To delete these credentials from a Secure Build container, you must either update the instance with new credentials or delete the instance. If a secure backup has been created for the container, then the backup must also be deleted. Be cautious about performing `delete` operations as the Docker Content Trust keys are also maintained within the container and secure backups.

Credential management has been designed following the Privacy by Design principles at IBM. For more information, see GDPR (General Data Protection Regulation).

Before you begin

- Ensure that you have the Docker Hub user ID and password that are used to register the base images. For example, `docker_base_user` and its password.
- Ensure that you have the Docker Hub user ID and password that will be used to push the application images to your repository with the Secure Build procedure. For example, `docker_writable_user` and its password.
- Ensure that the repository to host the container images for your application has been created on the remote Docker Registry, such as docker.io.
- Ensure that you create the `hosts` file under the `<installation_directory>/VS/hpvs-cli/config` directory as instructed in the Loading the Secure Build base image into the Secure Service Container partition topic.
- Refer to the checklist that you prepared for the Secure Build container on this topic Planning for the environment.

Procedure

On the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, complete the following steps under the `<installation_directory>/VS/securebuild-cli/config` directory with root user authority.

1. Copy the `hosts` file from the `<installation_directory>/VS/hpvs-cli/config` directory to the `<installation_directory>/VS/securebuild-cli/config` directory.

   ```
   cp -p <installation_directory>/VS/hpvs-cli/config/hosts <installation_directory>/VS/
   securebuild-cli/config
   ```

2. Create the `./config/securebuild.yaml` file with the settings for your Secure Build container. You can use the `./config/securebuild.yaml.example` file as the reference when configuring the `securebuild.yaml` file.

   The following `securebuild.yaml` example file shows a Secure Build container with an internal IP address `192.168.40.6` and external IP address `164.23.2.77`. The Secure Build container is assigned to build the s390x compatible docker image based on the source code from the dev branch of the source code repository `github.com:MyOrg/my-docker-app.git`.

```
secure_build_workers:
    -
        container:
            port: 443
            name: securebuild1
            ipaddress: 164.23.2.77
            quotagroup_name: myquotagroup
            imagetag: 1.2.0-release-abcdef
        lpar:
            ipaddress: 10.152.151.105
        network:
            name: encf900_internal_network
            ipaddress: 192.168.40.6
        github:
            url: git@github.com:MyOrg/my-docker-app.git
            branch: dev
        docker:
            repo: docker_base_user/MyDockerApp
            user: docker_writable_user
            password: passw0rd
            base_user: docker_base_user
            base_password: passw0rd
```

Note:

- If you don't want to assign the Secure Build container its own IP address, and your Secure Service Container partition supports port forwarding, remove the `network:` section, and choose an unused port of the partition for the value of `container:port:`. For more information, see Configuring the network for Secure Build containers.
- If you have more than one Secure Build container, add the parameter values for each container as a new entry in the `secure_build_workers` YAML array.
- Each Secure Build container can only be associated to one repository registered on the Secure Service Container partition.
- The quotagroup `myquotagroup` for the Secure Build container has a size of 50 GB by default.
- The value of `imagetag` is mandatory and must be copied from the `./config/securebuild.yaml.example` file.
- To recursively build submodules for the git repository, specify the value of `github:recurse_submodules` to `true`.
- Secure Build requires that the private key, used to secure access to the source Github repository, does not have a passphrase. For more information, see Secure Build failed to clone the github repository if a passphrase is associated with the private key.
- If specified, the value of `github:dockerfile_path:` is the path to the Dockerfile that will be used to build the image you are building. The path is specified relative to the root of the Github project so that the Dockerfile located at the URL `git@github.com:MyOrg/docker-build/Dockerfile` will be used to build the image if `github:dockerfile_path:` is specified as `/docker-build/Dockerfile`. If, as in the example `securebuild.yaml` file above, `github:dockerfile_path:` is not specified then the image will be built using the Dockerfile named `Dockerfile` located at the base directory of the Github project (provided that Dockerfile exists). So for the example above the Dockerfile located a the URL `git@github.com:MyOrg/Dockerfile` will be used to build the `git@github.com:MyOrg/my-docker-app.git` project.
- If specified, the value of `github:docker_build_path:` is the path to the directory within the Github project to use as the build context for the docker build that will build your image. If not specified the build context used will be the root directory of the Github project. The path is specified relative to the root of the Github project so the directory located at the URL `git@github.com:MyOrg/docker-build/` will be used as the build context if `github:docker_build_path:` is specified as `/docker-build/`.

- For a list of known supported Docker images from which you can create base images and `Dockerfile` for your workload during the Secure Build, see Open Source Software Development for IBM Z and LinuxONE website. Currently, Ubuntu 18.x images are supported.
- Git Large File Storage (LFS) is not supported.
- If the Secure Build container will be created for a different app developer or ISV, set the `container:cert_name` key in the `securebuild.yaml` file with the personal certificate from this app developer or ISV. For more information, see Using your own certificate for the Secure Build container.
- If you need to use key based authentication with your GitHub repository and add a private key for GitHub to your `securebuild.yaml` configuration file, the key value must be inserted as a block scalar using the correct block scalar syntax, as follows:

  ```
      key: |
         -----BEGIN RSA PRIVATE KEY-----
         MIIEogIBAAKCAQEAxYu4/PlJn+/DgY4MIMxpCbjodZ6nePiBfgKv9UsbKCeDGB1M
         ...
         e5jGeuNGkeqTvkQWG9uyaZ6z1UPGb+deQk9TDuGMtYO0d6DHQgCdlm4=
           -----END RSA PRIVATE KEY-----
  ```

  Note in the example syntax, the pipe character appears after the key name `key:` followed by a newline. The key value is NOT encased in single quotes in the block scalar. You must also use the correct indentation for each line of the block scalar parameter. For more information, see the block scalars in YAML. In addition, copy your entire private key file, including the header and footer, because various operating systems have different headers and footers.

3. Create a Secure Build container on the Secure Service Container partition by using the `securebuild create` command.

   The following command creates a Secure Build container based on the parameters in the `securebuild.yaml` file.

   ```
   docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-
   installer:1.2.0 securebuild create --name <secure-build-server-name>
   ```

   Where

   - `secure-build-server-name` is the name of the Secure Build container. For example, `securebuild1`.
   - If the `container:cert_name` key in the `securebuild.yaml` file is not specified, the command creates the Secure Build container on the Secure Service Container partition, initializes the Secure Build container to connect to the remote docker repository and source repository, and also creates a private key `securebuild-<container_name>-crt-cert` under the directory you mounted to `/securebuild-cli/config` when running the docker command (in this example, the current working directory returned by the pwd command is `securebuild-cli/config`). As long as you don't move the private key on your x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, and always mount the same directory to the same destination when running other `securebuild` commands, this private key is automatically picked up and used to send requests to the application API on a running Secure Build container.
   - If the `container:cert_name` key in the `securebuild.yaml` file is specified with the personal certificate from another app developer or ISV, the Secure Build container is created on the Secure Service Container partition, but not ready for use. You must send the `securebuild.yaml` file to this app developer or ISV, and ask the app developer or ISV to initialize the Secure Build container by using their own private key. After that, the Secure Build container can be only accessed by that app developer or ISV. For more information, see Using your own certificate for the Secure Build container.
   - For more information about the keys used by the Secure Build container, see List of keys used by the Secure Build.
   - If you plan to roll the keys used by the Secure Build container, see Rolling the keys in the Secure Build container

- To delete a Secure Build container, use the `securebuild delete --force` command as the following example. For more information, see Commands for Secure Build containers.

  ```
  docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-
  installer:1.2.0 securebuild delete --name <secure-build-server-name> --force
  ```

- If the Secure Build container is created and initialized correctly, a message containing keywords such as `initialized: OK` indicates the operation completed successfully.

4. Update the `Dockerfile` of your application code to include the base image.
   - For the development environment, use `FROM docker_base_user/hpvsop-base-ssh` to include the SSH daemon.
   - For the production environment, use `FROM docker_base_user/hpvsop-base` to exclude the SSH daemon.
5. Update the `Dockerfile` of your application code to ensure that the `/usr/bin/initHPVSoP.sh` script is part of initialization code defined with the `ENTRYPOINT` clause. The `/usr/bin/initHPVSop.sh` script must be invoked correctly to maintain the security and integrity of the Docker images that you are building. If the `ENTRYPOINT` clause is configured with your own script, then you must include a call to `/usr/bin/initHPVSop.sh` in your own initialization processing.

   The following Dockerfile example shows how to call the initialization code in the `hpvsop-base` parent Docker image. The initialization processing in the application `Dockerfile` is performed by the `<yourscript.sh>` script.

   ```
   FROM docker_base_user/hpvsop-base:1.2.0-release-abcdefg
   COPY <yourscript>.sh /usr/bin
   RUN chmod +x /usr/bin/<yourscript>.sh
   # Execute initialization code
   ENTRYPOINT ["/usr/bin/<yourscript>.sh"]
   ```

   In the `<yourscript>.sh` initialization script, you must include a call to `/usr/bin/initHPVSop.sh`.

   ```
   ...
   source /usr/bin/initHPVSoP.sh
   ...
   <Your initialization code>
   ...
   ```

6. Commit the changes to your application code into the Git repository.
7. Build your application code to a s390x compatible container image, and push it to your Docker Hub repository by using the following command.

   ```
   docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-
   installer:1.2.0 securebuild build --name <secure-build-server-name>
   ```

   Where

   - `secure-build-server-name` is the name of the Secure Build container. For example, `securebuild1`.
   - After the command completes, a message such as `"msg": "Build status - success"` indicates the build is completed successfully if you run `securebuild log --type build` command to check the complete log information.
   - You can also use the `securebuild status` command to check the build status of the Secure Build container.

     ```
     docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-
     installer:1.2.0 securebuild status -n <secure-build-server-name>
     ```

- To check the `securebuild build` command output log information, use the `securebuild log` command as in the following example.

  ```
  docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-
  installer:1.2.0 securebuild log -n <secure-build-server-name> --type build
  ```

The `securebuild build` command creates the following files for later use:

- An unsigned repository registration python file in the clear text, which you can use the credential and public key information when creating the repository definition file as described in Encrypting the repository definition file. Use the following command to retrieve this python file.

  ```
  docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
  securebuild regfile --regfile-type full --name <secure-build-server-name>
  ```

- An unsigned repository registration JSON file in the clear text, which you can use as an input to the `regfile encrypt` command with a self-generated GPG key pair to create a signed and encrypted repository definition file. See the topic Encrypting the repository definition file for details. Use the following command to retrieve this JSON file.

  ```
  docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
  securebuild regfile --regfile-type lite --name <secure-build-server-name>
  ```

  or

  ```
  docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
  securebuild regfile  --name <secure-build-server-name>
  ```

- A signed manifest file for the application and the public key used for signing the manifest. Use the following command to retrieve the signed manifest file and the public key used to verify the signature on the manifest file. If you configured IBM Cloud Object Store (COS) to store your manifest files, the manifest file for each build will be automatically pushed to COS after each build. For more information, see Archiving the application manifest files in the cloud Object storage.

  ```
  docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
  securebuild manifest --name <secure-build-server-name>
  docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
  securebuild public-key --name <secure-build-server-name>
  ```

Note:

- The following messages in the `securebuild status` command output indicate the progress of the Secure Build.
  - "msg": "Build status - github cloned"
  - "msg": "Build status - image built"
  - "msg": "Build status - success"

For a full list of supported actions and parameters in the `securebuild` commands or the `securebuild.yaml` file, see Commands for Secure Build containers and securebuild.yaml.

Next

You can encrypt the repository registration file before registering the repository for your applications on the Secure Service Container partition by following the instructions on Creating and encrypting the repository definition file.

Configuring the network for Secure Build containers

You can configure the Secure Build Container to use the existing network or create the network for it if the network does not exist.

This procedure is intended for users with the role *Application developer or ISV*.

Depending on your network configuration, you can choose one of the following options to configure the network for Secure Build container by using the `securebuild.yaml` file.

Note: When creating a new Secure Build instance, be aware that during a build, multiple external network connections will be made for the following tasks defined in the `securebuild.yaml` or `Dockerfile` file:

- Pulling and pushing images as specified in the `docker:repo` field of the `securebuild.yaml` file
- Cloning GitHub source code as specified in the `github:url` field of the `securebuild.yaml` file
- Pulling in components or packages to the image as specified in the RUN statements of the `Dockerfile` file

However it is possible to pass a Dockerfile engineered to connect to malicious endpoints during a build. To protect against the threat, the network must be configured to follow networking best practices and only allow egress over secure protocols to endpoints expected by the business. For further protection against malicious connections being established, network monitoring must also be used.

Procedure

- Use the port mapping on the Secure Service Container Partition

  By default, the docker uses the `bridge` network when it is started. You can map the Secure Build container to a port that is available on the Secure Service Container partition. Therefore, the value of `network:name` must be `bridge`. For more information about the `bridge` network, see Use bridge networks.

  To configure the port mapping, use the following code snippet in the `securebuild.yaml` file. Note that you must check with the *appliance administrator* to get the valid port number.

  ```
   ...
    - container:
         port: '20544'
      network:
         name: 'bridge'
    ...
  ```

  After the Secure Build container is created, you can access the Secure Build container via the `20544` port of the Secure Service Container partition for REST invocations.

- Allocate a dedicated IP address for external access

  To assign an external IP address to the Secure Build container, you must configure the Secure Build Container with both the internal IP address and external IP address information in the `securebuild.yaml` file. The `macvlan` network is used by the docker environment, and a dedicated OSA card is required for the external access to the Secure Build Container. Therefore, the value of `network:name` must be `staticIP`. For more information about the `macvlan` network, see Use macvlan networks.

  To configure the external and internal IP addresses, use the following code snippet in the `securebuild.yaml` file. Note that you must check with the *appliance administrator* and *cloud administrator* to get the valid internal and external IP addresses.

  ```
  ...
    - container:
        port: '443'
        ipaddress: '164.23.2.77'
      network:
        name: 'staticIP'
        ipaddress: '192.168.40.6'
  ...
  ```

  After the Secure Build container is created, external RESTful workload can access the Secure Build container via the external IP `164.23.2.7` and the default 443 port.

- Use the existing IP range and subnet on the Secure Service Container Partition

  To use an existing network on the Secure Service Container Partition, you must configure the Secure Build Container to use the same network name in the `securebuild.yaml` file. The value of `network:name` can be the connection name that is available on the Secure Service Container partition. For more information, see Configuring the network on the Secure Service Container partition.

To configure the Secure Build Container to use an existing network, use the following code snippet in the `securebuild.yaml` file. Note that you must check with the *cloud administrator* to get the valid network name and IP address information.

```
...
  - container:
      port: '443'
      ipaddress: '192.168.40.6'
    network:
      name: 'encf900_internal_network'
      ipaddress: '192.168.40.6'
...
```

After the Secure Build container is created, you can access the Secure Build container via its internal IP `192.168.40.6` and the default 443 port for REST invocations.

- Create and use a network on the Secure Service Container partition

  To use a new network that is not created on the Secure Service Container, you can configure the `securebuild.yaml` file so that Secure Build Container creates the network and then uses this network. The value of `network:driver` must be `macvlan` in order that the network can be created, and the value of `network:name` can be a connection name that is not used on the Secure Service Container partition.

  To configure the Secure Build container to create the network, use the following code snippet in the `securebuild.yaml` file. Note that you must check with the *appliance administrator* or *cloud administrator* to get the valid network name and IP address information.

```
...
 - container:
      port: '443'
      ipaddress: '192.168.40.6'
    network:
      name: 'encf900_new_internal_network'
      driver: 'macvlan'
      parent: 'encf900'
      subnet: '192.168.40.0/22'
      gateway: '192.168.40.1'
      ipaddress: '192.168.40.6'
...
```

  After the Secure Build container is created, you can access the Secure Build container via its internal IP `192.168.40.6` and the default 443 port for REST invocations.

Note:

- You can decide whether to delete the network when the Secure Build Container is deleted by using the `delete_network_on_container_delete` parameter in the `securebuild.yaml` file. By default, the value is no.
- The network used by other containers will not be deleted even if the values is set to `yes`.
- For a full list of supported parameters, see securebuild.yaml.

Using your own certificate for the Secure Build container

You can use your certificates for the Secure Build container to communicate with the Secure Service Container partition, so that the Secure Build container can only accept API calls encrypted by using your private key.

This procedure is intended for users with the role *Application developer or ISV*.

Before you begin

- Ensure you have the private key corresponding to the personal certificate that is used to create the Secure Build container. For example, your private key is `john_doe1.key` stored in the

<installation_directory>/VS/securebuild-cli/config directory. The command to create your own key pair will be as the following example.

```
openssl req -newkey rsa:2048 \
  -new -nodes -x509 \
  -days 3650 \
  -out john_doe1.cert \
  -keyout john_doe1.key \
  -subj "/C=GB/O=IBM/CN=john_doe1.example.com"
```

- Ensure the Secure Build container is created on the Secure Service Container partition by using your personal certificate. For example, the personal certificate `john_doe1.cert` is specified as the value of `container:cert_name` key in the `securebuild.yaml`, and used when creating the Secure Service container. For more information, see Building the container image for your application by using the Secure Build.
- Ensure you have the `securebuild.yaml` file used to create the Secure Build container, and the file is under the `<installation_directory>/VS/securebuild-cli/config` directory.

Procedure

Complete the following steps under the `<installation_directory>/VS/securebuild-cli/config` directory with root user authority.

1. Generate the PEM format certificate by using your private key and personal certificate.

```
cat john_doe1.cert john_doe1.key > john_doe1.pem
```

2. Update the `container:cert_name` key in the `securebuild.yaml` file with the resulting PEM certificate.

```
- containers:
      ...
      cert_name: john_doe1.pem
      ...
```

3. Initialize the Secure Build container by using the `securebuild init` command. After the command completes, this Secure Build container only accepts the API calls encrypted with your private key.

   The following command initializes the Secure Build container based on the parameters in the `securebuild.yaml` file.

```
docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-
installer:1.2.0 securebuild init --name <secure-build-server-name>
```

   After the command completes successfully, a message `"msg": "Build status - initialized"` indicates the Secure Build container is ready for use.

Next

You can use the Secure Build container to build s390x compatible images. For more information, see Building the container image for your application by using the Secure Build.

Archiving the application manifest files in the cloud Object storage

You can configure a Cloud Object Storage service to archive the application manifest files of your applications built by your Secure Build container. The manifest files can be later used for audit purposes, and are stored on the Secure Service Container if the Cloud Object Storage service is not configured.

This procedure is intended for users with the role *Application developer or ISV*.

Before you begin

- Ensure that you have the following information about your IBM Cloud Object Storage at hand.
  - The API Key to the cloud object storage service

- The object storage bucket to store the manifest
- The resource instance name of the cloud object storage service
- The authentication endpoint for the cloud object storage service
- The endpoint for the cloud object storage service

Procedure

Complete the following steps under the `<installation_directory>/VS/securebuild-cli/config` directory with root user authority.

1. Update the `securebuild.yaml` file with the configuration of the cloud object storage service.

   The following `securebuild.yaml` example file specifies a Secure Build container `securebuild1` to build the app and push the app into the remote docker repository `hpvs-my-id/MyDockerApp`, then upload the signed application manifest file to IBM Cloud Object Storage service.

   ```
   ...
     container:
       name: securebuild1
       ...
     repo:
       id: hpvs-my-id/MyDockerApp
       ...
     manifest_cos:
       bucket_name: my-cos-bucket1
       api_key: 0viPHOY7LbLNa9eLftrtHPpTjoGv6hbLD1QalRXikliJ
       resource_crn: crn:v1:...::1
       auth_endpoint: iam.cloud.ibm.com
       endpoint: s3.us-west.cloud-object-storage.test.appdomain.cloud
   ...
   ```

2. Run the `securebuild update` command to update the Secure Build container so that it uses the new configuration from your `securebuild.yaml` configuration file.

   ```
   docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-
   installer:1.2.0 securebuild update --name securebuild1
   ```

3. Run the `securebuild build` command to build the application and upload the application manifest file to the cloud object storage.

   ```
   docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-
   installer:1.2.0 securebuild build --name securebuild1
   ```

4. Log in to your cloud account and check the application manifest file has been transferred to its bucket in your Cloud Object Storage service.

Updating the configuration of a running Secure Build container

You can update a running Secure Build container to use a different configuration.

In the `securebuild.yaml` configuration file, for the Secure Build container you want to update, edit the parameters you want to update and save the configuration file. Note that not all parameters can be changed after a Secure Build container is created. For more information about which parameters can be updated, see configuration file parameters for Secure Build.

This procedure is intended for users with the role *Application developer or ISV*.

Before you begin

- Ensure that you backup the Secure Build container and your `securebuild.yaml` configuration file before you update its configuration so that you can revert any unwanted configuration changes.

Procedure

To update the Secure Build container to use the new values of the parameters in your configuration file, run the following command under the `<installation_directory>/VS/securebuild-cli/config` directory with root user authority.

```
docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
securebuild update --name <secure-build-server-name>
```

This command updates the Secure Build container named `<secure-build-server-name>` to use the updated values of the configuration parameters for that Secure Build container from your `securebuild.yaml` configuration file.

Updating the CPU or memory settings of a running Secure Build container

You can resize the CPU or memory settings of a running Secure Build container by using the `securebuild upgrade` command.

To apply other changes in the `securebuild.yaml` file, see the Updating the configuration of a running Secure Build container.

This procedure is intended for users with the role *Application developer or ISV*.

Before you begin

- Ensure that you backup the Secure Build container and your `securebuild.yaml` configuration file before you update its configuration so that you can revert any unwanted configuration changes.

Procedure

Complete the following steps under the `<installation_directory>/VS/securebuild-cli/config` directory with root user authority.

1. In the `<installation_directory>/VS/securebuild-cli/config/securebuild.yaml` configuration file, for the Secure Build container you want to update the CPU and memory configuration, edit the CPU and `memory` parameters under the `container` section.
2. Under the `<installation_directory>/VS/securebuild-cli/config` directory, check the Secure Build container is not currently building or pushing a Docker image by running the following command.

   ```
   docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-
   installer:1.2.0 securebuild status --name <secure-build-server-name>
   ```

   Where the value of `Display the overall status of the Docker build` in the output of this command should be one of the following, if a build is not in progress:

   - `"msg": "Build status - success"`
   - `"msg": "Build status - initialized"`
3. Under the `<installation_directory>/VS/securebuild-cli/config` directory, update the Secure Build container to use the new CPU and memory configurations by running the following command.

   ```
   docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-
   installer:1.2.0 securebuild upgrade --name <secure-build-server-name>
   ```

   This command restarts the Secure Build container named `<secure-build-server-name>`. The restarted instance of the Secure Build container will use the Secure Build container image given by the updated CPU and memory configuration for that Secure Build container from your `securebuild.yaml` configuration file.

Updating the base image of a Secure Build container

You can upgrade a running Secure Build container to a new version of the Secure Build image, in order to take advantage of fixes and new features in new versions of Secure Build.

All customer data on the Secure Build container, including private keys used to push Docker images built using the Secure Build container to their repository using Docker Content Trust, and the private key used to sign the build manifest file, are preserved over the upgrade.

This procedure is intended for users with the role *cloud administrator*.

Before you begin

- Ensure that you backup the Secure Build container and your `securebuild.yaml` configuration file before you update its configuration so that you can revert any unwanted configuration changes.
- Ensure that the new version of the Secure Build container image is loaded into the Secure Service Container partition. For more information, see Loading the Secure Build base image on to the Secure Service Container partition.

Procedure

Complete the following steps under the `<installation_directory>/VS/hpvs-cli/config` directory with root user authority.

1. Check that the new version of the Secure Build container image is available under the repository ID `SecureDockerBuild` on the Secure Service Container partition. For more information about the `image` commands, see Commands for images.

   ```
   docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 image
   list --lpar [LPARNAMEORADDRESS] --repoid SecureDockerBuild
   ```

2. In the `<installation_directory>/VS/securebuild-cli/config/securebuild.yaml` configuration file, for the Secure Build container you want to update, edit the `imagetag` parameter under `secure_build_workers` and `container` to be the image tag or the version of Secure Build you want to update the container to.
3. Under the `<installation_directory>/VS/securebuild-cli/config` directory, check the Secure Build container is not currently building or pushing a Docker image by running the following command.

   ```
   docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-
   installer:1.2.0 securebuild status --name <secure-build-server-name>
   ```

   Where the value of `Display the overall status of the Docker build` in the output of this command should be one of the following, if a build is not in progress:

   - `"msg": "Build status - success"`
   - `"msg": "Build status - initialized"`
4. Under the `<installation_directory>/VS/securebuild-cli/config` directory, update the Secure Build container to use the new Secure Build image version by running the following command.

   ```
   docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-
   installer:1.2.0 securebuild upgrade --name <secure-build-server-name>
   ```

   This command restarts the Secure Build container named `<secure-build-server-name>`. The restarted instance of the Secure Build container will use the Secure Build container image given by the updated imagetag for that Secure Build container from your `securebuild.yaml` configuration file.

Rolling keys in a Secure Build container

You can roll the keys used by a Secure Build container when being required by your security policies, or when the private keys get compromised by malicious attacks.

For the keys that might be impacted or rolled, see List of keys used during the Secure Build.

In order to roll such keys, you must create a new Secure Build container and an updated repository registration file for your applications images to be created.

This procedure is intended for users with the role *Application developer or ISV*.

Before you begin

- Ensure you install the latest IBM Hyper Protect Virtual Servers CLI tool on your x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.

Procedure

Complete the following procedure on the management server with root user authority.

1. Load the latest Secure Build base image by following the instructions on the Loading the Secure Build base image into the Secure Service Container partition.
2. Create and initialize a new Secure Build container by following the instructions on the Building the container image for your application by using the Secure Build.
3. Contact the image repository host (DockerHub or IBM Cloud Container Registry) to reset the repository state. See the **Lost keys** section on the Manage keys for content trust.
4. Update your application to use the latest Secure Build container, and then build the application by using the `securebuild build` command. For more information, see Building the container image for your application by using the Secure Build. Note that if the remote repository has been reset properly, the new images will be in the repository signed with the newly generated private key.
5. Update the repository on the Secure Service Container partition by using the new signing key. For more information, see Refreshing registered repositories with a new signing key pair.

Creating and encrypting the repository definition file

You must create a repository definition file, and encrypt the repository definition file before sending it to your cloud administrator to register the repository on the Secure Service Container partition.

You can encrypt the repository definition file using a private key generated by IBM Hyper Protect Virtual Servers CLI tool or an existing private key that is stored on the Linux management server. The recommended procedure is to use IBM Hyper Protect Virtual Servers CLI tool to generate the private key, however if you want to generate your own key, you can follow the instructions in the Creating your own public and private key to encrypt the repository definition file topic.

When you encrypt the repository definition file by using the IBM Hyper Protect Virtual Servers CLI tool without providing the pre-existing GPG key pair, the generated GPG key pair will be named as `isv_user.pub` for the public key and `isv_user.private` for the private key. And the key pair will be available on the Linux management server in your `<installation_directory>/VS/regfile-cli` directory after the `regfile create` or `regfile encrypt` command completes. You must store the generated public key securely and not share it.

Note: Keep the private keys used for creating and encrypting the repository definition files to yourself only, and rotate your keys when necessary.

This procedure is intended for users with the role *Application developer or ISV*.

Before you begin

- Ensure that you complete all the steps as instructed in the Creating the container image for your application by using the Secure Build topic.
- Optionally, ensure that you have retrieved the cleartext JSON repository registration file for your Secure Build container.
- Refer to the checklist that you prepared for the repository definition file on this topic Planning for the environment.

Procedure

Complete the following steps under the <installation_directory>/VS/regfile-cli/config directory with root user authority depending on whether you have the cleartext JSON repository registration file for your Secure Build container at hand.

- If you do not have the cleartext JSON repository registration file, create the signed and encrypted repository definition file by using the `regfile create` command, which requires you to manually input all the parameters required to create the repository definition file as part of the `regfile create` CLI command.

  The following command creates a repository definition file `MyDockerApp.json-rel.enc` for the remote repository `docker_base_user/MyDockerApp` on the `docker.io` site. The file is signed using a unique GPG key pair generated by IBM Hyper Protect Virtual Servers CLI tool with the passphrase `over-the-lazy-dog`, and the IBM public key provided by the CLI tool. This file will be used to create a `docker_base_user/MyDockerApp` repository on the Secure Service Container partition.

  ```
  docker run --rm -v $(pwd):/regfile-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 regfile
  create -u docker_readonly_user -p docker_password -s docker.io -r docker_base_user/
  MyDockerApp -ph over-the-lazy-dog -n MyDockerApp -kn <a_unique_key_name>
  ```

  Note:

  – You can use a Docker Hub ID with read-only access to generate the repository definition file.

  – The `-n` parameter specifies the name of the repository definition file. If the name is not provided in the command, the default file name is `isv_definition_template.json-rel.enc`.

  – The encrypted repository definition file is encrypted using the IBM public key, which is provided within the CLI tool. The default repository definition file is production ready and its name contains a keyword `rel`. For example, `MyDockerApp.json-rel.enc`.

  – A unique key pair is generated with the `-kn` option.

  – If you want to use your pre-existing key pair, you can provide IBM Hyper Protect Virtual Servers CLI tool by using the `-K/--private-key`, and `-k/--public-key` as in the following command example. The private key path for the `-K` parameter and the public key path for the `-k` parameter must be the relative path within your current working directory.

  ```
  docker run --rm -v $(pwd):/regfile-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
  regfile create -u docker_readonly_user -p docker_password -s docker.io -r docker_base_user/
  MyDockerApp -ph over-the-lazy-dog -n MyDockerApp -K isv_user.private -k isv_user.pub
  ```

- If you have the cleartext JSON repository registration file generated by the Secure Build container on your management server, encrypt the JSON repository registration file by using the `regfile encrypt` command. After the command completes, an encrypted repository definition file is created under the current directory.

  The following command creates a repository definition file `MyDockerApp.json-rel.enc` from the cleartext JSON repository definition file `MyDockerApp.json`. The file is signed using a unique GPG key pair generated by IBM Hyper Protect Virtual Servers CLI tool, with the passphrase `over-the-lazy-dog` and the IBM public key provided by the CLI tool.

  ```
  docker run --rm -v $(pwd):/regfile-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 regfile
  encrypt -ph over-the-lazy-dog -i MyDockerApp.json -kn <a_unique_key_name>
  ```

  Note:

  – The contents of an example of a cleartext JSON repository registration file, `MyDockerApp.json`, is displayed as the following code snippet. This JSON repository registration file will be used to create a `docker_base_user/MyDockerApp` repository on the Secure Service Container partition, and registered under the repository ID `MyDockerApp`.

  ```
    {
      "repository_name": "docker.io/docker_base_user/MyDockerApp",
      "docker_username": "dockerUserID",
      "docker_password": "dockerPassword",
  ```

```
    "public_key_id": "<PUBLIC_KEY_ID>",
    "public_key": "<PUBLIC_KEY>",
    "envs_whitelist": ["PATH","SHELL"]
  }
```

In this JSON repository registration file, `public_key` is the Docker Content Trust public key for the named repository.

- Ensure to use one repository for one type of workloads to avoid the repository file to be overwritten. The workloads in the same repository can have different image tags.
- If you want to use your pre-existing key pair, you can provide IBM Hyper Protect Virtual Servers CLI tool using the `-K/--private-key` and `-k/--public-key` parameters as in the following command example.

```
docker run --rm -v $(pwd):/regfile-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 regfile
encrypt -K isv_user.private -k isv_user.pub -ph over-the-lazy-dog -i MyDockerApp.json
```

- If you want to create a JSON repository registration file with a list of environment variables, run the following example command with your own docker credential.

```
docker run --rm -v $(pwd):/regfile-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 regfile
create \
-u dockerUserID -s docker.io -r ibmzcontainers/appliance-unit-test \
-p dockerPassword --public-key isv_user.pub  --private-key isv_user.private \
-ph 'over-the-lazy-dog' -
e='SSH_PUBLIC_KEY,SERVER_KEY,SERVER_CRT,CLIENT_KEY,CLIENT_CRT,LOGTARGET,\
ROOTFS_LOCK,DB2_PATH,JAVA_PATH'
```

The `env_whitelist` field in the result repository registration file contains an array of those environment variables as in the following code snippet.

```
...
"envs_whitelist": ["SSH_PUBLIC_KEY", "SERVER_KEY", "SERVER_CRT", "CLIENT_KEY", "CLIENT_CRT",
"LOGTARGET", "ROOTFS_LOCK", "DB2_PATH", "JAVA_PATH"]
...
```

- If you want to create a JSON repository registration file to replace its signing key, run the following example command to revoke the old key. For more information, see Refreshing registered repositories with a new signing key pair.

```
docker run --rm -v $(pwd):/regfile-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 regfile
create \
-u dockerUserID -s docker.io -r docker.io/docker_base_user/MyDockerApp \
-p ockerPassword --public-key isv_user.pub  --private-key isv_user.private \
-ph 'over-the-lazy-dog' -nk new_isv_user.pub -nK new_isv_user.private \
--revocation-cert isv_user.asc
```

For a full list of supported actions and parameters in the `regfile` commands, see Commands for repository definition files.

Next

You can send the repository definition file to the Cloud Administrator, so that the repository `MyDockerApp` can be registered on the Secure Service Container partition and ready to host your applications built into Hyper Protect Virtual Server containers. For more information, see Registering the repository on the Secure Service Container.

Creating your own public and private key pair to encrypt the repository definition file

You can create your own public and private keys to encrypt the repository definition file.

This procedure is intended for users with the role *Application developer or ISV*.

Before you begin

- Ensure that you install GnuPG or a similar tool on your x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server. For more information, see The GNU Privacy Handbook.

Procedure

On your x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, complete the following steps with root user privilege.

1. Generate the key pair for signing the repository definition file by using the GnuPG tool. If you encounter a problem, refer to the Troubleshooting topic for some known issues and workarounds.

   The following commands create a GPG key pair, export the public key `isv_user.pub` and the private key `isv_user.private`. The key pair is protected by using the passphrase `over-the-lazy-dog`.

   ```
   export keyName=isv_user
   export passphrase=over-the-lazy-dog
   cat >isv_definition_keys <<EOF
         %echo Generating registration definition key
         Key-Type: RSA
         Key-Length: 4096
         Subkey-Type: RSA
         Subkey-Length: 4096
         Name-Real: isv_user
         Expire-Date: 0
         Passphrase: over-the-lazy-dog
         # Do a commit here, so that we can later print "done" :-)
         %commit
         %echo done
    EOF
    gpg -a --batch --generate-key isv_definition_keys
    gpg --armor --pinentry-mode=loopback --passphrase  ${passphrase} --export-secret-keys $
   {keyName} > ${keyName}.private
    gpg --armor --export ${keyName} > ${keyName}.pub
   ```

2. Copy the generated key pair `isv_user.pub` and `isv_user.private` to the `<installation_directory>/VS/regfile-cli` directory on your x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.

Next

You can use the generated key pair to encrypt the repository registration file as instructed in the Creating and encrypting the repository definition file topic.

Creating repository definition files for your existing images

You can create repository registration files for the images that already exists in the docker hub.

This procedure is intended for users with the role *Application developer or ISV*.

Before you begin

- Ensure that these images are pushed to the docker hub by using the Docker content Trust (DCT). For example, the images are under the `docker_base_user/MyDockerApp` repository on the `docker.io` site.
- Ensure that you have the credential to access those images on the docker repository. For example, `docker_readonly_user` and its password `docker_password`.
- Ensure that you have the key pair and passphrase to encrypt the repository registration file. For example, `isv_user.private`, `isv_user.pub`, and the passphrase `over-the-lazy-dog`.

Procedure

Complete the following steps under the `<installation_directory>/VS/regfile-cli/config/` directory with root user authority.

1. Create the repository registration file for the image by using the following example command.

```
docker run --rm -v $(pwd):/regfile-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 regfile
create \
-u docker_readonly_user -p docker_password -s docker.io -r docker_base_user/MyDockerApp \
-ph over-the-lazy-dog -n MyDockerApp -K isv_user.private -k isv_user.pub
```

1. Once the encrypted repository registration file is available, you can create the repository on the Secure Service Container partition by using the following example command under the `<installation_directory>/VS/hpvs-cli/config` directory.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
repository create \
--lpar [LPARNAMEORADDRESS] --repoid MyDockerApp --repofile MyDockerApp.json-rel.enc`
```

1. Deploy your image into the Hyper Protect Virtual Server container by using the Secure Build. For more information, see Building the container image for your application by using the Secure Build.

Registering the repository on the Secure Service Container

You must register the repository on the Secure Service Container partition, so that the partition can pull down images from the repository and create Hyper Protect Virtual Server instances of those images on the partition.

This procedure is intended for users with the role *Cloud administrator*.

Before you begin

- Check that your application developer or ISV has sent you the encrypted repository definition file. For example, the file name is `MyDockerApp.json-rel.enc`.
- Check that you have the name or IP address of the Secure Service Container partition to host the containerized images for your application developers or ISVs.

Procedure

Complete the following steps under the `<installation_directory>/VS/hpvs-cli/config` directory with root user authority.

1. Copy the encrypted repository definition file from the `<installation_directory>/VS/regfile-cli/config` directory into the `<installation_directory>/VS/hpvs-cli/config` directory.

   ```
   cp -p MyDockerApp.json-rel.enc <installation_directory>/VS/hpvs-cli/config
   ```

2. Check that the repository name is available on the Secure Service Container partition by using the `repository list` command.

   The following command lists all the repositories on the Secure Service Container partition.

   ```
   docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
   repository list --lpar [LPARNAMEORADDRESS]
   ```

   Where LPARNAMEORADDRESS is the name or IP address of the Secure Service Container partition.

3. Register the repository on the Secure Service Container partition by using the `repository create` command and the repository definition file received from your application developer or ISV.

   The following command validates the signature of the repository registration file `MyDockerApp.json-rel.enc` by using the IBM private key in the hosting appliance, and then creates a `MyDockerApp` repository on the Secure Service Container partition.

   ```
   docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
   repository create --lpar [LPARNAMEORADDRESS] --repoid MyDockerApp --repofile
   MyDockerApp.json-rel.enc
   ```

4. Run the `repository list` command again to check that the repository is registered on the Secure Service Container partition.

   ```
   docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
   repository list --lpar [LPARNAMEORADDRESS]
   ```

5. (Optional) To delete a repository that is registered on the Secure Service Container partition, run the `repository delete` command.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
repository delete --lpar [LPARNAMEORADDRESS] --repoid MyDockerApp
```

For a full list of commands to manage the repositories on the Secure Service Container partition, see Commands for Repositories.

Next

You can deploy your application into the Secure Service Container partition as a Hyper Protect Virtual Server container by following the instructions on Deploying your application into the Hyper Protect Virtual Server container.

Deploying your application into the Hyper Protect Virtual Server container

You can deploy your application code into a Hyper Protect Virtual Server container, and then access the application by using an external IP address configured for the Hyper Protect Virtual Server container.

This procedure is intended for users with the role *Cloud administrator*.

Before you begin

- Check that you have the following configuration for the Secure Service Container partition.
  - IP address
  - Master user ID and password
- Check that you create the `hpvs-config.yaml` file under the `<installation_directory>/VS/hpvs-cli/config` directory as instructed in Loading the Secure Build base image into the Secure Service Container partition.
- Check that you create at least 100 GB for each Hyper Protect Virtual Server container by following the steps in Configuring the storage on the Secure Service Container partition.
- Ensure the value of the `repoid` field in the `hpvs-config.yaml` file is the repository ID when Registering the repository on the Secure Service Container.
- Ensure the value of the `imagetag` field in the `hpvs-config.yaml` file is the image tag when Registering base images in the remote registry server.
- Refer to the checklist that you prepared for the Hyper Protect Virtual Server on this topic Planning for the environment.

Note: Data can be passed into a Hyper Protect Virtual Server container as environment variables when a Hyper Protect Virtual Server container is created. This mechanism should never be used to pass in secrets (for example, a private key) to the Hyper Protect Virtual Server container. Instead, the application running as a Hyper Protect Virtual Server container should be built to provide a trusted channel to enable any secrets it requires at runtime to be configured at runtime.

Procedure

Complete the following steps under the `<installation_directory>/VS/hpvs-cli/config` directory with root user authority.

1. Create the Hyper Protect Virtual Server container by using the `hpvs create` command.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 hpvs
create
```

Note:

- To override the configuration of the Hyper Protect Virtual Server container defined in the `hpvs-config.yaml` file, specify the parameters when using the `hpvs` command. For more information about the `hpvs` commands, see Commands for Hyper Protect Virtual Server containers.
- To manage the quotagroups used by the Hyper Protect Virtual Server containers, use the `quotagroup` commands. For more information, see Commands for quotagroups.
- To back up or restore the Hyper Protect Virtual Server container with your application, see Backing up and restoring IBM Hyper Protect Virtual Servers.
- To debug your application in the Hyper Protect Virtual Server container by using the secure shell, check the following conditions:
  - The `FROM docker_base_user/hpvsop-base-ssh` clause is used in the `Dockerfile` of your application code.
  - Use the `hpvs create` command with `--env_file` parameter, or specify the `env_file` parameter in the `hpvs-config.yaml` file. Both parameters defines a `.json` file with a public SSH key so that you can connect to the Hyper Protect Virtual Server container by using the secure shell. For more information, see Commands for Hyper Protect Virtual Server containers.

2. Check the Hyper Protect Virtual Server status by running the `hpvs get` command.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 hpvs
get
```

3. (Optional) To delete the Hyper Protect Virtual Server container, run the `hpvs delete` command.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 hpvs
delete --name <Virtual_Server_Container_name>
```

Next

You can access your application by using the IP address of the Hyper Protect Virtual Server container, or debug the application if necessary by following the instructions on Debugging your application in the Hyper Protect Virtual Server container.

Debugging your application in the Hyper Protect Virtual Server container

You can debug your application deployed in the Hyper Protect Virtual Server container before publishing the application into your production environment.

This procedure is intended for users with the role *Application developer or ISV*.

Before you begin

- Ensure that your application is in the development environment.
- Ensure that your application is based on the Hyper Protect Virtual Server base image with SSH daemon. For example, `FROM docker_base_user/hpvsop-base-ssh` in the `Dockerfile` of your application.

Procedure

Complete the following steps under the `<installation_directory>/VS/hpvs-cli/config` directory with root user authority. For the Hyper protect Virtual Server container to be debugged, refer to the `<installation_directory>/VS/hpvs-cli/config/hpvs-config.yaml` file for the file name value of the `env_file` field. In this topic, the file name value of the `env_file` key is `hpvs-env.json`.

1. Log in to your remote docker repository and check that the `hpvsop-base-ssh` base image is available in the repository.

2. Create the SSH key pair for the communication with the Hyper Protect Virtual Server container that has your application. For example, create the SSH key pair with the name `debug-myapp-test`.

```
ssh-keygen -t rsa -b 4096 -f ~/.ssh/debug-myapp-test -N ""
```

3. Copy the SSH public key value from the `~/.ssh/debug-myapp-test.pub` file into the SSH_PUBLIC_KEY field in the `hpvs-env.json` file. For more information about the `hpvs-env.json` file, see hpvs-env.json.

```
{
   ...
   "SSH_PUBLIC_KEY": "<input your public key value>"
   ...
}
```

4. Update the Hyper Protect Virtual Server container to accept the SSH public key.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 hpvs
update --name <Hyper_Protect_Virtual_Server_Container_name> --lpar [LPARNAMEORADDRESS]
```

5. Connect to the Hyper Protect Virtual Server container with its IP address or name by using the secure shell.

```
ssh -i ~/.ssh/debug-myapp-test root@<Hyper_Protect_Virtual_Server_Container_IP>
```

6. (Optional)To securely copy data (a file) to the home directory of the Hyper Protect Virtual Server container, use the following `scp` command.

```
scp -i ~/.ssh/debug-myapp-test <filename> root@<Hyper_Protect_Virtual_Server_Container_IP>:/
home
```

Updating Hyper Protect Virtual Server containers

You can upgrade one or multiple Hyper Protect Virtual Server containers to use different resource settings, such as CPU, memory, a new quotagroup, or a different image tag.

This procedure is intended for users with the role *cloud administrator*.

Before you begin

- Ensure that you back up the Hyper Protect Virtual Server container by using the `snapshot create` command. For more information about the `snapshot` command, see Commands for snapshots.

Procedure

Complete the following steps under the `<installation_directory>/VS/hpvs-cli/config` directory with root user authority.

1. Get the Hyper Protect Virtual Server container name from the result of the `hpvs list` command by specifying the Secure Service Container partition name or IP address. For example, `stvn-aug27-10001` is the container name listed as the `Names` field in the command output. Note that you can use the `--lpar_index` parameter to list the container names on multiple Secure Service Container partitions, the value of the `--lpar_index` parameter is based on the partitions list in the `hosts` file. For more information, see Commands for Hyper Protect Virtual Server containers.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 hpvs
list --lpar [LPARNAMEORADDRESS]
```

1. Update the Hyper Protect Virtual Server container with a different resource configuration.

   (1). Update CPU, memory, or both settings for the Hyper Protect Virtual Server container. The following command example updates the Hyper Protect Virtual Server container `stvn-aug27-10001` to use one CPU thread and 1024 MB memory. You can also use the `hpvs-config.yaml` file to specify the different resource configuration, and avoid using the `--cpu` or `--memory` parameters in the command line. Note that the configuration values in the `hpvs-config.yaml` file are used if the corresponding parameters are not specified when running the command.

```
 docker run --rm -it --net host -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-
installer:1.2.0 hpvs update --name stvn-aug27-10001 --cpu 1 --memory 1024 --lpar
[LPARNAMEORADDRESS]
```

(2). Update the size of the quotagroup used by the Hyper Protect Virtual Server container. The following command example updates the Hyper Protect Virtual Server container `stvn-aug27-10001` to resize the quotagroup `qg-test` to 150 GB.

```
docker run --rm -it --net host -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-
installer:1.2.0 hpvs update --name stvn-aug27-10001 --quotagroup qg-test --quotagroup-
storage 150G --lpar [LPARNAMEORADDRESS]
```

Note: If you update the size of the quotagroup of a Hyper Protect Virtual Server container by using the `quotagroup update` command, you must manually restart the Hyper Protect Virtual Server to apply the change. For example,

```
docker run --rm -it --net host -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-
installer:1.2.0 quotagroup update --name qg-test --size 150
docker run --rm -it --net host -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-
installer:1.2.0 hpvs update --name stvn-aug27-10001 --lpar [LPARNAMEORADDRESS]
```

2. If you want to update the Hyper Protect Virtual Server container to use a different image tag, use the following command example.

```
docker run --rm -it --net host -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-
installer:1.2.0 hpvs update --name stvn-aug27-10001 --imagetag newtag --lpar
[LPARNAMEORADDRESS]
```

3. Check the Hyper Protect Virtual Server container is updated with the configuration. The following command example checks the Hyper Protect Virtual Server container `stvn-aug27-10001`.

```
docker run --rm -it --net host -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-
installer:1.2.0 hpvs get --name stvn-aug27-10001
```

Note: * For CPU updates, check the value for the `"RUNQ_CPU="` string in the output. * For Memory updates, check the value for the `"RUNQ_MEM="` string in the output.

For more information about the `hpvs` commands, see Commands for Hyper Protect Virtual Server containers.

Refreshing registered repositories with a new signing key pair

You can update the repositories on the Secure Service Partition with a new signing key pair, and revoke the access from an existing key pair.

Note:

- You must create the new signing key pair by using the same user ID associated with the key pair to be invoked. For more information, see Generating a new keypair.
- If this task is being performed because the original key pair was compromised, the generation and loading of the new encrypted repository file signed by the new key must be done using a trusted channel.

This procedure is intended for users with the role *Cloud administrator*.

Before you begin

- Ensure that you have the following information of the key pair to be revoked.
  - The private key file

  - The public key file
  - The passphrase of the private key
- Ensure you have a list of repositories registered by using the key to be revoked.
- Ensure that you install GnuPG or a similar tool on your x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server. For more information, see The GNU Privacy Handbook.

Procedure

Complete the following steps under the `<installation_directory>/VS/regfile-cli/config` directory with root user authority.

1. Retrieve the key fingerprint of the key pair to be revoked. Note that if you have multiple gpg key pairs under your user ID, ensure that you specify the correct key in the command.

   For example, the following command retrieves the key fingerprint of `isv_user.private` with its private passphrase `over-the-lazy-dog`.

   ```
   gpg --pinentry-mode=loopback --passphrase over-the-lazy-dog --import isv_user.private
   gpg --fingerprint "isv_user"
   ```

2. Generate the revocation certificate for the key pair to be invoked.

   ```
   gpg --output isv_user.asc --gen-revoke ${old_key_fingerprint}
   ```

3. Generate a new signing key pair with an identical user ID that is used when creating the key pair to be revoked. Note the passphrase can be different from the existing one.

   For example, the following commands create a GPG key pair, export the public key `isv_new_user.pub` and the private key `isv_new_user.private`. The user ID `Name-Real` of the existing key pair and the new key pair must use the same value `isv_user`. The new key pair is protected by using a passphrase `over-the-new-lazy-dog`.

   ```
   cat >isv_definition_keys <<EOF
     %echo Generating registration definition key
     Key-Type: RSA
     Key-Length: 4096
     Subkey-Type: RSA
     Subkey-Length: 4096
     Name-Real: isv_user
     Expire-Date: 0
     Passphrase: over-the-new-lazy-dog
     # Do a commit here, so that we can later print "done" :-)
     %commit
     %echo done
   EOF
   gpg -a --batch --generate-key isv_definition_keys
   gpg --fingerprint "isv_user"
   gpg --armor --pinentry-mode=loopback --passphrase over-the-new-lazy-dog --export-secret-keys
   ${new_key_fingerprint} > isv_new_user.private
   gpg --armor --export ${new_key_fingerprint} > isv_new_user.pub
   ```

4. Create a new repository definition file with the revocation certificate, the existing key pair information, and new key pair information. Note that the passphrase of the new private key is not needed.

   For example, the following example creates and signs a new repository definition file `MyDockerApp.json-rel.enc` with the new key pair `isv_new_user.pub` and `isv_new_user.private`. The `MyDockerApp.json-rel.enc` file contains information that the old key pair `isv_user.pub` and `isv_user.private` are to be revoked.

   ```
   docker run --rm -v $(pwd):/regfile-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
   regfile create \
   -u docker_readonly_user -p [yourDockerPassWord] -s docker.io \
   -r docker_base_user/MyDockerApp -k isv_user.pub -K isv_user.private \
   -ph over-the-lazy-dog -nk isv_new_user.pub -nK isv_new_user.private \
   --revocation-cert isv_user.asc
   ```

5. Go to the `<installation_directory>/VS/hpvs-cli/config` directory, copy the repository definition file into the directory, and then update the repositories on the Secure Service Container by using the new repository definition file to apply the changes. After the command completes, only the images signed by the new key can be pushed into the repositories.

   For example, the following command instructs the repository `MyDockerApp` on the Secure Service Container partition to only accept the images signed by the new key pair in the repository definition file `MyDockerApp.json-rel.enc`.

   ```
   cd <installation_directory>/VS/hpvs-cli/config
   cp -p <installation_directory>/VS/regfile-cli/config/MyDockerApp.json-rel.enc .
   docker run --rm -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
   repository update \
   --repoid MyDockerApp --repofile MyDockerApp.json-rel.enc --lpar [LPARNAMEORADDRESS]
   ```

Monitoring IBM Hyper Protect Virtual Servers

You can monitor a wide range of components with the monitoring infrastructure provided by IBM Hyper Protect Virtual Servers.

Note:

- The monitoring metrics are collected from Secure Service Container partitions.
- Only Hyper Protect hosting appliance and Secure Service Container partition level metrics are supported for IBM Hyper Protect Virtual Servers v1.2.0.
- The IBM Hyper Protect Virtual Servers monitoring functionality may be used to monitor IBM Secure Service Container partitions running IBM Secure Service Container for IBM Cloud Private.

For more information about the commands and metrics about the monitoring infrastructure, see Commands for monitoring and Metrics collected by the monitoring infrastructure.

This procedure is intended for users with the role *cloud administrator*.

Before you begin

- Ensure you have the IP address of the Secure Service Container partition.
- Ensure that port 8443 is available for the monitoring infrastructure on the Secure Service Container partition.
- Ensure that IBM Hyper Protect Virtual Servers CLI tools are installed on the x86 or Linux on IBM Z/ LinuxONE (such as s390x architecture) management server. For more information, see Installing the IBM Hyper Protect Virtual Servers CLI tool.
- Refer to the checklist that you prepared for the Hyper Protect Virtual Server on this topic Planning for the environment.

Procedure

Complete the following steps under the `<installation_directory>/VS/monitoring-cli/config` directory with root user authority.

1. Create a `./config/hosts` file with connection information for the underlying RESTful API calls to the Secure Service Container partition(s) to have the monitoring infrastructure created on. You can use the `hosts.example` file as the reference when configuring the `hosts` file.

The following hosts example file contains the Master user ID and password of the Secure Service Container partition with IP address `10.152.151.105`.

```
[lpars]
10.152.151.105 rest_user="someuser" rest_pass="somepassword"
```

2. Generate certificates for the secure communication between the Hyper Protect monitoring infrastructure (server) and the monitoring client. The monitor client invoke the `collectd-exporter` endpoint on the server to show the collected metrics. Note that when you generate certificates, use `collectdhost-<metric-dn-suffix>.<dns-name>` or `*.<dns-name>` as the common name. A wild card certificate with `*.<dns-name>` common name can be used across multiple partitions.

   - To generate self signed certificates, see Creating self signed certificates for the monitoring infrastructure.
   - To generate CA signed certificates, see Creating CA signed certificates for the monitoring infrastructure.

3. Create a `./keys` directory or a similar directory under the `<installation_directory>/VS/monitoring/config` directory.

```
mkdir <installation_directory>/VS/monitoring-cli/config/keys
```

4. Copy the certificate and key files for the monitoring infrastructure into the `./keys` directory. The certificate and key are used by monitoring infrastructure to encrypt the metric data in transit. If you create the client certificate to enable the client authentication, you can also copy the client certificate to the `./keys` directory.

```
cp -p server.key ./keys/server.key
cp -p server-certificate.pem ./keys/server-certificate.pem
cp -p client-certificate.pem ./keys/client-certificate.pem
```

5. Create the `./config/monitoring.yaml` file with the settings for your monitoring infrastructure. You can use the `./config/monitoring.yaml.example` file as the reference when configuring the `monitoring.yaml` file. For more information about the parameters, see monitoring.yaml.

```
LPARS:
 - ipaddress: 10.152.151.105
   containers:
   - template: monitoring-host
     private-key-server: /monitoring-cli/config/keys/server.key
     public-cert-server: /monitoring-cli/config/keys/server-certificate.pem
     public-cert-client: /monitoring-cli/config/keys/client-certificate.pem
     metric-dn-suffix: first
     dns-name: example.com
   - template: collectd-host
monitoring-host:
  name: monitoring-host
  cpu: 1
  memory: 512
  imagetag: 1.2.0
collectd-host:
  name: collectd-host
  imagetag: 1.2.0
```

   Where:

   - `ipaddress` is the IP address of the Secure Service Container partition.
   - `imagetag` is required and identical to the version number of IBM Hyper Protect Virtual Servers . For example, 1.2.0. The `imagetag` version number value can be found in the `<installation_directory>/version` file.
   - You can only create one set of `monitoring-host` and `collectd-host` containers on one IBM Hyper Protect Virtual Servers installation.
   - You must specify both `monitoring-host` and `collectd-host` containers in the `monitoring.yaml` file.

- If you want to configure multiple Secure Service Container partitions with the monitoring infrastructure, refer to the `./config/monitoring.yaml.example` file for the details.
- For more supported parameters in the `monitoring.yaml` file, see monitoring.yaml.

6. Update the `/etc/hosts` file to include an entry for the `collectd-host` container as the following code snippet. The entry is constructed in the format `<LPAR_IP> collectdhost-<metric-dn-suffix>.<dns-name>`. If you have monitoring containers being created on the different Secure Service Container partitions, you must include the collectd-host container for each partition.

```
10.152.151.105 collectdhost-first.example.com
```

7. Create the monitoring infrastructure by using the `monitoring create` command.

```
docker run --rm -it -v $(pwd):/monitoring-cli/config ibmzcontainers/hpvs-cli-
installer:1.2.0 monitoring create
```

Note:

- After the command completes, a `monitoring-configuration.yaml` file is generated as the following example.

```
         LPARS:
         - containers:
           - cpu: 1
             dns_name: example.com
             imagetag: 1.2.0
             metric_dn_suffix: first
             name: monitoring-host
             private_key_server: /monitoring-cli/config/keys/server.key
             public_cert_client: /monitoring-cli/config/keys/client-certificate.pem
             public_cert_server: /monitoring-cli/config/keys/server-certificate.pem
           - imagetag: 1.2.0
             name: collectd-host
           ipaddress: 10.152.151.105
```

- After the `monitoring create` command completes, monitoring infrastructure registers the `monitoring-host` container in the repository `ibmzcontainers/monitoring` with the repository ID `Monitoring` , and `collectd-host` container in the repository `ibmzcontainers/collectd-host` with the repository ID `CollectdHost` on the Secure Service Container partition.
- To verify both `Monitoring` and `CollectdHost` repositories are created, run the `repository list` command under the `<installation_directory>/VS/hpvs-cli/config` directory to check the output.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
repository list --lpar [LPARNAMEORADDRESS]
```

- For more information about the `repository` command, see Commands for repositories.
- For more information about the metrics being collected by the monitoring infrastructure, see Metrics collected by the monitoring infrastructure.

8. (Optional) To stop the monitoring infrastructure, use the `monitoring stop` command as the following example.

```
docker run --rm -it -v $(pwd):/monitoring-cli/config ibmzcontainers/hpvs-cli-
installer:1.2.0 monitoring stop
```

9. (Optional) To delete the `monitoring-host` and `collectd-host` containers of the monitoring infrastructure, use the `monitoring delete` command as the following example.

```
docker run --rm -it -v $(pwd):/monitoring-cli/config ibmzcontainers/hpvs-cli-
installer:1.2.0 monitoring delete
```

10. (Optional) To completely delete the monitoring infrastructure and monitoring repositories created on the Secure Service Container partition, use the `monitoring cleanup` command as the following example.

```
docker run --rm -it -v $(pwd):/monitoring-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
monitoring cleanup
```

For more information on how to manage the status of monitoring infrastructure or other supported actions in the `monitoring` command, see Commands for monitoring.

Next

You can configure any client tools that use the `collectd-exporter` endpoint to collect the monitoring metrics from the monitoring infrastructure.

- The following example file `prometheus.yml` shows how you can configure Prometheus to use the metrics collected by the monitoring infrastructure in IBM Hyper Protect Virtual Servers.

  ```
  global:
    scrape_interval: 10s
  scrape_configs:
    - job_name: 'prometheus'
      static_configs:
            - targets: ['collectdhost-first.example.com:8443']
      scheme: https
      tls_config:
         ca_file: /etc/prometheus/keys/server-certificate.pem
         cert_file: /etc/prometheus/keys/client-certificate.pem
         key_file:  /etc/prometheus/keys/client.key
         server_name: collectdhost-first.example.com
  ```

  Note:

  – With a properly configured `prometheus.yml` file, and properly configured, created, and running `monitoring-host` and `collectd-host` containers on the Secure Service Container partition, the targets view of the prometheus server will show the target Secure Service Container partition "State" as "UP" with a default color green.

  – To access the targets view of the Prometheus server, enter the following link with the actual IP address or the hostname of the Prometheus server in your browser. `http://<prometheus_server_IP_address_or_hostname>:9090/targets`

- The following example shows how you can view the current monitoring metrics for the `collectdhost-first.example.com` target Secure Service Container partition by using the `wget` utility. In this example, the `wget` command is executed from the directory containing the `prometheus.yml` file's keys, with the output written to the `metrics` file, or a derivative file if the `metrics` file already exists.

  ```
  wget https://collectdhost-first.example.com:8443/metrics --ca-certificate=server-
  certificate.pem --certificate=client-certificate.pem --private-key=client.key
  ```

  Note: You can also use the `wget` utility with the `--no-check-certificate` option to skip the SSL certificate validation when retrieving the monitoring metrics from the target Secure Service Container partition.

  ```
  wget https://collectdhost-first.pok.stglabs.ibm.com:8443/metrics --ca-certificate=server-
  certificate.pem --certificate=client-certificate.pem --private-key=client.key --no-check-
  certificate
  ```

Generating certificates for secure communication

You can use one of the following topics when you create certificates for secure communication between the monitoring infrastructure (server) and monitoring tools (client).

- Creating self signed certificates for the monitoring infrastructure
- Creating CA signed certificates for the monitoring infrastructure

Creating self signed certificates for the monitoring infrastructure

You can generate self-signed certificates for the monitoring infrastructure by using the `openssl` utility or any other certificate generation tools that comply with your organization rules.

This procedure is intended for users with the role *Application developer or ISV*.

Before you begin

- Ensure that you install the OpenSSL or similar tool on a workstation that you can use to generate the certificates.

Procedure

Complete the following steps on your workstation with root user authority.

1. Make a note of the details to generate certificates such as the Common Name (CN) and Subject Alternative Name (SAN) that you intend to set in the certificate. For example, `example.com`, `myorg.example.com`. For more information, see OpenSSL configuration examples
2. Create a directory on your workstation to run the `openssl` command or any similar tool.

   ```
   mkdir /home/myuser/certificates
   cd /home/myuser/certificates
   ```

3. Create a private key by using the following command. After the command completes, a private key will be created under the current directory.
   - For a server certificate, use the following command.

   ```
   ```

   openssl genrsa -out server.key 4096

   ```
   * For a client certificate, use the following command.
   ```

   openssl genrsa -out client.key 4096

   ```
   ```

4. Create a Certificate Signing Request (CSR) based on the private key you just created. You will be asked to enter values for various certificate fields such as Organization Unit (OU), Common Name (CN), Email, Country Code, State or Province name, City, Organization or Company Name. After the command completes, a CSR file is created under the current directory.

   a). If you choose to enter the values for the certificate fields as prompted, then run the following command to create a server certificate.

   ```
   openssl req -new -key server.key -out server-certificate.csr
   ```

   Or run the following command to create a client certificate.

   ```
   openssl req -new -key client.key -out client-certificate.csr
   ```

   b). If you choose to avoid entering these fields on command prompt in an interactive manner, then create a configuration file such as `server-certificate.cnf` and provide the list of these fields and their values as in the following the command for a server certificate.

   ```
   openssl req -new -config server-certificate.cnf -key server.key -out server-certificate.csr
   ```

Or a `client-certificate.cnf` configuration file as in the following command for a client certificate.

```
openssl req -new -config client-certificate.cnf -key client.key -out client-certificate.csr
```

Note:

- To create a server certificate, include the entry `extendedKeyUsage=serverAuth` in the `server-certificate.cnf` file.
- To create a client certificate, include the entry `extendedKeyUsage=clientAuth` in the `client-certificate.cnf` file.
- For the sample configuration files, see OpenSSL configuration examples.

5. Create a self-signed certificate based on the CSR you just created. After the command completes, the certificate is created under the current directory.

   - For a self-signed server certificate, use the following command.

   ```
   openssl x509 -req -days 365 -in server-certificate.csr -signkey server.key -out server-
   certificate.pem
   ```

   - For a self-signed client certificate, use the following command.

   ```
   openssl x509 -req -days 365 -in client-certificate.csr -signkey client.key -out client-
   certificate.pem
   ```

   Note: You can choose the certificate extension to be `.cer`, `.pem`, or `crt`. For example, `server-certificate.cer`.

Next

You can proceed with configuring of the monitoring infrastructure as instructed in the Monitoring IBM Hyper Protect Virtual Servers topic.

Creating CA signed certificates for the monitoring infrastructure

You can generate Certificate Authority (CA) Root and CA signed certificates for the monitoring infrastructure by using the `openssl` utility or any other certificate generation tools that comply with your organization rules.

This procedure is intended for users with the role *Application developer or ISV*.

Before you begin

- Ensure that you install the OpenSSL or similar tool on a workstation that you can use to generate the certificates.

Procedure

Complete the following steps on your workstation with root user authority.

1. Create a directory on your workstation to run the `openssl` command or any similar tool.

   ```
   mkdir /home/myuser/ca-certificates
   cd /home/myuser/ca-certificates
   ```

2. Create CA Root certificates by using the following procedure. The root CA certificate will be used to sign CA certificates.

   a. Create the CA root private key. After the command completes, the CA root private key `myrootCA.key` is generated under the current directory. For example, `/home/myuser/ca-certificates/myrootCA.key`.

   ```
   openssl genrsa -out myrootCA.key 4096
   ```

b. Create the Certificate Signing Request (CSR) based on the CA root private key. After the command completes, the CSR `myrootCA.csr` is generaterd under the current directory. For example, `/home/myuser/ca-certificates/myrootCA.csr`.

- The command prompts you to enter values for various certificate fields, such as Organization Unit (OU), Common Name (CN), Email, Country Code, State/Province name, City, Organization or Company Name.

```
openssl req -verbose -new -key myrootCA.key -out myrootCA.csr -sha256
```

- If you want to avoid entering each value when the command runs, you can use a OpenSSL configuration file to create the self signed CSR. For example, `/home/myuser/ca-certificates/myca.cnf`. For more information about the OpenSSL configuration file, see OpenSSL configuration examples.

c. Create other required configuration and OpenSSL database by using the following command.

```
  cd /home/myuser/ca-certificates/
  touch index.txt
  touch index.txt.attr
  touch serial
  mkdir crl
  mkdir newcerts
```

Note:

- Those files are required to successfully create a CA root certificate.
- You must update the file `serial` and enter a number in the file. For example, `1000`. This number signifies the serial number of the certificates being created.

d. Create the CA root certificate by using the following command. After the command completes, the CA root certificates `myrootCA.crt` is created under the current directory.

```
 openssl ca --config /home/myuser/ca-certificates/myca.cnf -out myrootCA.crt -keyfile
myrootCA.key -verbose -selfsign -md sha256 -infiles myrootCA.csr
```

e. Validate the CA root certificate by using the following command. After the command completes, the details of the CA root certificate is printed in the output.

```
 openssl x509 -noout -text -in myrootCA.crt
```

3. Create the CSR for the CA signed server certificate or client certificate by following the instructions on Creating self signed certificates for the monitoring infrastructure. After the commands complete, the CSR is created as the `/home/myuser/certificates/server-certificate.csr` file or `/home/myuser/certificates/client-certificate.csr` file.
4. Create the CA signed certificates by using the CA root certificate.

   - To create the CA signed server certificate, run the following command.

```
 openssl x509 -req -days 365 -in /home/myuser/certificates/server-certificate.csr -CA /home/
myuser/ca-certificates/myrootCA.crt -CAkey /home/myuser/ca-certificates/myrootCA.key -
CAcreateserial -out ./server-certificate.crt
```

   - To create the CA signed client certificate, run the following command.

```
 openssl x509 -req -days 365 -in /home/myuser/certificates/client-certificate.csr -CA /home/
myuser/ca-certificates/myrootCA.crt -CAkey /home/myuser/ca-certificates/myrootCA.key -
CAcreateserial -out ./client-certificate.crt
```

Next

You can proceed with configuring of the monitoring infrastructure as instructed in the Monitoring IBM Hyper Protect Virtual Servers topic.

Integrating with the EP11 library

You can connect to your (Enterprise PKCS #11) EP11 instantiation using a gRPC (GREP11) container on the Secure Service Container partition, and then use the Hardware Security Module (HSM) to perform numerous cryptographic operations, such as generating asymmetric (public and private) key pairs for digital signing and verification, or generating symmetric keys for encrypting data as needed by the deployed applications. For more information, see EP11.

This procedure is intended for users with the role *cloud administrator*.

Before you begin

- Check with your system administrator that the crypto express domain is configured in the EP11 mode. For more information, see **Chapter 8 - Using the Crypto Module Notebook to administer EP11 crypto modules** in the Cryptographic Services ICSF Trusted Key Entry Workstation (TKE) User's Guide.
- Check with your system administrator that the master key is initialized. For more information, see TKE workstation demo on Youtube, Master key setup - TKE EP11 Configuration Tasks video on youtube, and the **Reviewing and changing current logical partition cryptographic controls** topic in the Processor Resource/Systems Manager Planning Guide.
- Ensure that IBM Hyper Protect Virtual Servers CLI tools are installed on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server. For more information, see Installing the IBM Hyper Protect Virtual Servers CLI tool.
- Refer to the checklist that you prepared for the Hyper Protect Virtual Server on this topic Planning for the environment.

Procedure

Complete the following steps under the `<installation_directory>/VS/grep11-cli/config` directory with root user authority.

1. Generate certificates for the secure communication between the Hyper Protect GREP11 container and the grep11 client. For more information, see Creating certificates for GREP11 containers.
2. Update the `hosts` file with the Secure Service Container partition information.

```
[lpar]
grep11 ip="10.152.151.105" rest_user="someuser" rest_pass="somepassword"
```

3. Check the available crypto domains on the HSM by using the `crypto list` command. For more information about the `crypo` commands, see Commands for crypto domains.

```
docker run --rm -it -v $(pwd):/grep11-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
crypto list
```

The command might show the following output indicating that the crypto domain `09.000a` or `09.000b` are available on the HSM.

```
"domains": {
            "vfio_ap": [
                "07.000a",
                "07.000b",
                "09.000a",
                "09.000b"
            ]
        },
        "mdevs": {
                "matrix": [
                    "07.000a"
                ]
            },
                "matrix": [
                    "07.000b"
                ]
          }
```

4. Update the `grep11-config.yaml` file for your GREP11 containers.

```
grep11:
  name: 'grep11-container-2'
  imagetag: '1.2.0'
  crypto_domain: '09.000b'
  memory: '5000'
  network:
    network_name: 'my-private-network-name'
    port: '9876'
    ip: '192.168.10.106'
  mutual_tls: true
  certificate: '/grep11-cli/config/server.pem'
  key: '/grep11-cli/config/server-key.pem'
  #if mutual_tls is true
  cacertificate: '/grep11-cli/config/ca.pem'
```

Note:

- If you set the value of `network_name` key to be `default`, then you do not have to set the value for `ip`. However, you must set the value for the `port` to enable the port forwarding on the Secure Service Container partition.
- If you want the GREP11 container in your existing network of IBM Hyper Protect Virtual Servers and accessible internally, you must set the key `network_name` and `ip` to be part of your existing IBM Hyper Protect Virtual Servers network, and the port will be always 9876. The `network_name` can be created as VLAN or Ethernet type connection on the Secure Service Container partition. For more information, see Configuring the network on the Secure Service Container partition.
- If you want the GREP11 container to be accessible by the external workload, and set the value of `network_name` key to be default, then you do not have to set a value for the key `ip`. However, you must set a value for the key `port` to enable the port forwarding on the Secure Service Container partition.
- Ensure to copy your certificates and keys into the `grep11-cli/config` directory to match the configuration in the `grep11-config.yaml` file.
- For more information of supported parameters, see grep11-config.yaml.

5. Create the GREP11 container by using the `grep11 create` command. For more information about the `grep11` commands, see Commands for GREP11.

```
docker run --rm -it -v $(pwd):/grep11-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
grep11 create
```

6. Check the status of the GREP11 container by using the `grep11 status` command.

```
docker run --rm -it -v $(pwd):/grep11-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
grep11 status
```

7. (Optional) To stop the GREP11 container, use the `grep11 stop` command as the following example.

```
docker run --rm -it -v $(pwd):/grep11-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
grep11 stop
```

8. (Optional) To delete the GREP11 container, use the `grep11 delete` command as the following example.

```
docker run --rm -it -v $(pwd):/grep11-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
grep11 delete
```

For more information about the TKE, check out the following videos on youtube.

- TKE Introduction Videos 1 Introduction to TKE

- TKE Introduction Videos 2 Workstation Setup Wizard
- IBM Z TKE 9.1 Introduction To TKE Policy Wizard Family
- IBM Z TKE 9.1 Introduction To The TKE Smart Card 00RY790
- IBM Z TKE 9.1 TKE Smart Card Wizard
- IBM Z TKE 9.1 TKE Workstation Logon Profile Wizard
- IBM Z TKE 9.1 CCA Setup Module Policy Wizard

Next

You can update your application to use the asymmetric key pairs provided by the GREP11 containers. For more informations, refer to Accessing the GREP11 container within your code.

Creating certificates for GREP11 containers

When configuring the GREP11 container, you can create certificates for the one way or mutual authentication over TLS.

This procedure is intended for users with the role *cloud administrator*.

Procedure

Complete the following procedure depending on how you want to configure the communication between GREP11 container and client application.

- For mutual TLS communication, you can only create CA signed certificates.

  To create the CA certificate for mutual authentication over TLS, use the `golang-massl` utility as in the following steps. For more information, see golang-massl.

  1. Install the cfssl toolkit either by using the `apt-get install golang-cfssl` command or following the instructions on this page.
  2. Generate the CA certificate and private key by using the `golang-mass` utility and the certificate from the trusted CA.

     ```
     git clone https://github.com/wolfeidau/golang-massl.git
     cd golang-massl/certs
     cfssl gencert -initca ca-csr.json | cfssljson -bare ca
     ```

  3. Generate a server certificate by using the provided CSR.

     ```
     cfssl gencert \
      -ca=ca.pem  \
      -ca-key=ca-key.pem \
      -config=ca-config.json  \
      -hostname=domain-name:port,ip \
      -profile=massl server-csr.json | cfssljson -bare server
     ```

  4. Generate a client certificate by using the provided CSR.

     ```
     cfssl gencert \
      -ca=ca.pem  \
      -ca-key=ca-key.pem \
      -config=ca-config.json  \
      -profile=massl client-csr.json | cfssljson -bare client
     ```

  5. Copy the server certificate, server public key, and CA certificate into the `<installation_directory>/VS/grep11-cli/config` directory.

     ```
     cp -p server.pem grep11-cli/config/
     cp -p server-key.pem grep11-cli/config/
     cp -p ca.pem grep11-cli/config/
     ```

6. Use the following files in your applications.
   - CA certificate: `ca.pem`
   - Client public key: `client.pem`
   - Client private key: `client-key.pem`

- For one way TLS communication, you can only create self-signed certificates.
  - To create the self-signed certificate for one way authentication over TLS for a reserved or private IP address assigned to the GREP11 container, use the `certstrap` utility as in the following steps. For more information, see certstrap.
    1. Clone the `certstrap` utility to your x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.

    ```
    git clone https://github.com/square/certstrap.git
    cd certstrap
    git checkout tags/v1.2.0
    ```

    1. Replace the lines beginning with GOARCH= in the `build` file with the following one to let the Linux detect the architectural environment (x86 or s390x) that the build process is executing within.

    ```
    GOARCH=$([[ $(uname -p) == "x86_64" ]] && echo "amd64" || echo "s390x") GOOS=linux go build
    -ldflags "-X main.release=$BUILD_TAG" -o bin/certstrap ${REPO_PATH}
    ```

    1. Build the `certstrap` utility.

    ```
    ./build
    ```

    1. Initialize a new certificate authority for self signing. Note that the value of `--common-name` must be consist with the host name of the GREP11 container.

    ```
    ./bin/certstrap init --common-name "grep11.example.com"
    ```

    1. Copy the keys into the `<installation_directory>/VS/grep11-cli/config` directory.

    ```
    cp -p out/grep11.example.com.crt grep11-cli/config/cert.pem
    cp -p out/grep11.example.com.key grep11-cli/config/key.pem
    ```

  - To create the self-signed certificate for one way authentication over TLS with an unreserved or public IP, use the `openssl` utility as in the following example.

    ```
    openssl req -newkey rsa:2048 \
      -new -nodes -x509 \
      -days 3650 \
      -out cert.pem \
      -keyout key.pem \
      -subj "/C=GB/O=IBM/CN=grep11.example.com"
    ```

  The command generates your certificate `cert.pem` and private key `key.pem`, which must be placed under the `<installation_directory>/VS/grep11-cli/config` directory.

Next

You can create the GREP11 container by following the instructions in Integrating with the EP11 library.

Accessing the GREP11 container within your code

You can use the Enterprise PKCS #11 (EP11) API over gRPC (also referred to as GREP11 API) to remotely access the GREP11 container on the Secure Service Container partition for data encryption and management.

This procedure is intended for users with the role *cloud administrator*.

Before you begin

- Ensure that you have the connection information to the GREP11 container on the Secure Service Container partition. For example, `grep11.example.com:21876`.
- Ensure that you have the client certificates to authenticate with the GREP11 container.
  - For one-way TLS communication, the public certificate `cert.pem`
  - For two-way TLS communicattion, the client private key `client-key.pem`, the root certificate `ca.pem`, the public certificate `cert.pem`

Procedure

The following Golang code examples show how to generate asymmetric (public and private) key pairs by using the GREP11 API, and assume that additional required Golang packages are included via import statements, such as the gRPC, `http`, and `pb "github.com/ibm-developer/ibm-cloud-hyperprotectcrypto/golang/grpc"` statement used by GREP11 to perform API function calls.

For more code examples, see this sample repository.

Note: You must use certificate-based authentication as in the following code to access the GREP11 container on the Secure Service Container partition.

- For one-way TLS authentication, use the following code snippet.

```
# The URL of GREP11 container in this example grep11.example.com:21876
const address = "grep11.example.com:21876"
# cert.pem is the same server certificate because it is one way TLS
var cert, _ = credentials.NewClientTLSFromFile("cert.pem", "")
var callOpts = []grpc.DialOption{
         grpc.WithTransportCredentials(cert),
 }
func Example_getMechanismInfo() {
    conn, err := grpc.Dial(address, callOpts...)
    if err != nil {
        panic(fmt.Errorf("Could not connect to server: %s", err))
    }
    defer conn.Close()

    cryptoClient := pb.NewCryptoClient(conn)

    mechanismListRequest := &pb.GetMechanismListRequest{}
    mechanismListResponse, err := cryptoClient.GetMechanismList(context.Background(),
mechanismListRequest)
    if err != nil {
        panic(fmt.Errorf("Get mechanism list error: %s", err))
    }
    fmt.Printf("Got mechanism list:\n%v ...\n", mechanismListResponse.Mechs[:1])

    mechanismInfoRequest := &pb.GetMechanismInfoRequest{
        Mech: ep11.CKM_RSA_PKCS,
    }
    _, err = cryptoClient.GetMechanismInfo(context.Background(), mechanismInfoRequest)
    if err != nil {
        panic(fmt.Errorf("Get mechanism info error: %s", err))
    }
}
```

- For mutual TLS authentication, use the following code snippet.

```
# The URL of GREP11 container in this example grep11.example.com:21876
const address = "grep11.example.com:21876"
var (
    # client certificate
    crt = "client.pem"
    # client private key
    key = "client-key.pem"
    # CA certificate
    ca  = "ca.pem"
)
var certificate, _ = tls.LoadX509KeyPair(crt, key)
var certPool = x509.NewCertPool()
var cacert, _ = ioutil.ReadFile(ca)
var a = certPool.AppendCertsFromPEM(cacert)
var creds = credentials.NewTLS(&tls.Config{
    ServerName:   address, // NOTE: this is required!
    Certificates: []tls.Certificate{certificate},
    RootCAs:      certPool,
})
var callOpts = []grpc.DialOption{
        grpc.WithTransportCredentials(creds),
}
func Example_getMechanismInfo() {
    conn, err := grpc.Dial(address, grpc.WithTransportCredentials(creds))
    if err != nil {
        panic(fmt.Errorf("Could not connect to server: %s", err))
    }
    defer conn.Close()

    cryptoClient := pb.NewCryptoClient(conn)

    mechanismListRequest := &pb.GetMechanismListRequest{}
    mechanismListResponse, err := cryptoClient.GetMechanismList(context.Background(),
mechanismListRequest)
    if err != nil {
        panic(fmt.Errorf("Get mechanism list error: %s", err))
    }
    fmt.Printf("Got mechanism list:\n%v ...\n", mechanismListResponse.Mechs[:1])

    mechanismInfoRequest := &pb.GetMechanismInfoRequest{
        Mech: ep11.CKM_RSA_PKCS,
    }
    _, err = cryptoClient.GetMechanismInfo(context.Background(), mechanismInfoRequest)
    if err != nil {
        panic(fmt.Errorf("Get mechanism info error: %s", err))
    }
}
```

Backing up and restoring IBM Hyper Protect Virtual Servers

You can create backups and restore from those backups as part of your disaster recovery plan.

Note: Backup and restore feature is supported by the following components:

- Hyper Protect hosting appliance
- Hyper Protect virtual server containers
- Secure Build containers

Procedure

To back up and restore IBM Hyper Protect Virtual Servers, complete the following procedure according to your role.

- As a system or appliance administrator, back up and restore the hosting appliance by using the Secure Service Container user interface. For more information, download IBM z Secure Service Container User's Guide from the About page.
  - To create a backup, use the `Export` button on the navigation pane. The configuration file `export.data` will be stored on your local file system.
  - To restore the hosting appliance from a backup, use the `Import` button on the navigation pane, and then upload the exported configuration file `export.data` as instructed. After the restore is completed, the Hyper Protect hosting appliance will be restarted.
- As a cloud administrator, you can back up and restore the Secure Build containers by using the `securebuild` commands. For more information, see Commands for Secure Build containers.

- To create a backup by taking a snapshot of the Secure Build container, run the `securebuild backup` command under the `<installation_directory>/VS/securebuild-cli/config` directory with root user authority.

```
docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
securebuild backup -b <unique-backup-name> -n <secure-build-container-name>
```

Note: The snapshots of the Secure Build containers are stored on the Secure Service Container partition.

- To restore the Secure Build container from a backup, run the `securebuild restore` command under the `<installation_directory>/VS/securebuild-cli/config` directory with root user authority.

```
docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
securebuild restore -b <unique-backup-name> -n <secure-build-container-name>
```

• As an application developer or ISV, you can back up and restore the Hyper Protect Virtual Server containers with your application code by using the `snapshot` commands. For more information, see Commands for snapshots.

- To create a backup for a Hyper Protect Virtual Server container with your application, use the `snapshot create` command as in the following command example.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
snapshot create --snapshot hpvs_snapshot1 --container <hyper-protect-virtual-server-name> --
lpar [LPARNAMEORADDRESS]
```

Note: The snapshots of the Hyper Protect Virtual Server containers are stored on the Secure Service Container partition.

- To restore the Hyper Protect Virtual Server container from a snapshot, use the `snapshot revert` command as in the following command example. You must restart the Hyper Protect Virtual Server container after it is restored from a snapshot.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
snapshot revert --snapshot hpvs_snapshot1 --container <hyper-protect-virtual-server-name> --
lpar [LPARNAMEORADDRESS]
```

Upgrading IBM Hyper Protect Virtual Servers

You can upgrade IBM Hyper Protect Virtual Servers by downloading the latest version from the IBM Passport Advantage website.

Before you begin

• Ensure that you back up all the workload and configuration data. For more information, see Backing up and restoring IBM Hyper Protect Virtual Servers.

Procedure

Complete the following steps on your x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server with root user authority.

1. Download the latest IBM Hyper Protect Virtual Servers. For more information, see Downloading IBM Hyper Protect Virtual Servers.

2. Install the latest Hyper Protect hosting appliance and import the configuration data from the `export.data` file. For more information, see Installing the Hyper Protect hosting appliance and Backing up and restoring IBM Hyper Protect Virtual Servers.
3. Register the Hyper Protect Virtual Server base images `hpvsop-base-ssh` and `hpvsop-base` into the remote repository. Note that the `imagetag` of both images must be from the latest version. For more information, see Registering base images in the remote registry server
4. Load the Secure Build base image into the Secure Service Container partition under the repository ID `SecureDockerBuild`. For more information, see Loading the Secure Build base image on to the Secure Service Container partition.
5. Upgrade the Secure Build base image with the latest version. For more information, see Updating the base image of a Secure Build container.
6. Update the Dockerfile of your application to use the latest Hyper Protect Virtual Server base image version number, and the build the application. For more information, see Building the container image for your application by using the Secure Build.
7. Update the running Hyper Protect Virtual Server containers to use the latest base image version number. For more information, see Updating Hyper Protect Virtual Server containers

Uninstalling IBM Hyper Protect Virtual Servers

You can uninstall IBM Hyper Protect Virtual Servers . Note that you must back up your own workload first.

1. Uninstalling IBM Hyper Protect Virtual Servers CLI tools
2. Uninstalling Secure Service Container partitions

Uninstalling IBM Hyper Protect Virtual Servers CLI tools

You can uninstall IBM Hyper Protect Virtual Servers CLI tools and the modules included in the CLI tools.

This procedure is intended for users with the role *cloud administrator*.

Before you begin

- Ensure that you back up the Hyper Protect Virtual Server container by using the `hpvs backup` command. For more information about the `hpvs` command, see Commands for Hyper Protect Virtual Server containers.
- Ensure that you back up the Secure Build container by using the `securebuild backup` command. For more information about the `securebuild` command, see Commands for Secure Build containers.

Procedure

Complete the following steps under the `<installation_directory>` directory with root user authority.

1. Remove the Hyper Protect Virtual Server container and the quotagroup assigned to the container by using the `hpvs delete` command. For example,

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 hpvs
delete --name myvs --quotagroup qg-for-myvs
```

2. Remove the Secure Build container and its CRT key from your client by using the `securebuild delete` command. For example,

```
docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-
installer:1.2.0 securebuild delete --name securebuild1 --force
```

3. Remove the repository that you registered on the Secure Service Container partition by using the `repository delete` command. For example,

   ```
   docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
   repository delete --repoid myrepo --lpar [LPARNAMEORADDRESS] --force
   ```

4. Uninstall the command line tool by using the `uninstall` command.

   ```
   docker run --rm -it -v $(pwd)/config:/hpvs-cli-installer/config ibmzcontainers/hpvs-cli-
   installer:1.2.0 uninstall
   ```

Uninstalling Secure Service Container partitions

You can stop, deactivate, or delete the Secure Service Container partitions on the IBM Z or LinuxONE machine.

This procedure is intended for users with the role *system administrator*.

Before you begin

- Check whether your host system is running in standard mode (that is, with Processor Resource/System Manager or PR/SM) or has Dynamic Partition Manager (DPM) enabled.
- Check that you download Secure Service Container User's Guide from the About topic.

Procedure

1. (Optional) Export the Secure Service Container configuration by following the description in section *Exporting or importing appliance configuration data* of chapter 14 "Using the Secure Service Container user interface".
2. Stop/deactivate or delete the Secure Service Container partition:
   - On a standard mode system, chapter 7 - Deactivating or deleting a Secure Service Container partition on a standard mode system.
   - On a DPM-enabled system, chapter 12 - Stopping or deleting a Secure Service Container partition on a DPM-enabled system.

Working with Secure Service Container for IBM Cloud Private

You can use the following topics to learn how to work with Secure Service Container for IBM Cloud Private.

- Architecture of Secure Service Container for IBM Cloud Private
- System requirements of Secure Service Container for IBM Cloud Private
- Known issues and limitations of Secure Service Container for IBM Cloud Private
- Installing Secure Service Container for IBM Cloud Private
- Installing IBM Cloud Private cluster
- Upgrading Secure Service Container for IBM Cloud Private
- Reverting the Secure Service Container for IBM Cloud Private upgrade
- Uninstalling Secure Service Container for IBM Cloud Private
- Troubleshooting Secure Service Container for IBM Cloud Private

Architecture of Secure Service Container for IBM Cloud Private

When using the Secure Service Container for IBM Cloud Private, you need to prepare an x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server as the master node for the IBM Cloud Private cluster. The cloud administrator downloads the Secure Service Container for IBM Cloud Private to the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, and initiates the installation of IBM Cloud Private. During the installation, the worker or proxy nodes are provisioned in the Secure Service Container of IBM Z or LinuxONE servers.

Once the IBM Cloud Private cluster is set up, you can deploy containerized IBM Middleware applications as well as use common management tooling for deploying home-grown or other third-party Docker and Kubernetes based applications.

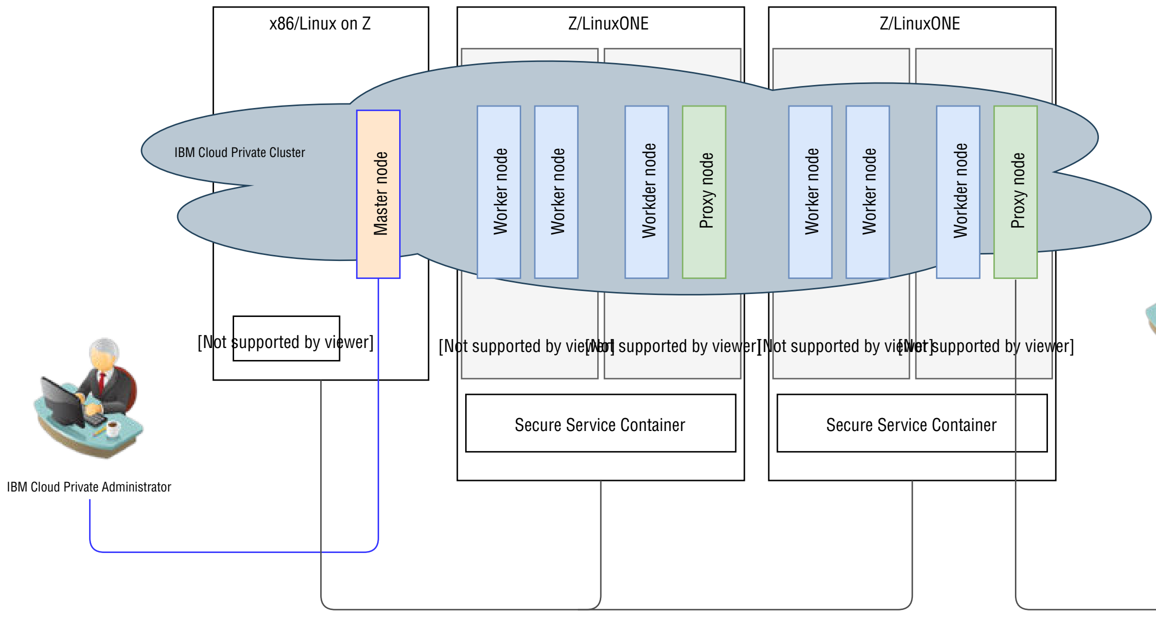


Figure 1. Secure Service Container for IBM Cloud Private - Architecture

System requirements for IBM Secure Service Container for IBM Cloud Private

Software, hardware, and system configuration settings that are required for setting up an IBM Cloud Private cluster on Secure Service Container.

Hardware requirements for the 64-bit x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server

The x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server is used to download Secure Service Container for IBM Cloud Private and IBM Cloud Private installation binary, install the Secure Service Container for IBM Cloud Private and IBM Cloud Private, configure the network for the IBM Cloud Private cluster, and also act as the master and boot node in the IBM Cloud Private cluster.

Table 1. 64-bit x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server requirements

Minimal requirement
8 or more cores with at least 2.4 GHz

Minimal requirement
16 GB RAM
300 GB disk space

Hardware requirements for Secure Service Container partition

You can configure Secure Service Container partitions on the following IBM Z and LinuxONE systems:

- IBM z15 (z15)
- IBM z14 (z14) (machine type 3906 or 3907)
- IBM LinuxONE III (LinuxONE III)
- IBM LinuxONE Emperor II (Emperor II), or IBM LinuxONE Rockhopper II (Rockhopper II)

The suggested practice is to use the latest available firmware for Secure Service Container, which is identified by the engineering changes (ECs) in the following table. To find the latest available EC microcode control levels (MCLs) for Secure Service Container, use the instructions for hardware updates in "Prerequisites for using Secure Service Container" after you download Secure Service Container User's Guide from the About topic.

Table 2. Engineering changes by machine type

Machine Type	Version / Driver	Bundle	Engineering Changes
8561	Version 2.15.0Driver 41	S10 or later	• SE-BCBASE P46639 • SE-BCBOOT P46640 • SE-BCINST P46655
3906 or 3907	Version 2.14.1 Driver 36	S12 or later	• SE-BCBASE P41453 • SE-BCBOOT P41454 • SE-BCINST P41467
3906 or 3907	Version 2.14.0 Driver 32	S53 or later	• SE-BCBASE P42638 • SE-BCBOOT P42639 • SE-BCINST P42652

Note:

- For each worker node running on the Secure Service Container partition, you need to allocate as least 200 GB in the storage pool (140 GB for the docker file system and 60 GB for the root file system). See Hardware requirements and recommendations of IBM Cloud Private for more details.
- For each Secure Service Container partition, you can use either SCSI or extended count key data (ECKD) disks as the storage subsystem. You can allocate 50 GB for the appliance, and at least 200 GB distributed over one or more disks for the storage.
- Secure Service Container for IBM Cloud Private supports hostPath persistent volumes as the storage solution. For more information about the hostPath volume, see hostPath.
- Secure Service Container for IBM Cloud Private also supports GlusterFS as the persistent volumes. You need to allocate at least 80 GB for each GlusterFS node. For more information, see Deploying GlusterFS.

Networking

The Secure Service Container for IBM Cloud Private requires two levels of network to work properly.

- Network among cluster nodes by using the internal IP addresses
- Network for proxy nodes for external requests to the services inside the cluster

Table 3. Supported network interfaces on the Secure Service Container partitions

Interface	Layer 2 network	Layer 3 network
Ethernet	Yes	Yes
VLAN	Yes	Yes
Bond	Yes	Yes

For more information, see Configuring the network for worker and proxy nodes and Configuring the network on the master node.

Supported operating systems and platforms

The operating system for running the worker and proxy nodes is Ubuntu 18.04, which is encapsulated into the Secure Service Container for IBM Cloud Private and will be installed into the Secure Service Container partition as a docker image during the installation.

However, you must set up an x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server to host the master node, which is configured with one of the supported operating systems in the following table.

Table 4. Supported operating systems and platforms

Platform	Operating system
Linux 64-bit	Red Hat Enterprise Linux (RHEL) 7.3, 7.4, 7.5, and 7.6
	Ubuntu 16.04 LTS and 18.04 LTS
	SUSE Linux Enterprise Server (SLES) 12 SP4, and 15

Linux Unified Key Setup (LUKS) hardware encryption on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server can protect the hardware from faulty access. When installing the Ubuntu onto the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, select the **Encrypt the new Ubuntu installation for Security** option to encrypt the hard disk.

Software requirements

You must invest in the following software infrastructure to run the Secure Service Container for IBM Cloud Private solution.

- IBM Cloud Private
- IBM Secure Service Container for IBM Cloud Private, which you can get from IBM Passport Advantage site.
- Feature Code 0104 (Container Hosting Foundation), which is required by the IBM Secure Service Container, and can be ordered on the IBM Z14, IBM LinuxONE Emperor II, and IBM LinuxONE Rockerhopper II servers from the eConfig fulfillment system.

You can contact your sales representatives to obtain the required access to IBM Passport Advantage site and eConfig system.

Supported Docker versions

The Docker release required by Secure Service Container for IBM Cloud Private is identical to the requirements of IBM Cloud Private. See IBM Cloud Private Supported Docker Versions for more details.

You must install one of supported Docker versions on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server. Note the supported Kubernetes version for the master node is 1.11.

The Docker/Kubernetes environment on the Secure Service Container partition is configured during the installation of Secure Service Container for IBM Cloud Private.

Supported IBM Cloud Private versions

The Secure Service Container for IBM Cloud Private solution is tested and developed on the following IBM Cloud Private bundles.

Table 5. Supported IBM Cloud Private versions

Version	Enterprise Edition	Cloud Native	Community Edition
3.2.1	Y	N	Y
3.2.0	Y	N	Y
3.1.2	Y	N	Y
3.1.1	Y	N	Y

Note:

- To install the IBM Cloud Private Community edition, the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server must have the internet access to install the required docker images that are hosted on this external site.
- The installation packages of IBM Cloud Private Enterprise Edition can be acquired from the IBM Passport Advantage site.
- The code snippets or links to IBM Cloud Private documentation uses 3.2.0 to maintain consistency. You can also use a different supported version number or refer to its document if needed.

Required ports

The required ports of Secure Service Container for IBM Cloud Private are identical to the ones of IBM Cloud Private. For more information , see Required ports.

Known issues and limitations

Review the known issues and limitations for Secure Service Container for IBM Cloud Private version 1.2.0.

- Secure Service Container for IBM Cloud Private cannot use IPv6 networks. You have to comment out the settings in the `/etc/hosts` file on each cluster node to remove the IPv6 settings.
- Secure Service Container for IBM Cloud Private requires one OSA card with two virtual devices (one for internal and one for external data traffic) for each IBM Cloud Private cluster.
- GlusterFS persistent volume supported by Secure Service Container for IBM Cloud Private is a technology-preview feature. For more information, For more information, see Deploying GlusterFS.
- You cannot dynamically add or remove glusterFS nodes after the GlusterFS cluster is deployed.
- You cannot enable GlusterFS storage on a Secure Service Container for IBM Cloud Private environment that is upgraded from a previous version.
- You have to manually clean up the GlusterFS resources after you delete the GlusterFS PV (persistent volume) or persistent volume claims (PVCs) by using the IBM Cloud Private CLI or user interface. For more information, see Preparing your nodes for a reinstallation of GlusterFS or IBM Cloud Private.
- The total amount of time to create GlusterFS PVC resources might vary even for a 1GB volume.
- The upgrade of Secure Service Container for IBM Cloud Private from Beta to 1.2.0 is not supported.
- The upgrade of Secure Service Container for IBM Cloud Private from 1.1.0 to 1.2.0 is supported. However, the master node must be on the x86 server after the upgrade.
- You cannot add additional nodes into an existing IBM Private Cloud cluster that is installed by using Secure Service Container for IBM Cloud Private.

- You cannot configure the High Availability (HA) on the IBM Cloud Private cluster that is installed by using Secure Service Container for IBM Cloud Private.
- You cannot use one x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server as the master node for another cluster.
- You must ensure the settings in the `ssc4icp-config.yaml` file are exclusive for different clusters to be installed on the same Secure Service Container partition.
- You must not use `temp` as a cluster name in the `ssc4icp-config.yaml` file.
- Installation of IBM Cloud Private through a firewall is not supported in the Secure Service Container for IBM Cloud Private. Ensure that the `firewall_enable` parameter in the `config.yaml` is set to `false`.
- If you use the Secure Service Container for IBM Cloud Private 1.2.0 bundled in the Hyper Protect Virtual Server 1.1.0, and plan to install the Secure Service Container for IBM Cloud Private to another LPAR, ensure that you remove, rename, or move the `<installation_directory>/SSC4IVP/config/<cluster_name>/cluster-configuration.yaml` file to another directory in order that the installation runs successfully.
- The following table shows the features that are supported on the IBM Cloud Private on IBM Z or LinuxONE version 3.2.0 and Secure Service Container for IBM Cloud Private 1.2.0. For more information about supported features for IBM Cloud Private on different platforms, see Supported features on IBM Cloud Private

Table 1. Supported features matrix

Feature	**IBM Cloud Private on IBM Z or LinuxONE 3.2.0**	**Secure Service Container for IBM Cloud Private 1.2.0**	**Note**
IBM Multicloud Manager	N	N	You must disable IBM Multicloud Manager components before upgrade or install. See Customizing the cluster with the `config.yaml` file to find `multicluster-hub` settings.
Cloud Foundry	N	N	
Cloud Automation Manager	Y	N	
Installation	Y	Y	Installation supported on master nodes or dedicated boot nodes only.
Management console	Y	Y	Management console runs on master nodes only.
Managed from IBM Z	Y	Y*	* Only supported for master, management, or boot node running on a Linux on Z or LinuxONE LPAR with worker or proxy nodes running in a Secure Service Container partition.

Feature	IBM Cloud Private on IBM Z or LinuxONE 3.2.0	Secure Service Container for IBM Cloud Private 1.2.0	Note
Mixed architecture	Y	Y*	*Only master node on x86 or Linux on IBM Z/ LinuxONE (such as s390x architecture) management server is supported.
Logging	Y*	Y	*Logging is available as a technology preview and requires post-installation configuration changes to enable its functionality. For more information, see IBM Cloud Private logging with Linux on IBM Z or LinuxONE.
Monitoring • Prometheus • Grafana	Y*	Y	*While Prometheus and Grafana run on only master or management nodes, data from worker nodes is collected by using the node exporter.
Security and RBAC	Y	Y	
Vulnerability advisor	Y	N	This feature is not available in the Community Edition. **Note**: VA does not support SLES.
IPsec	Y	Y	
Networking: Calico	Y	Y	
Networking: NSX-T	N	N	
Storage: GlusterFS	Y	Y*	*GlusterFS is a technology preview feature on Secure Service Container for IBM Cloud Private.
Storage: VMware	N	N	
Storage: Minio	Y	N	
Metering	Y	Y	
Helm repo or API	Y	Y	
Nvidia GPU support	N	N	
Cluster healthcheck service	N	N	

Installing Secure Service Container for IBM Cloud Private

The following tasks are required to set up the Secure Service Container for IBM Cloud Private environment to install an IBM Cloud Private cluster.

1. Planning for Secure Service Container for IBM Cloud Private installation
2. Downloading the installation package
3. Creating the Secure Service Container Partition
4. Installing the Hyper Protect hosting appliance
5. Installing the CLI tool on the Linux management server

Note: If you plan to have the IBM Cloud Private worker node and proxy node on different Secure Service Container partitions, you must repeat steps 3 to 4 for each partition.

Planning for Secure Service Container for IBM Cloud Private

Before you start with the Secure Service Container for IBM Cloud Private, you can use a worksheet to get an overall understanding on what information you will need to run the Secure Service Container for IBM Cloud Private ,and where to get such information.

The example values in the checklist are based on the following network topology for the Secure Service Container for IBM Cloud Private. You can use different values in the checklist according to your actual network configuration.

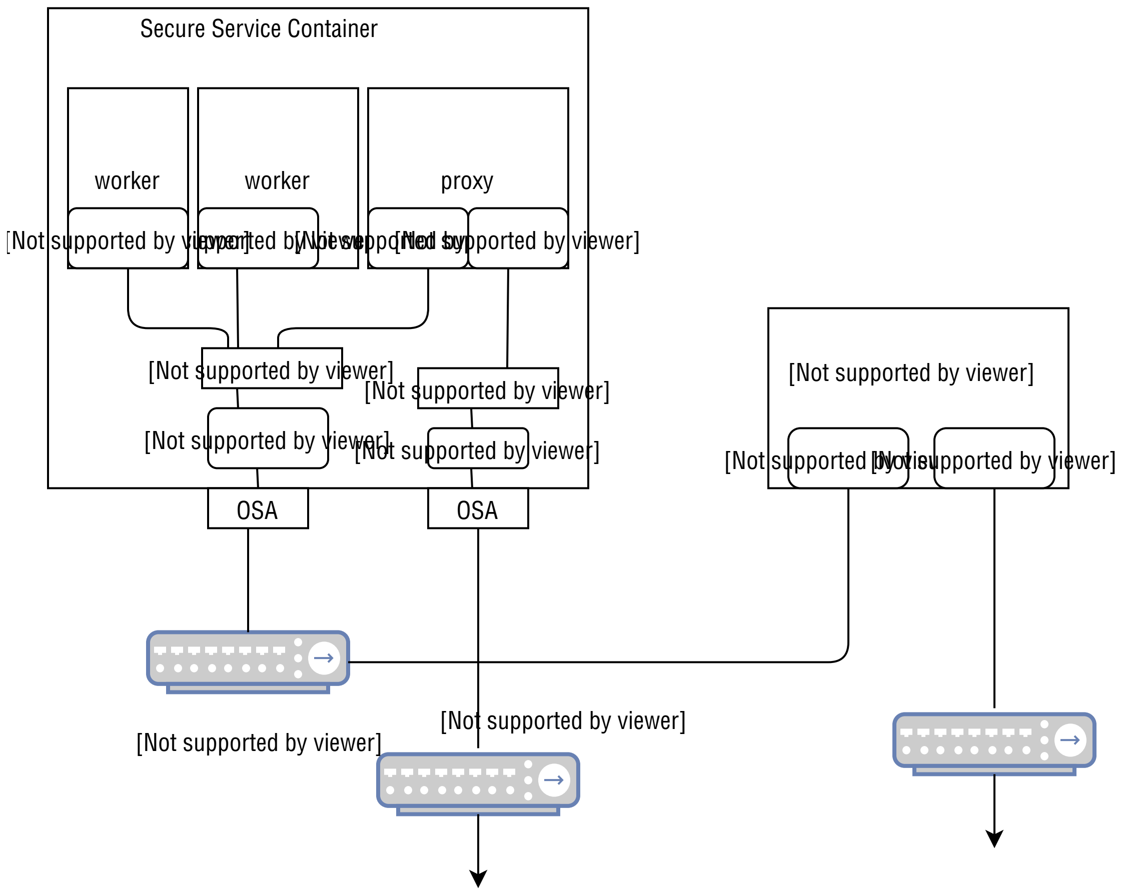


Figure 1. Network topology example for Secure Service Container for IBM Cloud Private

Secure Service Container partition

The following table shows the required information you will need when Configuring Secure Service Container storage.

Table 1. Secure Service Container partition checklist

Resource	The actual value	Example	Where to get
Partition IP address		`10.152.151.105`	System administrator
Master ID		`ssc_master_user`	System administrator
Master password		`ssc_master_password`	System administrator
Storage disks for quotagroups resizing		`3600507630affc427000000000002000` (FCP) or `0.0.78CA` (FICON DASD)	System administrator

Note: If you plan to use multiple Secure Service Container partitions, make sure you have a checklist for each partition.

An IBM Cloud Private cluster

An IBM Cloud Private cluster must have at least three types of cluster node: master, worker, and proxy. The master node is hosted on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, and the worker and proxy nodes are on the Secure Service Container partitions.

The cluster nodes communicates with each other by using internal IP addresses. For more information, see Configuring the network for worker and proxy nodes.

Cluster information

The following table shows the basic information that you need to know or use when Installing IBM Cloud Private cluster.

Table 2. Cluster basic information checklist

Resource	The actual value	Example	Where to get
Cluster name		`DemoCluster`	Cloud administrator
Number of worker node		2	Cloud administrator
Number of proxy node		1	Cloud administrator

Master node

The following table shows the required information for a master node.

Table 3. Master node checklist

Resource	The actual value	Example	Where to get
Architecture	x86 or Linux on Z	x86	Cloud administrator
Host name		`master`	`hostname`
Primary Network interface Controller (NIC)		eth1	`ifconfig -a`
External IP address for NIC		10.152.151.100	`ifconfig -a` (inet addr parameter in the result)

Resource	The actual value	Example	Where to get
Password for the user `root`		`root_user_password`	System administrator
Internal IP address		`192.168.0.251`	Network administrator
NIC for internal network		`eth0`	Network administrator
Subnet mask for internal IP		`192.168.0.0/24`	Network administrator
Gateway for internal IP		`192.168.0.1`	Network administrator

Proxy node

The following table shows the required information for a proxy node.

Table 4. Proxy node checklist

Resource	The actual value	Example	Where to get
Architecture	s390x (Secure Service Container)	s390x (Secure Service Container)	Cloud administrator
Number of nodes		1	Cloud administrator
Number of CPU		3	Cloud administrator
Memory size		1 GB	Cloud administrator
Storage for root file system		60 GB	See `root_storage` for configuring the cluster resources
Storage for cluster runtime		140 GB	See `icp_storage` for configuring the cluster resources
Port range		16000	Cloud administrator
Parent device for external access		encf900	Appliance administrator
External IP address		172.16.0.4	Network administrator
Subnet mask for external IP		`172.16.0.0/24`	Network administrator
Gateway for external IP		`172.16.0.1`	Network administrator
Password for the user `root`		`root_user_password`	System administrator
Parent device for internal access		• encf700 (Ethernet-type connection) • vxlan0f300.1121 (VLAN-type connection)	Appliance administrator
Internal IP address		`192.168.0.254`	Network administrator (See Configuring the network for worker and proxy nodes)

Resource	The actual value	Example	Where to get
Subnet mask for internal IP		`192.168.0.0/24`	Network administrator
Gateway for internal IP		`192.168.0.1`	Network administrator

Worker nodes

The following table shows the required information for two worker nodes on one Secure Service Container partition.

Table 5. Worker nodes checklist

Resource	The actual value	Example	Where to get
Architecture	s390x (Secure Service Container)	s390x (Secure Service Container)	Cloud administrator
Number of nodes		2	Cloud administrator
Number of CPU		4	Cloud administrator
Memory size		4 GB	Cloud administrator
Storage for root file system		60 GB	See `root_storage` for configuring the cluster resources
Storage for cluster runtime		140 GB	See `icp_storage` for configuring the cluster resources
Port range		15000	Cloud administrator
Parent device for internal access		• encf700 (Ethernet-type connection) • vxlan0f300.1121 (VLAN-type connection)	Appliance administrator
Internal IP address		`192.168.0.252`, `192.168.0.253`	Network administrator (See Configuring the network for worker and proxy nodes)
Subnet mask for internal IP		`192.168.0.0/24`	Network administrator
Gateway for internal IP		`192.168.0.1`	Network administrator

GlusterFS nodes

The following table shows the required information for one GlusterFS node on one Secure Service Container partition.

Table 5. Gluster nodes checklist

Resource	The actual value	Example	Where to get
Number of nodes		3	Cloud administrator
Number of CPU		8	Cloud administrator
Memory size		8 GB	Cloud administrator

Resource	The actual value	Example	Where to get
Storage		80 GB	See `size` for Deploying GlusterFS
Port range		17000	Cloud administrator
Parent device for internal access		• encf700 (Ethernet-type connection) • vxlan0f300.1121 (VLAN-type connection)	Appliance administrator
Internal IP address		`192.168.0.245`, `192.168.0.246`, `192.168.0.247`	Network administrator
Subnet mask for internal IP		`192.168.0.0/24`	Network administrator
Gateway for internal IP		`192.168.0.1`	Network administrator

Tasks roadmap for different user roles

To install the cluster by using the Secure Service Container for IBM Cloud Private, you need to work with at least three different user roles. You can click each step on the following clickable diagram to read detailed instructions.

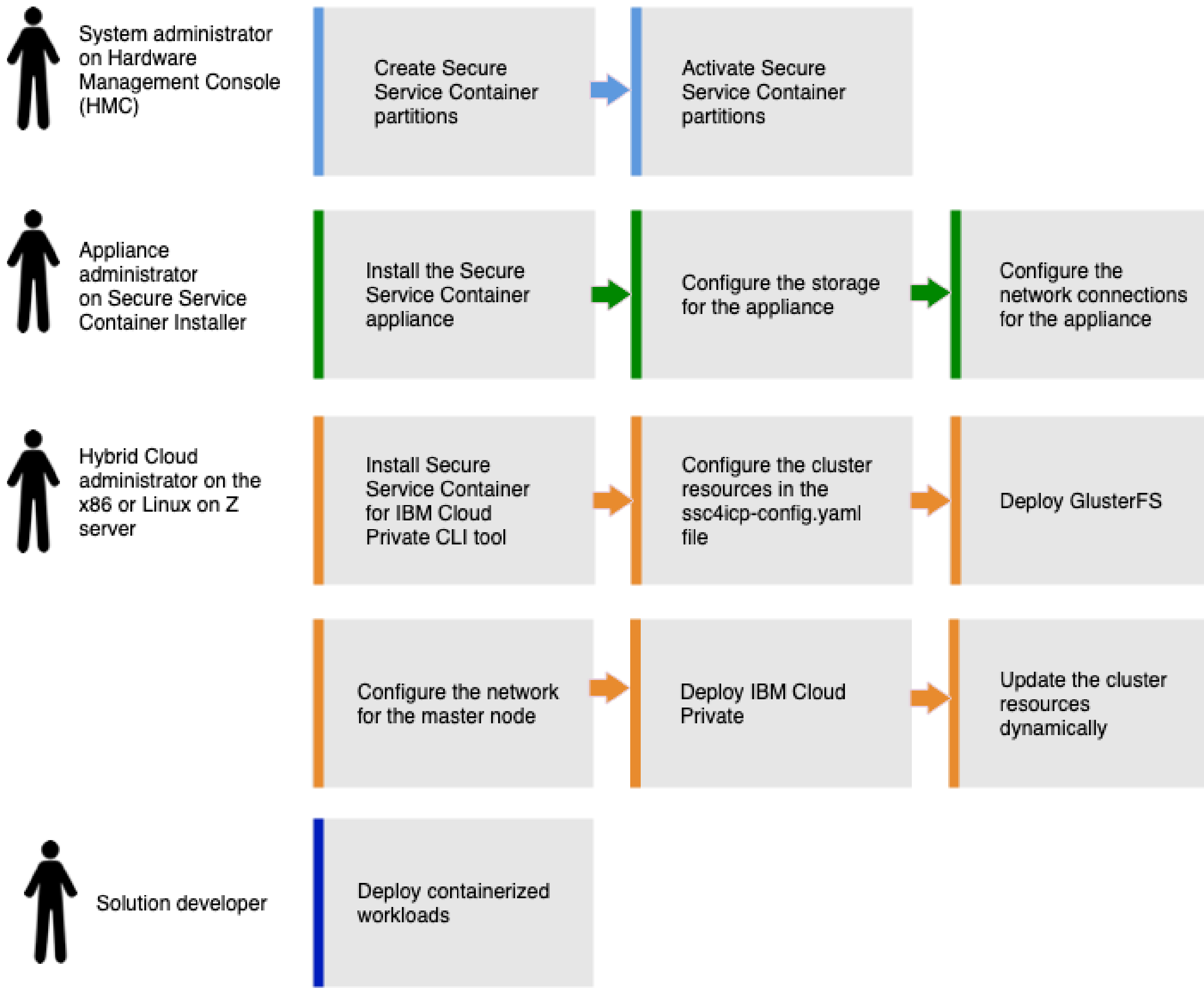

System administrator on Hardware Management Console (HMC)
Create Secure Service Container partitions
Activate Secure Service Container partitions
Appliance administrator on Secure Service Container Installer
Install the Secure Service Container appliance
Configure the storage for the appliance
Configure the network connections for the appliance
Hybrid Cloud administrator on the x86 or Linux on Z server
Install Secure Service Container for IBM Cloud Private CLI tool
Configure the cluster resources in the ssc4icp-config.yaml file
Deploy GlusterFS
Configure the network for the master node
Deploy IBM Cloud Private
Update the cluster resources dynamically
Solution developer
Deploy containerized workloads

Figure 1. Secure Service Container for IBM Cloud Private tasks by user roles

Downloading the installation package

You can get IBM Hyper Protect Virtual Servers software package from the IBM Passport Advantage.

This procedure is intended for users with the role *Cloud administrator*.

Before you begin

- Ensure you have the management server ready with one of the supported architectures as required in the **Hardware requirements for management server** section.
- Ensure that you install the OpenSSL or similar tool on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.

Procedure

On your x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, complete the following steps with root user authority.

1. Log in to IBM Passport Advantage website by using your IBM account ID and password. Contact your sales representative if you do not have one.
2. Go to **My Programs**, and then select the **IBM Hyper Protect Virtual Servers** program.
3. Download IBM Hyper Protect Virtual Servers image `CC37UEN.tar.gz` to your x86 or Linux on IBM Z/ LinuxONE (such as s390x architecture) management server.
4. On the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server with root user authority, create an installation directory to store IBM Hyper Protect Virtual Servers image and configuration files.

   ```
   mkdir /opt/<installation_directory>
   ```

5. Change to the installation directory, and extract the compressed file on the x86 on Linux on Z server.

   ```
   cd /opt/<installation_directory>
   tar -zxvf CC37UEN.tar.gz
   ```

   You will get the following files in the current directory:

   - `CC37UEN.tar.gz`, the offering image tar file.
   - `CC37UEN.sig`, the signature file for the offering image.
   - `CC37UEN.`pub, the public key issued by IBM for the offering image.

6. To verify the integrity of IBM Hyper Protect Virtual Servers image tar file, run the following example command by using the signature file with the `.sig` suffix and the public key issued by IBM with the suffix `.`pub along with the image tar file.

   ```
   openssl dgst -sha256 -verify CC37UEN.pub -signature CC37UEN.sig CC37UEN.tar.gz
   ```

7. Extract the compressed tar file on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.

   ```
   cd /opt/<installation_directory>
   gunzip CC37UEN.tar.gz
   tar -xvf CC37UEN.tar
   ```

Result

After extracting the `CC37UEN.tar` file, you can see the similar layout of files and directories under the `<installation_directory>` directory.

```
|- CC37UEN.tar
|- hpvs-cli-installer.docker-image.tar
|- License
|- readme.txt
|- secure-service-container-for-icp.appliance.1.1.18.img.gz
|- SSC4ICP
|   |-config
|   |   |- ICPIsolatedvm.tar.gz
|   |- readme.txt
|- version
|- VS
    |- grep11-cli
    |   |- config
    |       |-grep11-config.yaml.example
    |       |-hosts.example
    |       |-hpcsKpGrep11_runq.tar.gz
    |- hpvs-cli
    |   |- config
    |       |- hosts.example
    |       |- hpvs-config.yaml.example
    |       |- hpvs-config.yaml.example.VLAN.L2
    |       |- hpvs-config.yaml.example.VLAN.L3
    |       |- HpvsopBaseSSH.tar.gz
    |       |- HpvsopBase.tar.gz
    |- monitoring-cli
    |  |-config
    |       |- CollectdHost.tar.gz
    |       |- hosts.example
```

```
    |       |- monitoring-configuration-delete.yaml.example
    |       |- monitoring-configuration-service.yaml.example
    |       |- monitoring-configuration.yaml.example
    |       |- Monitoring.tar.gz
    |       |- monitoring.yaml.example
    |- readme.txt
    |- regfile-cli
    |   |- config
    |- securebuild-cli
        |- config
           |- hosts.example
           |- hosts.readme
           |- securebuild.yaml.example
           |- securebuild.yaml.example.VLAN.L2
           |- securebuild.yaml.example.VLAN.L3
           |- securebuild.yaml.readme
           |- SecureDockerBuild.tar.gz
```

Note:

- `readme.txt`, which is the general README file for IBM Hyper Protect Virtual Servers .
- `License`, a directory that contains the license files of IBM Hyper Protect Virtual Servers .
- `version`, which states the current version of IBM Hyper Protect Virtual Servers .
- `hpvs-cli-installer.docker-image.tar`, which contains the command line tools required to set up IBM Hyper Protect Virtual Servers environment, or set up the Secure Service Container for IBM Cloud Private. For the details about the Secure Service Container for IBM Cloud Private, see Working with Secure Service Container for IBM Cloud Private.
- `secure-service-container-for-icp.appliance.1.1.18.img.gz`, which is the hosting appliance to be installed on the IBM Z or LinuxONE system. Note that this hosting appliance image is identical with Hyper Protect hosting appliance version v3.7.7.
- `./VS/grep11-cli/config`, a directory that contains configuration example of `hosts` and `grep11-config.yaml` files.
- `./VS/grep11-cli/config/hpcsKpGrep11_runq.tar.gz`, which is the base image of the GREP11 container and must be under the `./VS/grep11-cli/config`.
- `./VS/hpvs-cli/config/HpvsopBase.tar.gz`, which is the base image of a Hyper Protect Virtual Server container without the secure shell (SSH) access.
- `./VS/hpvs-cli/config/HpvsopBaseSSH.tar.gz`, which is the base image of a Hyper Protect Virtual Server container with the secure shell (SSH) access.
- `./VS/hpvs-cli/config`, a directory that contains configuration example of `hosts` and `hpvs-config.yaml` files for Hyper Protect Virtual Server containers.
- `./VS/monitoring-cli/config`, a directory that contains configuration example of `hosts` and `monitoring.yaml` files for the monitoring infrastructure.
- `./VS/monitoring-cli/config/CollectdHost.tar.gz`, which is the base image of collectd-host container of the monitoring infrastructure. The file must be under the `./VS/monitoring-cli/config` directory.
- `./VS/monitoring-cli/config/Monitoring.tar.gz`, which is the base image of monitoring-host container of the monitoring infrastructure. The file must be under the `./VS/monitoring-cli/config` directory.
- `./VS/regfile-cli/config`, which is an empty directory upon initial extract, and can be used to contain configuration files for repository definition files.
- `./VS/securebuild-cli/config/SecureDockerBuild.tar.gz`, which is the docker image of the Secure Build container.
- `./VS/securebuild-cli/config`, a directory that contains configuration example files (with the `.example` extension) and readme files (with the `.readme` extension) of the `hosts` and `securebuild.yaml` files for Secure Build containers.
- `./SSC4ICP/config`, a directory that contains configuration example files for IBM Secure Service Container for IBM Cloud Private.

- `./SSC4ICP/config/ICPIsolatedvm.tar.gz`, which is the isolated VM image for hosting proxy and worker nodes of Secure Service Container for IBM Cloud Private. For more details, see Working with Secure Service Container for IBM Cloud Private.

Next

You can create and start the Secure Service Container partition as instructed in the Creating the Secure Service Container partition topic.

Creating the Secure Service Container partition

Use this procedure to configure a Secure Service Container partition on a host system and afterwards install IBM Hyper Protect Virtual Servers on that partition.

This procedure is intended for users with the role *system administrator*.

Before you begin

- Ensure that the hosting system is one of supported servers as required in the **Hardware requirements for Secure Service Container partition** section.
- Check that you download Secure Service Container User's Guide from the About topic.
- Refer to the checklist that you prepared on this topic Planning for the environment.

Procedure

To create and manage Secure Service Container partitions, you can use specific tasks on the Hardware Management Console (HMC) for a host system running either in standard mode (that is, with Processor Resource/System Manager or PR/SM), or with Dynamic Partition Manager (DPM) enabled. For more information about the host system, see the appropriate overview on the IBM® Redbooks® website at http://www.redbooks.ibm.com/. For example, for the z15, see the IBM z15 Technical Introduction, SG24-8850.

- For a host system (CPC) running in standard mode
  1. Open the **Customize/Delete Activation Profiles** task, and then select **SSC** mode on the **Customize Image Profiles** page.
  2. Configure the processor requirements on the **Processor** page, specify partition security options on the **Security** page, and specify the amount of storage required on the **Storage** page.
  3. Provide or modify any cryptographic controls as appropriate on the **Crypot** page.
  4. On the **SSC** page, ensure the **Secure Service Container installer** option is selected under **Boot selection** if you create the partition for the first time, and then provide values for the default master user ID, password, and IP address of the network adapter for the Secure Service Container partition.
  5. Click **Save** to save the changes and wait for the partition to be created.
  6. Select the image of the Secure Service Container partition, and start the Secure Service Container partition by using the **Activate** task.
- For a host system with DPM enabled
  1. Open the HMC **New Partition** task, and then select **Secure Service Container** as the **Partition Type** from the list.
  2. Provide the values for the master user ID and password as prompted.
  3. Define the number of virtual processors for the partition on the **Processor** page, define the initial and maximum amounts of memory to be assigned to the partition on the **Memory** page. The minimal initial amount of a Secure Service Container partition is 4 GB.
  4. Define all of the network interface cards (NICs) for the partition on the **Network** page. For a Secure Service Container partition, you must also specify at least one NIC for communication with the Secure Service Container web interface.
  5. Attach storagegroups or create host bus adapters (HBAs) for the partition on the **Storage** page.

6. Configure the cryptographic features on the **Cryptos** page as needed.
7. On the **Boot** page, note that option set in the "Boot from" menu is **Secure Service Container**. This boot option cannot be changed unless you first change the partition type.
8. Click **Finish** to save the partition definition, and then wait for DPM creating the partition.
9. Select the image of the Secure Service Container partition, and activate the partition by selecting **Yes** on the **Start** task page.

Note:

- Write down the following values specified in the image profile (standard mode system) or the partition definition (DPM-enabled system) for the Secure Service Container partition when you configure the Secure Service Container. You will need them when configuring the appliance network and creating cluster nodes.
  – Master user ID
    - Master password
    - IP address
- For more detailed information on how to create and start the Secure Service Container partition, see Secure Service Container User's Guide from the About topic.

Next

You can install the hosting appliance onto the Secure Service Container partition by following the instructions on Installing the Hyper Protect hosting appliance.

Installing the Hyper Protect hosting appliance

Use this procedure to install and start the Hyper Protect hosting appliance in a {{site.date.keyword.ssc}} partition on the IBM z or LinuxONE server.

This procedure is intended for users with the role *appliance administrator*.

Note:

- The Hyper Protect hosting appliance is an enhanced version of the IBM Secure Service Container software appliance.
- The Hyper Protect hosting appliance displays with the name **IBM Secure Service Container** on the Secure Service Container user interface.
- The Hyper Protect hosting appliance uses all of the IBM Secure Service Container documentation and techniques to install, administer, and maintain.
- The Hyper Protect hosting appliance version numbering scheme is unique to the Hyper Protect hosting appliance, as opposed to the general Secure Service Container verion numbering scheme.
- Only one appliance can be installed and run in a Secure Service Container partition at any given time; this type of partition does not support running multiple appliances simultaneously. You can define more than one Secure Service Container partitions on the same system, and run instances of the same appliance in each one. In this case, each partition must use separate storage devices.

Before you begin

- Check that you have the appliance image `secure-service-container-for-icp.appliance.<version_number>.img.gz` in the installation directory. For instructions, see Downloading the installation package
- Check that you have the Secure Service Container partition created to install the hosting appliance as instructed in the Creating the Secure Service Container partition topic.
- Check that you download Secure Service Container User's Guide from the About topic.

Procedure

Complete the following tasks through the browser of your choice.

1. Log in to the Secure Service Container installer by using the master user ID and password in your browser. For example, `https://<secure_service_container_partition_ip_address>`.
2. On the main page, click the plus (+) icon to install image files from local disk. The page display changes to the **Install Software Appliance** page.
3. On the **Install Software Appliance** page, select the **Upload image to target disk** option, and then locate the appliance image file on your local disk under the **Local Installation Image** section.
4. Under **Target Disk on Server**, select the device type **FICON DASD** or **FCP**, and then click **Apply** to upload the appliance image to the target disk on the server. **Note:**
   - You can only specify one type of disk (either DASD or FCP) during the appliance installation stage.
   - Target FCP disks must be large enough to fit the uncompressed appliance, with an additional 2 GB for the Secure Service Container installer to use.
5. Click **Reboot** on the confirmation diaglog to have the installer automatically reactivate the partition. The Secure Service Container installer uploads the appliance image to the target disk, and prepares the partition to load the appliance after the next reboot. a. When the reboot process begins, the installer displays the Reboot window. b. If an IP address type other than DHCP is in use for the appliance page, the Secure Service Container installer redirects the browser to the software appliance page.
6. On the appliance page, accept the self-signed certificate for the SSL connection, and log in to the Secure Service Container user interface by using the master user ID and password.

For more detailed instructions, see the following topic after you download Secure Service Container User's Guide from the About topic.

- Chapter 13 - Installing a new software appliance in a Secure Service Container partition

Next

You can configure the storage on the Secure Service Container partition as instructed in the Configuring the storage on the Secure Service Container partition topic.

Installing the IBM Hyper Protect Virtual Servers CLI tool

IBM Hyper Protect Virtual Servers command line interface (CLI) tool provides commands to work with all the components in the offering. This includes management of Hyper Protect Virtual Server containers, securely building applications, generating signed and encrypted registration files, and automation to setup the IBM Cloud Private infrastructure. The IBM Cloud Private infrastructure automation works by creating all the necessary cluster nodes or isolated VMs, and provisions them with appropriate network, storage, CPU, memory resources.

This procedure is intended for users with the role *cloud administrator*.

Note: When setting up an instance of IBM Hyper Protect Virtual Servers CLI tool, you must be cautious of the file system that stores the CLI tool configuration because the file system contains sensitive information such as credentials to a Hosting Appliance or a Secure Build server container. The CLI tool can be run on an x86 or s390 Linux management server.

- It is recommended that the CLI tool is run on individual users machines to simplify the security setup and protect credentials against users without a need to know having access to them.
- If a shared management server setup is required, then you need to ensure that the management server is configured in such a way that only users that have a need to know those credentials have access to the directories where the configuration is stored. By doing so, you can protect against other Linux management server administrators using their admin privileges from exploiting those credentials.
- The `hosts` file under the `/config` directory of each CLI command contains the master password of the Secure Service Container partition in clear text as the `rest_password` parameter.
- Also some other parameters in the `securebuild.yaml` file, such as `github:key`, `manifest_cos:api_key`, `docker:password`, and `docker:base_password`, must be kept private and not shared.

Before you begin

- Check that you have IBM Hyper Protect Virtual Servers installation binary from the IBM Passport Advantage website retrieved on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.
- Check that the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server has the supported Docker environment installed. See Docker versions for more details.

Procedure

Complete the following steps under the `<installation_directory>` directory as the root user.

1. Go to the installation directory where you extract IBM Hyper Protect Virtual Servers archive file, and then install the docker images of the command line tool.

   ```
   cd /opt/<installation-directory>
   docker load -i hpvs-cli-installer.docker-image.tar
   ```

2. Run the `docker images` command to verify the CLI tool is installed correctly. You can see the similar output as the following example on your screen once the CLI tool installation completes.

   ```
   REPOSITORY                         TAG                     IMAGE ID
   CREATED               SIZE
   ibmzcontainers/hpvs-cli-installer    1.2.0.s390x             97c5b4b59bd7        2 weeks
   ago          447MB
   ibmzcontainers/hpvs-cli-installer    1.2.0                   481cf705fc57        2 weeks
   ago          475MB
   ```

   Where

   - `ibmzcontainers/hpvs-cli-installer` with the image tag `1.2.0`, which is the CLI docker image for the x86 Linux management server.
   - `ibmzcontainers/hpvs-cli-installer` with the image tag `1.2.0.s390x`, which is the CLI docker image for the s390x Linux management server.

You can run the commands for different modules in IBM Hyper Protect Virtual Servers . Those modules include:

- `hpvs` - The command line tool to manage Hyper Protect Virtual Server containers.
- `crypto` - The command line tool to check crypto domains on the Hardware Security Modules (HSM).
- `disk` - The command line tool to manage disks on the Secure Service Container partition.
- `grep11` - The command line tool to manage GREP11 containers for IBM Hyper Protect Virtual Servers .
- `image` - The command line tool to load container images into the Secure Service Container partitions.
- `monitoring` - The command line tool to manage monitoring infrastructure for IBM Hyper Protect Virtual Servers .
- `securebuild` - The command line tool to manage the Secure Build containers.
- `regfile` - The command line tool to manage the repository registration files for your applications.
- `repository` - The command line tool to manage the repositories on the Secure Service Container partitions.
- `quotagroup` - The command line to manage the quotagroups on the Secure Service Container partition.
- `snapshot` - The command line tool to manage the snapshots of Hyper Protect Virtual Server containers.

Next

- To deploy your workload with IBM Hyper Protect Virtual Servers, follow the instructions in Deploying your workloads with IBM Hyper Protect Virtual Serverse.
- To configure the environment for Secure Service Container for IBM Cloud Private, follow the instructions in Configuring the Secure Service Container for IBM Cloud Private CLI tool.

Installing IBM Cloud Private cluster

The following tasks are required to set up an IBM Cloud Private environment to deploy and manage dockerized applications.

1. Configuring the Secure Service Container for IBM Cloud Private CLI tool
2. Configuring Secure Service Container storage
3. Configuring the appliance network
4. Configuring the network for worker and proxy nodes
5. Configuring the cluster resources
6. Creating the cluster nodes
7. Configuring the network on the master node
8. Deploying the IBM Cloud Private
9. Deploying containerized applications

Configuring the Secure Service Container for IBM Cloud Private CLI tool

IBM Secure Service Container for IBM Cloud Private is a secure node infrastructure deployed on IBM Secure Service Container of IBM Z or LinuxONE, which has a REST API layer for creating nodes/isolated virtual machines (VMs) that can become the cluster nodes for IBM Cloud Private once installed.

The Secure Service Container for IBM Cloud Private command line interface (CLI) tool automates the infrastructure setup by creating all the necessary cluster nodes/isolated VMs and provision them with appropriate network, storage, CPU, memory resources.

The CLI tool configuration and installation is the prerequisites for an IBM Cloud Private installation on the Secure Service Container for IBM Cloud Private environment.

This procedure is intended for users with role *cloud administrator*.

Before you begin

- Check that you have installed the IBM Hyper Protect Virtual Servers CLI command. For more information, see Installing the IBM Hyper Protect Virtual Servers CLI tool.
- Check that the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server has a Python version 2.7 or later installed, and a soft link points to the correct Python version.

```
#ln -s /usr/bin/python3 /usr/bin/python
```

Procedure

On the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, complete the following steps under the `<installation_directory>/SSC4ICP` directory with root user authority.

1. Install `jq` and `network-manager` utilities. See jq and NetworkManager for more details.

   - For Ubuntu:

   ```
   apt-get install jq
   apt-get -y install network-manager
   /bin/systemctl start network-manager
   ```

   - For RedHat:

   ```
   yum install -y network-manager-applet.x86_64
   wget https://github.com/stedolan/jq/releases/download/jq-1.5/jq-1.5.tar.gz
   gunzip -d jq-1.5.tar.gz
   tar -xvf jq-1.5.tar
   cd jq-1.5/
   ./configure --disable-maintainer-mode
   ```

```
make
make install
```

2. Download the configuration templates to the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server by using the following command. After the command is complete, a `config` directory is created with a `hosts` and `ssc4icp-config.yaml` file. You will need to update both files when Configuring the cluster resources.

   ```
   docker run --network=host --rm -v $(pwd):/data ibmzcontainers/hpvs-cli-installer:1.2.0 cp -r
   config /data
   ```

Note: Ensure that the isolated VM archive file `ICPIsolatedvm.tar.gz` exists in the `/config` directory. If not, copy it from where you extracted the installation archive file. The CLI tool will use the isolated VM archive to create cluster nodes. For more information, see Creating the cluster nodes.

Next

Follow the instructions on Installing IBM Cloud Private cluster to plan the cluster resources and install the IBM Cloud Private cluster.

Configuring Secure Service Container storage

Use this procedure to make resources like storage devices and network connections assigned to the Secure Service Container partition available in the Secure Service Container for IBM Cloud Private. These resources can then be utilized by the containerized applications running on worker nodes inside the Secure Service Container. This procedure is intended for users with role *appliance administrator*.

Before you begin

- Check with the IBM Cloud Private operations administrator the list of requirements of the containerized application to assign sufficient resources (disk space, network adapters) to the Secure Service Container for IBM Cloud Private.
- Check with the IBM Z or LinuxONE system administrator that sufficient resources are assigned to the partition of type Secure Service Container to fit the requirements of the containerized application.
- Check that you download Secure Service Container User's Guide from the About topic.
- Refer to the checklist that you prepared on this topic Planning for Secure Service Container for IBM Cloud Private.

Procedure

1. To match the disk space requirements of the containerized application, add storage disks to the storage pool, which is provided by the Secure Service Container. For instructions, see the following topic after you download Secure Service Container User's Guide from the About topic:

   - Chapter 14, "Using the Secure Service Container user interface", section "Viewing and managing storage resources"

   Note: If you have added the disks to the appliance on the Secure Service Container portal, go to step 3 to configure the quotagroup size by using the REST API.
2. If required by the containerized application, additional network adapter can be configured for the Secure Service Container for IBM Cloud Private. For instructions, see the following topic after you download Secure Service Container User's Guide from the About topic:

   - Chapter 14, "Using the Secure Service Container user interface", section "Viewing and managing network connections"
3. Configure the quotagroup size by using the REST API. See the Secure Service Container for IBM Cloud Private System APIs for a full list of REST API endpoints.

Note:

- For each worker or proxy node running on the Secure Service Container, you need to allocate at least 200 GB for the storage.

- For each GlusterFS node running on the Secure Service Container, you need to allocate at least 80 GB for the storage.

  a. Generate the access token to the Secure Service Container by using the following command.

```
curl --request POST --url https://<appliance_IP>/api/com.ibm.zaci.system/api-tokens \
-H 'accept: application/vnd.ibm.zaci.payload+json' -H 'cache-control: no-cache' \
-H 'content-type: application/vnd.ibm.zaci.payload+json;version=1.0' \
-H 'zaci-api: com.ibm.zaci.system/1.0' --insecure \
--data '{ "kind" : "request", "parameters" : { "user" : "<master_id>", "password" :
"master_id_password" } }'
```

  Where:

  – **appliance_IP** is the Secure Service Container IP address.

  – **master_id** is the `Master user ID` in the image profile (standard mode system) or the partition definition (DPM-enabled system) for the Secure Service Container partition.

  – **master_id_password** is the `Master password` in the same profile or definition for the partition.

  b. For the disks identified in the worker or proxy node checklist in the Planning for Secure Service Container for IBM Cloud Private topic, run the following command to add disks to the storage pool if those disks are not added on the Secure Service Container user interface. Note that you can remove `--insecure` option from the command if you import the self-signed SSL certificate. See SSL Certificate Verification for more information.

```
curl -X POST https://<appliance_IP>/api/com.ibm.zaci.system/storagepools/lv_data_pool/
<subresource> \
-H 'accept: application/vnd.ibm.zaci.payload+json' -H 'cache-control: no-cache' \
-H 'content-type: application/vnd.ibm.zaci.payload+json;version=1.0' \
-H 'zaci-api: com.ibm.zaci.system/1.0' --insecure \
-H 'authorization: Bearer '<TOKEN>'' -d '{ "kind": "request", "parameters": { "addDisks":
[ "<disk_id1>", <"disk_id2"> ] } }'
```

  Where:

  – **appliance_IP** is the Secure Service Container IP address.

  – **subresource** is to identify the resource type to be added to the pool, which can be either `storage-devices` (to represent a DASD) or `fcp-disks` (to represent a SCSI disk/LUN).

  – **TOKEN** is the bearer token that you get in the previous step.

  – **disk_id1** and **disk_id2** are the disk identifiers to be added to the storage pool. You can get the values from your Secure Service Container partition administrator.

  c. Check the status of the disks added to the storage pool by using the following command. If the disks are successfully added to the pool, the status of each disk is shown as in `used` state.

```
curl -X GET https://<appliance_IP>/api/com.ibm.zaci.system/storagepools/lv_data_pool \
-H 'accept: application/vnd.ibm.zaci.payload+json' -H 'cache-control: no-cache' \
-H 'content-type: application/vnd.ibm.zaci.payload+json;version=1.0' \
-H 'zaci-api: com.ibm.zaci.system/1.0' --insecure \
-H 'authorization: Bearer '<TOKEN>''
```

  Where:

  – **appliance_IP** is the Secure Service Container IP address.

  – **TOKEN** is the bearer token that you get in the previous step.

  d. Increase the quotagroup size by using the following command.

```
curl -X PUT https://<appliance_IP>/api/com.ibm.zaas/quotagroups/appliance_data \
-H 'Content-Type: application/vnd.ibm.zaci.payload+json;version=1.0' \
-H 'zACI-API: com.ibm.zaci.system/1.0' --insecure \
-H 'authorization: Bearer '<TOKEN>'' -d '{ "size": "<SIZE>", "size_unit": "GB"}'
```

Where: * **appliance_IP** is the Secure Service Container IP address. * **TOKEN** is the bearer token that you get in the previous step. * **SIZE** is the number of disk size in GB that you want to allocate to the storage. The minimal size is 50 GB.

Configuring the appliance network

You can configure the network devices for the Secure Service Container for IBM Cloud Private appliance by using the Secure Service Container user interface. The cluster nodes on the Secure Service Container partitions communicate through the Ethernet-type or VLAN-type connections over the network devices bound to Open Systems Adapter-Express (OSA-Express) devices.

If you have both worker and proxy nodes on one Secure Service Container partition, you must configure two network devices with one for internal communication among the cluster nodes, and another for external access through the proxy node. You can configure one network device to each of the OSA-Express device on the Secure Service Container partitions, or multiple network devices on one OSA-Express device.

This procedure is intended for users with role *appliance administrator*.

Before you begin

- Check that you have the connection information to each Secure Service Container partition. For more information, see Creating Secure Service Container partitions.
- Check that you install the Secure Service Container for IBM Cloud Private appliance by following the instructions on Installing the Secure Service Container for IBM Cloud Private appliance.
- Refer to the checklist that you prepared on this topic Planning for Secure Service Container for IBM Cloud Private.

Procedure

Complete the following steps to configure the network devices.

1. Connect to the Secure Service Container partition through the browser of your choice. For example, `https://<secure_service_container_partition_ip_address>`.
2. On the Login page, enter the master use ID and password values that you supplied in the image profile (standard mode system) or the partition definition (DPM-enabled system), and click **Login**.
3. In the navigation pane, click the **Network** icon to display the network connections page.
4. Select one of the network devices to get the channel path identifier (CHPID) of the OSA-Express device. For example, `encf900` is the network device name, and AA is the CHPID.
5. Configure another network device on the Secure Service Container partition.

- For an ethernet-type connection:
  1. Click the plus (+) icon to add a new connection, and then select **Ethernet** as the connection type.
  2. Select a new network device from the drop-down list. Ensure that the CHPID in the Device Details section is different from the one in step 4. For example, the network device name is `encf700`, and the CHPID is AB.
  3. Use the default value for the **Port** field, and set the connection state to **Active**.
  4. Use **Automatic** for both IPV4 and IPV6 addresses fields.
- For a VLAN-type connection, ensure that your OSA device is tagged with an VLAN ID (for example, `1121`) and the OSA device is connected with the trunk port of the switch.
  1. Click the plus (+) icon to add a new connection, and then select **VLAN** as the connection type.
  2. Select a parent device (also known as a tagged OSA device) from the drop-down list. If the parent device is not available, click the plus (+) icon to create a parent device. For example, the parent device name is `encf300`.
  3. Enter the VLAN ID by which the OSA device is tagged. For example, `1121`.
  4. Use the auto-generated connection name. For example, `vxlan0f300.1121`.

5. If the DHCP is not configured in your network, select the **Manual** checkbox on the **IPv4** tab and assign an appropriate IP address according to your network.
6. Set the connection state on the **General** tab to **Active**.
7. Click the **ADD** button to save the changes.

Note:

- Repeat all the steps on each Secure Service Container partition that will be used to host the cluster nodes.
- Decide the network device name that will be used for internal communications among the cluster nodes, or by the proxy node for external access. You will use those values for the `parent` parameter in the `ssc4icp-config.yaml` file. For more information, see Configuring the cluster resources.

For more information, see the following topic after you download Secure Service Container User's Guide from the About topic:

- Chapter 14, "Using the Secure Service Container user interface", section "Viewing and managing network connections"

Next

You can now follow the instructions in the Configuring the network for worker and proxy nodes topic to plan how the cluster nodes will communicate among each other and how the external requests can access the services inside the cluster.

Configuring the network for worker and proxy nodes

Use this procedure to configure the network for the worker and proxy nodes. You will use the network information in the `ssc4icp-config.yaml` file when creating the cluster resources. The cluster nodes can communicate among each other in a layer 2 or layer 3 network architecture. You also need to configure external IP addresses for the proxy nodes so that external requests to the services inside the IBM Cloud Private cluster can access the cluster.

This procedure is intended for users with role *cloud administrator*.

Before your begin

- Refer to the checklist that you prepared on this topic Planning for Secure Service Container for IBM Cloud Private.
- Ensure that the subnet range `172.31.0.1/16` is not used in your network because the IP addresses within this range are reserved for the appliance.

Procedure

Complete the following steps according to your network topology.

1. Configure a Layer 2 network for the cluster nodes as identified in the worker or proxy node checklist in the Planning for Secure Service Container for IBM Cloud Private topic, which means all the cluster nodes are available in the same subnet.

   a. Decide the private subnet that you want to use. For example, `192.168.0.0/24` is the subnet value and `192.168.0.1` is the gateway value. Then the available IP addresses in this subnet are from `192.168.0.2` to `192.168.0.254`. Use the subnet and gateway values for the `subnet` and `gateway` parameters under the `internal_network` section in the `ssc4icp-config.yaml` file.

   b. Choose IP addresses from the available values for your worker and proxy nodes. You can use any IP addresses within this private network because this private network is defined by yourself. For example, `192.168.0.252`, `192.168.0.253`, and so on.

   c. Choose one IP address from the range as the IP address for the master node. For example, `192.168.0.251`. Use the subnet value and this IP address for the `internal_ips` and `subnet` parameters under the `masterconfig` section in the `ssc4icp-config.yaml` file.

d. Use those IP values for the `internal_ips` parameter in the `ssc4icp-config.yaml` file when Configuring the cluster resources. See the following code snippet as an example.

```
...
cluster:
  masterconfig:
    internal_ips: ['192.168.0.251']
    subnet: "192.168.0.0/24"
LPAR
-ipaddress: '10.152.151.105'
   containers:
      -template: "template1"
      count: 2
      internal_ips: ['192.168.0.252','192.168.0.253']
-ipaddress: '10.152.151.105'
   containers:
      -template: "template2"
      count: 1
      internal_ips: ['192.168.0.254']
...
template1:
   type: "WORKER"
   internal_network:
     subnet: "192.168.0.0/24"
     gateway: "192.168.0.1
     parent: "encf700"
...
template2:
   type: "PROXY"
   internal_network:
     subnet: "192.168.0.0/24"
     gateway: "192.168.0.1"
     parent: "encf700"
...
```

2. If you want to configure the network for the cluster nodes by using your existing Layer 3 network, which means the cluster nodes are in two different subnets. For example, the worker and proxy nodes are available in the `10.152.151.0/24` subnet and the master node is in the `10.162.161.0/24` subnet.

   a. Check with your network administrator to get the subnet and gateway information. For example, `10.152.151.0/24` is the subnet value and `10.152.151.1` is the gateway value. Then the available IP addresses in this subnet are from `10.152.151.2` to `10.152.151.254`. Use the subnet and gateway values for the `subnet` and `gateway` parameters under the `internal_network` section in the `ssc4icp-config.yaml` file.

   b. Choose IP addresses from the available values for your worker and proxy nodes. Note that the IP address must not be used by any host in the subnet. For example, `10.152.151.110`, `10.152.151.111`, and so on.

   c. Use the master node IP address, for example, `10.162.161.107`, for the `internal_ips` and `subnet` parameters under the `masterconfig` section in the `ssc4icp-config.yaml` file.

   d. Use those IP values for the `internal_ips` parameter in the `ssc4icp-config.yaml` file when Configuring the cluster resources. See the following code snippet as an example.

```
    ...
    cluster:
      masterconfig:
        internal_ips: ['10.162.161.107']
        subnet: "10.162.161.0/24"
    LPAR
     -ipaddress: '10.152.151.105'
        containers:
           -template: "template1"
           count: 2
           internal_ips: ['10.152.151.110','10.152.151.111']
    -ipaddress: '10.152.151.105'
        containers:
           -template: "template2"
           count: 1
           internal_ips: ['10.152.151.112']
    template1:
       type: "WORKER"
```

```
         internal_network:
            subnet: "10.152.151.0/24"
            gateway: "10.152.151.1"
            parent: "encf700"
      template2:
         type: "PROXY"
         internal_network:
           subnet: "10.152.151.0/24"
           gateway: "10.152.151.1"
           parent: "encf700"
     ...
```

3. Configure the external network for the proxy node.

a. Check with your network administrator to get the subnet and gateway information of the external OSA device. For example, `172.16.0.0/24` is the subnet value and `172.16.0.1` is the gateway value. Then the available IP addresses in this subnet are from `172.16.0.2` to `172.16.0.254`. Use the subnet and gateway values for the `subnet` and `gateway` parameters under the `proxy_external_network` section in the `ssc4icp-config.yaml` file.

b. Choose an appropriate IP address within the range for the proxy node on the Secure Service Container partition. The proxy node transmits external request to the services created inside your cluster. If you have multiple proxy nodes on different Secure Service Container partitions, assign one IP address for each of proxy node. For example, `172.16.0.4`, `172.16.0.5`, and so on.

c. Use the external IP values for the `proxy_external_ips` parameter in the `ssc4icp-config.yaml` file. See the following code snippet as an example.

```
...
LPARS:
 -ipaddress: '10.152.151.105'
     containers:
       -template: "template2"
        ...
        proxy_external_ips: ['172.16.0.4']
template2:
     type: "PROXY"
     ...
     proxy_external_network:
        subnet: "172.16.0.0/24"
        gateway: "172.16.0.1"
        parent: "encf900"
 ...
```

For the complete `ssc4icp-config.yaml` example files, see Worker and proxy nodes in a random private Layer 2 network on two Secure Service Container partitions and Worker and proxy nodes in an existing Layer 3 network on two Secure Service Container partitions.

Next

You can now follow the instructions in the Configuring the cluster resources topic to set up the resource specifications for cluster nodes.

Configuring the cluster resources

You can configure the password authentication for the cluster nodes, and define the resource specifications for each node on the Secure Service Container partitions. The resource specification includes the CPU numbers, memory size, port range, and network settings.

This procedure is intended for users with role *cloud administrator*.

Before you begin

- Check that you complete the steps in the Installing the Secure Service Container for IBM Cloud Private CLI tool.
- Check that you create at least 200 GB for each cluster node on each Secure Service Container partition by following the steps in Configuring Secure Service Container storage.

- Check that you have the following configuration data for each Secure Service Container partition (also known as LPAR), which are received from the IBM Z or LinuxONE system administrator.
  - IP address
  - master user ID and password
- Refer to the checklist that you prepared on this topic Planning for Secure Service Container for IBM Cloud Private.

Procedure

On the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, complete the following steps under the `<installation_directory>/SSC4ICP` directory with root user authority.

1. Go to the `config` directory that you extracted the Secure Service Container for IBM Cloud Private archive file, and update the `hosts` file to configure the password authentication for cluster nodes. In the following `hosts` example file, the Master user ID in the login setting of the Secure Service Container partition with IP address `10.152.151.105` was set to `master` and the Master password to `somesecurepassword`.

   ```
   [master]
     10.152.151.100 ansible_user="root" ansible_ssh_pass="root_user_password"
   ansible_ssh_common_args="-oPubkeyAuthentication=no" ansible_become="true"
   [worker]
     10.152.151.105 rest_user="ssc_master_user" rest_pass="ssc_master_password"
   ```

   Where

   - The `[master]` section contains x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server details with IP address, username, password for IBM Cloud Private master installation.
   - The `[worker]` section contains Secure Service Container partition IP address, username, and password for the zAppliance REST API. If you use different Secure Service Container partitions to host the worker or proxy nodes, you must list each of partitions under the `[worker]` section. The zAppliance REST API username `rest_user` and password `rest_pass` values are the user ID and password used to log in to the Secure Service Container partition and UI, and initially set in the login setting of the Secure Service Container partition as:
     - Master user ID
     - Master password
2. Update the `ssc4icp-config.yaml` file to configure the nodes on the Secure Service Container partition with required CPU, memory, port range, and network specifications by using a specified template.

   The following `ssc4icp-config.yaml` example file shows that one cluster `DemoCluster` contains 2 worker nodes with resources specified in the `template1` on the Secure Service Container partition with an IP address `10.152.151.105`, and 1 proxy nodes with resources specified in the `template2` on the same Secure Service Container partition. See ssc4icp-config.yaml examples for more examples of the cluster configuration.

   ```
   cluster:
     name: "DemoCluster"
     datapool: "exists"
     masterconfig:
       internal_ips: ['192.168.0.251']
       subnet: "192.168.0.0/24"
   LPARS:
     -ipaddress: '10.152.151.105'
        containers:
         -template: "template2"
          count: 1
          internal_ips: ['192.168.0.254']
          proxy_external_ips: ['172.16.0.4']
     -ipaddress: '10.152.151.105'
        containers:
         -template: "template1"
   ```

```
        count: 2
        internal_ips: ['192.168.0.252','192.168.0.253']
 template1:
    name: "worker"
    type: "WORKER"
    cpu: "4"
    memory: "4098"
    port_range: '15000'
    root_storage: "60G"
    icp_storage: "140G"
    internal_network:
      subnet: "192.168.0.0/24"
      gateway: "192.168.0.1"
      parent: "encf700"
 template2:
    name: "proxy"
    type: "PROXY"
    cpu: "3"
    memory: "1024"
    port_range: '16000'
    root_storage: "60G"
    icp_storage: "140G"
    internal_network:
      subnet: "192.168.0.0/24"
      gateway: "192.168.0.1"
      parent: "encf700"
    proxy_external_network:
      subnet: "172.16.0.0/24"
      gateway: "172.16.0.1"
      parent: "encf900"
```

Note:

- `datapool` defines that the quotagroup still exists after the containers are deleted. If the value is not set for the `datapool` parameter, the CLI deletes the quotagroup after the uninstallation.
- `masterconfig` defines the network configurations for the master node.
- `internal_ips` defines an array of IP addresses for each worker and proxy node. See Configuring the network for proxy and worker nodes for information on how to plan the IP addresses for your worker and proxy nodes.
- `proxy_external_ips` defines an external IP addresses for the proxy node. The external workloads can use the value of `proxy_external_network_ip` to access the proxy node.
- `count` defines the number of nodes that will be created on the partition. Note that the value of `count` is an integer and can not be enclosed by using the quotation marks.
- `cpu` defines the number of CPU cores to be assigned for the node.
- `memory` defines the memory size (in MB) to be assigned for the node.
- `type` defines the type of node to be created by the CLI tool. The value must be `WORKER`, `PROXY`, or `STORAGE`.
- name defines the name of the proxy or worker node. The maximum length of a node name is 20 characters.
- `internal_network` defines the subnet, gateway, and parent network interface settings of the worker or proxy node.
- `proxy_external_network` defines the external subnet, gateway, and parent network interface settings of the proxy node.
- `parent` defines the parent network device name that data traffic will physically go through on the node. See Configuring Secure Service Container network devices for the instructions on how to get those values.
- `port_range` defines the starting port number on each Secure Service Container partition to be assigned to each node.
- `root_storage` defines the size of storage (G for GB or M for MB) allocated to the root file system. It must be set to at least what is required by IBM Cloud Private. IBM Cloud Private requires storage under the root file system that is used to store temporary files during the installation. For example,

IBM Cloud Private 3.2.0 requires 50 GB under the root file system. Therefore, the `root_storage` parameter must be set to 50 GB plus an adequate buffer for the operating system on the node itself.

- `icp_storage` defines the size of the storage (G for GB or M for MB) allocated to the IBM Cloud Private node runtime. It must be set to at least the sum of the directory sizes used by a node at runtime, as specified by the IBM Cloud Private system requirements. For example, based on IBM Cloud Private 3.2.0 system requirements, a node requires at runtime:
  - at least 100 GB under /var/lib/docker
  - at least 10 GB under /var/lib/kubelet Therefore, the `icp_storage` parameter must be set to 110 GB plus an adequate buffer to run custom kubernetes applications. The default value is 140 GB.

Next

You can now create worker and proxy nodes on the Secure Service Container partitions by following the steps in the Creating the cluster nodes topics.

If you want to enable GlusterFS on your cluster, proceed to Deploying GlusterFS topic.

ssc4icp-config.yaml examples

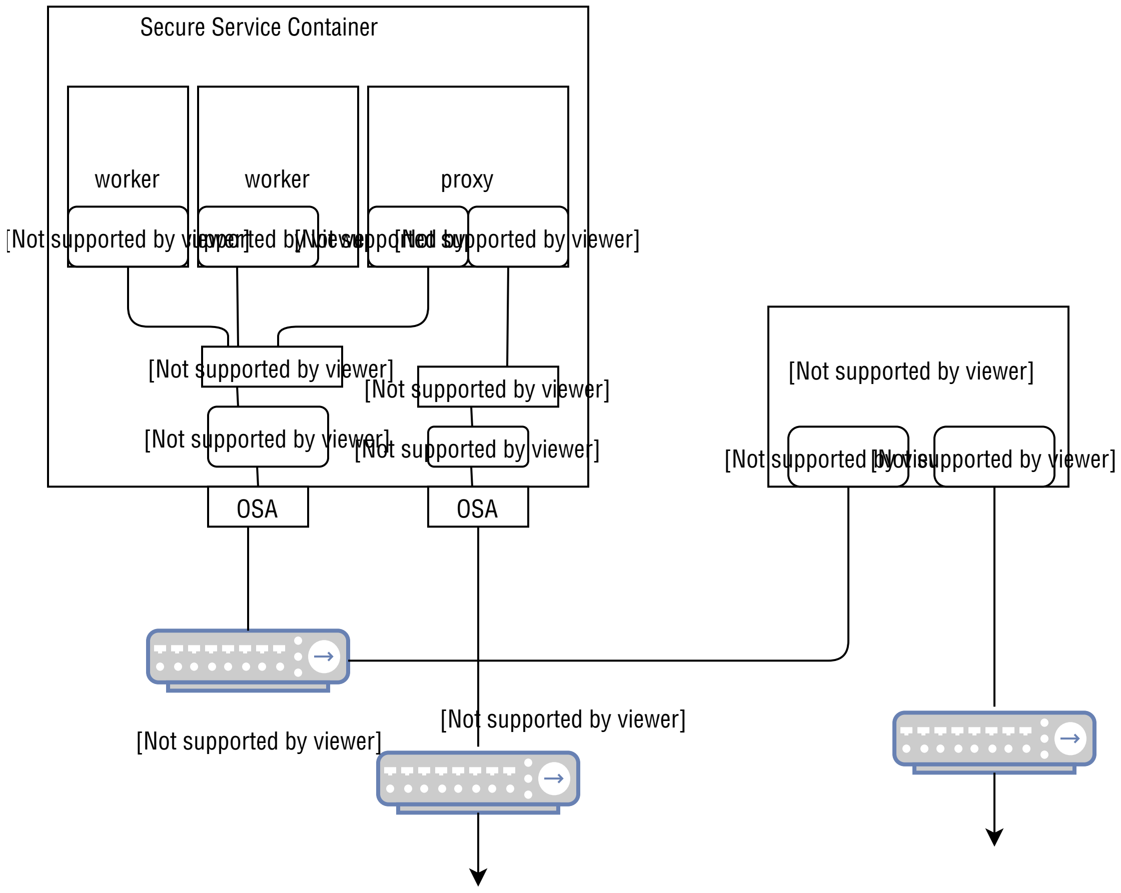
You can use the following `ssc4icp-config.yaml` example files to set up the cluster nodes and resources. The `ssc4icp-config.yaml` file is generated in the `config` folder on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server after you complete the tasks in Installing the Secure Service Container for IBM Cloud Private CLI tool. The `hosts` file in the same `config` directory must be updated based on the actual network configuration as well.

- Worker and proxy nodes on one Secure Service Container partition
- Worker and proxy nodes in a random Layer 2 private network on two Secure Service Container partitions
- Worker and proxy nodes in an existing Layer 3 network on two Secure Service Container partitions

Note: The IP addresses and network parent device names in the example files must be updated based on your actual network configurations.

Worker and proxy nodes on one Secure Service Container partition

The following table shows the network configuration that you want to create for the cluster `DemoCluster`, which hosts all the worker and proxy nodes on one Secure Service Container partition.



Secure Service Container
worker
worker
proxy
[Not supported by viewer]
[Not supported by viewer]
[Not supported by viewer]
[Not supported by viewer]
[Not supported by viewer]
[Not supported by viewer]
[Not supported by viewer]
[Not supported by viewer]
OSA
OSA
[Not supported by viewer]
[Not supported by viewer]
[Not supported by viewer]
[Not supported by viewer]
[Not supported by viewer]

Figure 1. The cluster nodes on one Secure Service Container partition

Table 1. Cluster network configuration for one partition

Node	Assigned IP	External IP	Remote port	Internal IP	Parent device for internal	Parent device for external	Resource template
x86 or Linux on IBM Z/ LinuxONE (such as s390x architecture) management server (master)	10.152.151.100	N/A	4789	192.168.0.251	eth0	N/A	N/A
Secure Service Container partition 1 (worker)	10.152.151.105	N/A	15001	192.168.0.252	encf700	N/A	template1

Node	Assigned IP	External IP	Remote port	Internal IP	Parent device for internal	Parent device for external	Resource template
Secure Service Container partition 1 (worker)	10.152.151.105	N/A	15002	192.168.0.253	encf700	N/A	template1
Secure Service Container partition 1 (proxy)	10.152.151.105	172.16.0.4	16001	192.168.0.254	encf700	encf900	template2

The configuration can be translated into the following `hosts` and `ssc4icp-config.yaml` files, which can be used when Configuring the cluster resources. Note that you must replace the credentials for the x86 server and Secure Service Container partition in the `hosts` file based on the actual values.

- The `hosts` example

```
master]
  10.152.151.100 ansible_user="root" ansible_ssh_pass="pass4chain" ansible_ssh_common_args="-
oPubkeyAuthentication=no" ansible_become="true"
[worker]
  10.152.151.105 rest_user="master" rest_pass="somesecurepassword"
```

- The `ssc4icp-config.yaml` example that shows all the worker nodes use identical resource setting as in `template1`.

```
cluster:
    name: "DemoCluster"
    datapool: "exists"
    masterconfig:
      internal_ips: ['192.168.0.251']
      subnet: "192.168.0.0/24"
LPARS:
  -ipaddress: '10.152.151.105'
     containers:
      -template: "template2"
       count: 1
       internal_ips: ['192.168.0.254']
       proxy_external_ips: ['172.16.0.4']
  -ipaddress: '10.152.151.105'
     containers:
      -template: "template1"
       count: 2
       internal_ips: ['192.168.0.252','192.168.0.253']
template1:
     name: "worker"
     type: "WORKER"
     cpu: "4"
     memory: "4098"
     port_range: '15000'
     root_storage: "60G"
     icp_storage: "140G"
     internal_network:
       subnet: "192.168.0.0/24"
       gateway: "192.168.0.1"
       parent: "encf700"
template2:
     name: "proxy"
     type: "PROXY"
     cpu: "3"
     memory: "1024"
     port_range: '16000'
     root_storage: "60G"
     icp_storage: "140G"
     internal_network:
       subnet: "192.168.0.0/24"
       gateway: "192.168.0.1"
       parent: "encf700"
     proxy_external_network:
       subnet: "172.16.0.0/24"
       gateway: "172.16.0.1"
       parent: "encf900"
```

Worker and proxy nodes in a random Layer 2 private network on two Secure Service Container partitions

The following `ssc4icp-config.yaml` example file shows one cluster `DemoCluster` with different types of cluster nodes on two Secure Service Container partitions. The network for the cluster nodes is configured in a random private Layer 2 network.

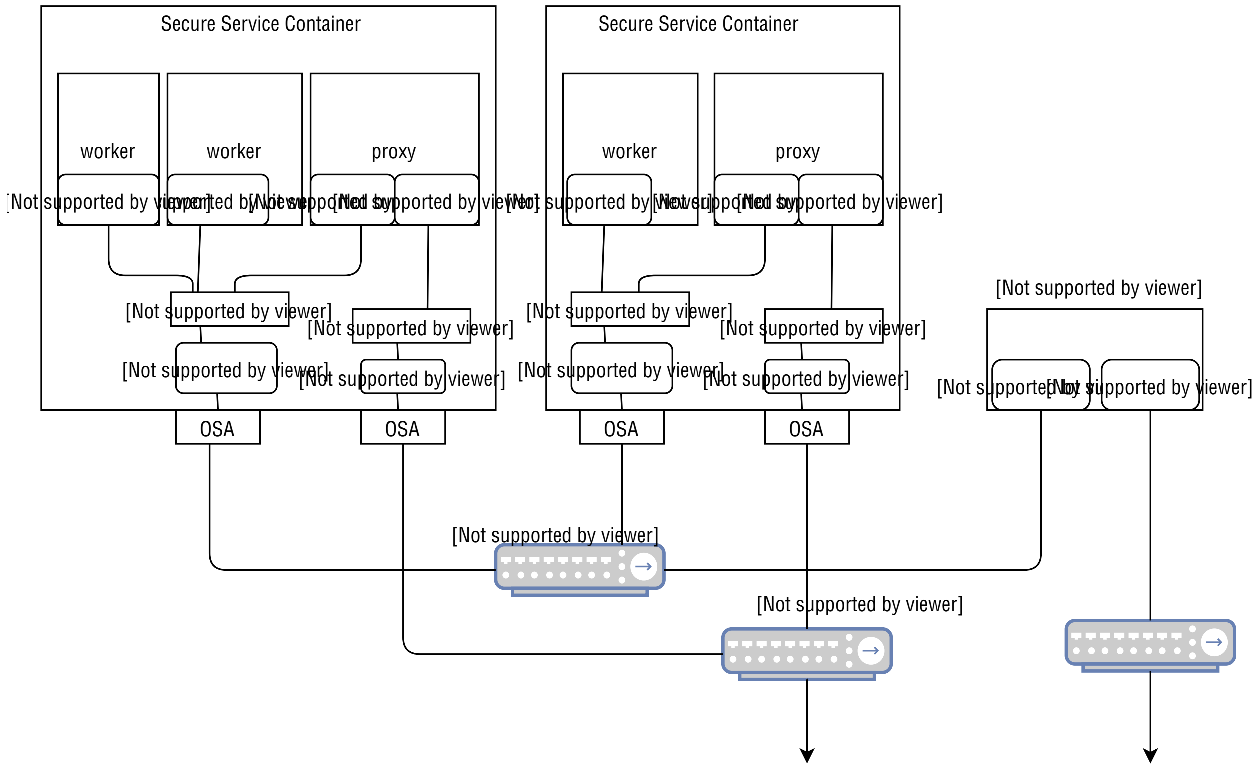


Figure 2. The cluster nodes in a Layer 2 network

Table 2. Cluster network configuration for two partitions on a private random Layer 2 network

Node	Assigned IP	External IP	Remote port	Internal IP	Parent device for internal	Parent device for external	Resource template
x86 or Linux on IBM Z/ LinuxONE (such as s390x architecture) management server (master)	10.152.151.107	N/A	4789	192.168.0.251	enov1	N/A	N/A

Node	Assigned IP	External IP	Remote port	Internal IP	Parent device for internal	Parent device for external	Resource template
Secure Service Container partition 1 (worker)	10.152.151.105	N/A	15001	192.168.0.252	enc8192	N/A	template1
Secure Service Container partition 1 (worker)	10.152.151.105	N/A	15002	192.168.0.253	enc8192	N/A	template1
Secure Service Container partition 1 (proxy)	10.152.151.105	172.16.0.4	16001	192.168.0.254	enc8192	enc8193	template2
Secure Service Container partition 2 (worker)	10.152.151.106	N/A	17001	192.168.0.250	enc8194	N/A	template3
Secure Service Container partition 2 (proxy)	10.152.151.106	172.16.0.5	18001	192.168.0.249	enc8194	enc8195	template4

The configuration can be translated into the following `hosts` and `ssc4icp-config.yaml` files, which can be used when Configuring the cluster resources. Note that you must replace the credentials for the x86 server and Secure Service Container partition in the `hosts` file based on the actual values.

- The `hosts` example

```
[master]
  10.152.151.107 ansible_user="root" ansible_ssh_pass="pass4chain" ansible_ssh_common_args="-
oPubkeyAuthentication=no" ansible_become="true"
[worker]
  10.152.151.105 rest_user="master" rest_pass="somesecurepassword"
  10.152.151.106 rest_user="root" rest_pass="anothersecurepassword"
```

- The `ssc4icp-config.yaml` example that shows the cluster nodes are configured in a random private network.

```
cluster:
  name: "DemoCluster"
  datapool: "exists"
  masterconfig:
    internal_ips: ['192.168.0.251']
    subnet: "192.168.0.0/24"
LPARS:
-ipaddress: '10.152.151.105'
   containers:
    -template: "template1"
     count: 2
     internal_ips: ['192.168.0.252','192.168.0.253']
-ipaddress: '10.152.151.105'
   containers:
    -template: "template2"
     count: 1
     internal_ips: ['192.168.0.254']
     proxy_external_ips: ['172.16.0.4']
-ipaddress: '10.152.151.106'
   containers:
    -template: "template3"
     count: 1
     internal_ips: ['192.168.0.250']
-ipaddress: '10.152.151.106'
   containers:
    -template: "template4"
     count: 1
     internal_ips: ['192.168.0.249']
     proxy_external_ips: ['172.16.0.5']
template1:
   name: "worker"
   type: "WORKER"
   cpu: "4"
   memory: "4098"
   port_range: '15000'
   root_storage: "60G"
   icp_storage: "140G"
   internal_network:
     subnet: "192.168.0.0/24"
     gateway: "192.168.0.1"
     parent: "enc8192"
template2:
   name: "proxy"
   type: "PROXY"
   cpu: "3"
   memory: "1024"
   port_range: '16000'
   root_storage: `60G`
   icp_storage: `140G`
   internal_network:
     subnet: "192.168.0.0/24"
     gateway: "192.168.0.1"
     parent: "enc8192"
   proxy_external_network:
     subnet: "172.16.0.0/24"
     gateway: "172.16.0.1"
     parent: "enc8193"
template3:
   name: "worker"
   type: "WORKER"
   cpu: "4"
   memory: "4098"
   port_range: '17000'
   root_storage: "60G"
   icp_storage: "140G"
   internal_network:
     subnet: "192.168.0.0/24"
     gateway: "192.168.0.1"
     parent: "enc8194"
template4:
   name: "proxy"
   type: "PROXY"
   cpu: "3"
   memory: "1024"
   port_range: '18000'
   root_storage: "60G"
   icp_storage: "140G"
   internal_network:
     subnet: "192.168.0.0/24"
     gateway: "192.168.0.1"
     parent: "enc8194"
   proxy_external_network:
     subnet: "172.16.0.0/24"
     gateway: "172.16.0.1"
     parent: "enc8195"
```

Worker and proxy nodes in an existing Layer 3 network on two Secure Service Container partitions

You can also configure the network for cluster nodes by using your existing Layer 3 network.

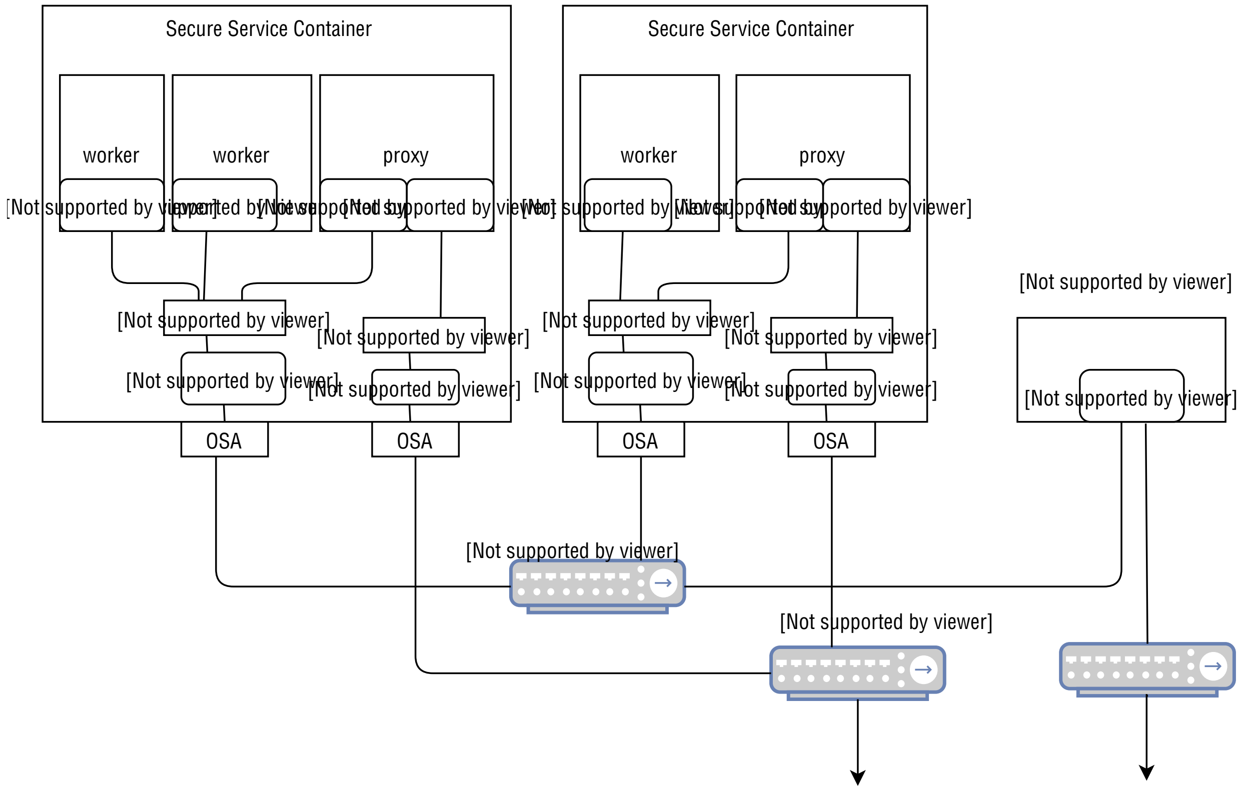


Figure 3. The cluster nodes in a Layer 3 network

Table 3. Cluster network configuration for two partitions on an existing Layer 3 network

Node	Assigned IP	External IP	Remote port	Internal IP	Parent device for internal	Parent device for external	Resource template
x86 or Linux on IBM Z/ LinuxONE (such as s390x architecture) management server (master)	10.152.151.107	N/A	4789	10.162.161.107	enov1	N/A	N/A
Secure Service Container partition 1 (worker)	10.152.151.105	N/A	15001	10.152.151.110	enc8192	N/A	template1
Secure Service Container partition 1 (worker)	10.152.151.105	N/A	15002	10.152.151.111	enc8192	N/A	template1

Node	Assigned IP	External IP	Remote port	Internal IP	Parent device for internal	Parent device for external	Resource template
Secure Service Container partition 1 (proxy)	10.152.151.105	172.16.0.4	16001	10.152.151.112	enc8192	enc8193	template2
Secure Service Container partition 2 (worker)	10.152.151.106	N/A	17001	10.152.151.113	enc8194	N/A	template3
Secure Service Container partition 2 (proxy)	10.152.151.106	172.16.0.5	18001	10.152.151.114	enc8194	enc8196	template4

- The `hosts` example

```
[master]
  10.152.151.107 ansible_user="root" ansible_ssh_pass="pass4chain" ansible_ssh_common_args="-
oPubkeyAuthentication=no" ansible_become="true"
[worker]
  10.152.151.105 rest_user="master" rest_pass="somesecurepassword"
  10.152.151.106 rest_user="root" rest_pass="anothersecurepassword"
```

- The `ssc4icp-config.yaml` example that shows the cluster nodes are configured in an existing network.

```
cluster:
  name: "DemoCluster"
  datapool: "exists"
  masterconfig:
    internal_ips: ['10.162.161.107']
    subnet: "10.162.161.0/24"
LPARS:
-ipaddress: '10.152.151.105'
   containers:
    -template: "template1"
     count: 2
     internal_ips: ['10.152.151.110','10.152.151.111']
-ipaddress: '10.152.151.105'
   containers:
    -template: "template2"
     count: 1
     internal_ips: ['10.152.151.112']
     proxy_external_ips: ['172.16.0.4']
-ipaddress: '10.152.151.106'
   containers:
    -template: "template3"
     count: 1
     internal_ips: ['10.152.151.113']
-ipaddress: '10.152.151.106'
   containers:
    -template: "template4"
     count: 1
     internal_ips: ['10.152.151.114']
     proxy_external_ips: ['172.16.0.5']
template1:
   name: "worker"
   type: "WORKER"
   cpu: "4"
   memory: "4098"
   port_range: '15000'
   root_storage: "60G"
   icp_storage: "140G"
```

```
    internal_network:
      subnet: "10.152.151.0/24"
      gateway: "10.152.151.1"
      parent: "enc8192"
 template2:
    name: "proxy"
    type: "PROXY"
    cpu: "3"
    memory: "1024"
    port_range: '16000'
    root_storage: "60G"
    icp_storage: "140G"
    internal_network:
      subnet: "10.152.151.0/24"
      gateway: "10.152.151.1"
      parent: "enc8192"
    proxy_external_network:
      subnet: "172.16.0.0/24"
      gateway: "172.16.0.1"
      parent: "enc8193"
 template3:
    name: "worker"
    type: "WORKER"
    cpu: "4"
    memory: "4098"
    port_range: '17000'
    root_storage: "60G"
    icp_storage: "140G"
    internal_network:
      subnet: "10.152.151.0/24"
      gateway: "10.152.151.1"
      parent: "enc8194"
 template4:
    name: "proxy"
    type: "PROXY"
    cpu: "3"
    memory: "1024"
    port_range: '18000'
    root_storage: "60G"
    icp_storage: "140G"
    internal_network:
      subnet: "10.152.151.0/24"
      gateway: "10.152.151.1"
      parent: "enc8194"
    proxy_external_network:
      subnet: "172.16.0.0/24"
      gateway: "172.16.0.1"
      parent: "enc8195"
```

Deploying GlusterFS

You can use GlusterFS storage either by deploying GlusterFS on your IBM Cloud Private cluster nodes, or by integrating a GlusterFS storage cluster that is deployed outside the IBM Cloud Private environment.

The following information shows how to prepare dedicated nodes for a GlusterFS cluster in the Secure Service Container for IBM Cloud Private environment. Then you can configure GlusterFS during the IBM Cloud Private installation. For more information about how the GlusterFS works, see GlusterFS.

You can also configure GlusterFS on worker nodes by following the instructions on Configuring GlusterFS during IBM Cloud Private installation, or configure GlusterFS by using helm charts or as an add-on service as in Configuring GlusterFS after IBM Cloud Private installation.

Note:

- GlusterFS as persistent volume is a technology preview feature on the Secure Service Container for IBM Cloud Private environment.
- You cannot dynamically add or remove glusterFS nodes from the cluster after the GlusterFS cluster is deployed.
- You cannot enable GlusterFS storage on a Secure Service Container for IBM Cloud Private environment that is upgraded from a previous version.

This procedure is intended for users with role *cloud administrator*.

Before you begin

- Check with the system administrator that at least 80 GB disk space for each GlusterFS node is assigned on the Secure Service Container partition.
- Refer to the GlusterFS nodes checklist that you prepared on this topic Planning for Secure Service Container for IBM Cloud Private.

Procedure

On the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, update the `config/ssc4icp-config.yaml` file to configure the GlusterFS nodes on the Secure Service Container partition with required CPU, memory, port range, and network specifications by using a specified template.

The following `ssc4icp-config.yaml` example file shows `template5` and `template6` are defined for GlusterFS nodes in the `DemoCluster` with required resources. The `template5` depicts the configuration of two storage quotagroups and `template6` depicts the configuration of single storage quotagroup.

```
cluster:
    name: "DemoCluster"
    datapool: "exists"
    masterconfig:
      internal_ips: ['192.168.0.251']
      subnet: "192.168.0.0/24"
LPARS:
  -ipaddress: '10.152.151.105'
     containers:
      -template: "template2"
       count: 1
       internal_ips: ['192.168.0.254']
       proxy_external_ips: ['172.16.0.4']
  -ipaddress: '10.152.151.105'
     containers:
      -template: "template1"
       count: 2
       internal_ips: ['192.168.0.252','192.168.0.253']
  -ipaddress: '10.152.151.105'
     containers:
      -template: "template5"
       count: 1
       internal_ips: ['192.168.0.245']
  -ipaddress: '10.152.151.105'
     containers:
      -template: "template6"
       count: 2
       internal_ips: ['192.168.0.246','192.168.0.247']
template1:
     name: "worker"
     type: "WORKER"
     # CPU defined by number of threads
     cpu: "4"
     # Memory in MB
     memory: "4098"
     port_range: '15000'
     root_storage: "60G"
     icp_storage: "140G"
     internal_network:
       subnet: "192.168.0.0/24"
       gateway: "192.168.0.1"
       parent: "encf700"
template2:
     name: "proxy"
     type: "PROXY"
     # CPU defined by number of threads
     cpu: "3"
     # Memory in MB
     memory: "1024"
     port_range: '16000'
     root_storage: "60G"
     icp_storage: "140G"
     internal_network:
       subnet: "192.168.0.0/24"
       gateway: "192.168.0.1"
       parent: "encf700"
     proxy_external_network:
       subnet: "172.16.0.0/24"
       gateway: "172.16.0.1"
       parent: "encf900"
template5:
    name: "storage"
    type: "STORAGE"
    # Configure storage provisioner
    provisioner:
      - name: "glusterfs"
        size: "40G"
      - name: "glusterfs"
        size: "40G"
    # CPU defined by number of threads
    cpu: "4"
    # Memory in MB
    memory: "4098"
    port_range: '17000'
    root_storage: "60G"
    icp_storage: "140G"
    internal_network:
      subnet: "192.168.0.0/24"
      gateway: "192.168.0.1"
      parent: "encf700"
template6:
    name: "storage"
    type: "STORAGE"
    # Configure storage provisioner
    provisioner:
      - name: "glusterfs"
        size: "80G"
    # CPU defined by number of threads
    cpu: "4"
    # Memory in MB
    memory: "4098"
    port_range: '18000'
    root_storage: "60G"
    icp_storage: "140G"
    internal_network:
      subnet: "192.168.0.0/24"
      gateway: "192.168.0.1"
      parent: "encf700"
```

Note:

- `internal_ips` defines the IP addresses of GlusterFS node.
- `provisioner` indicates the template is used for the glusterfs node. The value of name under the `provisioner` section must be `glusterfs`.
- `size` defines the disk size of glusterfs quotagroup.
- For the rest of configuration in the `ssc4icp-config.yaml` file, see Configuring the cluster resources.

Next

You can create the GlusterFS nodes as instructed in Creating the cluster nodes.

Creating the cluster nodes

You can use the Secure Service Container for IBM Cloud Private command line interface (CLI) tool to create all the necessary cluster nodes and provision them with appropriate network, storage, CPU, memory resources.

The CLI tool configuration and installation is the prerequisites for IBM Cloud Private installation on the Secure Service Container for IBM Cloud Private environment.

This procedure is intended for users with role *cloud administrator*.

Before you begin

- Check that you copy the isolated VM archive `ICPIsolatedvm.tar.gz` into the `/config` directory.
- If you have a firewall installed on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, check that the `iptables` on the server is configured to allow network traffic from and to docker containers. When the command line tool is running inside a docker container, it has to communicate with the remote hosting appliance on Secure Service Container partitions. The firewall

rules for docker user must contain configurations for DOCKER-USER. You can use the following commands to configure the firewall rules for docker containers on the server.

```
iptables -I FORWARD -j DOCKER-USER
iptables -A DOCKER-USER -j ACCEPT
```

- Refer to the checklist that you prepared on this topic Planning for Secure Service Container for IBM Cloud Private.

Procedure

On the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, complete the following steps under the `<installation_directory>/SSC4ICP/config` directory with root user authority..

1. Run the command line tool to create the cluster nodes. Note that the command must be run in the parent directory of the `config` directory. For example, `/opt/<installation-directory>`.

   ```
   docker run --network=host --rm -it -v $(pwd)/config:/ssc4icp-cli-installer/config
   ibmzcontainers/hpvs-cli-installer:1.2.0 install
   ```

2. After the installation completes, the following directories and files are created under the `config` directory for future reference and use.

   - `Logs` - A directory that contains logs for all operations being performed.
   - `Cluster-status.yaml` - A file that is used to capture the current status of installation.
   - `DemoCluster` - A directory with your cluster name that contains the following files for the cluster:
     - `cluster-configuration.yaml` - A file indicates that the cluster configurations in the `ssc4icp-config.yaml` file are applied successfully.
     - `quotagroup-symlink.yaml` - A file contains details of storage containers, quotagroups and device names if you specify GlusterFS configurations in the `ssc4icp-config.yaml` file. For more information, see Deploying GlusterFS.
     - `ipsec.conf` - A file that contains the network topology of the cluster.
     - `ipsec.secret` - A file that contains a randomly generated Pre-Shared-Key (PSK) that will be used as an authorization token to the IPSec network.

- An ssh key pair `ssh_key` and `ssh_key.pub` files that will provide SSH access for the IBM Cloud Private installer to all the cluster nodes. In order for the IBM Cloud Private installer to access your master or boot node over SSH and also to use the generated SSH key, you must follow the instructions in the **Before you begin** section of Deploying IBM Cloud Private.

The following `cluster-configuration.yaml` example file is generated based on the cluster configuration specified in the `ssc4icp-config.yaml` file.

```
LPARS:
- containers:
  - cpu: '4'
    icp_storage: 140000M
    internal_network:
      gateway: 192.168.0.1
      ip: 192.168.0.252
      parent: encf900
      subnet: 192.168.0.0/24
    memory: '4098'
    name: worker-15001
    port: '15001'
    root_storage: 60000M
  - cpu: '4'
    icp_storage: 140000M
    internal_network:
      gateway: 192.168.0.1
      ip: 192.168.0.253
      parent: encf900
      subnet: 192.168.0.0/24
    memory: '4098'
    name: worker-15002
```

```
    port: '15002'
    root_storage: 60000M
  ipaddress: 10.152.151.105
- containers:
  - cpu: '3'
    icp_storage: 140000M
    internal_network:
      gateway: 192.168.0.1
      ip: 192.168.0.254
      parent: encf900
      subnet: 192.168.0.0/24
    memory: '1024'
    name: proxy-16001
    port: '16001'
    proxy_external_network:
      gateway: 172.16.0.1
      ip: 172.16.0.4
      parent: encf700
      subnet: 172.16.0.0/24
    root_storage: 60000M
  ipaddress: 10.152.151.105
cluster:
  ipsecsecrets: ApcQn3Gmuo4sbwLRZfLxo3hD9MEP21u4HAvOQyoQ2dEYVifWEdE98dIF9Panc2/
gJP6nSXQu3NHgASbb/VlT3w==
  masterconfig:
    internal_ips:
    - 192.168.0.251
    subnet: 192.168.0.0/24
  name: DemoCluster
  repoid: ICPIsolatedvm
  sshpubkey: ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQDSBAz0Pf+vPWiZCfLZ/
NKIrbvFy9+4iav0ihQJ89zIRrwIasQHKPepRPzXYH0h/3g8iAIKymZLuBl2bSn6/
tGNN1stl5nsIdZ5Vr8yKd9a7YAHpBYgzkgq9qcuZHIP5PRJwlrcfwIiCLVdfp73Z2ZCfdzjbfMmd1pb2egv78XlTJLyizxAN
1jV7PyVkiJdjFkxKXIqbWfuYMoMYrwSsBB0gjn66KiQlptrbem9hvYkeMX7d
+zOLjd46C3F7+0gbSzabE0IScfkZdXbjmN9ldIa+70H1/ruGdIuoMN3+1m7Wjj0Xh4sPyXVkmaqRHvJ/OC/
cHkxRweNsztM1GbMkXQXXRhxykYlWNqo/E+U2hPScCDMsf+WU9di4pjYc/JDfVHXFmN/vtk+WFkQqJMwy/
hVR5lOjnSrRA1RTU97uRzOsCgzqXpV9vnF9Xr+20RR4Ml/tP6pEHOScEsjA5ztn/PYROEKM/
3zaV3O2Px6bWP24SWWvOARqjRipHda6/3GzMylp1bL4DMyvKSJ
+m5WlwlIDcrYZ0z2xM2zbzmnjgbDiYWYvu9AgaHSQO8Vdep8wcM0ZO6zXqDT6awPEFBNKkcuxYrv1K5Vf48w0O8saceQSX0V
hNuBk4Kf4PWAy30TxJC4KaCq1yf7zE1545ImCjxIKjY2PPieDgNC0rNCyejfVw==
```

Note:

- Those generated directories and files must not be deleted because the uninstallation procedure needs to validate those files when resetting the environment.
- The private key `ssh_key` must be protected because it provides SSH access to all of the cluster nodes. Only the *cloud administrator* who will install the IBM Cloud Private can have the access privilege to the `ssh_key` file.
- If you run the command line tool again with a same cluster name but different configurations, you must delete the `config/<ClusterName>` directory first.
- The `ipsec.secrets` file contains the IPSec key generated by the command line tool, and will be used when creating the network configuration for each node. You can use LUKS (Linux Unified Key Setup) hardware encryption to protect the key from ambiguous access other than the root user on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.

The following `quotagroup-symlink.yaml` example file is generated based on the configuration of `template5` and `template6` in the `ssc4icp-config.yaml` file. In the example, two quotagroups `storage_17001_glusterfs1_qg` and `storage_17001_glusterfs1_qg` are attached to the GlusterFS node `storage-17001`, `storage_18001_glusterfs1_qg` attached to node `storage-18001`, and `storage_18002_glusterfs1_qg` attached to node `storage-18002`.

```
container = storage-17001, quotagroup = storage_17001_glusterfs1_qg,  symbolic_link = /dev/
disk/by-runq-id/storage_17001_glusterfs1_qg
container = storage-17001, quotagroup = storage_17001_glusterfs2_qg,  symbolic_link = /dev/
disk/by-runq-id/storage_17001_glusterfs2_qg
container = storage-18001, quotagroup = storage_18001_glusterfs1_qg,  symbolic_link = /dev/
disk/by-runq-id/storage_18001_glusterfs1_qg
container = storage-18002, quotagroup = storage_18002_glusterfs1_qg,  symbolic_link = /dev/
disk/by-runq-id/storage_18002_glusterfs1_qg
```

Result

Verify the cluster status by using the `docker ps` command, the nodes for both IBM Cloud Private cluster and GlusterFS are listed as the following.

```
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
7750d5343991        77b4f35dbc74        "entry.sh init"     2 days ago          Up 2
days                                storage-18002
8fd222737e87        77b4f35dbc74        "entry.sh init"     2 days ago          Up 2
days                                storage-18001
bbdf0695c317        77b4f35dbc74        "entry.sh init"     2 days ago          Up 2
days                                storage-17001
121fe6ba774d        77b4f35dbc74        "entry.sh init"     2 days ago          Up 2
days                                proxy-16001
5b157b60cf0c        77b4f35dbc74        "entry.sh init"     2 days ago          Up 2
days                                worker-15001
6e1579fcef84        77b4f35dbc74        "entry.sh init"     2 days ago          Up 2
days                                worker-15002
```

Next

Follow the instructions in Configure the network on the master nodes to ensure that cluster nodes are connected in the cluster.

Configuring the network on the master node

You can configure the network on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server to ensure that the master node is connected with other cluster nodes on the IBM Z or LinuxONE system.

This procedure is intended for users with role *cloud administrator*.

Before you begin

- Check that the `cluster-configuration.yaml` file contains the network information for the master node. For example,

```
...
masterconfig:
  internal_ips:
  - 192.168.0.251
  subnet: 192.168.0.0/24
...
```

- Refer to the checklist that you prepared on this topic Planning for Secure Service Container for IBM Cloud Private.

Procedure

On the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, complete the following steps under the `<installation_directory>/SSC4ICP/config` directory with root user authority..

1. Configure the master node to persist the network configuration. Note that if you want to configure multiple aliases to one network interface controller (NIC) on the master node, see IP-Aliasing.
   - For Ubuntu 16.04:
     – If the master node is connected to an ethernet-type connection, use the following procedure.

     a. Add the following interface and network information into the `/etc/network/interfaces` file.

```
# Static IP
auto eth0
iface eth0 inet static
  address 192.168.0.251
  netmask 255.255.255.0
  gateway 192.168.0.1
  dns-nameservers 8.8.8.8 8.8.4.4
```

```
# Static route
up route add -net 192.168.10.0/24 gw 192.168.10.1 dev enp0s3
```

b. Restart the networking service to apply the changes.

```
  sudo /etc/init.d/networking restart
```

c. If the IP address is not updated after running the restart command, reboot the master node.

– If the master node is connected with the trunk port of the switch, use the following procedure.

a. Add the following interface and network information into the `/etc/network/interfaces` file.

```
# Static IP
auto eth0
iface ens224.1121 inet static
  address 192.168.0.251
  netmask 255.255.255.0
  gateway 192.168.0.1
  dns-nameservers 8.8.8.8 8.8.4.4
# Static route
up route add -net 192.168.10.0/24 gw 192.168.10.1 dev ens224
```

b. Restart the networking service to apply the changes.

```
  sudo /etc/init.d/networking restart
```

c. If the IP address is not updated after running the restart command, reboot the master node.

- For Ubuntu 18.04:

  – If the master node is connected to an ethernet-type connection, use the following procedure.

  a. Create a `Mycluster.yaml` file under the `/etc/netplan` directory, and add the following interface and network information.

```
network:
   version: 2
   ethernets:
      ens224:
         addresses: [192.168.0.251/24]
         gateway4: 192.168.0.1
         nameservers:
             search: [ibm.com]
             addresses: [8.8.8.8, 1.1.1.1]
        #For the L3 network
        routes:
            - to: 192.168.20.0/24
              via: 192.168.10.1
```

b. Apply the changes by using the network management tool.

```
  sudo netplan apply
```

c. If the IP address is not updated after running the `netplan` command, reboot the master node.

- If the master node is connected with the trunk port of the switch, , use the following procedure.

  a. Create a `Mycluster.yaml` file under the `/etc/netplan` directory, and add the following interface and network information.

```
network:
   version: 2
   ethernets:
      ens224:
         addresses: [192.168.0.251/24]
         gateway4: 192.168.0.1
         nameservers:
             search: [ibm.com]
             addresses: [8.8.8.8, 1.1.1.1]
        #For the L3 network
        routes:
            - to: 192.168.20.0/24
```

```
             via: 192.168.10.1
    vlans:
       vlan.1121:
            id: 1121
            link: ens224
            addresses: [192.168.0.10/24]
      vlan.1122:
            id: 1122
            link: ens224
            addresses: [192.168.0.20/24]
```

b. Apply the changes by using the network management tool.

```
sudo netplan apply
```

c. If the IP address is not updated after running the `netplan` command, reboot the master node.

- For Redhat:
  - If the master node is connected to an ethernet-type connection, add the IP address of the master node as the following.

    a. create a file `ifcfg-eth0` in the `/etc/sysconfig/network-scripts/` directory, where `eth0` is device name.

    ```
    vi /etc/sysconfig/network-scripts/ifcfg-eth0
    ```

    b. Add following content into the `ifcfg-eth0` file.

    ```
    TYPE="Ethernet"
    BOOTPROTO="none"
    DEVICE="eth0"
    ONBOOT="yes"
    IPADDR="192.168.0.251"
    PREFIX="24"
    ```

  c. Restart network service on the master node by using the `systemctl restart network` command.

  - If the master node is connected with the trunk port of the switch, create the VLAN interface that is connected to the trunk port of the switch.

    a. Create a file `ifcfg-ens224.1121` under the `/etc/sysconfig/network-scripts/` directory, where ens224 is device name and vlan ID is 1121.

    ```
    vi /etc/sysconfig/network-scripts/ifcfg-ens224.1121
    ```

    b. Add following content into the `ifcfg-ens224.1121` file.

    ```
    DEVICE=ens224.1121
    BOOTPROTO=none
    ONBOOT=yes
    IPADDR=192.168.0.251
    PREFIX=24
    VLAN=yes
    ```

    c. Restart network service on the master node by using the `systemctl restart network` command.

- In case of layer 3 (where more than one subnets are available), you need to add the routing rule for other subnets.

  a. Create a file `route-eth0` under the `/etc/sysconfig/network-scripts/` directory, where `eth0` is device name.

  ```
  vi /etc/sysconfig/network-scripts/route-eth0
  ```

b. Add the following content into the `iroute-eth0` file.

```
GATEWAY0=10.162.161.0
NETMASK0=255.255.255.0
ADDRESS0=10.152.151.0
```

c. Restart network service on the master node by using the `systemctl restart network` command.

- For SUSE Linux:

  – If the master node is connected to an ethernet-type connection, add the IP address of the master node as the following.

    a. Disable the default network management service `NetworkManager` by using the following commands:

    ```
    systemctl stop NetworkManager
    systemctl disable NetworkManager
    ```

    b. create a file `ifcfg-eth0` in the `/etc/sysconfig/network` directory, where `eth0` is device name.

    ```
    vi /etc/sysconfig/network/ifcfg-eth0
    ```

    c. Add following content into the `ifcfg-eth0` file.

    ```
    BOOTPROTO='static'
    IPADDR='192.168.0.251'
    NETMASK='255.255.255.0'
    STARTMODE='auto'
    ONBOOT='yes'
    ```

    d. Enable and start `Wicked` network management service by using the following commands.

    ```
    systemctl enable wicked
    systemctl start wicked
    ```

- If the master node is connected with the trunk port of the switch, create the VLAN interface that is connected to the trunk port of the switch.

  a. Create a file `ifcfg-vlan1121` under the `/etc/sysconfig/network/` directory, where vlan ID is 1121.

  ```
  vi /etc/sysconfig/network/ifcfg-vlan1121
  ```

  b. Add following content into the `ifcfg-vlan1121` file.

  ```
  BOOTPROTO='static'
  IPADDR='192.168.0.251'
  NETMASK='255.255.255.0'
  STARTMODE='auto'
  ONBOOT='yes'
  VLAN='yes'
  ETHERDEVICE='eth0'
  ```

  c. Create a file `ifroute-eth0` under the `/etc/sysconfig/network/` directory with the following routing information.

  ```
  DESTINATION GATEWAY NETMASK INTERFACE
  192.168.20.0/24 0.0.0.0 - eth0
  default 192.168.0.1
  ```

  d. Restart network service on the master node by using the `systemctl start wicked` command.

2. Configure IPSec to ensure that the data traffic within the network is encrypted. IPSec can operate in two different modes: transport or tunnel. The transport mode is sufficient for encryption of the

provided IP traffic. To configure IPSec, you must ensure that the `strongswan` daemon is installed. See strongSwan for more details.

a. Install the `strongswan` daemon on your x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server. The version of `strongswan` must be 5.6.2 or later.

- For Ubuntu 16.04, you have to replace the bundled strongswan binary with version 5.6.2 or later.

```
sudo apt-get remove strongswan
sudo apt-get purge strongswan
sudo apt-get autoremove
apt-get -y install build-essential libunbound-dev libldns-dev libgmp3-dev
wget http://download.strongswan.org/strongswan-5.6.2.tar.bz2
tar xjvf strongswan-5.6.2.tar.bz2
cd strongswan-5.6.2/
./configure --prefix=/usr --sysconfdir=/etc
make
make install
ipsec version
ipsec start
```

- For Ubuntu 18.04:

```
apt-get install strongswan
```

- For Redhat on x86:

```
yum install http://ftp.nluug.nl/pub/os/Linux/distr/fedora-epel/7/x86_64/Packages/e/epel-
release-7-11.noarch.rpm
yum install strongswan
```

- For Redhat on IBM Z, you have to download the strongswan source package and build the binary on the IBM Z system. For information on how to build the strongswan package, see strongSwan Installation Documentation.

```
yum install gmp-devel
wget http://download.strongswan.org/strongswan-5.6.2.tar.bz2
tar xjvf strongswan-5.6.2.tar.bz2
cd strongswan-5.6.2/
./configure --prefix=/usr --sysconfdir=/etc
make
make install
ipsec version
ipsec start
```

Note: You might experience network connectivity problems, because of a known issue Using /32 groups in ipsec causing leaks, when the master node runs on IBM Z with Redhat 7.5 or 7.6 and strongswan v5.6.2. To workaround the problem, create a cron job to run `ipsec restart` command every 30 minutes on the master node.

- For SUSE Linux: ``` wget http://download.strongswan.org/strongswan-5.6.2.tar.bz2 tar xjvf strongswan-5.6.2.tar.bz2 cd strongswan-5.6.2/ ./configure --prefix=/usr --sysconfdir=/etc --disable-gmp --enable-openssl make make install ipsec version ipsec start

```
```

b. Copy the following two files into the `/etc` (on Ubuntu and SUSE Linux) or `/etc/strongswan` (on RedHat) directory. Those two files are generated in the `config/<ClusterName>` directory after the Secure Service Container for IBM Cloud Private CLI tool is installed. * `config/<ClusterName>/ipsec.conf`, this file contains the network topology of the cluster. * `config/<ClusterName>/ipsec.secret`, this file contains a randomly generated Pre-Shared-Key (PSK) that will be used as an authorization token to the IPSec network.

c. Start the `strongswan` daemon to apply the changes. `service strongswan restart` **Note:** You might have to run the command again for some Linux distributions if you reboot the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.

1. Test the internal and external connection to each cluster node on the IBM Z or LinuxONE system by using the `ping` command. For example,

   ```
   ping 192.168.0.252
   ping 192.168.0.253
   ping 192.168.0.254
   ping 172.16.0.4
   ```

Next

Follow the instructions in the Installing IBM Cloud Private topic to deploy the IBM Cloud Private on your cluster nodes.

Deploying IBM Cloud Private

Use this procedure to deploy the IBM Cloud Private cluster to the cluster nodes on x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, and Secure Service Container partitions. Note that the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server is the master node and boot node for the IBM Cloud Private cluster.

This procedure is intended for users with role *cloud administrator*.

Before you begin

- Check that you have completed the steps in Configure the network on the master node topics.
- Create an installation directory to store the extracted IBM Cloud Private installation files. For example, `mkdir /opt/icp320/cluster`.
- Add the SSH key public key `ssh_key.pub` file to the authorized_keys for the root user of your master node by running the following command, where `/opt/<installation-directory>/config` is the location of the generated SSH public key for your cluster. Note that you need to enter the root password for your master node when prompted. If the command completes successfully, you will see a response message such as `Number of key(s) added: 1`.

```
ssh-copy-id -i /opt/<installation-directory>/config/ssh_key.pub root@localhost
```

- Copy the SSH private key `ssh_key` file for the cluster to the IBM Cloud Private installation directory by running this command, where `/opt/<installation-directory>/config/ssh_key` is the location of the generated SSH private key for your cluster and `/opt/icp320/cluster` is the directory configured for IBM Cloud Private installation.

```
cp -p /opt/<installation-directory>/config/ssh_key /opt/icp312/cluster
```

Procedure

1. Download the IBM Cloud Private installation package to the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.
   - For the community edition, pull the docker image by using the following command. See Installing IBM Cloud Private-CE for more details

   ```
   sudo docker pull ibmcom/icp-inception:3.2.0
   ```

   - For the enterprise edition, download the following files from IBM Passport Advantage site.
     - `ibm-cloud-private-x86_64-3.2.0.gz`, which is for a Linux 64-bit IBM Cloud Private cluster if you want to run the master node on the x86 server.
     - `ibm-cloud-private-s390x-3.2.0.gz`, which is for a cluster that uses IBM Z nodes.
2. Install the IBM Cloud Private runtime environment based on the release of your choice.
   - For the community edition, follow the instructions on Installing an IBM Cloud Private-CE.

  - For the enterprise edition, follow the instructions on Installing an IBM Cloud Private Enterprise environment.

Note:

- When customizing the IBM Cloud Private cluster, the value of `cluster_lb_address` parameter in the `<icp_installation_directory>/cluster/config.yaml` file must be set to the public IP address of master node that will be accessed, and the `proxy_lb_address` parameter set to the public IP address of proxy node.
- You must config the `offline_pkg_copy_path: /var/tmp` value pair in the `config.yaml` file as the staging directory for IBM Cloud Private installation files on all the nodes, and then use the `-e ANSIBLE_REMOTE_TEMP=/var/tmp` option with the install command.
  - For an IBM Cloud Private community edition, run the following command.

  ```
  sudo docker run --network=host -t -e LICENSE=accept -v "$(pwd)":/installer/cluster ibmcom/icp-
  inception:3.2.0 install -e ANSIBLE_REMOTE_TEMP=/var/tmp
  ```

  - For an IBM Cloud Private enterprise edition, run the following command.

  ```
  sudo docker run --network=host -t -e LICENSE=accept -v "$(pwd)":/installer/cluster ibmcom/icp-
  inception-amd64:3.2.0-ee install -e ANSIBLE_REMOTE_TEMP=/var/tmp
  ```

- When configuring the calico network settings in the `<icp_installation_directory>/cluster/config.yaml` file, use `1350` as the value of `calico_tunnel_mtu` parameter. For more information about the network settings in the `config.yaml` file, see Network settings. See the following code snippet as an example. Note that `ens7` in the example is the device name of master node. You need to replace `ens7` with the actual name of the primary network device on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.

```
## Calico Network Settings
calico_ipip_enabled: true
calico_tunnel_mtu: 1350
calico_ip_autodetection_method: interface=eth0,eth1,ens7
```

- You can configure the `skip_host_ip_check: true` value pair in the `config.yaml` file to skip checking IP addresses in the `hosts` file. The hosts file is generated with the IP address and name of each cluster node by the Secure Service Container for IBM Cloud Private CLI tool.
- After the installation completes, you might want to wait a few minutes before accessing the IBM Cloud Private console. The UI services of IBM Cloud Private needs time to fully up and running. For more details, see Troubleshooting Secure Service Container for IBM Cloud Private.
- When accessing the IBM Cloud Private cluster, use the IBM Cloud Private console URL `https://<cluster_lb_address>:8443` with the default user name and password that is displayed on the screen.
- High Availability (HA) is not supported in the Secure Service Container for IBM Cloud Private.
- Do not perform any activities on Isolated VMs or the cluster nodes other than documentation instructions advised.
- You must use a separate s390x architecture Linux LPAR to build docker images relevant to your workloads, and then load them on to the docker image registry on the master node for deployments.
- After the installation completes, you must ensure the pod security policy is set to `restricted` on the IBM Cloud Private. If it is not set as `restricted`, use the command `cloudctl cm psp-default-set restricted` to enable the policy. For more information, see Enabling pod isolation.
- If you use dedicated nodes for GlusterFS by following the instructions on Deploying GlusterFS, you must update the `config.yaml` and `hosts` files with the storage configuration. For more information, see Configuring GlusterFS during IBM Cloud Private installation. For example:

- In the `config.yaml` file, add the `storage-glustefs` section to specify quotagroups for each GlusterFS node configured in the `ssc4icp-config.yaml` file. For the details of each device, refer to the `config/quotagroup-symlink.yaml` file.

  ```
  storage-glusterfs:
    nodes:
      - ip: 192.168.0.245
        devices:
          - /dev/disk/by-runq-id/storage_17001_glusterfs1_qg
          - /dev/disk/by-runq-id/storage_17001_glusterfs2_qg
      - ip: 192.168.0.246
        devices:
          - /dev/disk/by-runq-id/storage_18001_glusterfs1_qg
      - ip: 192.168.0.247
        devices:
          - /dev/disk/by-runq-id/storage_18002_glusterfs1_qg
  ```

- In the `hosts` file, add `hostgroup-glusterfs` section with the IP address of each GlusterFS nodes.

  ```
  [hostgroup-glusterfs]
  192.168.0.245
  192.168.0.246
  192.168.0.247
  ```

- If you restart one of GlusterFS nodes, you must update the glusterfs daemonset file to avoid the OCI (Open Containers Initiative) runtime error. For more information, see OCI runtime error after GlusterFS node restarted.
- You have to manually clean up the GlusterFS resources after you delete the GlusterFS PV (persistent volume) or persistent volume claims (PVCs) by using the IBM Cloud Private CLI or user interface. For more information, see Preparing your nodes for a reinstallation of GlusterFS or IBM Cloud Private.
- The total amount of time to create GlusterFS PVC resources might vary even for a 1GB volume.

Result

1. Check the IBM Cloud Private console at the URL `https://<cluster_lb_address>:8443`.
2. Check the GlusterFS nodes by using the command `kubectl get po -n kube-system | grep gluster`.

   ```
   storage-glusterfs-glusterfs-daemonset-cs8rz                    1/1     Running
   0          39m
   storage-glusterfs-glusterfs-daemonset-tmfr4                    1/1     Running
   0          39m
   storage-glusterfs-glusterfs-daemonset-xgqnb                    1/1     Running
   0          39m
   storage-glusterfs-glusterfs-heketi-deployment-5446d748f9-6clmb   1/1     Running
   0          39m
   ```

3. Check the `storageclass` information by using the command `kubectl get storageclass -n kube-system`.

   ```
   NAME                       PROVISIONER                  AGE
   glusterfs                  kubernetes.io/glusterfs      41m
   ...
   ```

Next

You can manually install the Helm charts for your applications or other products that are included with your IBM Cloud Private package. See Deploying containerized applications for more details.

Deploying containerized applications

To deploy your own workloads to the IBM Cloud Private cluster on a Z or LinuxONE system, you need to install the bundled components into the IBM Cloud Private Catalog, and then use Helm charts for your applications.

This procedure is intended for users with role *cloud administrator*.

Before you begin

- Check that you have installed the IBM Cloud Private cluster successfully, and the IBM Cloud Private console is accessible.
- Check that you have installed the IBM Cloud Private CLI and log in to your cluster.

Procedure

1. Install the bundled components into the IBM Cloud Private Catalog. See Installing bundled products for more details. The following bundled components are supported by IBM Cloud Private on IBM Z or LinuxONE architecture, and listed by its application category on the IBM Cloud Private Catalog.
   - Data Services
     - IBM Db2 Direct Advanced Edition 11.1 with Data Server Manager
     - IBM Db2 Advanced Enterprise Server Ed. 11.1 with Data Server Manager
     - IBM Db2 Warehouse Enterprise 2.0
     - MongoDB
     - PostgreSQL
     - MariaDB
   - Data Science and Business Analytics
     - IBM Data Science Experience Local 1.1
   - Toolchains & Runtimes
     - IBM Urban Code Deploy
     - Microclimate 18.03
     - Jenkins
     - IBM WebSphere Liberty 17.0.0.4, 18.0.0.x
     - IBM SDK for Node.js V6, V8
     - Open Liberty
     - Swift runtime

- Modernization Tools
  - IBM Transformation Advisor 1.5.1
- Messaging
  - IBM MQ Advanced 9.0 & v.next
  - Rabbit MQ
- Digital Business Automation
  - IBM Operational Decision Manager 8.9.2

1. Install your own component into the IBM Cloud Private Catalog by using Helm charts. See IBM Cloud Private enablement guide for ISV and open source software for more details.

Updating the cluster resources dynamically

You can change the CPU or memory resources on the cluster nodes while the cluster is still running.

This procedure is intended for users with role *cloud administrator*.

Before you begin

- Check the cluster configuration file `ssc4icp-config.yaml` file to understand how many worker or proxy nodes will be impacted after updating the CPU or memory configurations. For more information, see Configuring the cluster resources.

Procedure

On the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, complete the following steps under the `<installation_directory>/SSC4ICP/config` directory with root user authority.

1. Identify the node IP addresses that you want to update the resources by using the following command. You can also get the information from the `ssc4icp-config.yaml` file by checking the node IP addresses that uses the template to be updated.

   ```
   kubectl get nodes
   ```

2. Configure each of those to-be-updated cluster nodes to the maintenance mode by following the instructions on Node maintenance.

   ```
   kubectl cordon <node_IP_address>
   kubectl drain <node_IP_address> --grace-period=300 --ignore-daemonsets=true
   ```

3. Update the `config/ssc4icp-config-update.yaml` file to include the nodes to be updated and resource settings you want to apply. For example, if you want to change the CPU number to 5 for one of worker nodes `192.168.0.252`, the `ssc4icp-config-update.yaml` only needs to specify the configuration related to this node `192.168.0.252`, such as LPAR configuration, and the template for this worker node.

   ```
   cluster:
     name: "temp"
     datapool: "exists"
     masterconfig:
       internal_ips: ['192.168.0.251']
       subnet: "192.168.0.0/24"
   LPARS:
     -ipaddress: '10.152.151.105'
      containers:
       -template: "template1"
        count: 1
        internal_ips: ['192.168.0.252']
   template1:
      name: "worker"
      type: "WORKER"
      cpu: "5"
      memory: "4098"
      port_range: '15000'
      root_storage: "60G"
      icp_storage: "140G"
      internal_network:
        subnet: "192.168.0.0/24"
        gateway: "192.168.0.1"
        parent: "encf700"
   ```

   Note:

   - Only the changes to the CPU and `Memory` settings will be applied. Any other changes in the `ssc4icp-config-update.yaml` file will be ignored. Ensure the total CPU or memory of your cluster do not exceed the assigned resources on the Secure Service Container partitions.
   - The cluster name in the `ssc4icp-config-update.yaml` file must be `temp`.
   - The value of `count` setting under the `containers` configuration is 1 because only one worker node will be updated.
   - For more information of the existing cluster settings in the `ssc4icp-config.yaml` file, see Configuring the cluster resources.

4. Apply the changes to the cluster node by using the following command.

   ```
   docker run --rm -it --net host -v $(pwd)/config:/ssc4icp-cli-installer/config ibmzcontainers/
   hpvs-cli-installer:1.2.0 update
   ```

   Note:

- The `cluster-configuration.yaml` file of the cluster will be refreshed with the new configuration. For more information of the `cluster-configuration.yaml` file, see Creating the cluster nodes.

5. After the command completes, the updated cluster nodes are in the `NotReady` state. Run the following commands to reactivate those cluster nodes.

   a. Remove those cluster nodes by following the instructions on Removing an IBM Cloud Private cluster node. For example, remove the updated worker node `192.168.0.252` from the existing cluster by running the following command on the Linux on IBM Z/LinuxONE (such as s390x architecture) management server.

   ```
   docker run -e LICENSE=accept --net=host -t -e LICENSE=accept -v "$(pwd)":/installer/
   cluster ibmcom/icp-inception-s390x:3.2.0-ee uninstall -l 192.168.0.252
   ```

   b. Configure the `/cluster/hosts` file under the IBM Cloud Private installation directory to include the cluster node information. For example, for a cluster with two worker nodes and one proxy node, the `cluster/hosts` file resembles the following settings.

   ```
   [master]
   <master_node_IP_address> ansible_user="root"
   [worker]
   <worker_node_1_IP_address> ansible_user="root"
   <worker_node_2_IP_address> ansible_user="root"
   [proxy]
   <proxy_node_IP_address> ansible_user="root"
   ```

   c. Log in to the updated cluster nodes by using the `ssh` utility, and configure the `/etc/hosts` file by adding the IP address and node name of each cluster node. You can get the IP address and node name information from the `cluster-configuration.yaml` file. For example, the `/etc/hosts` file for a cluster with two worker nodes and one proxy node resembles the following settings.

   ```
   ...
   <master_node_IP_address> <master_node_host_name>
   <worker_node_1_IP_address> <worker_node_1_host_name>
   <worker_node_2_IP_address> <worker_node_2_IP_host_name>
   <proxy_node_IP_address> <proxy_node_host_name>
   ```

   d. Add those cluster nodes back into the cluster by following the instructions on Adding an IBM Cloud Private cluster node. For example, add the updated worker node `192.168.0.252` back into the existing cluster by running the following command on the Linux on IBM Z/LinuxONE (such as s390x architecture) management server.

   ```
   docker run -e LICENSE=accept --net=host -t -e LICENSE=accept -v "$(pwd)":/installer/
   cluster ibmcom/icp-inception-s390x:3.2.0-ee worker -l 192.168.0.252
   ```

Upgrading Secure Service Container for IBM Cloud Private

The following tasks are required to upgrade the Secure Service Container for IBM Cloud Private environment. Note that the upgrade of Secure Service Container for IBM Cloud Private must be coordinated between the Appliance administrator and Cloud administrator.

1. Planning for the upgrade
2. Upgrading the Secure Service Container for IBM Cloud Private appliance
3. Upgrading the Secure Service Container for IBM Cloud Private CLI tool
4. Upgrading cluster nodes with the isolated VM
5. Upgrading the IBM Cloud Private

Planning for the upgrade

Before you start upgrading the Secure Service Container for IBM Cloud Private, you can use a worksheet to get an overall understanding on what information you will need to complete the upgrade procedure.

This procedure is intended for users with role *cloud administrator*.

Before you begin

- Ensure that the appliance administrator is notified upon the upcoming upgrade.
- Ensure that the workload owners in the cluster are notified upon the upcoming upgrade.

Procedure

- Check whether the target version is supported for your existing Secure Service Container for IBM Cloud Private version. Note that the upgrade only supports master node on the x86 server.

Table 1. Secure Service Container for IBM Cloud Private upgrade path

Current version	Target version	Upgrade supported
Beta	1.1.0	No
Beta	1.2.0	No
1.1.0	1.2.0	Yes

- Check whether the operation system on the x86 server is supported for upgrade. For more information, see Supported operating systems and platform.
- Check whether the version of `Docker-CE` and `Strongswan` on the master node are supported for the upgrade.

Table 2. Components version supported for upgrade

Strongswan	Docker-CE
5.6.2 or later	18.06.3-CE

- Download the latest Secure Service Container for IBM Cloud Private binary to the x86 server, and extract the binary to an installation directory. For example, `./1.2.0`.
- Download the upgrade script `CC2VREN.sh` from the IBM Passport Advantage website, and put it into the current Secure Service Container for IBM Cloud Private installation directory. For example, `./1.1.0`.
- On the x86 server, mark the worker and proxy nodes in preparation of maintenance.
  1. Use the `kubectl get nodes` command to get a list of cluster nodes, and make notes of the IP addresses of worker and proxy nodes.
  2. Run the following commands to drain and cordon each worker and proxy node.

  ```
  kubectl drain <node_IP_address>
  kubectl cordon <node_IP_address>
  ```

  Where `<node_IP_address>` is the IP address of each worker or proxy node that you get by running the `kubectl get nodes` command.

Next

You can now work with the appliance administrator to upgrade the Secure Service Container for IBM Cloud Private appliance.

Upgrading the Secure Service Container for IBM Cloud Private appliance

You can upgrade the Secure Service Container partitions with the latest Secure Service Container for IBM Cloud Private appliance.

This procedure is intended for users with role *appliance administrator*.

Before you begin

- Check that you have the IP address of Secure Service Container partitions, and master ID and password.
- Check that you have the latest appliance image `secure-service-container-for-icp.appliance.<version_number>.img.gz` in the installation directory.
- Ensure that you complete the instructions in the Planning for the upgrade topic.

Procedure

1. Connect to the Secure Service Container installer through the browser of your choice, and log in with the master ID and password.
2. Export the appliance configuration data by following the instructions after you download Secure Service Container User's Guide from the About topic.
   - Chapter 14, "Using the Secure Service Container user interface", section " Exporting or importing appliance configuration data"
3. Install the latest appliance image by following the instructions after you download Secure Service Container User's Guide from the About topic.
   - Chapter 13, "Installing a new software appliance in a Secure Service Container partition"
4. Once the installation completes, import the appliance configuration data by following the instructions after you download Secure Service Container User's Guide from the About topic.
   - Chapter 14, "Using the Secure Service Container user interface", section " Exporting or importing appliance configuration data"

Note: You must repeat all the steps on each Secure Service Container partition that are used to host the cluster.

Next

You can now follow the instructions on Upgrading the Secure Service Container for IBM Cloud Private CLI tool to upgrade the Secure Service Container for IBM Cloud Private CLI tool.

Upgrading the Secure Service Container for IBM Cloud Private CLI tool

You can upgrade the Secure Service Container for IBM Cloud Private CLI tool by downloading the latest offering from the IBM Passport Advantage website.

This procedure is intended for users with role *cloud administrator*.

Before you begin

- Ensure that you complete the instructions in the Upgrading the Secure Service Container for IBM Cloud Private appliance.
- Ensure that you download the upgrade script image `CC2VREN.sh` from the IBM Passport Advantage as instructed in the Planning for the upgrade.

Procedure

On the x86 server, complete the following steps under the `<installation_directory>/SSC4ICP/config` directory with root user authority..

1. Back up the version files for your existing Secure Service Container for IBM Cloud Private CLI tool, by running the following command under the current Secure Service Container for IBM Cloud Private installation directory.

   ```
   docker run --rm -v $(pwd):/data ibmzcontainers/hpvs-cli-installer:<installed_version> cp
   ssc4icp-config/version /data/
   ```

   Where the `<installed_version>` is the previous CLI tool version number. For example, `1.1.2`.

2. Rename the upgrade script that you get from the IBM Passport Advantage.

   ```
   mv CC2VREN.sh ISSC_4_ICP_1103_Up_Script.sh
   ```

3. Make the upgrade script executable by using the following command.

   ```
   chmod +x ISSC_4_ICP_1103_Up_Script.sh
   ```

4. (Optional) If you update the CLI tool from `v1.1.2` to `v1.2.0`, you must create a file named as `version` with the following content current Secure Service Container for IBM Cloud Private installation directory.

   ```
   Version: 1.1.2
   ```

5. Run the upgrade script to update the isolated VM images under the `/config` directory, and also create a backup of your existing `/config` directory for future use. Note that the cluster configuration file `ssc4icp-config.yaml` will be reused when upgrading the cluster nodes.

   ```
   ./ISSC_4_ICP_1103_Up_Script.sh <fully_qualified_path_to_cli_install_package>
   ```

   Where:

   - **fully_qualified_path_to_cli_install_package** is the fully qualified path to the Secure Service Container for IBM Cloud Private image in `gzip` format that you download from the IBM Passport Advantage.

6. Check the cluster nodes are either in `Running` or `Paused` state by using the `kubectl get nodes` command again.
7. Go to the directory where you extract the latest Secure Service Container for IBM Cloud Private binary, and stop the cluster nodes with the following command.

   ```
   docker run --rm -it --net host -v $(pwd)/config:/ssc4icp-cli-installer/config ibmzcontainers/
   hpvs-cli-installer:1.2.0 stop
   ```

   Note:

   - You must ensure that the network configuration is persisted on the master node by followig the instructions on Configuring the network for the master node.If the command fails, you can reboot the master node
   - If the command fails because the connection to the worker or proxy node does not work, you can reboot the master node to ensure the connection is recovered.
   - If the problem still exists, restart the `strongswan` service and have another try to test the connection to the worker or proxy node.

Next

You can now upgrade each cluster node by following the instructions in Upgrading isolated VMs topic.

Upgrading cluster nodes with the isolated VM

You can upgrade the cluster nodes by using the latest isolated VM image.

This procedure is intended for users with role *cloud administrator*.

Before you begin

- Check that you have the IP addresses of all cluster nodes.
- Check that the Secure Service Container for IBM Cloud Private CLI tool is upgraded to the latest version. For more information, see Upgrading the Secure Service Container for IBM Cloud Private CLI tool.
- Check with the appliance administrator that the Secure Service Container partitions are upgraded to the latest version. For more information, see Upgrading the Secure Service Container for IBM Cloud Private appliance.

Procedure

On the x86 server, complete the following steps under the `<installation_directory>/SSC4ICP/config` directory with root user authority..

1. Use the Secure Service Container for IBM Cloud Private CLI tool to initiate the upgrade of cluster nodes.

   ```
   docker run --rm -it --net host -v $(pwd)/config:/ssc4icp-cli-installer/config ibmzcontainers/
   hpvs-cli-installer:1.2.0 upgrade
   ```

2. Go to the IBM Cloud Private installation directory, and complete the following steps:

   a. Uninstall the IBM Cloud Private nodes.

   ```
   docker run -e LICENSE=accept --net=host \
   -v "$(pwd)":/installer/cluster \
   ibmcom/icp-inception:<version_number> uninstall -l \
   <ip_address_list_of_cluster_nodes>
   ```

   Note:

   - Replace `<ip_address_list_of_cluster_nodes>` with IP addresses of cluster nodes separated by using a comma `,`. For example, `192.168.0.252,192.168.0.253,192.168.0.254`.
   - Replace `<version_number>` with the existing IBM Cloud Private version number. For example, `3.1.2` or `3.2.0`.

   b. Upgrade the proxy nodes by using the latest isolated VM.

   ```
   docker run -e LICENSE=accept --net=host \
   -v "$(pwd)":/installer/cluster \
   ibmcom/icp-inception:<version_number> worker -l \
   <ip_address_list_of_proxy_nodes>
   ```

   Note:

   - Replace `<ip_address_list_of_proxy_nodes>` with IP addresses of cluster nodes separated by using a comma `,`. For example, `192.168.0.254`.
   - Replace `<version_number>` with the existing IBM Cloud Private version number. For example, `3.1.2` or `3.2.0`.

   c. Upgrade the worker nodes by using the latest isolated VM.

   ```
   docker run -e LICENSE=accept --net=host \
   -v "$(pwd)":/installer/cluster \
   ibmcom/icp-inception:<version_number> worker -l \
   <ip_address_list_of_worker_nodes>
   ```

   Note:

   - Replace `<ip_address_list_of_worker_nodes>` with IP addresses of cluster nodes separated by using a comma `,`. For example, `192.168.0.252,192.168.0.253`.
   - Replace `<version_number>` with the existing IBM Cloud Private version number. For example, `3.1.2` or `3.2.0`.

Next

You can now upgrade the IBM Cloud Private by following the instructions on Upgrading IBM Cloud Private topic.

Upgrading the IBM Cloud Private

You can upgrade the IBM Cloud Private from specific previous versions.

This procedure is intended for users with role *cloud administrator*.

The supported upgrade paths are:

- IBM Cloud Private version 3.1.1 to 3.1.2
- IBM Cloud Private version 3.1.1 to 3.2.0
- IBM Cloud Private version 3.1.2 to 3.2.0

Procedure

1. Depending on the IBM Cloud Private release you have installed, follow the instructions to complete the upgrade.
   - For IBM Cloud Private Community edition, see Upgrading IBM Cloud Private-CE.
   - For IBM Cloud Private Enterprise edition, see Upgrading IBM Cloud Private.
2. After the upgrade is completed, clean up the temporary files that are created during the Secure Service Container for IBM Cloud Private upgrade by using the CLI tool.

   ```
   ${pwd}./ssc4icp-cli-installer-upgrade-<version>.sh clean_up
   ```

   Note:

   - The `cleanup` command must be executed under the `/config` directory of the Secure Service Container for IBM Cloud Private installation directory.
   - You must replace the `<version>` with the correct CLI version number. For example, `1.2.0`.
3. To dynamically update the CPU or memory resources on the cluster nodes, complete the following steps.

   a. Create a `tmp` directory in the home directory, and go to this `tmp` directory.

   ```
   mkdir ~/tmp && cd tmp
   ```

   a. Download the configuration template files into the `tmp` directory.

   ```
   docker run --network=host --rm -v $(pwd):/data ibmzcontainers/hpvs-cli-installer:1.2.0 cp -r
   config /data
   ```

   a. Copy the `ssc4icp-config-update.yaml` file into the Secure Service Container for IBM Cloud Private installation directory.

   ```
   cp config/ssc4icp-update.yaml to the SSC4ICP_INSTALLATION_FOLDER/config/
   ```

   a. (Optional) Update the resources for the cluster nodes. For more information, see Updating the cluster resources dynamically.

Reverting the Secure Service Container for IBM Cloud Private

Reverting the Secure Service Container for IBM Cloud Private includes multiple manual steps, and must be coordinated between the *appliance administrator* and *cloud administrator*.

You must follow the exact instructions by reverting the IBM Cloud Private first. See the instructions in the Rollback IBM Cloud Private section.

If any failure occurs during the upgrade, you must rollback the Secure Service Container for IBM Cloud Private infrastructure to its pre-upgrade state.

The following table shows the sequence of rollback actions when a failure occurs during the upgrade procedure.

Table 1. Rollback operations for each upgrade

A failure occurred during...	Rollback operations to be followed
IBM Cloud Private Cluster upgrade	• Rollback isolated VMs on cluster nodes • Rollback CLI installer • Downgrade Secure Service Container for IBM Cloud Private appliance • Undo upgrade preparation steps – Restore components on the master node – Restart all containers – Uncordon all cluster nodes
Isolated VM upgrade	• Rollback CLI installer • Downgrade Secure Service Container for IBM Cloud Private appliance • Undo upgrade preparation steps – Restore components on the master node – Restart all containers – Uncordon all cluster nodes
CLI installer upgrade	• Downgrade Secure Service Container for IBM Cloud Private appliance • Undo upgrade preparation steps – Restore components on the master node – Restart all containers – Uncordon all cluster nodes
Appliance upgrade	• Undo upgrade preparation steps – Restore components on the master node – Restart all containers – Uncordon all cluster nodes
Upgrade preparation	• Restore components on the master node • Restart all containers • Uncordon all cluster nodes

Note:

- When the upgrade fails, it is recommended to try the upgrade steps.
- If Retrying the upgrade does not work, you can either contact IBM Support to resume the upgrade, or perform the rollback operations.
- If the upgrade fails at a given sub-step, you need to ensure the system is take to the state prior to running this sub-step, then perform the rollback operations.

For example, if a failure occurs during the Secure Service Container for IBM Cloud Private CLI tool upgrade, you must ensure that the previous CLI tool is in place. Then proceed with the rollback steps. Use the following command to validate the CLI tool version.

```
 docker run --rm -it --net host $(pwd):/data ibmzcontainers/hpvs-cli-
installer:<installed_version> -version
```

Where the `<installed_version>` is the previous CLI tool version number. For example, `1.1.2`.

Undo upgrade preparation steps

This task is intended for users with role *cloud administrator*.

1. Log in to the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, and downgrade the `Docker-CE` and `strongswan` to the previous version.
2. Restart all the containers by using the Secure Service Container for IBM Cloud Private CLI tool.

```
docker run --rm -it --net host -v $(pwd)/config:/ssc4icp-cli-installer/config ssc4icp/
ssc4icp-cli-installer start
```

3. Get the list of node IP addresses and uncordon each of them.

```
kubectl get nodes
kubectl uncordon <node_IP_address>
```

Downgrade the Secure Service Container for IBM Cloud Private appliance

This task is intended for users with role *appliance administrator*.

1. Log in to the Secure Service Container user interface via your choice of browser. For example, `https://<LPAR_IP_Address>`.
2. Install the Secure Service Container for IBM Cloud Private appliance from your previous Secure Service Container for IBM Cloud Private installation directory, by following the instructions after you download Secure Service Container User's Guide from the About topic.
   - Chapter 13, "Installing a new software appliance in a Secure Service Container partition"
3. Import the configuration data that is exported when Upgrading the appliance, by following the instructions after you download Secure Service Container User's Guide from the About topic.
   - Chapter 14, "Using the Secure Service Container user interface", section " Exporting or importing appliance configuration data"
4. Notify the *cloud administrator* to run the steps as instructed in Undo the upgrade preparation.

Rollback the Secure Service Container for IBM Cloud Private CLI tool

This task is intended for users with role *cloud administrator*.

1. On the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server, run the following command under the `/config` directory of the Secure Service Container for IBM Cloud Private installation directory.

```
sh ssc4icp-cli-installer-upgrade.sh rollback_installer
```

2. Notify the *appliance administrator* to run the steps as instructed in Downgrade the Secure Service Container for IBM Cloud Private appliance.
3. Run the steps as instructions in Undo the upgrade preparation.

Rollback the isoldated VMs on each cluster node

This task is intended for users with role *cloud administrator*.

1. Run the steps as instructed in Rollback the Secure Service Container for IBM Cloud Private CLI tool.
2. Notify the *appliance administrator* to run the steps as instructed in Downgrade the Secure Service Container for IBM Cloud Private appliance.
3. Run the steps as instructions in Undo the upgrade preparation.

Rollback IBM Cloud Private

This task is intended for users with role *cloud administrator*.

1. Following the instructions based on the previous IBM Cloud Private version before you upgrade.
   - Reverting IBM Cloud Private from version 3.1.2
   - Reverting IBM Cloud Private from version 3.2.0
2. Run the steps as instructed in Rollback the isoldated VMs on each cluster node.
3. Notify the *appliance administrator* to run the steps as instructed in Downgrade the Secure Service Container for IBM Cloud Private appliance.
4. Run the steps as instructions in Undo the upgrade preparation.

Uninstalling Secure Service Container for IBM Cloud Private

You can uninstall the Secure Service Container for IBM Cloud Private offering. Note that you must back up your own workload first, and the Secure Service Container for IBM Cloud Private configuration cannot be reused on another Secure Service Container partition.

1. Uninstalling the IBM Cloud Private
2. Uninstalling the Secure Service Container for IBM Cloud Private CLI tool
3. Uninstalling Secure Service Container partitions

Uninstalling IBM Cloud Private

Use this procedure to uninstall the IBM Cloud Private.

This procedure is intended for users with role *cloud administrator*.

Procedure

1. Log in as a root user.
2. Run the `docker save` command to back up all the images for your workloads.
3. Uninstall the IBM Cloud Private runtime environment by following the instructions on Uninstalling IBM Cloud Private Community Edition or Uninstalling IBM Cloud Private Enterprise Edition.

Uninstalling the Secure Service Container for IBM Cloud Private CLI tool

Use this procedure to uninstall the Secure Service Container for IBM Cloud Private CLI from the x86 or Linux on Z master node server.

This procedure is intended for users with role *cloud administrator*.

Procedure

1. Log in as a root user.
2. Deleting all the nodes by run the following command under the `config` folder.

   ```
   docker run --network=host --rm -it -v $(pwd)/config:/ssc4icp-cli-installer/config
   ibmzcontainers/hpvs-cli-installer:1.2.0 uninstall
   ```

   The uninstallation uses the configuration in the `ssc4icp-config.yaml` file to delete all the cluster nodes and configurations that are created on the Secure Service Container partitions. The command also removes the SSH key files from your Secure Service Container for IBM Cloud Private `config` directory.

3. Run the following command to perform the unregistration process on the Secure Service Container partitions.

   ```
   docker run --network=host --rm -it -v $(pwd)/config:/ssc4icp-cli-installer/config
   ibmzcontainers/hpvs-cli-installer:1.2.0 cleanup
   ```

   If you want to completely remove any active node on the Secure Service Container partitions, run the forcing cleanup by using the `--force` option with the `cleanup` parameter to delete these nodes during the unregistration process. For example,

   ```
   docker run --network=host --rm -it -v $(pwd)/config:/ssc4icp-cli-installer/config
   ibmzcontainers/hpvs-cli-installer:1.2.0 cleanup --force
   ```

   Note: The quotagroups of these active nodes will not be deleted after the forcing cleanup.

4. (Optional) Delete the SSH key pair generated for the cluster and remove the associated SSH access from your master node. If you do not delete the SSH key pair, then when you install the IBM Cloud Private again by using the same cluster installation directory, the existing SSH keys will be used.

   a. Find the public SSH key in your `authorized_keys` file, including the line number, by using the following command. You need to modify the path of `~/config/ssh_key.pub` file to match the public key location inside your Secure Service Container for IBM Cloud Private `config` directory when you run the command.

   ```
   grep -n "$(cat ~/config/ssh_key.pub)" /root/.ssh/authorized_keys
   ```

   b. Remove the identified line from your `authorized_keys` file by using a text editor. For example:

   ```
       vi /root/.ssh/authorized_keys
   ```

   c. Remove the SSH private key from the IBM Cloud Private installation directory by using the following command. You need to modify the path of `~/cluster/ssh_key` file to match the SSH private key location in your IBM Cloud Private installation directory when you run the command.

   ```
   rm -rf ~/cluster/ssh_key
   ```

Uninstalling Secure Service Container partitions

You can stop, deactivate, or delete the Secure Service Container partitions on the IBM Z or LinuxONE machine.

This procedure is intended for users with role *system administrator*.

Before you begin

- Check whether your host system is running in standard mode (that is, with Processor Resource/System Manager or PR/SM) or has Dynamic Partition Manager (DPM) enabled.
- Check that you have *IBM Z Secure Service Container User's Guide* at hand.

Procedure

1. (Optional) Export the Secure Service Container configuration by following the description in section *Exporting or importing appliance configuration data* of chapter 14 "Using the Secure Service Container user interface".
2. Stop/deactivate or delete the Secure Service Container partition:
   - On a standard mode system, chapter 7 - Deactivating or deleting a Secure Service Container partition on a standard mode system.
   - On a DPM-enabled system, chapter 12 - Stopping or deleting a Secure Service Container partition on a DPM-enabled system.

Troubleshooting IBM Hyper Protect Virtual Servers

When you run into problems with IBM Hyper Protect Virtual Servers, you can refer to the following information:

- For Secure Service Container partitions, use the Secure Service Container user interface to view the logs.
- For components such as Secure Build containers, use the `securebuild log` command to retrieve the build logs, or `securebuild status` command to check the progress of the Secure Build.
- For the command line tools provided by the product, enable the `verbose` option to view the detailed log. For example, use the `-v` docker command option together with the `-v*`, `-vv*`, or `-vvv*` IBM Hyper Protect Virtual Servers CLI command option to see the detailed output from the `quotagroup` command.

```
docker run --rm -it -e -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
quotagroup list --all --language EN -v*
```

- For the docker commands executed when using the product, you can find an individual log file created for each `docker run` command under the `/logs` folder of the current directory.
- For the commands that require a yaml configuration file, check the formatting of the yaml file by comparing with the example yaml file in the `/config` directory of each command.

For the known issues about the IBM Secure Service Container for IBM Cloud Private, see Troubleshooting IBM Secure Service Container for IBM Cloud Private.

`standard_init_linux.go:<integer>:` exec user process caused "exec format error"

You might notice one of the following errors when running the IBM Hyper Protect Virtual Servers CLI tool.

- standard_init_linux.go:211: exec user process caused "exec format error", or
- standard_init_linux.go:190: exec user process caused "exec format error"

The problem is most likely caused by the CLI commands for x86 architecture being executed on the Linux on IBM Z/LinuxONE (such as s390x architecture) management server, or the s390x architecture CLI commands being executed on the x86 Linux system.

To workaround the problem, ensure that you use the CLI tool with the correct tag for the management server.

- The IBM Hyper Protect Virtual Servers CLI tagged with 1.2.0.s390x must be executed on the s390x architecture management server.
- The IBM Hyper Protect Virtual Servers CLI tagged with 1.2.0 must be executed on the x86 architecture management server.

"gpg: Invalid option" errors when generating the GPG key pair

You might encounter an error messages such as `gpg: Invalid option "--pinentry-mode=loopback"` or `gpg: Invalide opiton "--generate-key"` when generating the GPG key pair on the s390x Linux management server.

The problem is most likely caused by a different version of GnuPG tools that you have installed, such as `gnupg 1.4.20-1ubuntu3.3`, or `gnupg2 2.1.11-6ubuntu2.1`.

To resolve the problem, use `--gen-key` option instead of `--generate-key` in the command, or upgrade GnuPG to a later version such as 2.2.17.

"Not enough random bytes available." error when generating the GPG key pair

You might encounter an error messages such as `Not enough random bytes available. Please do some other work to give the OS a chance to collect more entropy! (Need 188 more bytes)` when generating the GPG key pair on the s390x Linux management server.

The problem is most likely caused by a missing utility haveged on the Linux management server. For more information, see Stackoverflow.

To resolve the problem, install the haveged utility with the following command:

```
apt-get install -y haveged
```

Configuration files not found when running the command

You might encounter one of the following error messages when running the command such as `securebuild create`, which requires the hosts and `securebuild.yaml` configuration file.

- `fatal: [localhost]: FAILED! => {"changed": false, "meta": {"yaml_path": "/securebuild-cli/config/securebuild.yaml"}, "msg": "Could not open Secure Build containers configuration file /securebuild-cli/config/securebuild.yaml for reading. [Errno 2] No such file or directory: '/securebuild-cli/config/securebuild.yaml'"}`. Or,
- `fatal: [localhost]: FAILED! => {"changed": false, "msg": "Response: {'failed': True, 'msg': 'The task includes an option with an undefined variable. ... The error appears to have been in \\'/securebuild-cli/playbook/roles/hostingappliance-base/tasks/retrieve-apitoken.yaml\\': line 13, column 5, but may\\nbe elsewhere in the file depending on the exact syntax problem.\\n\\nThe offending line appears to be:\\n\\n- block:\\n - name: \"{{ RETRIEVE_API_TOKEN }} - {{ https_uri }}/{{ rest_api_url_api_token }}\"\\n ^ here\\nWe could be wrong, but this one looks like it might be an issue with\\nmissing quotes. Always quote template expression brackets when they\\nstart a value. For instance:\\n\\n with_items:\\n - {{ foo }}\\n\\nShould be written as:\\n\\n with_items:\\n - \"{{ foo }}\"\\n'}"}`

The problem is most likely caused by the incorrect location of the configuration files required by the command. The IBM Hyper Protect Virtual Servers CLI tool has a number of configuration elements. For example, the `securebuild create` command requires the following two files:

- The `hosts` file, which describes the Secure Service Container partition and the REST API credentials.
- The `securebuild.yaml` file, which describes the Secure Build container itself.

These configuration files can, in principle, be located anywhere on the file system of the management server. One of the best practices is to use a dedicated sub-directory to host those configuration files. Especially if the same CLI tool will be used to manage multiple Secure Build containers across one or more Secure Service Container partitions.

For example, a directory such as `~/cli/config` can be used to host all the configuration files, which means that this `cli/config` directory is placed within the home directory.

In addition, the CLI tool requires that you must specify the path to the configuration root directory when running the CLI tool, either explicitly:

```
docker run --rm -it -v ~/cli/config:/securebuild-cli/config ibmzcontainers/hpvs-cli-
installer:1.2.0 securebuild create --name securebuild_1 --language EN*
```

or implicitly: -

```
docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
securebuild create --name securebuild_1 --language EN*
```

where `$(pwd)` refers to the current directory, which requires you to change the directory before running the CLI.

```
cd ~/cli/config*
docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-installer:latest
securebuild create --name securebuild_1 --language EN*
```

Malformed configuration file issues when running the command

You might encounter one of the following error messages when running the command such as `securebuile create`, and the command requires a yaml configuration file.

- `TypeError: list indices must be integers or slices, not string`
- `TypeError: string indices must be integers`
- `"msg": "Could not parse Secure Build containers configuration file / securebuild-cli/config/securebuild.yaml...`

The problem is most likely caused by the incorrect formatting in the yaml configuration file, such as arrays, scalars, and collections.

To workaround the problem, check the yaml file format by comparing with the yaml example file provided in the `/config` directory of each command. A correctly formatted `securebuild.yaml` file as in the following example can resolve such problems.

```
secure_build_workers:
  - container:
      port: '22443'
      name: 'securebuild_1'
      ipaddress: 'demossc.example.com'
      quotagroup_name: 'qg1'
      repoid: 'SecureDockerBuild'
      imagetag: '1.2.0'
    lpar:
      ipaddress: 'demossc.example.com'
    network:
      name: 'bridge'
    github:
      url: 'git@github.com:dummy/HelloNginx.git'
    docker:
      repo: 'dummy/hello-nginx-awesome-fabbo'
      user: 'dummy'
      password: 'password'
      base_user: 'dummy'
      base_password: 'password'
```

Secure Build failed to clone the Github repository if a passphrase is associated with the private key

You might encounter the following error message or a similar one when the Secure Build tries to build the source code from a Github repository by using the private key with a passphrase.

```
Could not read from remote repository.
```

The problem is most likely caused by a known limitation that the Secure Build requires the private key used to secure access to the source Github repository does not have a passphrase.

To workaround the problem, consider one of the following options.

- Generate a new SSH key pair with the `-N` parameter and an empty passphrase as in the following command example. Note that both private key and public key are generated. The `-m pem` parameter is optional and ensures the private key is generated with a `RSA PRIVATE KEY` comment line.

```
ssh-keygen -t rsa -b 4096 -f /tmp/id_rsa -N "" -m pem
```

- Overwrite the private key with an empty passphrase as in the following command example. Note that the public key is not changed.

```
openssl rsa -in id_rsa -out id_rsa
```

After you generate the new key pair or overwrite the private key, ensure that you update the `github:key` value with the new private key, and then run the `securebuild update` command to apply the changes. For more information, see Updating the configuration of a running Secure Build container.

Invalid IP address for the Secure Build container when running the `securebuild create` command

You might encounter the following error message when you run the `securebuild create` command.

```
PLAY [Initialize the Secure Build container]
***********************************************************************************************
********************************************

TASK [securebuild-base : Initialize the Secure Build container using the values from
securebuild.yaml https://9.20.129.10:22443/image]
**************************************************
fatal: [localhost]: FAILED! => {"censored": "the output has been hidden due to the fact that
'no_log: true' was specified for this result", "changed": false}

TASK [securebuild-base : Print failure from previous task]
***********************************************************************************************
******************************
fatal: [localhost]: FAILED! => {"changed": false, "msg": "Init response: {'content': '',
'redirected': False, 'url': 'https://9.20.129.10:22443/image', 'msg': 'Status code was -1 and
not [201]: Request failed: <urlopen error timed out>', 'status': -1, 'failed': True, 'changed':
False}"}
```

The problem is most likely caused by an invalid IP address specified in the `securebuild.yaml` file for the Secure Build container. For example, the `securebuild.yaml` has the following information.

```
secure_build_workers:
  - container:
      port: '22443'
      name: 'securebuild_1'
      ipaddress: '9.20.129.10'
```

The configuration means that the Secure Build container is configured with port mapping on the Secure Service Container partition, while the IP address `9.20.129.10` is not in the subnet to which the underlying Secure Service Container partition has access.

To workaround the problem, complete the following steps.

1. Update the `securebuild.yaml` file with the correct IP address or port mapping configuration. For example, use the underlying Secure Service Container partition IP address or host name to map from port 443 inside the Secure Build container to port 22443 on the partition.

   ```
   secure_build_workers:
     - container:
       port: '22443'
       name: 'securebuild_1'
       ipaddress: 'demossc.example.com'
   ```

2. Delete the Secure Build container that reported the error message for the invalid IP address configuration.

```
docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-
installer:1.2.0 securebuild delete --name securebuild_1 --language EN --force
```

3. Create the Secure Build container again. Note that the output message such as `The Secure Build Container was successfully initialized: OK` indicates the command completes successfully.

```
docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-
installer:1.2.0 securebuild create --name securebuild_1 --language EN
```

4. (Optional) Validate the connection to the Secure Build Container by using the following cURL command.

```
curl --insecure -X GET https://demossc.example.com:22443/image --cert securebuild-
securebuild_1-crt-cert
```

Note:

- The certificate `securebuild-securebuild_1-crt-cert` used to authenticate to the Secure Build container is generated when the container is created, and is named to align with the name of the container itself, unless being overridden by using the `cert_name` field in the `securebuild.yaml` file. For more information, see Using your own certificate for the Secure Build container.
- The `-insecure` option is merely to indicate to the cURL command that the certificate is self-signed, rather than signed by a Certificate Authority.

Key pair issues when creating or registering a repository definition file

You might encounter one of the following error messages when an error message such as `CHZ00083E Could not decrypt input data` when registering the repository.

- `CHZ00083E Could not decrypt input data` when registering the repository.
- `CHZ00213E Invalid repository definition json file: invalid vendor key, found 2 key(s).` when running the `repository create` command.
- `CHZ00258E Invalid repository definition json file: only one repo signing key is expected` when running the `repository create` command.
- `gpg: Can't check signature: No public key` or `error: the private key and public key is not a pair` when running the `regfile create` command.

The problem is most likely caused by a mismatch key pair used when running the `regfile create` command. The app developer or ISV might create multiple GPG key pairs on the management server, and the repository definition file is generated containing multiple keys.

To resolve the problem, take the following steps.

1. Check the keys in the local public key ring by using the `gpg --list-keys` command.
2. Check the private keys in the local secret key ring by using the `gpg --list-secret-keys` command.
3. Check that if multiple key pairs are created by using the same `uid`.
4. Delete the public and private key pairs with the same `uid` by using `gpg --delete-key` and `gpg --delete-secret-keys` command to remove those key pairs.
5. Creating a new key pair by using the GnuPG utility. A set of exported keys should be available with a similar layout as the following example under the directory that you store the key pairs.

```
-rw-r--r--@ 1 myId  users  7487 23 Sep 14:51 isv_user.private
-rw-r--r--@ 1 myId  users  3866 23 Sep 14:51 isv_user.pub
```

For more information about the GnuPG utility, see Using the GNU Privacy Guard.

"Internal Server Error" message when running the `image load` command

You might encounter an error message `CHZ00006E Internal Server Error` when running the `image load` command to load the Secure Build container image into the Secure Service Container partition.

The problem is most likely caused by insufficient size of the quotagroup `appliance_data` on the the Secure Service Container partition. The Secure Build container image needs at least 8 GB of disk space on the `appliance_data` quotagroup. If you log in to the Secure Service Container user interface, and check the images log, the following information might exist.

```
...
ImageLoadError: ApplyLayer exit status 1 stdout:  stderr: write /usr/bin/docker-containerd-ctr:
no space left on device
...
```

To resolve the problem, complete the following steps:

1. Increase the quotagroup size of `appliance_data` to 30 GB or above by following the instructions in Configuring the storage on the Secure Service Container partition.
2. Run the `image load` command again by following the instructions in Loading the SecureBuild image on the management server and Secure Service Container partition.

Invalid password problem when running `regfile create` command

You might encounter an error message like `/bin/sh: !9: not found` when running the command `regfile create` to create the signed and encrypted repository definition file.

The problem is most likely caused by special characters in your docker password. The `regfile create` command requires you to provide all the necessary parameters and values to create the signed and encrypted repository definition file in JSON format.

```
docker run --rm -it -v $(pwd):/regfile-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
regfile create --private-key isv_user.private --public-key isv_user.pub --username username --
password password
```

If your docker password contains special characters, such as `ub$V3^JNlk#%2*np9X8AgGsh9y1!9mK0`, the command fails when interpreting the value of the `--password` parameter.

The problem still exists if you wrap the password either in double quotes `"` or single quotes `'`.

To workaround the problem, consider one of the following options:

- Use a simpler password without special characters
- Use an access token generated via Docker Hub
- Use a combination of single quotes `'` and escape symbols `\` for the special characters. For example. `'ub\$V3\^JNlk\#\%2\*np9X8AgGsh9y1\!9mK0'`

"root.json not found" message when running the `regfile create` command

You might encounter an error message like `root.json not found for hpvstest/myubuntu:1.0` when running the command `regfile create` to create the signed and encrypted repository definition file.

The problem is most likely caused by an invalid value for the `-r` or `--repo` parameter within the command. For example, a image tag `1.0` was specified to the repository name by mistake when running the command.

```
docker run --rm -v $(pwd):/regfile-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 regfile
create -u hpvstest -p pass4ibm@2020 -s docker.io -r hpvstest/nmyubuntu:1.0 -ph dummypass -n
ubuntutest -kn isv_user
```

To solve the problem, ensure that the `-r` or `--repo` parameter in the `regfile create` command does not contain any version information. For example, `-r hpvstrest/myubuntu` is the valid form for defining a repository. The image tag value must be specified in the `hpvs-config.yaml` file of each Hyper Protect container that are to be deployed into the repository.

The repository signing key already exists in appliance when running `repository create` command

You might notice the error message `CHZ00258E Invalid repository definition json file: the repo signing key exists in appliance already` when running the `repository create` command.

The problem might be caused by the repository signing key already exists in the Hyper Protect hosting appliance again, and for each repository to be registered on the Secure Service Container partition, you must use a unique key pair to encrypt the repository definition file.

To workaround the problem, consider the following procedure:

1. Generate a unique signing key pair by using the `regfile encrypt` command, and encrypt the repository definition file with this unique key pair.

   ```
   docker run --rm -v $(pwd):/regfile-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
   regfile encrypt -i isv_definition_template.json -ph [PASSPHRASE_PRIVATE_KEY] -kn
   <a_unique_key_name>
   ```

2. Run the `repository create` command again to register the repository on the Secure Service Container partition.

Signature validation failed when running `repository update` command

You might notice the error message `CHZ00258E Invalid repository definition json file: signature validation failed` when running the `repository update` command.

The problem might happen when you update a repository with an updated repository registration file, and the repository registration file is encrypted by a different signing key.

To workaround the problem, consider the following procedure:

1. Encrypt the repository definition file by using the same signing key for the repository. Note that one repository must be signed and validated by using a unique signing key.

   ```
   docker run --rm -v $(pwd):/regfile-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
   regfile encrypt -K same_key.private -k same_key.pub -ph [PASSPHRASE_PRIVATE_KEY] -i
   isv_definition_template.json
   ```

2. Run the `repository update` command again to update the repository on the Secure Service Container partition.

CHZ00106E Not enough disk space available when running hpvs create command

You might notice an error message similar to `CHZ00106E Not enough disk space available for file '/var/lib/quotagroups/lv_data_pool/democontainer_10001_root_qg/democontainer_10001/democontainer_10001_root_qg.raw'` when you run the `hpvs create` command.

The problem is caused by a known limitation that actual root quotagroup size available for a Hyper Protect Virtual Server container is always 1 GB less than the assigned size when creating the root quotagroup for the container.

To workaround the problem, complete the following instructions.

1. Remove all the orphan quotagroups by using the following command.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 \
quotagroup delete --all-orphan --lpar <lpar_ip>
```

2. Create the root quotagroup with a larger size for the container first, so that the Hyper Protect Virtual Server container will use this existing quotagroup when the container is being created again. For example, use 12 GB other than 10 GB to create the quotagroup `democontainer_10001_root_qg`.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 \
quotagroup create --name democontainer_10001_root_qg --size-unit GB --size 12 --lpar
<lpar_ip>
```

3. Create the Hyper Protect Virtual Server container again by using the `hpvs create` command.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 \
hpvs create --lpar <lpar_ip>
```

Hyper Protect Virtual Server instance restarting continuously when running hpvs get command

You might notice that the Hyper Protect Virtual Server container has been in the restarting state continuously if you run the `hpvs get` command.

The problem is most likely caused by the excessive memory setting assigned to the Hyper Protect Virtual Server container. The memory size allocated to the Hyper Protect Virtual Server container cannot exceed the available memory resource on the Secure Service Container partition.

To workaround the problem, consider one of the following options

- Ask the appliance or system administrator to allocate sufficient memory on the Secure Service Container partition before creating or updating the Hyper Protect Virtual Server container.
- Change the memory setting of the Hyper Protect Virtual Server container to a valid value, and then use the `hpvs update` command to update the container.

Container already exists when running monitoring create command

You might encounter an error saying that the container already exists when running the `monitoring create` command.

The problem is most likely caused by duplicate container configurations in the `monitoring.yaml` file.

To workaround the problem, remove the redundant configuration for the `monitoring-host` or `collectd-host` container from the `monitoring.yaml` file, and run the `monitoring create` command again.

IBM Hyper Protect Virtual Servers CLI tool fails to work after AppArmor upgrade on the management server

You might notice that IBM Hyper Protect Virtual Servers CLI tool does not work after you upgrade the Ubuntu operation system on the management server with the latest AppArmor updates.

The problem is most likely caused by the AppArmor updates, which confine the CLI tool to a limited set of resources. For more information, see apparmor man page.

To workaround the problem, you must disable the AppArmor and restart it on the management server.

```
# /etc/init.d/apparmor stop
# /etc/init.d/apparmor restart
```

Troubleshooting Secure Service Container for IBM Cloud Private

When you run into problems with Secure Service Container for IBM Cloud Private, you can refer to the following information:

- For Secure Service Container partitions, use the Secure Service Container user interface to view the logs.
- For Secure Service Container for IBM Cloud Private command line tool, go to the `/config/logs` directory on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server to view the logs.
- For IBM Cloud Private, see the troubleshooting and support topic of IBM Cloud Private.
- For the issues that you might encounter during the upgrade or rollback of Secure Service Container for IBM Cloud Private, see the Troubleshooting upgrade or rollback.

You can refer to the following information if you have problems using the Secure Service Container for IBM Cloud Private offering.

Secure Service Container for IBM Cloud Private command line tool failed when configuring IBM Cloud Private nodes

When you run the Secure Service Container for IBM Cloud Private command line tool to create IBM Cloud Private nodes, you might encounter some failures and the procedure cannot continue.

To resolve this issue, you can take the following steps on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server:

1. Remove the `config/cluster-status.yaml` file manually.
2. Run the uninstall command to clean up the stale entries from the failed installation.

   ```
   docker run --network=host --rm -it -v $(pwd)/config:/ssc4icp-cli-installer/config
   ibmzcontainers/hpvs-cli-installer:1.2.0 uninstall
   ```

3. Run the command line tool again as instructed in the Creating IBM Cloud Private nodes by using the command line tool topic.

IBM Cloud Private installation failed with the host unresolved error

During the IBM Cloud Private installation, you might encounter the following error message:

```
TASK [check : Validating Hostname is resolvable] ********************************
fatal: [192.168.60.3]: FAILED! => {"changed": false, "msg": "Please configure your hostname to
resolve to an externally reachable IP"}
```

This error occurs because the hostname of each IBM Cloud Private node cannot be connected during the IBM Cloud Private installation. You have to log in to each IBM Cloud Private node by using the SSH utility and update the `/etc/hosts` file by following the instructions on Configuring your cluster.

HTTP request error when accessing the IBM Cloud Private console right after the installation

If you access the IBM Cloud Private cluster console immediately after the IBM Cloud Private installation completes, you might see an error like the following in your browser:

```
Error making request: Error: getaddrinfo EAI_AGAIN platform-identity-provider:4300 POST http://
platform-identity-provider:4300/v1/auth/token HTTP/1.1 Accept: application/json Content-Type:
application/json {"client_id":"34ddcf35f1ef7f42a23678feb8b96e8b","client_secret":
"56dbb5f1a355082692fca4b91a7a8eea","code":"WAAvWJXE8sEVUWYhERlwedeC3xA0Jv",
"redirect_uri":"https://10.152.150.205:8443/auth/liberty/
callback","grant_type":"authorization_code","scope":"openid email profile"} Error: getaddrinfo
EAI_AGAIN platform-identity-provider:4300
```

This error occurs because the UI services of IBM Cloud Private is not fully up and running, and it takes some time to be ready. Therefore, you can wait a few minutes after the IBM Cloud Private installation completes, then try accessing the console again.

"500 internal server error" when accessing the IBM Cloud Private master node

After reinstalling an IBM Cloud Private cluster and then attempting to access the IBM Cloud Private master node using port 8443, the error message "500 internal server error" might be displayed in your browser.

To resolve this issue, You can try one of the following options as a workaround.

- Clear the cache of the current web browser session, then open a new private web browser window for the same web browser type (for example, Mozilla Firefox) and access the IBM Cloud Private master node URL with port 8443.
- Open a new web browser session on a different web browser type and then enter the IBM Cloud Private master node URL with port 8443. For example, if you encounter "500 internal server error" message on a Mozilla Firefox window, open a new web browser window using Microsoft Edge.

An IBM Cloud Private node on the recycled Secure Service Container partition cannot ping all of the other IBM Cloud Private nodes in the IBM Cloud Private cluster using its network interface

After a Secure Service Container partition recycle (including a CEC recycle), you might notice that any IBM Cloud Private node on this Secure Service Container partition cannot connect to other IBM Cloud Private nodes by using the IP addresses.

To resolve this issue, you must restart each IBM Cloud Private node on this Secure Service Container partition with the following command once the Secure Service Container partition is reactivated and back online.

```
shutdown -r now
```

Worker or proxy nodes stop responding after the cluster has been running smoothly for a while or restarted

After the IBM Cloud Private cluster has been running for a while or you restart the cluster nodes for some reason, you might notice that the nodes on the Secure Service Container partition stops responding.

To resolve the issue, you can restart the cluster nodes on each Secure Service Container partition by using REST APIs. See the Secure Service Container for IBM Cloud Private System APIs for a full list of REST API endpoints.

1. Generate the access token to the Secure Service Container by using the following command.

```
curl --request POST --url https://<appliance_IP>/api/com.ibm.zaci.system/api-tokens \
-H 'accept: application/vnd.ibm.zaci.payload+json' -H 'cache-control: no-cache' \
-H 'content-type: application/vnd.ibm.zaci.payload+json;version=1.0'
-H 'zaci-api: com.ibm.zaci.system/1.0' --insecure \
--data '{ "kind" : "request", "parameters" : { "user" : "<master_id>", "password" :
"master_id_password" } }'
```

   Where:

   - **appliance_IP** is the Secure Service Container IP address.
   - **master_id** is the `Master user ID` in the image profile (standard mode system) or the partition definition (DPM-enabled system) for the Secure Service Container partition.
   - **master_id_password** is the `Master password` in the same profile or definition for the partition.

2. Restart each cluster node on the Secure Service Container partition by using the following command.

```
curl -X POST https://<appliance_IP>/api/com.ibm.zaas/containers/<node_name>/restart \
-H 'accept: application/vnd.ibm.zaci.payload+json' -H 'cache-control: no-cache' \
-H 'content-type: application/vnd.ibm.zaci.payload+json;version=1.0' \
-H 'zaci-api: com.ibm.zaci.system/1.0' \
-H 'authorization: Bearer '<TOKEN>'' --insecure
```

   Where:

   - **appliance_IP** is the Secure Service Container IP address.
   - **TOKEN** is the access token from the previous step.
   - **node_name** is the name of worker or proxy node that you have to restart. You can get the node name from the `config/cluster-configuration.yaml` file on the x86 or Linux on IBM Z/ LinuxONE (such as s390x architecture) management server. For example, `worker-15001` is the name of one worker node.

Gateway on the proxy node is not configured automatically after the CLI installation

After the Secure Service Container for IBM Cloud Private CLI installation completes, you might notice that the gateway on the proxy node is not added even though the gateway was specified in the `config/ssc4icp-config.yaml` file. For example,

```
...
proxyIPConfig:
   - ipaddress: "172.16.0.4"
     subnet: "172.16.0.0/24"
     gateway: "172.16.0.1"
     parent: "vxlan0f300.1121"
...
```

To resolve the issue, you can manually add the gateway into the routing table on the proxy node by using the following command:

```
route add -net <target_subnet> gw <gateway_IP_address>
```

Where:

- `target_subnet` is the destination/target IP subnet you wish to establish connectivity with. For example, where the client laptop is located. The value is `172.16.0.0/24` as indicated in the `ssc4icp-config.yaml` example file.
- `gateway_IP_address` is the gateway currently accessible from the Proxy node that provides this routing connectivity. The value is `172.16.0.1` as indicated in the `ssc4icp-config.yaml` example file.

DSN Server loopback issue during the IBM Cloud Private installation

During the installation of IBM Cloud Private, you might encounter an error when the IBM Cloud Private installer is validating the DNS server with the following message.

```
fatal: [x.x.x.x] => A loopback IP is used in your DNS server configuration. For more details,
see https://ibm.biz/dns-fails.
```

The problem might be caused by the loopback IP (127.0.0.1 or 127.0.1.1) as the DNS server. Or, the cluster node that is specified in the error message does not have a `/etc/resolv.conf` file.

To resolve the issue, you can set the master node IP addresses as the name server in the `/etc/resolve.conf` file on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server. For example,

```
...
nameserver 192.168.15.240
nameserver 10.152.151.100
#nameserver 127.0.0.53
...
```

False error messages about the storage requirements when installing the IBM Cloud Private

When you install the IBM Cloud Private, you might see the following error messages about the disk space validation.

```
TASK [check : Validating /var directory disk space on worker, proxy and custom nodes]
*********************************************************************************
fatal: [192.168.19.252]: FAILED! => changed=true
cmd: |
disk=$(df -lh -BG --output=avail /var | sed '1d' | grep -oP '\d+')
[[ $disk -ge 110 ]] || (echo "/var directory available disk space ${disk}GB, it should be
greater than or equal to 110 GB" 1>&2 && exit 1)
...
stdout_lines: <omitted>
...ignoring
```

If you have allocated sufficient disk space in the `ssc4icp-config.yaml` file when creating the cluster nodes, the message can be safely ignored because this is a known issue of IBM Cloud Private.

The Catalog page is empty after the IBM Cloud Private cluster is started

After you log into the IBM Cloud Private console, you might notice that the **Catalog** page is empty and an error message **Error loading charts** pops up. If you check the logs of the DNS pod by using the command `kubectl logs <dns_pod_name> -n kube-system`, you might see the following errors in the logs. Note that <dns_pod_name> is the name of DNS pod.

```
2018/12/04 15:19:20 [ERROR] 2 ... A: unreachable backend: read udp 127.0.0.1:40613-
>127.0.0.53:53: i/o timeout
2018/12/04 15:19:20 [ERROR] 2 ... A: unreachable backend: read udp 127.0.0.1:54930-
>127.0.0.53:53: i/o timeout
2018/12/04 15:19:20 [ERROR] 2 ... A: unreachable backend: read udp 127.0.0.1:54521-
>127.0.0.53:53: i/o timeout
2018/12/04 15:19:20 [ERROR] 2 ... A:  unreachable backend: read udp 127.0.0.1:45653-
>127.0.0.53:53: i/o timeout
2018/12/04 15:19:20 [ERROR] 2 ... A:  unreachable backend: read udp 127.0.0.1:60436-
>127.0.0.53:53: i/o timeout
```

The problem might be caused by the DNS in the kubernetes not being resolved correctly.

To resolve issue, try the following steps on the x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.

1. Open the `/etc/systemd/system/kubelet.service` file, and add the `--resolv-conf=/run/systemd/resolve/resolv.conf \` line into the file.
2. Restart the `docker` and `kubelet` services.

3. Run the command `kubectl get pods` to ensure all the pods are in the **Running** state, and then log into the IBM Cloud Private console.
4. Click **Helm Repositories** option on the **Manage** Menu, and then click the **Sync repositories** button.
5. Go to the **Catalog** page to check if all the helm charts are available.

Actions to perform after the restart of the Secure Service Container for IBM Cloud Private components

The Secure Service Container for IBM Cloud Private consists of different components and you might need to perform specific actions on each type of components after the component is restarted.

- After a master node is restarted, follow those instructions to ensure the connectivity among cluster nodes.
  1. If the master node is configured by using the ethernet interface, run this command to ensure the connectivity to other cluster nodes.

     ```
     ip addr add 192.168.0.251/24 dev eth0
     ```

  2. If the master node is configured by using the VLAN interface, run these commands to ensure the connectivity to other cluster nodes.

     ```
     ip link add link ens224 name ens224.1121 type vlan id 1121
     ip addr add 192.168.0.251/24 dev ens224.1121
     ip link set up ens224.1121
     ```

  3. Check the IPSec service status by using this command.

     ```
     service strongswan status
     ```

  4. If the IPSec service is not running, enable the service by using this command.

     ```
     service strongswan start
     ```

- If the cluster nodes are in a layer 3 network, run the command `ip route add 10.162.161.0/24 via 10.152.151.1` on the proxy node to configure the connectivity to the master node.
- After a Secure Service Container partition is restarted, you have to run all the required commands on each worker or proxy node that are hosted on this partition.

"502 Bad Gateway" when accessing the application on IBM Cloud Private v3.1.2

After you deploy the application on the IBM Cloud Private version 3.1.2, you might notice the "502 Bad Gateway" error message when accessing the application. After checking the logs, you might see similars error messages from the `ngins-ingress-controller` pod.

```
2019/03/27 07:11:29 [error] 58#58: *48168 upstream prematurely closed connection while reading
response header from upstream, client: 127.0.0.1, server: _, request: "GET /favicon.ico HTTP/
2.0", upstream: "http://10.1.125.134:9443/favicon.ico", host: "9.20.36.107", referrer: "https://
9.20.36.107/"
2019/03/27 07:11:29 [error] 58#58: *48168 upstream prematurely closed connection while reading
response header from upstream, client: 127.0.0.1, server: _, request: "GET /favicon.ico HTTP/
2.0", upstream: "http://10.1.125.134:9443/favicon.ico", host: "9.20.36.107", referrer: "https://
9.20.36.107/"
```

The problem is caused by a new prefix in the NGINX ingress version 0.9.0, which is part of IBM Cloud Private version 3.1.2 `nginx.ingress.kubernetes.io`. You can refer to Enable Ingress Controller to use a new annotation prefix for the details.

To resolve the problem, try the following steps:

1. Apply the workaround as described in the Enable Ingress Controller to use a new annotation prefix.

2. Check the `ingress` pod of the application to ensure `nginx.ingress.kubernetes.io/backend-protocol: HTTPS` annotation is available by using the following command.

   ```
   kubectl get ingress oldone-ibm-open-liberty -o yaml
   ```

3. If the annotation is not displayed in the command result, add the annotation into the ingress deployment by using the following command.

   ```
   kubectl edit ingress oldone-ibm-open-liberty -o yaml
   ```

4. Access the application again.

OCI runtime error after GlusterFS node restarted

After you restart one of the GlusterFS node, you might encounter the OCI runtime error with the glusterfs daemon. For example, run the command `kubectl -n kube-system exec storage-glusterfs-glusterfs-daemonset-8n9g6 -- ll /var/lib/glusterd`.

The error message might be as the following:

```
OCI runtime exec failed: exec failed: container_linux.go:348: starting container process caused
"exec: \"ll\": executable file not found in $PATH": unknown
command terminated with exit code 126
```

The problem is caused by missing configurations in the glusterfs daemonset.

To resolve the problem, complete the following tasks on the master node server.

1. Retrieve the glusterfs daemonset name.

   ```
   kubectl --all-namespaces get daemonset | grep glusterfs
   ```

2. Open the daemonset editor by using the glusterfs daemonset name. For example,

   ```
   kubectl edit daemonset <gluster_daemonset_name> --all-namespaces
   ```

3. Apply the following changes for the daemonset.
4. Update the `livenessProbe` and `readinessProbe` sections with the following configuration.

   ```
   failureThreshold: 50
   initialDelaySeconds: 40
   periodSeconds: 25
   successThreshold: 1
   timeoutSeconds: 3
   ```

5. Add the following lines into the `volumeMounts` section.

   ```
   - mountPath: /lib/modules
     name: kernel-modules
     readOnly: true
   - mountPath: /etc/ssl
     name: glusterfs-ssl
     readOnly: true
   ```

6. Add the following lines into the `volumes` section.

   ```
   - name: glusterfs-ssl
     hostPath:
       path: "/etc/ssl"
   - name: kernel-modules
     hostPath:
       path: "/lib/modules"
   ```

7. Save the changes and exit.

The changes will be applied automatically. By monitoring the output of the command `kubectl -n kube-system get po -o wide|grep glusterfs`, you can see glsuterfs daemonset pods terminating and coming up gradually.

Troubleshooting upgrade and rollback

When you run into problems during the upgrade or rollback of Secure Service Container for IBM Cloud Private, you can refer to the following information:

Operation failed when cordoning the cluster nodes.

When you run the command to cordon the cluster nodes as instructed in the Planning the upgrade, you might encounter the following error message:

```
Error from server: Operation cannot be fulfilled on nodes "<node-address>": the object has been
modified; please apply your changes to the latest version and try again
```

This error occurs because the node status is not update properly when the `kubectl cordon` command is executed. You can try the command again to fix the problem. If the problem persists, see kubernetes documentation to verify the prerequisites.

Errors occurred when draining the cluster nodes

When you run the command to drain the cluster nodes as instructed in the Planning the upgrade, you might notice the command hangs as it waits for all the nodes to be terminated.

This problem might occur when some pods are stuck in the `terminating` state.

To resolve the problem, take the following steps:

1. Check whether some pods are stuck in the `terminating` state by using the following command syntax.

```
kubectl get pods -o wide --all-namespaces | grep <node_name>
```

1. Delete such pods manually by using the following command syntax.

```
kubectl delete pod <pod_name> -n=<namespace> --grace-period=0 --force
```

Errors occurred when stopping the cluster nodes

You might encounter problems when stopping the cluster nodes as instructed in the Upgrading the Secure Service Container for IBM Cloud Private CLI tool. An error message might indicate the problem is from the Secure Service Container for IBM Cloud Private appliance. The problem might be due to the following reasons such as:

- API authentication failure with the Secure Service Container partition
- Internal errors from the appliance API
- LPAR network outage
- Lost containers on the cluster nodes

To resolve the problem, you can take either of the following options:

- Restart all the containers on the cluster nodes by using the Secure Service Container for IBM Cloud Private CLI tool with the following command. And then try the stop command again.

```
docker run --network=host --rm -it -v $(pwd)/config:/ssc4icp-cli-installer/config
ibmzcontainers/hpvs-cli-installer:1.1.2 start
```

- Manually invoke the `stop` REST API against the problematic container with the `curl` command. For more information, see `stop` API.

Errors occurred when exporting or importing the appliance data

You might encounter problems when exporting or importing the appliance data from the Secure Service Container user interface. The problem might be due to the following reasons such as:

- API authentication failure with the Secure Service Container partition

- Internal errors from the appliance API

To resolve the problem, take the following steps:

1. Use the Secure Service Container user interface to view the logs.
2. Refer to the **Appendix A. Codes from the Secure Service Container installer** section after you download Secure Service Container User's Guide from the About topic to verify that the problem is not an internal server error, and then try to fix the problem.
3. If the problem persists, contact IBM support.

Errors occurred when upgrading the Secure Service Container for IBM Cloud Private appliance

You might encounter problems when following the instructions in the Upgrading the Secure Service Container for IBM Cloud Private appliance. The problem might be due to the following reasons such as:

- API authentication failure with the Secure Service Container partition
- Internal errors from the appliance API

If you want to continue with the upgrade, Contact IBM support. Or you can install the previous version of the appliance, and import the configuration that was backed up during the upgrade procedure. For more information, See **Exporting or importing appliance configuration data** section after you download Secure Service Container User's Guide from the About topic.

Errors occurred when upgrading the Secure Service Container for IBM Cloud Private CLI tool

You might encounter problems when following the instructions in the Upgrading the Secure Service Container for IBM Cloud Private CLI tool. The problem might be due to issues from different aspects such as:

- Storage
- Permission
- Docker runtime environment

If you want to continue with the upgrade, Contact IBM support. Or you can reverting the CLI tool to a pervious version by following the instructions in the Rollback CLI installer section.

Errors occurred when upgrading the isolated VM

You might encounter problems when following the instructions in the Upgrading cluster nodes with the isolated VM. The problem might be due to issues from different aspects such as:

- Docker runtime environment
- Image packaging or naming
- Supported version

If you want to rollback the upgrade, follow the instructions in the Reverting the Secure Service Container for IBM Cloud Private. If you need further help, Contact IBM support.

Errors occurred when upgrading the IBM Cloud Private

You might encounter problems when following the instructions in the Upgrading the IBM Cloud Private. The problem might be due to issues from different aspects such as:

- Docker runtime environment
- Image package or naming
- Supported Version
- Kubernetes
- Storage
- Permission

To resolve such problems, you can try the following options:

- Refer to the **UPGRADE FAILED error** section in the Troubleshooting Key Management Service topic of IBM Cloud Private document.
- Revert your upgraded IBM Cloud Private as instructed in the Reverting to an earlier version of IBM Cloud Private.
- Revert your upgraded Secure Service Container for IBM Cloud Private as instructed in the Reverting the Secure Service Container for IBM Cloud Private.

If the problem persists, Contact IBM support.

Frequently asked questions

FAQs about Secure Service Container for IBM Cloud Private are listed here.

Architecture limits

How many IBM Cloud Private nodes are permitted inside a single Secure Service Container partition?

You can have three to five worker nodes on each LPAR and one IBM Cloud Private proxy node on each Secure Service Container partition.

When a Secure Service Container partition is established and in use, can additional IBM Cloud Private nodes be added later to the existing Secure Service Container partition?

No. This is not currently supported.

Can IBM Cloud Private nodes be removed from a Secure Service Container partition without impacting other IBM Cloud Private nodes running in the Secure Service Container partition?

Yes. Removing an IBM Cloud Private node does not impact other IBM Cloud Private nodes. IBM Cloud Private can continue to run without the other active nodes.

Will the process of creating a Secure Service Container partition and installing IBM Cloud Private be programatically scripted for automation?

The Secure Service Container partition has to be setup manually. You can refer to the following chapters after you download Secure Service Container User's Guide from the About topic.

- Chapter 3 - Configuring a Secure Service Container partition on a standard mode system
- Chapter 4 - Starting a Secure Service Container partition on a standard mode systems
- Chapter 8 - Configuring a Secure Service Container partition on a DPM-enabled system
- Chapter 9 - Starting a Secure Service Container partition on a DPM-enabled system

IBM Cloud private

Can multiple master nodes be deployed to one IBM Cloud Private cluster?

No. This is not currently supported.

GDPS

Does Secure Service Container integrate with GDPS?

No. Geographically Dispersed Parallel Sysplex (GDPS) is not currently supported..

Updates

Does the customer need to patch Linux or the Hypervisor in a Secure Service Container?

No. The Secure Service Container for IBM Cloud Private offering has its own embedded Linux distribution and hypervisor. Both are inaccessible to the user and are updated as part of offering updates. The customer does not have to manage those components separately.

Security

Can a customer integrate their Crypto Express Card with the Secure Service Container?

No. This is not currently supported.

Can a customer create their own encryption algorithms?

You can set up any custom encryption or certification within your own docker workload. However, you cannot change the encryption algorithms used for storage and data transport in the offering. See Encryption algorithms for more information.

Is Multi-Factor Authentication supported with Secure Service Container?

No. This is not currently supported.

Can a customer integrate their own certificates with Secure Service Container?

No. This is not currently supported.

Are there security certifications supported for Secure Service Container?

The Secure Service Container is deployed in a LPAR on the IBM Z or LinuxONE architecture, and both architectures are rated up to EAL5+ for isolation. Security certifications are intent to be pursued in the future based on customer and industry demand.

Database

Are Oracle databases supported in a Secure Service Container environment?

No. Oracle databases are not currently supported. You can use IBM Db2, PostgreSQL, and other open source databases in the environment.

References

Refer to the following topics when you use IBM Hyper Protect Virtual Servers.

- File and directory structure of IBM Hyper Protect Virtual Servers
- Commands
- Configuration files
- Others
- Hyper Protect Hosting Appliance REST APIs

File and directory structure of IBM Hyper Protect Virtual Servers

After you download and extract the `CC37UEN.tar` image file, you can see the similar layout of files and directories under the `<installation_directory>` directory.

For more information about how to get the IBM Hyper Protect Virtual Servers image file, see Downloading the installation package.

```
|- CC37UEN.tar
|- hpvs-cli-installer.docker-image.tar
|- License
|- readme.txt
|- secure-service-container-for-icp.appliance.1.1.18.img.gz
|- SSC4ICP
|    |-config
|    |    |- ICPIsolatedvm.tar.gz
|    |- readme.txt
|- version
|- VS
     |- grep11-cli
     |    |- config
     |        |-grep11-config.yaml.example
     |        |-hosts.example
     |        |-hpcsKpGrep11_runq.tar.gz
     |- hpvs-cli
     |    |- config
```

```
    |       |- hosts.example
    |       |- hpvs-config.yaml.example
    |       |- hpvs-config.yaml.example.VLAN.L2
    |       |- hpvs-config.yaml.example.VLAN.L3
    |       |- HpvsopBaseSSH.tar.gz
    |       |- HpvsopBase.tar.gz
    |- monitoring-cli
    |  |-config
    |       |- CollectdHost.tar.gz
    |       |- hosts.example
    |       |- monitoring-configuration-delete.yaml.example
    |       |- monitoring-configuration-service.yaml.example
    |       |- monitoring-configuration.yaml.example
    |       |- Monitoring.tar.gz
    |       |- monitoring.yaml.example
    |- readme.txt
    |- regfile-cli
    |   |- config
    |- securebuild-cli
        |- config
           |- hosts.example
           |- hosts.readme
           |- securebuild.yaml.example
           |- securebuild.yaml.example.VLAN.L2
           |- securebuild.yaml.example.VLAN.L3
           |- securebuild.yaml.readme
           |- SecureDockerBuild.tar.gz
```

Note:

- `readme.txt`, which is the general README file for IBM Hyper Protect Virtual Servers .
- `License`, a directory that contains the license files of IBM Hyper Protect Virtual Servers .
- `version`, which states the current version of IBM Hyper Protect Virtual Servers .
- `hpvs-cli-installer.docker-image.tar`, which contains the command line tools required to set up IBM Hyper Protect Virtual Servers environment, or set up the Secure Service Container for IBM Cloud Private. For the details about the Secure Service Container for IBM Cloud Private, see Working with Secure Service Container for IBM Cloud Private.
- `secure-service-container-for-icp.appliance.1.1.18.img.gz`, which is the hosting appliance to be installed on the IBM Z or LinuxONE system. Note that this hosting appliance image is identical with Hyper Protect hosting appliance version v3.7.7.
- `./VS/grep11-cli/config`, a directory that contains configuration example of `hosts` and `grep11-config.yaml` files.
- `./VS/grep11-cli/config/hpcsKpGrep11_runq.tar.gz`, which is the base image of the GREP11 container and must be under the `./VS/grep11-cli/config`.
- `./VS/hpvs-cli/config/HpvsopBase.tar.gz`, which is the base image of a Hyper Protect Virtual Server container without the secure shell (SSH) access.
- `./VS/hpvs-cli/config/HpvsopBaseSSH.tar.gz`, which is the base image of a Hyper Protect Virtual Server container with the secure shell (SSH) access.
- `./VS/hpvs-cli/config`, a directory that contains configuration example of `hosts` and `hpvs-config.yaml` files for Hyper Protect Virtual Server containers.
- `./VS/monitoring-cli/config`, a directory that contains configuration example of `hosts` and `monitoring.yaml` files for the monitoring infrastructure.
- `./VS/monitoring-cli/config/CollectdHost.tar.gz`, which is the base image of collectd-host container of the monitoring infrastructure. The file must be under the `./VS/monitoring-cli/config` directory.
- `./VS/monitoring-cli/config/Monitoring.tar.gz`, which is the base image of monitoring-host container of the monitoring infrastructure. The file must be under the `./VS/monitoring-cli/config` directory.
- `./VS/regfile-cli/config`, which is an empty directory upon initial extract, and can be used to contain configuration files for repository definition files.

- `./VS/securebuild-cli/config/SecureDockerBuild.tar.gz`, which is the docker image of the Secure Build container.
- `./VS/securebuild-cli/config`, a directory that contains configuration example files (with the `.example` extension) and readme files (with the `.readme` extension) of the `hosts` and `securebuild.yaml` files for Secure Build containers.
- `./SSC4ICP/config`, a directory that contains configuration example files for IBM Secure Service Container for IBM Cloud Private.
- `./SSC4ICP/config/ICPIsolatedvm.tar.gz`, which is the isolated VM image for hosting proxy and worker nodes of Secure Service Container for IBM Cloud Private. For more details, see Working with Secure Service Container for IBM Cloud Private.

Commands

The following topics list each command and the usage for IBM Hyper Protect Virtual Servers.

- Commands for crypto domains
- Commands for disks
- Commands for GREP11 containers
- Commands for Hyper Protect Virtual Server containers
- Commands for images
- Commands for quotagroups
- Commands for monitoring
- Commands for repositories
- Commands for repository registration files
- Commands for snapshots
- Commands for Secure Build containers

Commands for crypto domains

Commands to check the crypto domains on the HSM take the form as in the following example. Note these commands require root user authority.

```
docker run --rm -it -v $(pwd):/grep11-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 crypto
list
```

Where:

- `crypto` is the `SUBJECT`,
- `list` is the `ACTION`

Table 1. The full list of the `SUBJECT`, `ACTION` and `Parameters` for `crypto` commands

subject	action	parameter name	sample value	description
crypto	list		N/A	List all the crypto domains on the HSM

Commands for disks

Commands to manage disks of the storage pool on the Secure Service Container partition take the form as in the following example. Note these commands require root user authority.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 disk get
--lpar [LPARIPORADDRESS]
```

Where:

- `disk` is the `SUBJECT`,
- `get` is the `ACTION`
- `-lpar` is the extra parameter.

Table 1. The full list of the `SUBJECT`, `ACTION` and `Parameters` for `disk` commands

subject	action	parameter name	sample value	description
disk	N/A	--help, -h	N/A	List all command parameters
disk	get, list			Get the disks added to the Secure Service Container partition `LV Data Pool` storage pool
		--lpar	SSC_LPAR_NAME	The LPAR name or address to run this command
disk	add			Add disks to the the Secure Service Container partition `LV Data Pool` storage pool
		--sub-resource	storage-devices	Identify the resource type to be added to the pool, which can be either 'storage-devices' (to represent a DASD) or 'fcp-disks' (to represent a SCSI disk/LUN)
		--disk-id	diskid1 diskid2	A list of disk identifiers with the whitespace separator, which will be added to the storage pool
		--lpar	SSC_LPAR_NAME	The LPAR name or address to run this command

For more command examples, see Configuring the storage on the Secure Service Container partition.

Commands for GREP11

Commands to manage GREP11 containers on the Secure Service Container Host Appliance take the form as in the following example. Note these commands require root user authority.

```
docker run --rm -it -v $(pwd):/grep11-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 grep11
create
```

Where:

- `grep11` is the `SUBJECT`,
- `create` is the `ACTION`

Table 1. The full list of the `SUBJECT`, `ACTION` and `Parameters` for `grep11` commands

subject	action	parameter name	sample value	description
grep11	create		N/A	Create a GREP11 container
grep11	start			Start the GREP11 container
grep11	delete			Delete the GREP11 container
grep11	stop			Stop the GREP11 container
grep11	status			Check the status of the GREP11 container

Commands for Hyper Protect Virtual Server containers

Commands to manage Hyper Protect Virtual Server containers take the form as in the following examples. Note these commands require root user authority.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 hpvs
create --name hpvs1
```

Where:

- `hpvs` is the `SUBJECT`,
- `create` is the `ACTION`
- `--name  hpvs1` is extra parameters.

Table 1. The full list of the `SUBJECT`, `ACTION` and `Parameters` for `hpvs` commands

SUBJECT	ACTION	Parameter name	Sample value	Description
hpvs	N/A	--help, -h	N/A	List all command parameters
hpvs	create			create one or multiple Hyper Protect Virtual Server
		-n, --name	myvs	Container name
		-r, --repoid	HpvsopBaseSSH	Registered id of repository to create container from
		-t, --imagetag	latest	Image tag to create container from
		--cpu	4	Integer value x with 0 < x <= # of CPU's in the system

SUBJECT	ACTION	Parameter name	Sample value	Description
		--memory	4096	Integer value in megabytes x with x > 0
		--domainname	somecompany.co m	String with a valid domain name
		--hostname	hpvs.somecompan y.com	String with a valid hostname for the container
		--ports	'{"1234":"4321"}'	Dict of ports to forward to the container
		--labels	{"arch":"s390x"}	A JSON object of name-value labels
		--extra-hosts		Array with hostname:IP strings
		--log-type	json-file	Indicate which log driver to use
		--log-config	{"max-size":"10m"}	A driver-dependent configuration dictionary
		--quotagroup-storage	10G	Storage size with unit of the quotagroup object. Set to '0G' to ignore mounting quotagroup
		--container-networks	'[{"network_name" :"your-network-name","ip_address ":}]'	The network info for Hyper Protect Virtual Server. If the network doesn't exist, it will create the network according to template definition in hpvs-config.yaml
		--mount	/data	Mount point inside the container where the quotagroup is mounted
		--filesystem	raw	filesystem used for the mount inside the container, can be any string out of

SUBJECT	ACTION	Parameter name	Sample value	Description
		--quotagroup	qg-test	Specify existing quotagroup for mounting quotagroup
		--env-file	env.json	The JSON file name containing environment variables that will be set to container. For more information, see hpvs-env.json.
		--template	template2	Template name used to create Hyper Protect Virtual Server
		--lpar	SSC_LPAR_NAME	LPAR name or ipaddress
hpvs	delete			delete one or multiple Hyper Protect Virtual Server
		--name	myvs	Container name
		--names	myvs1,myvs2	Container names, seperated with comma
		--quotagroup	qg-test	Specify the quotagroup that will be deleted after VS deletion
		--quotagroup-keep	N/A	If true, keep the quotagroup after VS deletion
		--lpar	SSC_LPAR_NAME	LPAR name or ipaddress
hpvs	list,get			list multiple Hyper Protect Virtual Server containers summary info or list detail of one Hyper Protect Virtual Server
		--name	myvs	Container name.
		--lpar	SSC_LPAR_NAME	LPAR name or ipaddress

SUBJECT	ACTION	Parameter name	Sample value	Description
		--lpar_index	• 2-4 • 2,4	• `[STARTING_INDEX_NO]-[ENDING_INDEX_NO]`: Specify the partition index range as defined in the `hosts` file. For example, `2-4` • `[LPAR_INDEX_NO1], [LPAR_INDEX_NO2]`: Specify the comma separated partition indexes as defined in the `hosts` file. For example, `2,4`
hpvs	start			start a Hyper Protect Virtual Server
		--name	vs1	Container name
		--lpar	SSC_LPAR_NAME	LPAR name or ipaddress
hpvs	stop			start a Hyper Protect Virtual Server
		--name	vs1	Container name
		--lpar	SSC_LPAR_NAME	LPAR name or ipaddress
hpvs	restart			start a Hyper Protect Virtual Server
		--name	vs1	Container name
		--lpar	SSC_LPAR_NAME	LPAR name or ipaddress
hpvs	update			update one or multiple Hyper Protect Virtual Server
		-n, --name	myvs	Container name
		-r, --repoid	HpvsopBaseSSH	Registered id of repository to create container from
		-t, --imagetag	latest	Image tag to create container from

SUBJECT	ACTION	Parameter name	Sample value	Description
		--cpu	4	Integer value x with 0 < x <= # of CPU's in the system
		--memory	4096	Integer value in megabytes x with x > 0
		--domainname	somecompony.com	String with a valid domain name
		--hostname	hpvs.somecompony.com	String with a valid hostname for the container
		--ports	'{"1234":"4321"}'	Dict of ports to forward to the container
		--labels	{"arch":"s390x"}	A JSON object of name-value labels
		--extra-hosts		Array with hostname:IP strings
		--log-type	json-file	Indicate which log driver to use
		--log-config	{"max-size":"10m"}	A driver-dependent configuration dictionary
		--quotagroup-storage	10G	Storage size with unit of the quotagroup object. Set to '0G' to ignore mounting quotagroup
		--container-networks	'[{"network_name":"your-network-name","ip_address":}]'	The network info for Hyper Protect Virtual Server. If the network doesn't exist, it will create the network according to template definition in hpvs-config.yaml
		--mount	/data	Mount point inside the container where the quotagroup is mounted

SUBJECT	ACTION	Parameter name	Sample value	Description
		--filesystem	raw	filesystem used for the mount inside the container, can be any string out of
		--quotagroup	qg-test	Specify existing quotagroup for mounting quotagroup
		--env-file	env.json	The JSON file name containing environment variables that will be set to container. See hpvs-env.json.
		--template	template2	Template name used to create Hyper Protect Virtual Server
		--imagetag	newtag	Image tag of a different Hyper Protect Virtual Server base image
		--lpar	SSC_LPAR_NAME	LPAR name or ipaddress

Example

The following example shows the usage of the `ports` parameter with the `hpvs create` command. The `80` port of the `testhpvs-10001` container is mapped to `8080` port on the Secure Service Container partition `10.152.151.105`.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 hpvs
create --repoid TestNginxDemo --name testhpvs-10001 --lpar 10.152.151.105 --ports
'{"80":"8080"}'
```

Commands for images

Commands to manage images on the Secure Service Container Host Appliance take the form as in the following example. Note these commands require root user authority.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 image
load -i ${IMAGE}
```

Where:

- `image` is the `SUBJECT`,
- `load` is the `ACTION`
- `-i` is extra parameters.

Table 1. The full list of the `SUBJECT`, `ACTION` and `Parameters` for `image` commands

subject	action	parameter name	sample value	description
image	N/A	--help, -h	N/A	List all command parameters
image	load			Load an image into Host Appliance
		--image-bundle-path, -i	./temp/SecureDockerBuild.tar.gz	The image full path
		--lpar	SSC_LPAR_NAME	The LPAR name to run this command. will run the command on all LPARs configured if the parameter not provided.
image	list,get			List images on the LPAR
		--repoid	SecureDockerBuild	Docker repository ID on the LPAR
		--lpar	SSC_LPAR_NAME	The LPAR name to run this command

Note: To unload an image from the Secure Service Container partition, use the `repository delete` command to delete the image and its repository from the partition. For more information, see Commands for repositories.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
repository delete --repoid SecureDockerBuild --lpar [LPARNAMEORADDRESS] --force
```

Commands for quotagroups

Commands to manage quotagroups used by a Hyper Protect Virtual Server container take the form as in the following example. Note these commands require root user authority.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
quotagroup list --all --name-only
```

Where:

- `quotagroup` is the `SUBJECT`,
- `list` is the `ACTION`
- `--all` and `-name-only` are extra parameters.

Table 1. The full list of the `SUBJECT`, `ACTION` and `Parameters` for `quotagroup` commands

SUBJECT	ACTION	Parameter name	Sample value	Description
quotagroup	N/A	--help, -h	N/A	List all command parameters
quotagroup	list,get			list quotagroups
		--name-only	N/A	Only list quotagroup names

SUBJECT	ACTION	Parameter name	Sample value	Description
		--all-orphan	N/A	List all quotagroup not used by any Hyper Protect Virtual Server containers
		--all	N/A	List all quotagroup no matter used or not by Hyper Protect Virtual Server containers
		--lpar	SSC_LPAR_NAME	The LPAR name or address to run this command. will run the command on all LPARs configured if the parameter not provided
quotagroup	delete			delete one or multiple quotagroup
		--name	qg-test	Delete a specific quotagroup with the name
		--all-orphan	N/A	Delete all quotagroup not used by any Hyper Protect Virtual Server containers
		--lpar	SSC_LPAR_NAME	The LPAR name or address to run this command. will run the command on all LPARs configured if the parameter not provided
quotagroup	update			update a quotagroup. You must manually restart the Hyper Protect Virtual Server container for the quotagroup.
		--name	qg-test	The quotagroup name to be updated
		--size	10	The new quotagroup size

SUBJECT	ACTION	Parameter name	Sample value	Description
		--size-unit	GB	The size_unit, , default is GB
		--lpar	SSC_LPAR_NAME	The LPAR name or address to run this command. will run the command on all LPARs configured if the parameter not provided

Commands for monitoring

Commands to monitoring IBM Hyper Protect Virtual Servers take the form as in the following example. Note these commands require root user authority.

```
docker run --rm -it -v $(pwd):/monitoring-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
monitoring create
```

Where:

- `monitoring` is the `SUBJECT`
- `create` is the `ACTION`

Table 1. The full list of the `SUBJECT`, `ACTION` and `Parameters` for `monitoring` commands

subject	action	parameter name	sample value	description
monitoring	create	N/A	N/A	Creates/Deploys monitoring-host and collectd-host containers
monitoring	delete	N/A	N/A	Deletes monitoring-host and collectd-host containers
monitoring	cleanup	N/A	N/A	Removes monitoring-host and collectd-host containers as well as repositories for both containers
monitoring	start	N/A	N/A	Starts monitoring-host and collectd-host containers
monitoring	stop	N/A	N/A	Stops monitoring-host and collectd-host containers
monitoring	restart	N/A	N/A	Restarts monitoring-host and collectd-host containers

Note:

- The command requires a `monitoring.yaml` file under the `./monitoring-cli/config` folder. For more information, see monitoring.yaml.
- You cannot run the `monitoring` commands against a subset of Secure Service Container partitions if the `monitoring.yaml` includes the configuration of multiple partitions. To run the `monitoring` commands against an individual Secure Service Container partition, or multiple Secure Service Container partitions at the same time, only include the target Secure Service Container LPAR's stanza(s) for the intended `monitoring` command in the VS`/monitoring-cli/config/monitoring.yaml` configuration file.
- During the life cycle of the monitoring infrastructure, you might notice the following files are created under the `./monitoring-cli/config` directory.
  - `monitoring-configuration.yaml`, which is generated when the monitoring infrastructure is initialized. And the file is updated by removing containers when the `monitoring delete` command is executed. When the `monitoring create` is triggered again, or the `monitoring cleanup` command is executed, the file gets overwritten.
  - `monitoring-configuration-service.yaml`, which is generated and updated when the `monitoring start`, `monitoring stop`, or `monitoring restart` command is executed.
  - `monitoring-configuration-delete.yaml`, which is generated when the `monitoring delete` command is executed. The file is used to track the containers configuration that are deleted. The `monitoring-configuration.yaml` file is updated with the containers that are deleted.

Commands for repositories

Commands to manage repositories used by a Hyper Protect Virtual Server take the form as in the following example. Note these commands require root user authority.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
repository list --lpar [LPARNAMEORADDRESS]
```

Where:

- `repository` is the `SUBJECT`,
- `list` is the `ACTION`
- `--lpar` is an extra parameter.

Table 1. The full list of the `SUBJECT`, `ACTION` and `Parameters` for `repository` commands

SUBJECT	ACTION	Parameter name	Sample value	Description
repository	N/A	--help, -h	N/A	List all command parameters
repository	list,get			list repositories
		--repoid	MyDockerApp	List specified repository only
		--lpar	SSC_LPAR_NAME	List repositories on this lpar
repository	create			create a repository
		--repoid	MyDockerApp	The repoid to be registered
		--repofile	MyDockerApp.py-rel.enc	Path to Repository definition file
		--lpar	SSC_LPAR_NAME	register repository on this lpar
repository	update			update a repository

SUBJECT	ACTION	Parameter name	Sample value	Description
		--repoid	MyDockerApp	The repoid to be updated
		--repofile	MyDockerApp.py-rel.enc	Path to new Repository definition file
		--lpar	SSC_LPAR_NAME	register repository on this lpar
repository	delete	--force		delete a repository
		--repoid	MyDockerApp	repoid to be unregistered
		--lpar	SSC_LPAR_NAME	unregister repository on this lpar

Note: The value of `repoid` must be in the range a-z, A-Z, _ and 0-9.

Commands for repository definition files

Commands to manage your Repository Definition File in the Json format take the form as in the following example. Note these commands require root user authority.

```
docker run --rm -v $(pwd):/regfile-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 regfile
create -u [DOCKER_USER] -p [DOCKER_PASSWORD] -s [DOCKER_PUSH_SERVER] -r [DOCKER_REPO] -K
[PRIVATE_KEY] -k [PUBLIC_KEY] -ph [PRIVATE_KEY_PASSHPRASE]
```

Where:

- `regfile` is the SUBJECT
- `create` is the ACTION
- `-u [DOCKER_USER] -p [DOCKER_PASSWORD] -s [DOCKER_PUSH_SERVER] -r [DOCKER_REPO] -K [PRIVATE_KEY] -k [PUBLIC_KEY] -ph [PRIVATE_KEY_PASSHPRASE]` are extra parameters

After the `create` action is completed, one `isv_definition_template.json` file and the encrypted file `isv_definition_template.json-rel.enc` will be created in the current directory.

Table 1. The full list of the `SUBJECT`, `ACTION` and `Parameters` for `regfile` commands

SUBJECT	ACTION	Parameter name	Sample value	Description
regfile	N/A	N/A	N/A	List all command
regfile	N/A	-h/--help	N/A	List all command parameters
regfile	create			Generate the Repository Definition File in Json format and encrypt it
		-r/--repo	docker_base_user/MyDockerApp	Remote docker repository name
		-s/--server	docker.io	Docker push server name

SUBJECT	ACTION	Parameter name	Sample value	Description
		-u/--username	docker_user	User name to log in to docker repository
		-p/--password	docker_password	Password to log in to docker repository
		-K/--private-key	isv_user.private	The relative path of the ISV private key file to `/regfile-cli/config` directory
		-k/--public-key	isv_user.pub	The relative path of the ISV public key file to `/regfile-cli/config` directory
		-ph/--passphrase	passphrase	The passphrase of ISV private key
		-kn/--keyname	my_key_name	A unique ISV key name for signing the repository files.
		-e/--envs-whitelist	"VAR_A,VAR_B,VAR_C"	Variables that can be pass into containers associated with this repository definition file.
		-c/--revocation-cert	revoke.asc	The path to the revocation certification of the existing ISV key pair. To revoke an existing key, you must provide the new ISV definition key pair with `-nK` and `-nk` parameters.
		-nK/--new-private-key	isv_key.private	The path to the new ISV private key file
		-nk/--new-public-key	isv_key.pub	The path to the new ISV public key file

SUBJECT	ACTION	Parameter name	Sample value	Description
		-n/--name	MyDockerApp	The file name of the encrypted repository definition file. If not provided, the default file name is `isv_definition_template.json-rel.enc`.
regfile	encrypt			Sign the clear text Repository definition file with ISV private key and encrypt with IBM public key.
		-K/--private-key	isv_user.private	The path to the ISV private key file
		-k/--public-key	isv_user.pub	The path to the ISV public key file
		-i/--input_data	MyDockerApp.json	The path to the repository definition json file
		-ph/--passphrase	passphrase	The passphrase for ISV private key
		-kn/--keyname	my_key_name	A unique ISV key name for signing the repository files.

Note:

- When you create and encrypt a repository definition file by using your own key pair, ensure to use the key pair for one specific repository.
- When you create and encrypt a repository definition file by using the key pair generated by the `regfile` commands, ensure to specify a key name by using the `-kn` or `--keyname` option.

Commands for snapshots

Commands to manage snapshots used by a Hyper Protect Virtual Server take the form as in the following example. Note these commands require root user authority.

```
docker run --rm -it -v $(pwd):/hpvs-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0 snapshot
create --name [SNAPSHOT_NAME]
```

Where:

- `snapshot` is the `SUBJECT`. The snapshots of the Hyper Protect Virtual Server are stored on the Secure Service Container partition.
- `create` is one of values in the `ACTION` column from the following table.
- `--name` is extra parameters.

Table 1. The full list of the `SUBJECT`, `ACTION` and `Parameters` for `snapshot` commands

SUBJECT	ACTION	Parameter name	Sample value	Description
snapshot	N/A	--help, -h	N/A	List all command parameters
snapshot	list,get			list all snapshots for a Hyper Protect Virtual Server
		--lpar	SSC_LPAR_NAME	The LPAR name to run this command. will run the command on all LPARs configured if the parameter not provided
		--container	vs1	The name of the Hyper Protect Virtual Server
snapshot	create			Create a snapshot for a Hyper Protect Virtual Server on the LPAR.
		--snapshot	snapshot1	The name of snapshot to be created
		--container	vs1	The name of the Hyper Protect Virtual Server
		--lpar	SSC_LPAR_NAME	The LPAR name to run this command. will run the command on all LPARs configured if the parameter not provided
snapshot	delete			Delete a snapshot for a Hyper Protect Virtual Server
		--lpar	SSC_LPAR_NAME	The LPAR name to run this command. will run the command on all LPARs configured if the parameter not provided
		--container	vs1	The name of the Hyper Protect Virtual Server
		--snapshot	snapshot1	The name of the snapshot to be deleted

SUBJECT	ACTION	Parameter name	Sample value	Description
snapshot	revert			Revert to the snapshot for a Hyper Protect Virtual Server. You must restart the Hyper Protect Virtual Server container after the `snapshot revert` command.
		--lpar	SSC_LPAR_NAME	The LPAR name to run this command. will run the command on all LPARs configured if the parameter not provided
		--snapshot	snapshot1	The name of the snapshot to be reverted
		--container	vs1	The name of the Hyper Protect Virtual Server
		--quotagroup	qg1	The quotagroup name to be reverted

Commands for Secure Build containers

Commands to manage Secure Build server containers take the form as in the following example. Note these commands require root user authority.

```
docker run --rm -it -v $(pwd):/securebuild-cli/config ibmzcontainers/hpvs-cli-installer:1.2.0
securebuild create --name <secure-build-server-name> --language <EN|FR|CN>
[ADDITIONAL_PARAMETERS]
```

where

- `securebuild` is the `SUBJECT`.
- `create` is the `ACTION`.
- `ADDITIONAL_PARAMETERS` are the extra parameters for the corresponding action taken from the following table.
- The `--name` parameter is mandatory for all commands.
- The `--language` parameter is optional and can be used to override the default language setting used by the command line tool.

Table 1. The full list of the `SUBJECT`, `ACTION` and `Parameters` for `securebuild` commands

SUBJECT	ACTION	Parameters name	Sample values	Description
securebuild	N/A	--help, -h	N/A	List all command parameters

SUBJECT	ACTION	Parameters name	Sample values	Description
securebuild	build			Start a build on the Secure Build server container
securebuild	clean			Clean up all artifacts from a previous build
securebuild	create			Create the Secure Build server container
securebuild	init			initialize the Secure Build server container. Only required when the Secure Build server container is created by using a specific public certificate.
securebuild	status			Get the status of the most recent build
securebuild	log	--type	audit or build	Get logs from the Secure Build server container
securebuild	manifest			Get the application manifest file for an application built by the Secure Build server container
securebuild	public-key			Get the application manifest public key for an application built by the Secure Build server container

SUBJECT	ACTION	Parameters name	Sample values	Description
securebuild	regfile	--regfile-type	`lite` or `full`	Get the unsigned repository registration file in the clear text for an application built by the Secure Build server container, either in JSON (`lite`) or Python (`full`) format. The JSON repository definition file can be used as a signed and encrypted registration file with the `regfile` command by you or your ISV. The Python registration file will need IBM or your system administrator to sign and encrypt.
securebuild	update			Update a Secure Build server container to use the new configuration parameters other than CPU and `memory` written to the `securebuild.yaml` configuration file.
securebuild	backup	--backup-name	any	Take a snapshot of the given (Secure Build) container.
securebuild	restore	--backup-name	any	Restore the named snapshot of the given (Secure Build) container.
securebuild	backup-delete	--backup-name	any	Delete the named snapshot of the given (Secure Build) container.
securebuild	restart			Restart the Secure Build container.

SUBJECT	ACTION	Parameters name	Sample values	Description
securebuild	upgrade			Upgrade the Secure Build container with different base images, or different CPU or memory settings.
securebuild	delete	--force		Delete the Secure Build container.

Note:

- If you do not specify a value for the `container:cert_name` key in the `securebuild.yaml` file, the `securebuild create` command creates and initializes the Secure Build server container. Otherwise, the `securebuild init` must be run after the `securebuild create` command.
- Deleting a Secure Build container is a dangerous action because it will delete all private keys which that Secure Build container uses to push built images to their Docker repository, and deletes all backups of that container. Only use this action if you do not intend to push any more images to the Docker repository the Secure Build container to be deleted is using. To replace a Secure Build container with a container based on a newer Secure Build container image, use the `securebuild upgrade` action.

Configuration files

The following topics list the configuration files will be used when running a certain component of IBM Hyper Protect Virtual Servers.

- hosts
- hpvs-config.yaml
- hpvs-env.json
- securebuild.yaml
- monitoring.yaml
- OpenSSL configuration examples
- grep11-config.yaml

grep11-config.yaml

The `grep11-config.yaml` file defines the configuration of each GREP11 container. The following example is a yaml template, with descriptions of each parameter, that you can fill out with the configuration for your own GREP11 containers.

```
grep11:
  name: '<Enter the name of your GREP11 container. It could any name.>'
  imagetag: '<Enter the image tag value of GREP11 container. You must copy the value from the
grep11-config.yaml.example file>'
  crypto_domain: '<Enter the crypt domain name available on the HSM>'
  memory: '<Enter the memory size (In MB) of the grep 11 container>'
  network:
    network_name: '<Enter the network name for your GREP11 container>'
    port: '<Enter the port number if port mappting on LPAR is enabled>'
    ip: '<Enter the IP address of your GREP11 container>'
  # optional
  hostname: '<Enter the host name for your GREP11 container if you use the self-signed
certificate>'
  mutual_tls: '<Whether to enable mutual TLS. Default is False>'
  certificate: '<Enter the path to the server certificate. The server certificate must be
located under the grep11-cli/config directory>''
  certificate: '<Enter the path to the server private key file. The server private key file
must be located under the grep11-cli/config directory>'
```

```
  # If mutual TLS is true
  cacertificate: '<Enter the path to the CA certificate. The CA certificate must be located
under the grep11-cli/config directory>'
```

Hosts

The `hosts` file defines the RESTful API connection information and credential to the Secure Service Container partition used by the IBM Hyper Protect Virtual Servers CLI.

```
[lpars]
# Replace the example IP with your Secure Service Container partition IP address,
# And, replace the rest_* credentials with real values for rest api access.
10.152.151.105 rest_user="someuser" rest_pass="somepassword"
```

Note:

- To run the `hpvs`, `images`, `repository`, `regfile`, `snapshots`, `disks`, or `quotagroup` command, the `hosts` file must be available under the `<installation_directory>/VS/hpvs-cli/config/` directory.
- To run the `securebuild` command, the `hosts` file must be available under the `<installation_directory>/VS/securebuild-cli/config/` directory.
- To run the `monitoring` command, the `hosts` file must be available under the `<installation_directory>/VS/monitoring-cli/config/` directory.
- To run the `crypto` or `grep11` command, the `hosts` file must be available under the `<installation_directory>/VS/grep11-cli/config/` directory.

hpvs-config.yaml

The `hpvs-config.yaml` defines the configuration of a Hyper Protect Virtual Server container.

```
cluster:
  name: "vs"
LPARS:
  - ipaddress: '<Enter the LPAR IP Address here>'
    name: '<Enter the LPAR name here>'
    containers:
      - template: '<Enter the resource template for the server container>'
        count: '<Enter the number of the server containers for the resource template>'
        networks:
          - network_name: '<Enter the internal network name>'
            ip_address: ['<Enter the internal network IP address>]
          - network_name: ''<Enter the external network name>'
            ip_address: ['<Enter the external network IP address>]
template1:
 name: '<Enter the resource template name>'
 repoid: '<Enter the registered id of repository to create container from>'
 imagetag: '<Enter the image tag to create container from>'
 cpu: `<Enter the cpu defined by Number of threads>`
 memory: <Enter the memory defined in MB>
 vs_index: '<Enter the initial Hyper Protect Virtual Server index, if there is no names array
in containers, the ${name}_${vs_index} will be used as the Hyper Protect Virtual Server name,
like hyper-protect-vs-10001, hyper-protect-vs-10002>'
 domainname: '<Enter the optional domain name for the container>'
 hostname: '<Enter the optional host name for the container>'
 ports: `<Enter the dict of ports to forward to the container in this format {'FROM': 'TO'},
where FROM is the port inside the container such as 80 and TO is the port on the Secure Service
Container partition itself such as 8080>`
 labels: `<Enter the list of labels to set for the container in this format {"arch": "s390x"}>
 extra_hosts: `<Enter the list of hostname:IP in this format ["test.com:192.168.10.1"]>'
 log_type: `<Enter the logging type such as json-file>'
 log_config:
    max-size: '<Enter the maximum size of the log file such as 10M>`
 quotagroup_storage: `<Enter the size of the quotagroup such as 10G. The default value is 12
GB>'
 mount: '<Enter the mount point inside the container where the quotagroup is mounted. For
example, /data_pool>''
 filesystem: '<Enter the filesystem used for the mount inside the container. The value can be
btrfs, ext4, or xfs>'
 # Don't remove the quotagroup after container deletion if true
 quotagroup_keep: false
 # Optional. Specify existing quotagroup for mount, but not create new one
 #quotagroup: ""
 networks:
    # public network sample
    encf900_network:
      subnet: "9.20.4.0/22"
      gateway: "9.20.4.1"
      parent: "encf900"
      driver: "macvlan"
    # internal network sample
    encf900_internal_network:
      subnet: "192.168.40.0/24"
      gateway: "192.168.40.1"
      parent: "encf900"
```

hpvs-env.json

The `hpvs-env.json` file defines environment variables that can be passed into the `hpvs-config.yaml` file of the Hyper Protect Virtual Server container when the container is created. If you want to debug the Hyper Protect Virtual Server container, use your SSH public key value in the SSH_PUBLIC_KEY parameter.

```
{
    "LOGTARGET": "/dev/console",
    "ROOTFS_LOCK": "y",
    "SSH_PUBLIC_KEY": "<input your public key value>"
}
```

Refer to the following example when you configure the SSH_PUBLIC_KEY key in the `hpvs-env.json` file.

1. Retrieve the SSH public key value from the SSH public key file. For example, the public key file is `~/.ssh/debug-myapp-test.pub`.

```
#cat ~/.ssh/debug-myapp-test.pub
ssh-rsa AAAAB...D4k1w== root@testsys1
```

1. Configure the SSH public key value in the `hpvs-env.json` file.

```
{
  "LOGTARGET": "/dev/console",
  "ROOTFS_LOCK": "y",
  "SSH_PUBLIC_KEY": "AAAAB...D4k1w=="
}
```

monitoring.yaml

The `monitoring.yaml` file defines the configuration of a monitoring container on one {site.data.keyword.ssc}} Partition. The following example is a yaml template, with descriptions of each parameter, that you can fill out with the configuration for your own monitoring containers.

```
LPARS:
  - ipaddress: '<Enter the LPAR IP address>'
    containers:
    - template: "monitoring-host"
      private-key-server: '<Enter the private key used for encryption (HTTPS). Required. For 
example, /monitoring-cli/keys/key.pem>'
      public-cert-server: '<Enter the certification used for server-side authentication. 
Required. For example, /monitoring-cli/keys/certificate.pem>'
      public-cert-client: '<Enter Certification used for client-side authentication. Optional. 
if it's not provided, client authentication will be disabled and may incur potential security 
issue. For example, /monitoring-cli/keys/certificate.pem>'
      metric-dn-suffix: '<Enter the domain suffix of the monitoring infrastructure. Check with 
the system administrator for the value>'
      dns-name: `<Enter the DNS name. Check with the system administrator for the value>`
    - template: "collectd-host"
monitoring-host:
   name: "monitoring-host"'<This name must not be changed and given to any other container on 
LPAR>'
   cpu: '<Enter the number of thread for the monitoring container. The default value is 1>'
   memory: '<Enter the memory size (in MB) of the monitoring container. Optional. The default 
value is 512>'
```

```
   imagetag: '<Enter the imagetag of the monitoring container. Must be identical to the version
number of the IBM Hyper Protect Virtual Servers offering>'
collectd-host:
   name: "collectd-host"'<This name must not be changed and given to any other container on
LPAR>'
   imagetag: '<Enter the imagetag of the collectd container. Must be identical to the version
number of the IBM Hyper Protect Virtual Servers offering>'
```

securebuild.yaml

The `securebuild.yaml` file defines the configuration of each Secure Build container. The following example is a yaml template, with descriptions of each parameter, that you can fill out with the configuration for your own Secure Build containers.

```
secure_build_workers:
  - container:
      port: '<Enter the HTTPS port used by the Secure Build container here>'
      name: '<Enter the name of the Secure Build container here>'
      ipaddress: '<Enter the IP address on which the Secure Build container is accessible here.
If this is the same as the Hosting Appliance's IP address enter the Hosting Appliance's IP
address>'
      cpu: '<Enter the number of virtual CPUs to assign to the container>'
      memory: '<Enter the memory to assign to the container, in GB>'
      quotagroup_name: '<Enter the quotagroup to create container datapool on>'
      delete_quotagroup_on_container_delete: '<If this parameter is set to True the Secure
Build container's quotagroup will be deleted when the Secure Build container is deleted and
there are no other containers using its quotagroup. If set to False or undefined, the Secure
Build container's quotagroup will remain if the Secure Build container is deleted>'
      quotagroup_size: '<Enter the size of the quotagroup, in GB>'
      root_volume_size: '<Enter the size of the root volume datapool for the Secure Build
container>'
      docker_volume_size: '<Enter the size of the datapool for storing Docker images on the
Secure Build container>'
      data_volume_size: '<Enter the size of the datapool for storing logs and other data on the
Secure Build container>'
      repoid: '<Enter the ID of the repository that contains the Secure Build server container
image. Default is 'SecureDockerBuild'>'
      imagetag: '<Enter the tag of the Secure Build server container image to be used. Default
is 'latest'>'
      delete_network_on_container_delete: '<Enter whether to delete the network when the
container is deleted. Default is 'False'>'
      cert_name: '<Enter the name of a .PEM or .crt format client certificate, either to enable
the Secure Build server container container to generate a certificate in that name, or use a
user-provided client certificate>'
    lpar:
      ipaddress: '<Enter the IP address of Secure Service Container Partition to host this
Secure Build container>'
    network:
      name: '<Enter the name of the network on the Secure Service Container Partition to
connect the Secure Build container to>'
      ipaddress: '<Enter the internal IP address on the Secure Service Container Partition
network to assign to the Secure Build container>'
    regfile:
      id: '<Enter the file name of the JSON repository registration file. Default is
'MyApp.json'>'
    github:
      url: '<Enter the GitHub URL for the GitHub project you are building (GitHub URL starts
with 'git@')>'
      branch: '<Enter the branch to build in the GitHub project you are building. Default is
'master'>'
      key: '<Enter the private key file used to authenticate with your GitHub server and access
the GitHub project, including the header and footer of the file>'
      recurse_submodules: '<Enter 'True' if the build requires contents from the submodules of
your GitHub repository>'
      dockerfile_path: '<Enter the path to the Dockerfile (including the dockerfile name)
relative to the root of the Github project. Default is "./Dockerfile">'
      docker_build_path: '<Enter the path to the subdirectory within the Github project to be
used as the build context for the Docker build>'
    docker:
      repo: '<Enter the Docker repo to which the image you are building will be pushed>'
      image_tag_prefix: '<Enter the prefix for the image tag the built image will be tagged
with when it is pushed to its Docker repo>'
      user: '<Enter the user name used to authenticate with the Docker Registry to which the
image will be pushed>'
      password: '<Enter the password used to authenticate with the Docker Registry to which the
image will be pushed>'
      base_user: '<Enter the username used to authenticate with the Docker Registry where the
base image used to build the built image is pulled from>'
```

```
      base_password: '<Enter the password used to authenticate with the Docker Registry where
the base image used to build the built image is pulled from>'
      push_server: '<Enter the server for the Docker Registry server the image will be pushed
to - default is 'docker.io'>'
      content_trust_push_server: '<enter the server for the Docker Content Trust service used
to sign the built image. Default is 'https://notary.docker.io'>'
      base_server: '<Enter the server for the Docker Registry the base image used to build the
built image is pulled from. Default is 'docker.io'>'
      image_pull_user: '<Enter the username for the Docker user that will pull down the built
image from the Docker Registry to the SSC>'
      image_pull_password: '<Enter the password for the Docker user that will pull down the
built image from the Docker Registry to the SSC>'
      content_trust_base: '<Enter 'True' to validate the base image using Docker Content trust.
Enter 'False' if you do not want to validate the base image. Default is 'True'>'
      content_trust_base_server: '<Enter the server for the Docker Content Trust service used
to validate the base image. Default is 'https://notary.docker.io'>'
    manifest_cos:
      bucket_name: '<Enter the bucket name on the S3 object store where manifest files will be
transferred to after each build>'
      api_key: '<Enter the API key used to authenticate with the S3 object store>'
      resource_crn: '<Enter the resource instance ID for the S3 object store>'
      auth_endpoint: '<Enter the authentication endpoint for the S3 object store>'
      endpoint: '<Enter the endpoint for the S3 object store>'
    env:
      whitelist: '<Enter a comma separated list of environment variable names that you want to
be able to pass into a container instance of the built image when it is created on the Hosting
Appliance. For example, ["PATH","SHELL"]. If you do not configure the whitelist parameter,
remove the entry from the yaml file.>'
    build:
      args: '<Enter a list of parameters to pass to docker --build-arg, if a build script is
not being used'
```

Secure Build configuration parameters

The following tables list the parameters that can appear in the Secure Build configuration file, `securebuild.yaml`, whether the parameter is required, whether the parameter value can be updated after a Secure Build container has been created, and what the default value, if any, is for that parameter if it is not a required parameter.

Container parameters

Table 1. Parameters under the key `container:`

Key	Required	Can Be Updated	Default Value
port	No	No	443
name	Yes	No	N / A
ipaddress	Yes	No	
cpu	No	Yes	1
memory	No	Yes	8192
quotagroup_name	Yes	No	
quotagroup_size	No	No	50
delete_quotagroup_on_container_delete	No	Yes	False
root_volume_size	No	No	20
docker_volume_size	No	No	16
data_volume_size	No	No	4
repoid	No	No	SecureDockerBuild
imagetag	Yes	Yes	N / A

Key	Required	Can Be Updated	Default Value
delete_network_on_container_delete	No	Yes	False

Note:

- Volume and Quotagroup sizes are in units of GB.
- Memory sizes are in units of MB.
- The value of cpu is the number of virtual CPUs assigned to the container.

LPAR parameters

Table 2. Parameters under the key `lpar:`

Key	Required	Can Be Updated	Default Value
ipaddress	Yes	No	N / A

network parameters

Table 3. Parameters under the key `network:`

Key	Required	Can Be Updated	Default Value
name	No	No	(null)
driver	No	No	(null)
parent	No	No	(null)
subnet	No	No	(null)
gateway	No	No	(null)
ipaddress	No	No	(null)

Note: If no network parameters are specified, the Secure Build container is created without connecting it to a network.

regfile parameter

Table 4. Parameter under the key `regfile:`

Key	Required	Can Be Updated	Default Value
id	No	Yes	MyApp.json

Github parameters

Table 5. Parameters under the key `github:`

Key	Required	Can Be Updated	Default Value
url	Yes	Yes	N / A
branch	No	Yes	master
key	No	No	(null)
recurse_submodules	no	Yes	False
dockerfile_path	No	Yes	`docker_build_path`/ Dockerfile
docker_build_path	no	Yes	.

The default `docker_build_path` is the root directory of the Github project.

Docker parameters

Table 6. Parameters under the key `docker:`

Key	Required	Can Be Updated	Default Value
repo	Yes	No	N / A
image_tag_prefix	No	Yes	(null)
user	Yes	Yes	N / A
password	Yes	Yes	N / A
push_server	No	Yes	docker.io (Docker Hub)
content_trust_push_server	No	Yes	notary.docker.io
base_user	Yes	Yes	N / A
base_password	Yes	Yes	N / A
base_server	No	Yes	docker.io (Docker Hub)
image_pull_user	No	Yes	(null)
image_pull_password	No	Yes	(null)
content_trust_base	No	Yes	True
content_trust_base_server	No	Yes	notary.docker.io

The Secure Build container generates a randomised value for each passphrase (repository or root) that is not provided in the Secure Build configuration. Because the root and repository private keys used with Docker Content Trust are only used internally inside the Secure Build container, and are not accessible outside of it, users of the Secure Build server container or Hyper Protect Virtual Server containers do not need to know the values of these passphrases.

Manifest Cloud Object Store parameters

Table 7. Parameters under the key `manifest_cos:`

Key	Required	Can Be Updated	Default Value
bucket_name	No	Yes	(null)
api_key	No	Yes	(null)
resource_crn	No	Yes	(null)
auth_endpoint	No	Yes	(null)
endpoint	No	Yes	(null)

ENV parameters

Table 8. Parameters under the key `env:`

Key	Required	Can Be Updated	Default Value
whitelist	No	Yes	(null)

Build parameters

Table 9. Parameters under the key `build`:

Key	Required	Can Be Updated	Default Value
args	No	Yes	(null)

OpenSSL configuration examples

You can use the following example files with the `openssl` command if you want to avoid entering the values for each parameter required when creating certificates.

Note: You must update the configuration files with the actual values for your environment. For more information, see Creating self signed certificates or Creating CA signed certificates.

The sample configuration file to generate the Root CA certificate

```
[ ca ]
default_ca = CA_LOC

[ CA_LOC ]
prompt            = no
dir               = /home/myuser/ca
certs             = $dir/certs
crl_dir           = $dir/crl
new_certs_dir     = $dir/newcerts
database          = $dir/index.txt
serial            = $dir/serial
RANDFILE          = $dir/private/.rand
private_key       = $dir/private/myrootCA.key
certificate       = $dir/certs/myrootCA.crt
crlnumber         = $dir/crlnum
crl               = $dir/crl/mycrl.pem
default_crl_days  = 30
preserve          = no
policy            = policy
default_days      = 365

[ policy ]
commonName              = supplied
stateOrProvinceName     = supplied
countryName             = supplied
emailAddress            = supplied
organizationName        = supplied
organizationalUnitName  = supplied

[ req ]
default_bits        = 4096
distinguished_name  = req_distinguished_name

string_mask         = utf8only
default_md          = sha256
x509_extensions     = v3_ca

[ req_distinguished_name ]
countryName                     = AB
stateOrProvinceName             = CD
localityName                    = EF_GH
organizationName              = myorg
organizationalUnitName          = myorgunit
commonName                      = mycn
emailAddress                    = myemail@example.com

[ v3_ca ]
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true
keyUsage = critical, digitalSignature
```

The sample configuration file to generate the CSR for a server certificate

```

[ req ]
prompt                 = no
```

```
days                    = 365
distinguished_name      = req_distinguished_name
req_extensions          = v3_req


[ req_distinguished_name ]
countryName             = AB
stateOrProvinceName     = CD
localityName            = EFG_HIJ
organizationName        = MyOrg
organizationalUnitName = MyOrgUnit
commonName              = mycommname.com
emailAddress            = emailaddress@myemail.com

[ v3_req ]
basicConstraints        = CA:false
extendedKeyUsage        = serverAuth
subjectAltName          = @sans

[ sans ]
DNS.0 = localhost
DNS.1 = myexampleserver.com
```

The sample configuration file to generate the CSR for a Client certificate

```

[ req ]
prompt                  = no
days                    = 365
distinguished_name      = req_distinguished_name
req_extensions          = v3_req


[ req_distinguished_name ]
countryName             = AB
stateOrProvinceName     = CD
localityName            = EFG_HIJ
organizationName        = MyOrg
organizationalUnitName = MyOrgUnit
commonName              = mycommname.com
emailAddress            = emailaddress@myemail.com

[ v3_req ]
basicConstraints        = CA:false
extendedKeyUsage        = clientAuth
subjectAltName          = @sans

[ sans ]
DNS.0 = localhost
DNS.1 = myexampleclient.com
```

Others

The following topics list other miscellaneous topics that you can refer to when using IBM Hyper Protect Virtual Servers.

- Security of IBM Hyper Protect Virtual Servers
- List of docker images in the IBM Hyper Protect Virtual Servers
- List of keys used during the Secure Build
- Metrics collected by the monitoring infrastructure
- Creating self signed certificates for the monitoring infrastructure
- Creating CA signed certificates for the monitoring infrastructure
- Creating certificates for GREP11 containers

List of docker images in the IBM Hyper Protect Virtual Servers

Table 1. The full list of the docker image files in the IBM Hyper Protect Virtual Servers

Image file Name	Location	Used by which command(s)	Description	More information
`hpvs-cli-installer.docker-image.tar`	`<installation_directory>`	`docker load`	CLI for IBM Hyper Protect Virtual Servers	Installing the IBM Hyper Protect Virtual Servers CLI tool
`HpvsopBaseSSH.tar.gz`	`<installation_directory>/VS/hpvs-cli/config/<destination-folder-HpvsopBaseSSH>`	`docker load`	Base image for the Hyper Protect Virtual Server container with SSH daemon	Registering base images in the remote registry server
`HpvsopBase.tar.gz`	`<installation_directory>/VS/hpvs-cli/config/<destination-folder-HpvsopBase>`	`docker load`	Base image for the Hyper Protect Virtual Server container without SSH daemon	Registering base images in the remote registry server
`SecureDockerBuild.tar.gz`	`<installation_directory>/VS/securebuild-cli/config`	`docker run` + `image load`	Base image for the Secure Build container	Loading the Secure Build base image into the Secure Service Container partition
`CollectdHost.tar.gz`	`<installation_directory>/VS/monitoring-cli/config`	`docker run` + `monitoring create`	Base image for the `collect-host` container of the monitoring infrastructure	Monitoring IBM Hyper Protect Virtual Servers
`Monitoring.tar.gz`	`<installation_directory>/VS/monitoring-cli/config`	`docker run` + `monitoring create`	Base image for the `monitoring-host` container of the monitoring infrastructure	Monitoring IBM Hyper Protect Virtual Servers
`hpcsKpGrep11_runq.tar.gz`	`<installation_directory>/VS/grep11-cli/config`	`docker run` + `grep11 create`	Base image for the grep11 container	Integrating with the EP11 library

List of keys used during the Secure Build

Table 1. The full list of the keys used during the Secure Build and BYOI lifecycle.

Key Name	Key Function	Private Key Location	How Created	Owned by Whom
Image Signing Key	Pushing Docker images to a Docker repository	Encrypted volume on the Secure Build container	Created by the remote registry server on first push to the remote repository, and written to Secure Build container	ISV or application developer
Manifest Signing Key	Signing a manifest created by Secure Build	Encrypted volume on the Secure Build container	Created by the Secure Build container when an image is built	ISV or application developer
Session Key	Used to encrypt the Secure Build container backup image	Hosting Appliance zHSM	Generated by a request to backup container snapshots to Cloud Object Store. New key generated each time backup to Cloud Object Store is requested	Cloud administrator
Container Key	Used to encrypt the Session Key (additional layer of encryption)	Hosting Appliance zHSM	Generated by a call to the Hosting Appliance's POST /`disaster_recovery/keys/{container_name}` REST API endpoint. See here	Cloud administrator
Client CRT Key	Used by the cloud administrator to securely interact with the Secure Build REST API, contains certificate and private key	Client	Created on creation of the Secure Build container and provided to the client as the file specified in their CLIENT_CRT_KEY setting	Cloud administrator

Metrics collected by the monitoring infrastructure

The following table shows the metrics collected by the monitoring infrastructure provided in IBM Hyper Protect Virtual Servers .

Table 1. Monitoring metrics collected

#	Plugin Name	Metrics Name	Labels	Description
01	collectd	collectd_collectd_cache_size	collectd="cache",instance	The number of elements in the metric cache.

#	Plugin Name	Metrics Name	Labels	Description
02	collectd	collectd_collectd_derive_total	collectd="write_queue",type="dropped",instance	The number of metrics dropped due to a queue length limitation.
03	collectd	collectd_collectd_queue_length	collectd="write_queue",instance	The number of metrics currently in the write queue.
04	cpu	collectd_cpu_percent	cpu="idle",instance	Percentage of time that the CPU or CPUs were idle and the system did not have an outstanding disk I/O request.
05	cpu	collectd_cpu_percent	cpu="interrupt",instance	Percentage of time spent by the CPU or CPUs to service hardware interrupts.
06	cpu	collectd_cpu_percent	cpu="nice",instance	Percentage of time spent by the CPU or CPUs to run a niced guest. Nice is when the CPU is executing a user task having below-normal priority.
07	cpu	collectd_cpu_percent	cpu="softirq",instance	Percentage of time spent by the CPU or CPUs to service software interrupts.
08	cpu	collectd_cpu_percent	cpu="steal",instance	Percentage of time spent in involuntary wait by the virtual CPU or CPUs while the hypervisor was servicing another virtual processor.
09	cpu	collectd_cpu_percent	cpu="system",instance	Percentage of CPU utilization while the CPU is running kernel code. This includes device drivers and kernel modules.

#	Plugin Name	Metrics Name	Labels	Description
10	cpu	collectd_cpu_percent	cpu="user",instance	Percentage of CPU utilization while the CPU is running code in user-mode. This includes your application code.
11	cpu	collectd_cpu_percent	cpu="wait",instance	Percentage of time when the CPU or CPUs were waiting for an I/O operation to complete, and the CPU can't be used for anything else.
12	df	collectd_df_percent_bytes	df=`<MountPoint>`,type="free",instance	Free disk space on the file system, expressed as a percentage. MountPoints: root, /hostfs/var/lib/quotagroups/lv_data_pool/appliance_data
13	df	collectd_df_percent_bytes	df=`<MountPoint>`,type="reserved",instance	Reserved disk space on the filesystem, expressed as a percentage. MountPoints: root, /hostfs/var/lib/quotagroups/lv_data_pool/appliance_data
14	df	collectd_df_percent_bytes	df=`<MountPoint>`,type="used",instance	Used disk space on the file system, expressed as a percentage. MountPoints: root, /hostfs/var/lib/quotagroups/lv_data_pool/appliance_data
15	load	collectd_load_longterm	load="relative",instance	The average system load over a period of the last 15 minutes.

#	Plugin Name	Metrics Name	Labels	Description
16	load	collectd_load_midterm	load="relative",instance	The average system load over a period of the last 5 minutes.
17	load	collectd_load_shortterm	load="relative",instance	The average system load over a period of 1 minute.
18	memory	collectd_memory	memory="buffered",instance	Amount of memory used for buffering, mostly for I/O operations.
19	memory	collectd_memory	memory="cached",instance	Memory used for caching disk data for reads, memory-mapped files or tmpfs data.
20	memory	collects_memory	memory="free",instance	Total amount of unused memory.
21	memory	collectd_memory	memory="slab_recl",instance	Amount of reclaimable memory used for slab kernel allocations.
22	memory	collectd_memory	memory="slab_unrecl",instance	Amount of unreclaimable memory used for slab kernel allocations.
23	memory	collectd_memory	memory="used",instance	Total amount of memory used.
24	memory	collectd_memory_percent	memory="buffered",instance	Amount of memory used for buffering, mostly for I/O operations.
25	memory	collectd_memory_percent	memory="cached",instance	Memory used for caching disk data for reads, memory-mapped files or tmpfs data.
26	memory	collects_memory_percent	memory="free",instance	Total amount of unused memory.
27	memory	collectd_memory_percent	memory="slab_recl",instance	Amount of reclaimable memory used for slab kernel allocations.

#	Plugin Name	Metrics Name	Labels	Description
28	memory	collectd_memory_percent	memory="slab_unrecl",instance	Amount of unreclaimable memory used for slab kernel allocations.
29	memory	collectd_memory_percent	memory="used",instance	Total amount of memory used.
30	uptime	collectd_uptime	instance	Seconds since system boot.

Creating self signed certificates for the monitoring infrastructure

You can generate self-signed certificates for the monitoring infrastructure by using the `openssl` utility or any other certificate generation tools that comply with your organization rules.

This procedure is intended for users with the role *Application developer or ISV*.

Before you begin

- Ensure that you install the OpenSSL or similar tool on a workstation that you can use to generate the certificates.

Procedure

Complete the following steps on your workstation with root user authority.

1. Make a note of the details to generate certificates such as the Common Name (CN) and Subject Alternative Name (SAN) that you intend to set in the certificate. For example, `example.com`, `myorg.example.com`. For more information, see OpenSSL configuration examples
2. Create a directory on your workstation to run the `openssl` command or any similar tool.

   ```
   mkdir /home/myuser/certificates
   cd /home/myuser/certificates
   ```

3. Create a private key by using the following command. After the command completes, a private key will be created under the current directory.

   - For a server certificate, use the following command.

   ```
   ```

   openssl genrsa -out server.key 4096

   ```
   * For a client certificate, use the following command.
   ```

   openssl genrsa -out client.key 4096

   ```
   ```

4. Create a Certificate Signing Request (CSR) based on the private key you just created. You will be asked to enter values for various certificate fields such as Organization Unit (OU), Common Name (CN), Email, Country Code, State or Province name, City, Organization or Company Name. After the command completes, a CSR file is created under the current directory.

   a). If you choose to enter the values for the certificate fields as prompted, then run the following command to create a server certificate.

   ```
   openssl req -new -key server.key -out server-certificate.csr
   ```

Or run the following command to create a client certificate.

```
openssl req -new -key client.key -out client-certificate.csr
```

b). If you choose to avoid entering these fields on command prompt in an interactive manner, then create a configuration file such as `server-certificate.cnf` and provide the list of these fields and their values as in the following the command for a server certificate.

```
openssl req -new -config server-certificate.cnf -key server.key -out server-certificate.csr
```

Or a `client-certificate.cnf` configuration file as in the following command for a client certificate.

```
openssl req -new -config client-certificate.cnf -key client.key -out client-certificate.csr
```

Note:

- To create a server certificate, include the entry `extendedKeyUsage=serverAuth` in the `server-certificate.cnf` file.
- To create a client certificate, include the entry `extendedKeyUsage=clientAuth` in the `client-certificate.cnf` file.
- For the sample configuration files, see OpenSSL configuration examples.

5. Create a self-signed certificate based on the CSR you just created. After the command completes, the certificate is created under the current directory.

   - For a self-signed server certificate, use the following command.

   ```
   openssl x509 -req -days 365 -in server-certificate.csr -signkey server.key -out server-
   certificate.pem
   ```

   - For a self-signed client certificate, use the following command.

   ```
   openssl x509 -req -days 365 -in client-certificate.csr -signkey client.key -out client-
   certificate.pem
   ```

   Note: You can choose the certificate extension to be `.cer`, `.pem`, or `crt`. For example, `server-certificate.cer`.

Next

You can proceed with configuring of the monitoring infrastructure as instructed in the Monitoring IBM Hyper Protect Virtual Servers topic.

Creating CA signed certificates for the monitoring infrastructure

You can generate Certificate Authority (CA) Root and CA signed certificates for the monitoring infrastructure by using the `openssl` utility or any other certificate generation tools that comply with your organization rules.

This procedure is intended for users with the role *Application developer or ISV*.

Before you begin

- Ensure that you install the OpenSSL or similar tool on a workstation that you can use to generate the certificates.

Procedure

Complete the following steps on your workstation with root user authority.

1. Create a directory on your workstation to run the `openssl` command or any similar tool.

```
mkdir /home/myuser/ca-certificates
cd /home/myuser/ca-certificates
```

2. Create CA Root certificates by using the following procedure. The root CA certificate will be used to sign CA certificates.

   a. Create the CA root private key. After the command completes, the CA root private key `myrootCA.key` is generated under the current directory. For example, `/home/myuser/ca-certificates/myrootCA.key`.

```
openssl genrsa -out myrootCA.key 4096
```

   b. Create the Certificate Signing Request (CSR) based on the CA root private key. After the command completes, the CSR `myrootCA.csr` is generaterd under the current directory. For example, `/home/myuser/ca-certificates/myrootCA.csr`.

   - The command prompts you to enter values for various certificate fields, such as Organization Unit (OU), Common Name (CN), Email, Country Code, State/Province name, City, Organization or Company Name.

```
openssl req -verbose -new -key myrootCA.key -out myrootCA.csr -sha256
```

   - If you want to avoid entering each value when the command runs, you can use a OpenSSL configuration file to create the self signed CSR. For example, `/home/myuser/ca-certificates/myca.cnf`. For more information about the OpenSSL configuration file, see OpenSSL configuration examples.

   c. Create other required configuration and OpenSSL database by using the following command.

```
  cd /home/myuser/ca-certificates/
  touch index.txt
  touch index.txt.attr
  touch serial
  mkdir crl
  mkdir newcerts
```

   Note:

   - Those files are required to successfully create a CA root certificate.
   - You must update the file `serial` and enter a number in the file. For example, `1000`. This number signifies the serial number of the certificates being created.

   d. Create the CA root certificate by using the following command. After the command completes, the CA root certificates `myrootCA.crt` is created under the current directory.

```
 openssl ca --config /home/myuser/ca-certificates/myca.cnf -out myrootCA.crt -keyfile
myrootCA.key -verbose -selfsign -md sha256 -infiles myrootCA.csr
```

   e. Validate the CA root certificate by using the following command. After the command completes, the details of the CA root certificate is printed in the output.

```
 openssl x509 -noout -text -in myrootCA.crt
```

3. Create the CSR for the CA signed server certificate or client certificate by following the instructions on Creating self signed certificates for the monitoring infrastructure. After the commands complete, the CSR is created as the `/home/myuser/certificates/server-certificate.csr` file or `/home/myuser/certificates/client-certificate.csr` file.
4. Create the CA signed certificates by using the CA root certificate.

- To create the CA signed server certificate, run the following command.

```
 openssl x509 -req -days 365 -in /home/myuser/certificates/server-certificate.csr -CA /home/
myuser/ca-certificates/myrootCA.crt -CAkey /home/myuser/ca-certificates/myrootCA.key -
CAcreateserial -out ./server-certificate.crt
```

- To create the CA signed client certificate, run the following command.

```
 openssl x509 -req -days 365 -in /home/myuser/certificates/client-certificate.csr -CA /home/
myuser/ca-certificates/myrootCA.crt -CAkey /home/myuser/ca-certificates/myrootCA.key -
CAcreateserial -out ./client-certificate.crt
```

Next

You can proceed with configuring of the monitoring infrastructure as instructed in the Monitoring IBM Hyper Protect Virtual Servers topic.

Creating certificates for GREP11 containers

When configuring the GREP11 container, you can create certificates for the one way or mutual authentication over TLS.

This procedure is intended for users with the role *cloud administrator*.

Procedure

Complete the following procedure depending on how you want to configure the communication between GREP11 container and client application.

- For mutual TLS communication, you can only create CA signed certificates.

  To create the CA certificate for mutual authentication over TLS, use the `golang-massl` utility as in the following steps. For more information, see golang-massl.

  1. Install the cfssl toolkit either by using the `apt-get install golang-cfssl` command or following the instructions on this page.
  2. Generate the CA certificate and private key by using the `golang-mass` utility and the certificate from the trusted CA.

     ```
     git clone https://github.com/wolfeidau/golang-massl.git
     cd golang-massl/certs
     cfssl gencert -initca ca-csr.json | cfssljson -bare ca
     ```

  3. Generate a server certificate by using the provided CSR.

     ```
     cfssl gencert \
      -ca=ca.pem  \
      -ca-key=ca-key.pem \
      -config=ca-config.json  \
      -hostname=domain-name:port,ip \
      -profile=massl server-csr.json | cfssljson -bare server
     ```

  4. Generate a client certificate by using the provided CSR.

     ```
     cfssl gencert \
      -ca=ca.pem  \
      -ca-key=ca-key.pem \
      -config=ca-config.json  \
      -profile=massl client-csr.json | cfssljson -bare client
     ```

  5. Copy the server certificate, server public key, and CA certificate into the `<installation_directory>/VS/grep11-cli/config` directory.

     ```
     cp -p server.pem grep11-cli/config/
     cp -p server-key.pem grep11-cli/config/
     cp -p ca.pem grep11-cli/config/
     ```

  6. Use the following files in your applications.

    - CA certificate: `ca.pem`
    - Client public key: `client.pem`
    - Client private key: `client-key.pem`
- For one way TLS communication, you can only create self-signed certificates.
  - To create the self-signed certificate for one way authentication over TLS for a reserved or private IP address assigned to the GREP11 container, use the `certstrap` utility as in the following steps. For more information, see certstrap.
    1. Clone the `certstrap` utility to your x86 or Linux on IBM Z/LinuxONE (such as s390x architecture) management server.

```
git clone https://github.com/square/certstrap.git
cd certstrap
git checkout tags/v1.2.0
```

    1. Replace the lines beginning with GOARCH= in the `build` file with the following one to let the Linux detect the architectural environment (x86 or s390x) that the build process is executing within.

```
GOARCH=$([[ $(uname -p) == "x86_64" ]] && echo "amd64" || echo "s390x") GOOS=linux go build
-ldflags "-X main.release=$BUILD_TAG" -o bin/certstrap ${REPO_PATH}
```

    1. Build the `certstrap` utility.

```
./build
```

    1. Initialize a new certificate authority for self signing. Note that the value of `--common-name` must be consist with the host name of the GREP11 container.

```
./bin/certstrap init --common-name "grep11.example.com"
```

    1. Copy the keys into the `<installation_directory>/VS/grep11-cli/config` directory.

```
cp -p out/grep11.example.com.crt grep11-cli/config/cert.pem
cp -p out/grep11.example.com.key grep11-cli/config/key.pem
```

  - To create the self-signed certificate for one way authentication over TLS with an unreserved or public IP, use the `openssl` utility as in the following example.

```
openssl req -newkey rsa:2048 \
  -new -nodes -x509 \
  -days 3650 \
  -out cert.pem \
  -keyout key.pem \
  -subj "/C=GB/O=IBM/CN=grep11.example.com"
```

  The command generates your certificate `cert.pem` and private key `key.pem`, which must be placed under the `<installation_directory>/VS/grep11-cli/config` directory.

Next

You can create the GREP11 container by following the instructions in Integrating with the EP11 library.

Terminology

The following list explains the terms that might be used in this documentation.

- Appliance

    IBM Secure Service Container based appliance provided by an Appliance Vendor. From Hosting Appliance perspective, it is the combination of IBM Secure Service Container and Hosting Appliance.

- Appliance Administrator

  The person administrating an appliance which includes tasks, such as configuring storage, or memory to the appliance or performing other configuration tasks through the API provided by Secure Service Container or the Hosting Appliance.

- Appliance Operational Data

  Metrics, logs, appliance dump data, error logs, stack traces, kernel dump, etc.

- Appliance Protected Data

  Appliance secrets, workload data, configuration data, settings, and other internal information stored by an appliance.

- Appliance Vendor

  An internal, or external exploiter of Secure Service Container, packaging Secure Service Container into an appliance.

- BYOK

  The abbreviation of Bring Your Own Key, which allows you to import your existing keys to Hyper Protect Crypto Services service instances that protect your keys with advanced encryption.

- BYOI

  The abbreviation of **Bring Your Own Image**, which is a part of IBM Hyper Protect Virtual Servers solution to support the development and deployment of your own container images on top of the Secure Service Container framework.

- Container

  A runtime instance of an Open Container Image (OCI) compatible image.

- Datapool

  Synonyms for **Storage Pool**.

- EP11

  Enterprise PKCS #11 (EP11) is specifically designed for customers seeking support for open standards and enhanced security. The EP11 library provides an interface very similar to the industry-standard PKCS #11 API.

- GPG

  The abbreviation of Gnu Privacy Guard, which is an open standard used for signing, encrypting, and decrypting texts with public and private keys to increase the security of communications.

- GREP11

  GREP11 represents the Enterprise PKCS #11 (EP11) APIs over gRPC calls, which is designed to be a stateless interface for cryptographic operations on cloud.

- gRPC

  A modern open source high performance remote procedure call (RPC) framework that can connect services in and across data centers for load balancing, tracing, health checking, and authentication.

- Hardware security module

  A physical appliance that provides on-demand encryption, key management, and key storage as a managed service.

- Hosting appliance

  A technical component within IBM Secure Service Container based appliances, providing the enablement for running Docker-based workloads.

- Hyper Protect hosting appliance

An enhanced version of IBM Secure Service Container software appliance.

- Image

  Images are the basis of the containers. An image is an ordered collection of root file system changes and the corresponding execution parameters for use within a container runtime.

- ISV

  The abbreviation of **Independent Software Vendor**, who provides software solutions by developing and deploying containerized applications to the Secure Service Container partitions.

- Management server

  An x86 or Linux on IBM Z or LinuxONE (such as s390x architecture) management server used to run the commands provided by IBM Hyper Protect Virtual Servers , and administer the offering.

- Manifest

  A manifest is generated by the Secure Build for audit purpose, which contains a copy of the github project cloned by the Secure Build container, a copy of the build log, and a `build.json` with the build status.

- Manifest public key

  A manifest public key is used to verify the manifest generated by the Secure Build.

- Manifest private key

  A manifest private key is used to sign the manifest during the Secure Build.

- Namespace

  A namespace such as `ibmzcontainers`that contains a number of unique images. For examples, the images include `hpvsop-base`, `hyperpcons-worker`, `hyperpcons-riaas`, and so on.

- Partition

  A partition is the logic partition (LPAR) on the mainframe, and can be created by using the logic partitioning tools such as Hardware Management Console (HMC) or other logical partitioning tools.

- PKCS #11

  The abbreviation of Public-Key Cryptography Standards #11, which defines a platform-independent API to cryptographic tokens, such as HSM and smart cards.

- Quotagroup

  The storage assigned to a workload running on an appliance. The appliance administrator assigns FCP, or ECKD based storage to an appliance, and then creates quotagroups, representing parts of the underlying storage. The administrator finally assigns quotagroups to workloads through the appliance API.

- Registry

  A Registry is a hosted service containing repositories of container images that responds to the Registry API. For example, Docker Hub.

- Repository registration files

  A cleartext Python or JSON format file, which is generated by the Secure Build container when the container is created. The JSON format repository registration file can be used as the direct input to generate an encrypted repository definition file.

- Repository definition files

  An encrypted registration file or a repository definition file is used to register the repository, for authentication or validation reasons, such that a Hosting Appliance will trust that the image, when pulled from the registry, is authentic.

- Repository

  A repository is a set of containerized images. A repository can be shared by pushing it to a registry server. Different images in the repository can be labeled using tags. For example, `hpvsop-base`.

- runc

  A CLI tool for spawning and running containers according to the Open Container Initiative (OCI) specification.

- runq

  An open-sourced hypervisor-based Docker runtime environment, which is based on runc to run regular containerized images in a lightweight KVM or Qemu virtual machine.

- s390x

  The underlying architecture of IBM Z or LinuxONE mainframe.

- Secure Build

  The process of building the application code from a Git-like source repository into a container image for s390x architecture, signing the image by using the authentication keys, and publishing the image to the remote repository for later integration.

- Secure Service Container

  A container framework based on the runq technology, that is supported by the IBM Z or LinuxONE servers.

- Secure Service Container partition

  A type of logic partitions (LPAR) on the mainframe that runs the Secure Service Container framework.

- SSH

  The abbreviation of Secure Shell, which is a cryptographic network protocol for operating network services securely over an unsecured network by using public and private keys.

- Storage Pool

  A storage pool is a uniquely named collection of storage disks on which the appliance file system is mounted.

- System Administrator

  This role includes the system administrator of a machine, storage administrators, and network administrators.

- tag

  A tag is used to version images in a repository. For example, `latest`, `1.2.3.4-develop-a0d3aea`, or `s390x-develop-54a9045`.

- Workload

  The application and data provided and generated by a (running) Workload Image.

- Workload Data

  Workload user or workload client data, workload logs, workload secrets stored in the appliance.

- Workload Image

  A container-based image, provided by the Workload Vendor. An appliance only runs workload images which have been registered with the appliance through a repository definition file.

- Workload User

  The end user of a workload.

- Workload Vendor

  The creator of a Docker image running on top of Hosting Appliance.