

IBM Hyper Protect Platform: Applying Data Protection and Confidentiality in a Hybrid Cloud Environment

Bill White

Robbie Avill

Sandeep Batta

Abhiram Kulkarni

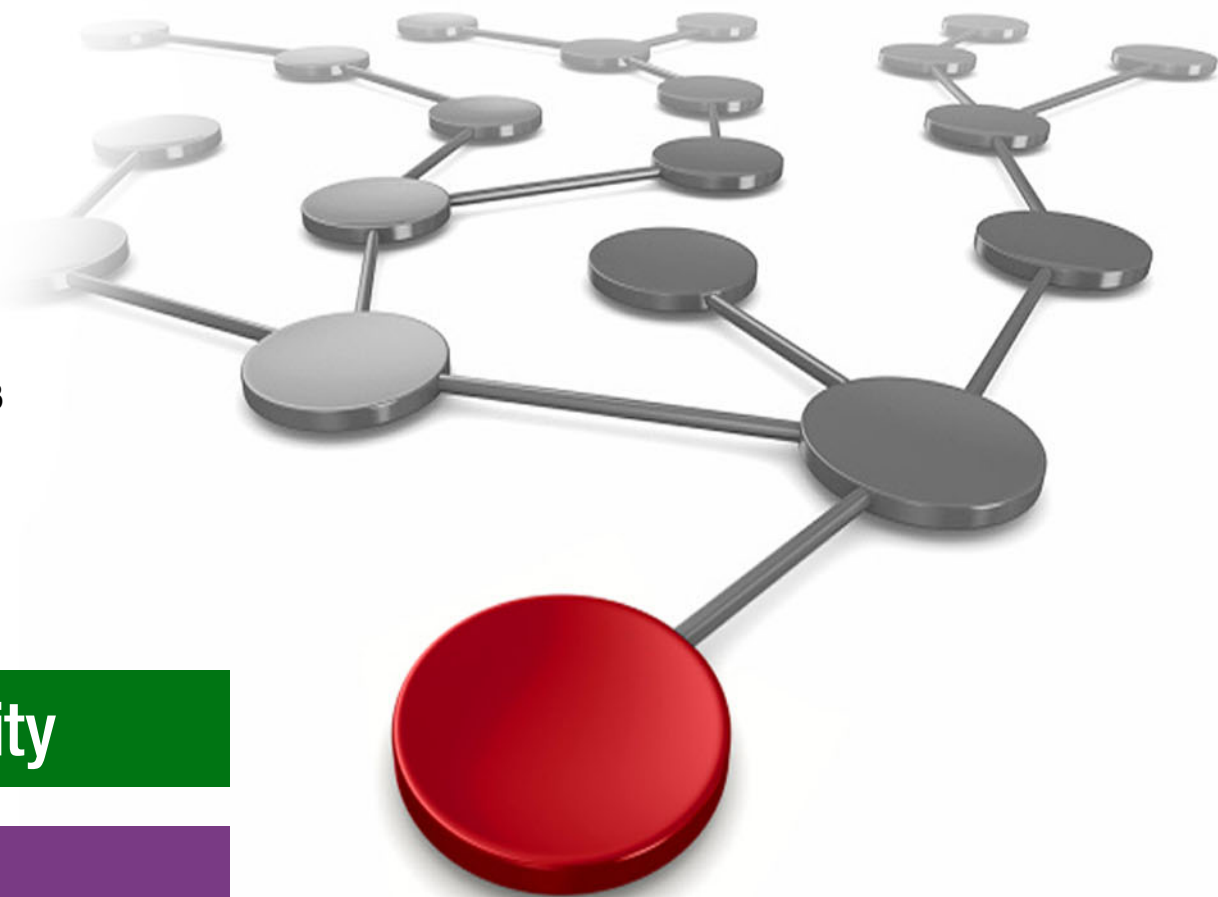
Timo Kußmaul

Stefan Liesche

Nicolas Mäding

Christoph Schlameuß

Peter Szmrecsányi



 **Security**

Cloud



IBM Redbooks

Applying Data Protection and Confidentiality in a Hybrid Cloud Environment

February 2024

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

First Edition (February 2024)

This edition applies to the IBM z15, IBM z16, IBM LinuxONE III, IBM LinuxONE 4, and IBM Hyper Protect Platform Second Generation.

© Copyright International Business Machines Corporation 2024. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	viii
Preface	ix
Authors	x
Now you can become a published author, too!	xi
Comments welcome	xii
Stay connected to IBM Redbooks	xii
Chapter 1. A hybrid cloud with data security in mind	1
1.1 Identifying the threat	2
1.2 Beyond regulatory and standard frameworks	3
1.3 Mitigating the threat	4
1.3.1 Technical assurance	4
1.3.2 A Trusted Execution Environment for your application	5
1.3.3 Reduced trust boundary and trusted computing base	6
1.3.4 Controlling your application with separation of duty	6
1.3.5 Exclusive and full control over your cryptographic key	7
1.3.6 Support for your application OCI images	7
1.3.7 Support for hybrid cloud	7
1.4 The solution explained	7
1.4.1 The technology underlying the Hyper Protect Platform	8
1.4.2 Features of the Hyper Protect Platform	10
1.4.3 Cryptography and Hyper Protect Crypto Service	13
1.4.4 Hyper Protect Secure Build	14
Chapter 2. Understanding the solution	15
2.1 IBM Hyper Protect services and a secure hybrid cloud	16
2.2 IBM Cloud Virtual Private Cloud	18
2.2.1 IBM Cloud virtual server instance on IBM LinuxONE	18
2.3 Hyper Protect Virtual Server	18
2.3.1 Bootloader	19
2.3.2 Volume encryption	19
2.3.3 Description of the contract	20
2.3.4 The attestation record	21
2.3.5 Logging	21
2.3.6 Hyper Protect layer services	21
2.3.7 Hyper Protect Virtual Server for VPC	22
2.3.8 Hyper Protect Virtual Server for IBM LinuxONE and IBM Z	23
2.3.9 Considerations when deploying workloads in HPVS instances	23
2.4 Hyper Protect Secure Build	24
2.5 Cryptographic agility is the key to SecDevOps	25
2.6 Hyper Protect Crypto Services	25
2.6.1 Accessing cryptographic services with HPCS	26
2.7 Crypto Express Network API for Secure Execution Enclaves	28
2.7.1 Security considerations	29
2.8 Storage and repositories in the cloud	29
2.8.1 Cloud object storage	29
2.8.2 Block storage	30

2.8.3	File storage	30
2.8.4	On-premises storage	30
2.9	Common usages	31
2.9.1	Securely bring applications to hybrid cloud.	31
2.9.2	Digital assets infrastructure.	32
2.9.3	Confidential AI.	33
2.9.4	Secure multi-party computation	34
2.9.5	Secure distributed cloud	35
Chapter 3. Making the infrastructure secure		37
3.1	The contract	38
3.1.1	The workload section	39
3.1.2	The workload volumes subsection	46
3.1.3	The env section.	47
3.2	Contract encryption	49
3.3	Contract certificates	53
3.4	Attestation	54
3.5	Logging for HPVS instances	56
3.6	Encrypting data volumes.	63
Chapter 4. Application development in a trusted environment		65
4.1	Securing the application lifecycle	66
4.1.1	Development.	67
4.1.2	Test	67
4.1.3	Build	67
4.1.4	Release.	68
4.1.5	Deployment.	68
4.1.6	Update	68
4.1.7	Application and service development	69
4.1.8	Working with the log	69
4.1.9	Deployment automation - Terraform	70
4.2	Build container image by using Hyper Protect Secure Build.	71
4.2.1	Determine readiness.	71
4.2.2	Install the secure build CLI	72
4.2.3	Create client and server certificates for secure build	72
4.2.4	Prepare user_data.yaml	73
4.2.5	Create the Hyper Protect Secure Build instance.	74
4.2.6	Configure the HPSB client with the HPVS IP address	75
4.3	Zero knowledge proofs: TLS server certificates and wrapped secrets	78
4.3.1	Passing secrets into a secure HPVS	78
4.3.2	Certificate benefits	78
4.3.3	Importing server certificate from contract	79
4.3.4	Random number generation	80
4.3.5	Reverse proxy	83
4.3.6	Basic web server (nginx) hardening	84
4.3.7	Offloading NGINX TLS to HPCS.	85
4.4	Trust in-depth based on boot flow attestation	85
4.5	Data storage	87
4.5.1	Encrypting block storage.	87
4.5.2	Encryption state	88
4.5.3	Upgrade, backup, and disaster recovery	91
4.5.4	High Availability.	91
4.6	Securing cloud native services	92

4.6.1 Confidential cluster	92
4.6.2 Confidential containers	93
4.6.3 Confidential service platform.	94
4.7 Secure supply chain with SLSA	95
4.7.1 Jenkins	96
4.7.2 Source-to-image (S2I).	96
4.7.3 GitHub Actions	97
Appendix A. Client contract setup sample files.	99
Sample YAML file with literal scalars	100
Sample YAML file with double-quoted scalars.	101
Sample script for certificate or key files	103
Appendix B. Creating a Hyper Protect Virtual Server for VPC	107
Using the IBM Cloud VPC UI	108
Appendix C. Additional examples for HPSB and HPVS	115
Hyper Protect Secure Build log	116
How to verify disk (volume) encryption with HPL13000I	118
Appendix D. Encryption keys explained.	121
What is a master key (MK)	122
What are data encryption keys (DEKs)	122
What are key encryption keys (KEKs)	123
Using and protecting keys	123
How encryption keys are created using GREP11	124

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.


COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <https://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

IBM®	IBM Z®	z/Architecture®
IBM Cloud®	IBM z16™	z/OS®
IBM Research®	Redbooks®	z/VM®
IBM Security®	Redbooks (logo)  ®	z16™
IBM Watson®	X-Force®	

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Zowe is a trademark of the Linux Foundation.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Red Hat, OpenShift, are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

Protecting workloads and sensitive data throughout their lifecycle is a great concern across all industries and organizations. Increasing demands to accelerate hybrid cloud adoption and integration are changing the way data is securely stored, processed, and accessed.

In addition, regulatory guidelines and standards are causing many businesses and organizations to implement zero trust policies and privacy enhancing techniques to restrict access to workloads as *state of least privilege* is established. A state of least privilege ensures that no user or workload has any more access to data than is necessary. Confidentiality and integrity assurance for *data at rest* and *data in transit* is typically provided through cryptography. Nevertheless, *data in use* is generally unencrypted while it is processed by the system, which can make data in use accessible to privileged users or workloads.

In the past, data owners relied upon operational assurance to control access to workloads and data. An operational assurance approach ensures that a service provider will not access customer workloads or data through specific operational procedures and measures. However, with today's constant, unpredictable, and always changing cyberthreats, operational assurance is not enough.

A more robust technical assurance approach that is hardware-based is needed. A Trusted Execution Environment (TEE) or confidential computing platform does just that. A TEE ensures that no one can access sensitive workloads and data while in use, not even the service provider. A TEE can also protect the CI/CD pipeline from bad actors, enforce supply chain protection, and provide code integrity through cryptographic proofs and encryption.

This IBM® Redbooks® publication outlines how to apply common concepts of data protection and confidentiality and make use of a privacy-enhancing technology-based solution that can be implemented in a hybrid cloud environment. It describes the TEE technologies that are offered with IBM Z® and IBM LinuxONE (such as IBM Secure Execution for Linux), and how the IBM Hyper Protect Platform uses them.

This publication discusses how the various IBM Hyper Protect services ensure zero trust data-centric security and data privacy end-to-end. It also illustrates the business value through specific use case scenarios, covering relevant aspects of workload creation and evidence collection for regulatory compliance of software supply chains.

This IBM Redbooks publication is for Chief Information Security Officers (CISOs), IT managers, security architects, security administrators, cloud application developers, and anyone who needs to plan, deploy, and manage data security and confidentiality in a hybrid cloud environment. The reader is expected to have a basic understanding of IT security and hybrid cloud concepts.

Authors

This book was produced by a working at IBM Redbooks.

Bill White is an IBM Redbooks Project Leader and Senior IT Infrastructure Specialist at IBM Poughkeepsie, New York.

Robbie Avill is a Solutions Architect with the IBM Z Client Acceleration Team for IBM Hyper Protect Services and IBM Cloud® products. He has worked at IBM for 8 years since joining as the first ever apprentice at the IBM Lab in the UK. Robbie has been involved in various projects over the years including IBM z/OS® Connect and IBM's Kubernetes service, as well as the Open Mainframe Projects, Galasa and Zowe.

Sandeep Batta started his career as a z/OS Systems Programmer. He consulted Fortune 100 clients in the NJ/NYC area, where he had a chance to work on game changing technologies that helped restore operations after disastrous events like 9/11. Sandeep created a “self-service” platform for developers in an IBM z/VM® farm, developed Cloud Deployment strategies, modernized front-end network architecture for IBM's internal infrastructure and developed offering management tools. Sandeep is currently a Lead Solutions Architect in the IBM Hyper Protect organization, where he works with clients in financial services, insurance, and digital assets sectors, to address their data-protection requirements.

Abhiram Kulkarni is the Software Architect for IBM Hyper Protect at IBM India Systems Development Lab, Bangalore. He received his Bachelor of Engineering degree at PESIT Engineering College, Bangalore. He has over 15 years of experience in Software development and joined IBM in 2013. In his recent roles, he has worked in various development projects across IBM Z. He has worked on projects that are related to Secure Service Container, Secure Execution catering toward the clients that need zero trust architecture and Confidential Computing in hybrid cloud environment.

Timo Kußmaul is a Solution Architect and Master Inventor at the IBM Research® and Development Lab in Böblingen, Germany. He holds a Dipl.-Inform. diploma in Computer Science from the University of Stuttgart, Germany. Timo works in the Hyper Protect Client Acceleration Team and has more than 25 years of experience in Software Development, application of AI and Search, Hybrid Cloud, Security and Confidential Computing. Timo is the author or coauthor of over 100 patents and multiple technical papers and books.

Stefan Liesche is an IBM Distinguished Engineer working with IBM Hybrid Cloud and Hyper Protect Services for IBM Z and IBM LinuxONE. His main focus is on security, transparency, and protection of data and services in flexible cloud environments. Stefan has worked in various areas as a technical leader within IBM, most recently as Chief Architect for IBM Cloud Hyper Protect Services and IBM Watson® Talent Portfolio. He designed and built AI driven solutions that transform recruiting and career decisions within global organizations, that not only enhance quality of decisions, but also allow HR functions to enhance fairness and tackle biases. Stefan also an innovator within the Exceptional Web Experience products for several years with a focus on open solutions and integration. He has 25 years of experience as a technical leader, collaborating with business partners and clients through joint projects, as well as within IBM's product development organization.

Nicolas Mäding is Principal Product Manager at the IBM Lab in Böblingen, Germany. He received his Dipl. Ing. Degree in Electrical and Information Technology at the Technical University of Chemnitz, Germany. He joined IBM in 2001 and worked in various development and management positions in IBM Systems Hardware Development. He joined the Z-as-a-Service organization as Release Manager of the Hyper Protect Hosting Appliance in 2018 and became product manager for the Secure Execution based offerings. In 2023, he was appointed as Principal Product Manager for the Hyper Protect Platform and Confidential Computing with Linux on Z. He is author or co-author of 12 patents and several technical papers.

Christoph Schlameuß is a Software Architect at the IBM Research and Development Lab in Böblingen, Germany. He holds a Dipl.-Inf. diploma in Computer Software Engineering from the University of Stuttgart, Germany. Christoph works in the Hyper Protect Services Innovation Team and has over 12 years of experience in professional software development. Including seven years of experience in developing Confidential Computing products like Secure Service Container and IBM Hyper Protect. He is the author or co-author of two patents.

Peter Szmrecsányi is a Solution Architect at the IBM Lab in Markham, Canada. He received an Electrical and Electronic Master of Engineering with Honours degree from The University of Birmingham in the United Kingdom. Peter has over 20 years of experience in the field of Information Technology. His areas of expertise include Hyper Protect Services (Confidential Computing on the IBM LinuxONE platform). Peter is the author or co-author of two patents.

Thanks to the following people for their contributions to this project:

Divya K Konoor (IBM Senior Technical Staff Member, IBM Hyper Protect Services IaaS)
IBM India

Rene Meyer (Principal IBM Cloud Technical Specialist)
IBM Germany

Louisa Muschal (Product Manager IBM Hyper Protect Services)
IBM Germany

Barry Silliman (Consulting IT Specialist)
IBM USA

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an IBM Redbooks residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at: ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, IBM Redbooks
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on LinkedIn:

<https://www.linkedin.com/groups/2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/subscribe>

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

<https://www.redbooks.ibm.com/rss.html>



A hybrid cloud with data security in mind

Moving workloads to a cloud infrastructure raises critical questions about data security and data privacy. A workload running in a cloud infrastructure owned by a service provider, maintained by the provider's administrators, and shared with other tenants requires a novel and complete approach to protecting data and developing applications in a trusted environment. The data security concerns might even be prohibitive when you consider applications for a hybrid cloud infrastructure, more so if you cannot validate the trust of all parties. There might also be regulatory compliance to adhere to or business-specific needs for higher levels of data security controls that mandate protection at every stage of a digital interaction or data lifecycle.

This chapter discusses the threats, concerns, requirements, and the solution for a hybrid cloud infrastructure with data security in mind. The following topics are discussed:

- ▶ 1.1, "Identifying the threat" on page 2
- ▶ 1.2, "Beyond regulatory and standard frameworks" on page 3
- ▶ 1.3, "Mitigating the threat" on page 4
- ▶ 1.4, "The solution explained" on page 7

1.1 Identifying the threat

As cyberattacks continue to increase, the cost and reputation impacts of data breaches remain a top concern across all businesses and organizations¹. To help understand how a cyberattack might threaten a system, application, or data, the well-known **STRIDE threat model** can be used. This model conceptualizes the potential threats into six categories:

1. Spoofing. An entity falsely identifies as another identity
2. Tampering with data. The malicious modification of data
3. Repudiation. An entity disputing its responsibility, ownership or authorship of data, resources or operations
4. Information disclosure. The exposure of information to unauthorized entities, this means to entities who are not supposed to have access
5. Denial of service. An attack that tries to make a system or resource unavailable
6. Elevation of privilege. An attack to get elevated privileges or privileged access to a system or resource

Another way to look at threats is the path that an attacker might use to gain access to a system, application, or data, referred to as an attack vector. There are many well-known attack vectors, some of which have been used to successfully attack applications and gain access to sensitive data (reference **IBM X-Force® Threat Intelligence Index 2023**). Typically, an attack vector corresponds to one or more of the STRIDE threats.

There are also different threats, mechanisms, and implications for managing and ensuring confidentiality, integrity, and availability depending on the state of the data. As shown in Figure 1-1, data is categorized in three distinct states:

1. Data in persistent storage is *at rest*.
2. Data traversing the network between a source and a destination is *in transit*, alternatively called *data in motion* or *data in flight*.
3. Data being processed by a system is *in use*. During processing, the data is typically stored in a non-persistent state in CPU cache or system memory.

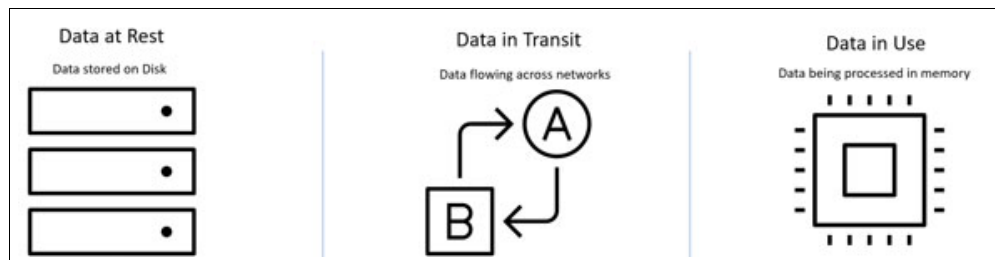


Figure 1-1 Data states

Today, cryptography is commonly used for data at rest and data in transit to provide both data confidentiality for stopping unauthorized viewing and data integrity for preventing or detecting unauthorized changes.

A well-established best practice is to protect data in transit by using cryptographic protocols like Transport Layer Security (TLS)². Such cryptographic protocols protect the confidentiality and integrity of the data in transit between endpoints in a public or private network.

¹ Reference: [Cost of a Data Breach Report 2023](#)

² Reference: [HTTPS encryption on the web, Google Transparency Report](#)

Data at rest should be stored in encrypted form only. [IBM Cloud Object Storage](#) or volume encryption, such as Linux Unified Key Setup (LUKS), encrypts the data before writing to persistent storage and decrypts it for reading. Thus, an attacker cannot access the data at rest even with access to the physical storage device.

However, data in use is generally unencrypted and easily accessible, as it is active data being processed by the system. For further discussion about protecting data in use, see 1.3, “Mitigating the threat” on page 4.

1.2 Beyond regulatory and standard frameworks

In addition to the concerns related to protecting data in use, data sovereignty³ states that data is bound to the laws and regulations of the country in which it is collected, stored, processed, and distributed and can be subject to data protection policies. Worldwide, there are many standards and regulations that are related to protecting data at rest and in transit. Although some of the standards specify the technologies required for compliance, encryption is applied to achieve compliance for most of them.

A broad framework like a zero trust architecture can also provide effective protection of an organization’s data. It works by assuming every connection, endpoint, and domain are considered a threat. See Figure 1-2. The goal of a zero trust strategy is to eliminate implicit trust and to continuously validate every stage of a digital interaction.

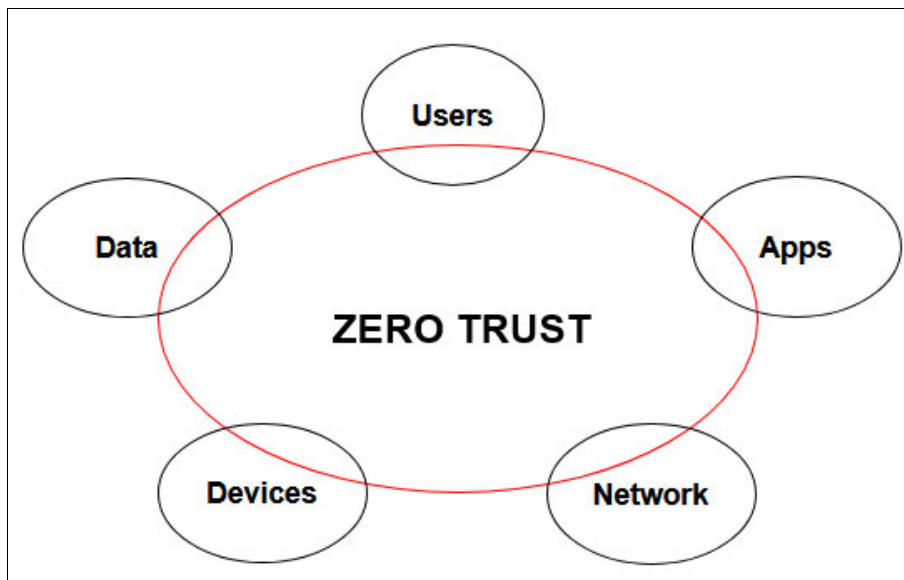


Figure 1-2 Zero trust framework threats

Common techniques for protection of data at rest and data in transit with a zero trust architecture are defined by the National Institute of Standards and Technology (NIST).⁴

However, other than the regulatory and standard frameworks, other areas of concern are provided in the following list:

- ▶ Protecting sensitive data that is in use by applications in a hybrid cloud infrastructure, even from privileged users

³ Refers to the notion that a country or jurisdiction has the authority and right to govern and control the data generated within its borders.

⁴ Reference: [NIST Special Publication 800-207, Zero Trust Architecture](#)

- ▶ Avoiding the need to trust potentially malicious privileged users or potentially vulnerable and compromised infrastructure and intermediary components in a hybrid cloud environment
- ▶ Achieving an air-gapped solution for applications and data from any potential malicious actor

In other words, identify the technologies and techniques that can help with the following requirements:

- ▶ Reduce hybrid cloud migration risk
- ▶ Secure sensitive data in use
- ▶ Control applications according to requirements
- ▶ Ensure compliance with regulatory requirements
- ▶ Cause zero additional effort for consumers
- ▶ Have zero impact on the functionality and availability of applications

1.3 Mitigating the threat

Cyberthreats are constant, unpredictable, and always changing. Therefore, many businesses and organizations use the US Department of Commerce [National Institute of Standards and Technology](#) (NIST) standards, guidelines, and recommendations as a baseline, then apply stronger policies and controls as needed or required. IBM also aligns with NIST guidelines and is committed to embedding security and privacy into the design of all products and services.⁵

A zero trust architecture can help prevent unauthorized access to data and services. However, in a hybrid cloud infrastructure, access control policies or operations to ensure isolation of applications and data in use might not satisfy all business-specific requirements. Ideally, data security and isolation are implemented in all layers of the hybrid cloud infrastructure, creating an air-gapped environment.⁶

The properties and features that are required to prevent or mitigate cyberthreats include the following characteristics:

- ▶ Technical assurance
- ▶ A Trusted Execution Environment for your application
- ▶ Reduced trust boundary and trusted computing base
- ▶ Controlling your application with separation of duty
- ▶ Exclusive and full control over your cryptographic key
- ▶ Support for your application OCI images
- ▶ Support for hybrid cloud

1.3.1 Technical assurance

“Computer security assurance is the degree of confidence one has that the security measures, both technical and operational, work as intended to protect the system and the information it processes.”⁷

Technical assurance, or security assurance by technical measures, can be distinguished from operational assurance, or security assurance by operational measures. Technical assurance

⁵ Reference: [IBM Security® and Privacy by Design \(SPbD@IBM\)](#)

⁶ An air-gapped environment has no direct connection to the internet or to any other computer that is connected to the internet.

⁷ <https://csrc.nist.gov/publications/nistpubs/800-12/800-12-html/chapter9.html>

is provided by the hardware, the firmware, and the software stack of a system and is included in the system technically. In contrast, operational assurance addresses whether required procedures and regulations are followed and are compliant to operational requirements.

The difference between technical assurance and operational assurance is illustrated in Figure 1-3. Operational assurance ensures that service providers do not access client workloads. Technical assurance ensures that service providers cannot access client workloads.

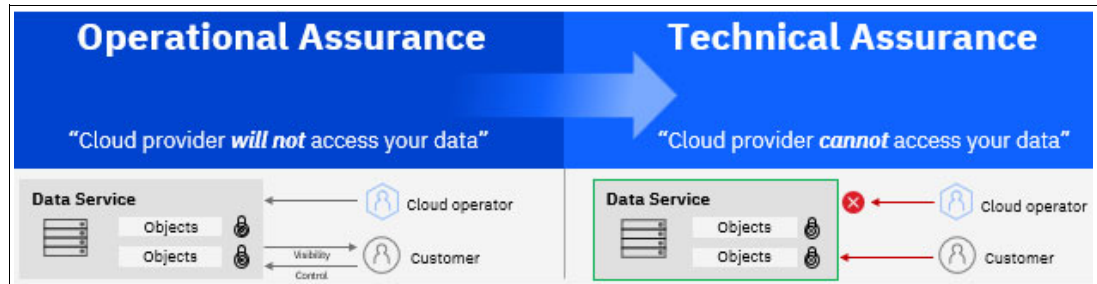


Figure 1-3 Technical assurance versus operational assurance

1.3.2 A Trusted Execution Environment for your application

A hardware-based Trusted Execution Environment (TEE) is an execution environment that provides hardware-based technical assurance of the following properties:

- ▶ Data confidentiality. Unauthorized actors cannot view data while it is in use within the TEE.
- ▶ Data integrity. Unauthorized actors cannot add, remove, or alter data while it is in use within the TEE.
- ▶ Code integrity. Unauthorized actors cannot add, remove, or alter code that is running in the TEE. Examples of unauthorized actors:
 - other applications on the host system
 - the host operating system and hypervisor
 - other tenants of the host system
 - the cloud provider
 - privileged actors like system administrators
 - parties with physical access to the hardware
- ▶ Code confidentiality. The TEE protects the code while in use from being viewed or accessed by unauthorized actors.
- ▶ Programmability. The TEE can be programmed with arbitrary code.
- ▶ Attestation. The process with which the TEE can provide evidence or measurements of its origin and current state. This evidence can be verified by another persona, for example the auditor persona, who can decide whether to trust the application running in the TEE or not. Typically, this evidence is represented as an attestation record. The contents of this record can be verified against workload and environment expectations. The attestation record is signed by a trusted key that is anchored in hardware that can be vouched for by a trusted manufacturer. This signature provides assurance that the attestation record was created by the correct component and was not altered by an unauthorized entity.

The TEE provides code and data confidentiality and integrity. The TEE thus isolates your application code and data from access and modification by unauthorized actors. The TEE isolates your application code and data from access and modification by other privileged or non-privileged actors, which includes potentially malicious administrators, and other tenants.

This process must be technically assured by an implementation that is anchored in hardware. No unauthorized entity is allowed to access or modify the code or the data of the application.

1.3.3 Reduced trust boundary and trusted computing base

You can run your application in a TEE with technically assured isolation that you can remotely verify through attestation. Instead of trusting all the components and actors in your hybrid cloud infrastructure, you reduce the trust assumptions to the hardware-based implementation of the underlying TEE and the attestation protocol. In this scenario, components and actors can include those of your public cloud provider.

1.3.4 Controlling your application with separation of duty

The infrastructure needs to ensure exclusive control over your application and its properties is provided to authorized personas only. This means that there is a process in place that ensures only you and authorized personas can control the application code, the properties of the runtime environment, and how data at rest is protected, such as controlling the seeds for volume encryption. In addition, only you and authorized personas can control cryptographic keys and can pass secrets, such as cryptographic seeds for key derivation, to the application.

Separation of duty

The infrastructure needs to ensure separation of duty that allows multiple personas by defining different aspects and properties of your application. Also, a process needs to be defined to allow the different personas to cooperate securely. A persona must be able to define the properties and secrets of your application or its environment. Other personas must be prevented from accessing, modifying, or overwriting these properties and secrets. The infrastructure provides technical assurance of separation of duties and enables secure cooperation of the various personas.

Multiple personas can securely pass secrets to your application

A persona must be able to pass secrets to the application in a secure way, which means the secret must be protected from other personas, the hybrid cloud infrastructure, and other actors.

For example, secrets can comprise TLS certificate and keys, cryptographic keys, cryptographic seeds used for key derivation, seeds used for volume encryption, and so on.

Separation of duty allows a first persona, a Workload Provider, to define the [Open Container Initiative \(OCI\)](#) image for the application that should be run in a confidential computing environment. A second persona, a Workload Deployer, can define the TLS certificate and key that should be used by the application. This way, both personas cooperate to setup the application in confidential computing. The separation of duty process ensures that the Workload Provider cannot access or tamper the TLS certificate and key, and the Workload Deployer cannot access or tamper the OCI image.

In addition, both personas can cooperate to define a cryptographic seed for LUKS based volume encryption or another type of cryptographic encryption or signature. Here, the first Workload Provider persona defines a first part of the seed, and the Workload Deployer defines a second part of the seed. The separation of duty process ensures that the seed parts are provided to the secure enclave without exposing them to the other persona by keeping them confidential from each other. In the reference architecture both parts of the seed are combined in the TEE to create the LUKS encryption key. This means that the resulting combined seed never leaves the TEE. Each persona knows only one part of the

seed. Thus this process prevents individual personas from being able to re-create the LUKS encryption key and access data at rest.

1.3.5 Exclusive and full control over your cryptographic key

A cloud hardware security module (HSM) must provide full and exclusive control over your cryptographic keys. Without such mechanisms, privileged actors in the hybrid cloud environment can use known attack vectors to gain access to your keys and cryptographic material. A malicious privileged actor might then tamper or misuse your keys and secrets, for example, by maliciously decrypting or signing sensitive data.

Note that control over your cryptographic keys must be technically *assured* by using hardware-based mechanisms.

1.3.6 Support for your application OCI images

The TEE must support common programming and deployment models, such as OCI images. The TEE does not require a specific programming model for your code. Also, it does not impose special limitations or requirements on your code. This means you can run your own OCI images in the TEE without code changes. The application developer does not need to learn a specific programming model. There is code compatibility between the TEE and non-confidential computing environments.

1.3.7 Support for hybrid cloud

The infrastructure must support a hybrid cloud environment, comprising support for on-premises environments and provide confidential computing as a service in the cloud.

In addition, the infrastructure must not impose limits on scalability of your application. This means, the infrastructure must scale with your performance and memory requirements.

1.4 The solution explained

Confidential computing⁸ can help enhance the zero-trust architecture by providing a trusted execution environment for applications, even in an untrusted environment. The hardware-based security mechanism that is offered by confidential computing enables the processing and storage of sensitive data in a safe enclave that is isolated from the host system and other potentially vulnerable components.

This isolation is technically assured. Hence, the hardware-based and firmware-based protection mechanisms provide technical means for ensuring that unauthorized actors are prevented from accessing your application and your data. The access is prevented even if an attacker overcomes the access control systems of the cloud provider or gains access to another vulnerable component in the cloud infrastructure. Confidential computing adds a layer of data security on top of a zero trust architecture by providing technically assured isolation, sometimes called a *virtual air-gap*, of applications in TEEs. This virtual air-gap prevents malicious inside actors like authorized administrators from dumping the memory of your application or directly accessing data of your application. The virtual air-gap prevents access to the file system or memory of your application.

⁸ Confidential computing is the protection of data in use by performing computation in a hardware-based, attested TEE. Reference: [A Technical Analysis of Confidential Computing](#), a publication of the Confidential Computing Consortium, November, 2022.

The Hyper Protect Platform as the confidential computing solution can help solve the data protection concerns in a hybrid cloud environment, an on-premises environment, and in an IBM Cloud-based SaaS model.

The solution consists of the following components:

- ▶ A Trusted Execution Environment for containerized applications in hybrid cloud environments.
It is available as IBM Hyper Protect Virtual Server on IBM LinuxONE and IBM Z platforms for on-premises environments or as IBM Hyper Protect Virtual Server for Virtual Private Cloud (VPC) in IBM Cloud.
- ▶ A hardware security module (HSM) such as IBM Hyper Protect Crypto Services in IBM Cloud and IBM LinuxONE or IBM Z platform with Crypto Express features for on-premises use.
- ▶ A component for secure CI/CD: IBM Hyper Protect Secure Build

The solution provides a set of secure processes and patterns that are explained in the subsequent sections. Also, the components that make up the solution are discussed in more detail in Chapter 2, “Understanding the solution” on page 15.

1.4.1 The technology underlying the Hyper Protect Platform

Modern data-serving systems must immediately scale up and scale out in size, performance, and features. Virtualization is a key technology that enables a hardware system to achieve this level of scaling performance and maintain dense packaging and optimal use of resources.

Virtualization is one of the core strengths of the IBM LinuxONE and IBM Z platforms, with the goal of maximizing usage of computing resources and lowering the overall cost and resource requirements for running critical workloads for enterprises.

The embedded architecture and hardware with the IBM LinuxONE and IBM Z platforms is designed around the ability to partition resources to be used independently in distinct virtualized environments. The resources include compute, memory, and I/O connectivity of both storage and network.

Hypervisors are a core part of the virtualization technology stack with the IBM LinuxONE and IBM Z platforms. They are designed to enable simultaneous execution of multiple operating systems and allocate the correct amount of virtualized resources. Hypervisors are necessary to securely run, manage, and isolate virtual servers or logical partitions on the IBM LinuxONE and IBM Z platforms.

IBM Secure Execution for Linux

The IBM Hyper Protect Platform uses Secure Execution for Linux. This is a hardware-based security technology, which was introduced with the IBM Z and IBM LinuxONE platforms specifically for Kernel-based Virtual Machine (KVM) guests⁹. It is designed to provide scalable isolation for individual workloads to help protect them from not only external attacks, but also insider threats. Secure Execution for Linux can help protect and isolate Linux workloads on-premises, or on IBM Z and IBM LinuxONE in hybrid cloud environments.

Secure Execution for Linux isolates and protects KVM guests from hypervisor access. The hypervisor administrator can manage guests and deploy workloads, but cannot view data.

⁹ Virtual machines (VMs) are also referred to as guests or images.

Multiple applications that are running in a logical partition (LPAR) under KVM have fully isolated environments.

To achieve this, the IBM LinuxONE and IBM Z firmware provides an *ultravisor*. The ultravisor is a trusted firmware component. It uses memory-protection hardware, and the owner of a given KVM guest can securely pass secret information to the ultravisor by using the public host key.

Figure 1-4 illustrates the Secure Execution for Linux environments running in KVM guests in an IBM LinuxONE or IBM Z platform.

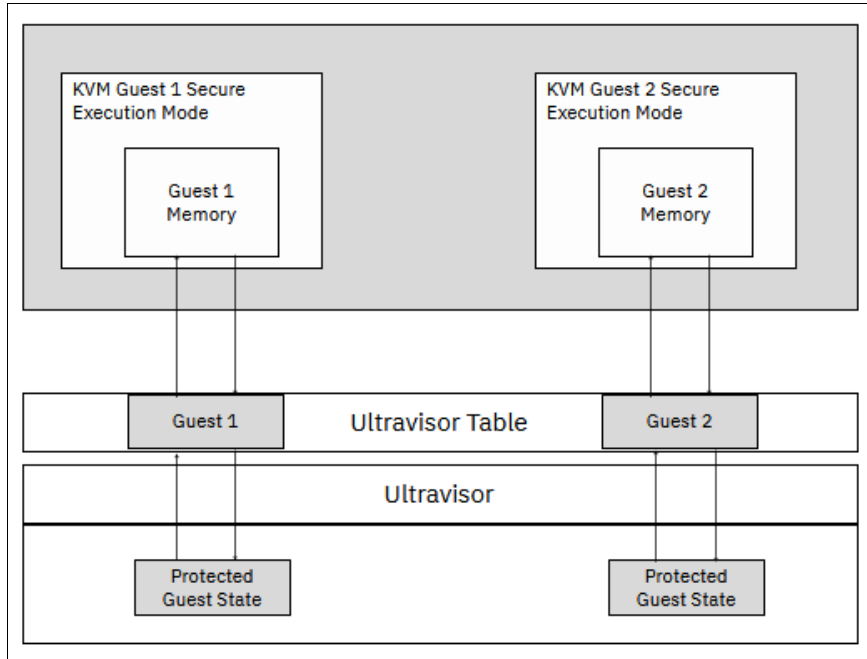


Figure 1-4 Secure Execution for Linux environment

For more information, refer to [IBM Secure Execution components](#).

Kernel-based virtual machines

Kernel-based virtual machine (KVM) is a key software technology for IBM LinuxONE and IBM Z platforms. It is a Type 2 hypervisor¹⁰ that provides simple, cost-effective virtualization technology for Linux workloads. It allows sharing and managing of allocated resources and can coexist with other types of virtualization technologies that are simultaneously running in IBM LinuxONE or IBM Z.

One of the advantages of KVM virtualization is the familiar standard Linux user interfaces for open source developers, which can help make adoption and integration easier with hybrid environments.

Together with Secure Execution for Linux, data and workloads that run in KVM guests are isolated and protected from being inspected or modified from the moment the guest is built, through the boot process, and during workload execution.

¹⁰ A Type 2 hypervisor runs as a software layer. It does not run directly on the underlying hardware, but rather like an application in an OS, sharing and managing its allocated resources with virtual machines.

KVM on IBM LinuxONE and IBM Z is supported through the following Linux distribution partners:

- ▶ Red Hat Enterprise Linux (RHEL)
- ▶ SuSE Linux Enterprise Server
- ▶ Canonical Ubuntu Server

Isolation through IBM Processor Resource/Systems Manager

The IBM LinuxONE and IBM Z platforms have a unique implementation of its hypervisor at the hardware and firmware level. It is part of the base system that fully virtualizes all system resources and runs without any extra software. This type 1 hypervisor¹¹ runs directly on bare metal. With it, multiple isolated partitioned environments can be created on the same physical server. These isolated environments are known as logical partitions (LPARs).

EAL 5+ isolation and cryptographic key protection

IBM LinuxONE and IBM Z platforms feature EAL 5+ isolation. EAL5+ is a regulatory certification for logical partitions that verifies the separation of each partition to improve security.¹² Therefore, you can run many virtual servers concurrently. The platforms can isolate and protect each LPAR, whether z/VM, z/OS, or KVM, as though they were running on physically separated servers.

The LPARs are isolated from each other, but VMs or containers within the same LPAR are not as isolated. Secure Execution for Linux is an IBM LinuxONE or IBM Z hardware capability. By using Secure Execution for Linux, a KVM hypervisor can isolate virtual machines and containers from each other within an LPAR.

A Processor Resource/Systems Manager-based LPAR is the only technology that is commercially available to provide this certified level of isolation between logical partitions.

In addition, cryptographic key protection that is used by Secure Execution for Linux, is achieved by dedicated cryptographic coprocessors. The CP Assist for Cryptographic Function (CPACF) delivers cryptographic and hashing capabilities in support of key protection operations. The Crypto Express adapter is used to create the fortified data perimeter by using the IBM LinuxONE or IBM Z protected key in which the keys that are used in the encryption process are not visible to the applications and operating system.

Each LPAR on an IBM LinuxONE or IBM Z platform has its own uniquely generated and assigned cryptographic keys that are held in a secure hardware area. This configuration provides a level of cryptographic isolation between secure environments that required by many regulatory compliance frameworks.

1.4.2 Features of the Hyper Protect Platform

The features that are offered by the Hyper Protect Platform create the foundation for an end-to-end secure environment. This platform provides protection of code and data and supports a consistent developer experience.

Bring your container runtime

For OCI images, Hyper Protect Virtual Server (HPVS) provides a trusted container runtime that provides the benefits and properties of a TEE. HPVS supports any OCI images that are built for IBM LinuxONE and IBM Z, which means images do not need to be adapted

¹¹ A type 1 hypervisor runs directly on the underlying computer's physical hardware, interacting directly with its CPU, memory, and physical storage and network I/O.

¹² Reference to EAL 5+ isolation:

https://www.bsi.bund.de/SharedDocs/Zertifikate_CC/CC/Serveranwendungen_Virtualisierung/1133.html

specifically for HPVS. The application code does not need to be changed and does not need to adhere to a specific programming model. You can run your existing application code and images and on the Hyper Protect Platform.

Virtual air-gap: memory protection and isolation

The IBM Hyper Protect Platform uses IBM Secure Execution for Linux to ensure data and code confidentiality and integrity of the deployed application, including against privileged users and infrastructure components.

Separation of duty

HPVS supports separation of duty with predefined personas, as described in Table 1-1. The predefined personas are based on least privilege and zero trust principles. There is no assumed trust for what is expected to be deployed.

Table 1-1 Personas for the separation of duties

Persona	Description
Container Image Provider	An application can consist of one or more container images. The Container Image Provider is responsible for building the images in a secure manner and according to best practices. This ensures that the images are valid and free from vulnerabilities. Logs and evidence of the build are retained for potential future use for auditing.
Workload Provider	This can be the same persona as the Container Image Provider. Alternatively, the Workload Provider is a separate persona, which can combine container images from different sources or different Container Image Providers. The Workload Provider persona defines one or more containers to be deployed and defines certain properties of the application environment, such as seeds used for data volume encryption. No one else can change the container images or redefine the application properties that are specified by the Workload Provider. For this purpose, the Workload Provider uses the workload section of the contract to define the container images and application properties in a secure manner. They can then pass the encrypted contract section to other personas without exposing the content of the workload contract section. Workload section is described under “Contract mechanism” on page 12.
Workload Deployer	This persona is responsible for deploying an HPVS instance for running the application in the hybrid cloud environment. The Workload Provider provides the encrypted workload contract section to the Workload Deployer. This allows the Workload Deployer to supplement the contract section that is provided by the Workload Provider with an additional environment contract section that defines further properties of the application and its environment. Some of the properties include information such as defining where the logs should be sent to and seeds for data volume encryption. The Workload Deployer is responsible for the application deployment and application availability. The following list describes some of the aspects of the Workload Deployer : <ul style="list-style-type: none"> ▶ It can control the networking, compute, and storage resources made available to the application. ▶ It can influence network traffic in and out of the application. ▶ It cannot change the container images to be deployed ▶ It cannot change the application properties that are defined by the Workload Provider. If the Workload Provider and the Workload Deployer specify seeds for data volume encryption, both seeds are combined. This means that individual personas do not have enough information for decrypting encrypted data volumes.

Persona	Description
Auditor	The Auditor is the persona with responsibility for verifying the integrity of the HPVS instance and the deployed application. For example, the auditor ensures that the expected container image and application properties are used by the application. It does this by obtaining and verifying a trusted attestation record. The contents of this attestation record can be verified against the expected contract sections, comprising the container image definition and the application properties. The Auditor, or possibly another persona, like the Workload Provider or Workload Deployer, can choose to sign a contract before it is passed as input. A contract signature is an optional feature that can be used with the contract. Contracts that are in plain text or encrypted can be signed.
Infrastructure/System admin	This role includes the system or cloud administrator and support personas of the infrastructure like a Site Reliability Engineer (SRE). This role has responsibility for the underlying hardware and infrastructure, such as networks, but must not be able to access confidential data or tamper with the application.

Contract mechanism

HPVS uses a contract mechanism to enable the Workload Provider and the Workload Deployer personas to define the container images and the properties of the application and its environment in a secure way. The contract is a document comprising multiple sections, which can be independently encrypted. The contract can be signed and is provided to HPVS during deployment. During the initialization process of the HPVS instance, the Hyper Protect Container Runtime (HPCR) decrypts the contract, verifies the contract signature, and creates the passphrase that is used for volume encryption based on the seeds that are contained in the contract. The container images are set up as defined in the contract according to the properties in the environment variables.

The contract has several sections, two of these are mandatory:

- ▶ **Workload (mandatory).** This section contains the definition of the application workload in the form of a docker compose file or a pod descriptor. It defines one or more container images, the container image registry where it resides, and the information and credentials that are required to download and validate the image.
The section can also comprise information about data volumes, the seed for deriving the volume encryption passphrase, and environment variables.
- ▶ **env (mandatory).** This section describes the environment for the application. It comprises several subsections to define information about logging, such as where the logs should be sent to; data volumes; another seed for deriving the volume encryption passphrase; environment variables; and optionally, the public part of the contract signing key.
- ▶ **attestationPublicKey (optional).** This section provides a public RSA key, which is used to encrypt the attestation document.
- ▶ **envWorkloadSignature (optional).** This section contains the signature of the other sections of the contract, pinning to a specific workload part to the env part. An Auditor, a Workload Provider, or a Workload Deployer persona can choose to sign a contract before it is passed as input.

Only the Hyper Protect Platform can decrypt an encrypted contract. Therefore, by using the contract mechanism, the Workload Provider persona can define and encrypt the workload section of the contract, then pass it to the Workload Deployer persona. This way, the Workload Provider can hide the content of the workload section of the contract, such as the actual container images of the application, from the Workload Deployer, and allow the Workload Deployer to provision the HPVS instance for the application.

The Workload Deployer can define and encrypt the env section of the contract. The Workload Deployer then combines the workload and the env section, optionally adds the envWorkloadSignature and the attestationPublicKey sections, and then deploys the HPVS instance using the contract. As both sections of the contract are encrypted, no intermediate infrastructure component and no other party including privileged actors can view the contents of the contract. By adding the envWorkloadSignature, the contract can be protected against modification or tampering.

Attestation

The HPVS instance provides an Attestation Record that is securely generated and signed by the HPCR during instance initialization. This Attestation Record is made available to the containers in this HPVS instance. The signing key is published and can be validated to a 3rd-party certificate authority. Optionally, the Attestation Record can be encrypted by a public key that is defined by the Workload Deployer and provided in the Contract. It is a best practice that the Auditor provides the public key to the Workload Deployer and is part of the signature of the envWorkloadSignature of the contract. The encryption provides proof to the Auditor that no one can replay the attestation record.

The Attestation Record contains measurements of the original base image, the compressed root filesystem, and the cloud initialization options, which include the contract of this HPVS instance.

Attestation enables the Auditor to validate the integrity of the HPVS instance and the integrity of the contract. The Auditor can compare the attestation record against the reference values specific for the HPCR. The Auditor can also compare the checksum of the user data element of the attestation record against the checksum of the expected contract. If this validation succeeds because the checksums are identical, then this proves that the HPVS instance uses the expected contract. This means that the HPVS instance will run the expected environment and container images that are defined in the workload section of the contract.

Data volume encryption

The data volume that can be attached to an HPVS instance is protected by Linux Unified Key Setup (LUKS) volume encryption. The LUKS passphrase is automatically derived from the seeds that are provided by the Workload Provider persona and the Workload Deployer persona. This means that the Workload Provider or the Workload Deployer individually cannot re-create the LUKS passphrase because they know only their respective seed. Also, no other persona or party can re-create the LUKS passphrase because they do not know any of the seeds that are used for passphrase derivation.

1.4.3 Cryptography and Hyper Protect Crypto Service

A hardware security module (HSM) is a device or service that safeguards and manages secrets, such as cryptographic keys, and performs cryptographic functions such as key creation, key derivation, encryption, decryption, and a signature. An HSM contains one or more cryptographic processors. A cloud HSM is a cloud service that provides the same functions as a physical HSM.

Hyper Protect Crypto Services (HPCS) is a hybrid cloud key management service, which is based on FIPS 140-2 Level 4 certified hardware on the IBM LinuxONE or IBM Z platform.

HPCS supports various programming models like PKCS11 or GREP11. GREP11 is a stateless programming model with which cryptographic functions are run in the HSM. Cryptographic material, such as keys, are created in the HSM but are stored outside of the HSM by the client application.

An application deployed to HPVS can use a stateless cryptographic API like GREP11 to create and use cryptographic keys. It can store the keys in the boundary of the HPVS instance on attached devices and use volume encryption and protection of memory and data in use.

You can use separation of duty that is enabled by the contract concept and cloud HSMs to design and implement advanced cryptographic mechanisms for deriving keys from combined seeds. Combined seeds are a combination of different seeds owned by Workload Provider and Workload Deployer.

1.4.4 Hyper Protect Secure Build

The application container images must be built in a standardized, repeatable manner, and built by using secure components. The Container Image Provider persona is responsible for following state-of-the-art CI/CD processes and best practices.

In addition, the Container Image Provider can use Hyper Protect Secure Build (HPSB) to build a trusted container image within a secure enclave that is provided by HPVS. The enclave is isolated such that developers can access the container only by using a specific API and the cloud administrator cannot access the contents of the container. Therefore, the image that is built can be highly trusted. Specifically, the build server cryptographically signs the image, and a manifest, which is a collection of materials that are used during the build and which can be used for audits. Because the enclave protects the signing keys, the signatures can be used to verify whether the image and manifest are from the build server, and not elsewhere.

In summary, HPSB uses HPVS to protect the build process, the build results, the build evidence, and the signing keys for signing the images and the manifest against malicious privileged actors. This prevents potential malicious insiders from inserting malware or otherwise tampering with the application container images and from stealing the signing keys for container image signature.

More detailed descriptions of the components that make up the Hyper Protect Platform, as well as some common use cases can be found in Chapter 2, “Understanding the solution” on page 15.



Understanding the solution

With the integration of IBM Hyper Protect services and confidential computing, you can achieve end-to-end data and workload protection in a hybrid cloud environment. The IBM Hyper Protect Platform is a feature of IBM LinuxONE and IBM Z that offers hardware-level security and isolation for virtual servers. It not only protects data and workloads in production, but also allows you to securely build, deploy, and manage mission-critical applications in a hybrid cloud environment.

The hybrid cloud solution that is discussed in this chapter consists of trusted virtual servers and services that ensure your workloads and data are always secure, private, and protected from internal and external threats.

This chapter contains the following topics:

- ▶ 2.1, “IBM Hyper Protect services and a secure hybrid cloud” on page 16
- ▶ 2.2, “IBM Cloud Virtual Private Cloud” on page 18
- ▶ 2.3, “Hyper Protect Virtual Server” on page 18
- ▶ 2.4, “Hyper Protect Secure Build” on page 24
- ▶ 2.5, “Cryptographic agility is the key to SecDevOps” on page 25
- ▶ 2.6, “Hyper Protect Crypto Services” on page 25
- ▶ 2.7, “Crypto Express Network API for Secure Execution Enclaves” on page 28
- ▶ 2.8, “Storage and repositories in the cloud” on page 29
- ▶ 2.9, “Common usages” on page 31

2.1 IBM Hyper Protect services and a secure hybrid cloud

IBM Hyper Protect services that are offered with IBM Cloud and the IBM Secure Execution for Linux¹ on IBM Z and IBM LinuxONE platforms can provide a solution that isolates workloads and protects sensitive data throughout its lifecycle, regardless if its state is at rest, in transit, or in use.

Figure 2-1 illustrates the components of an example secure hybrid cloud solution. The solution consists of a public cloud domain, represented by the IBM Cloud, and a private cloud domain, represented by an on-premises cloud. Each cloud domain has multiple components that provide specific isolation and protection capabilities.

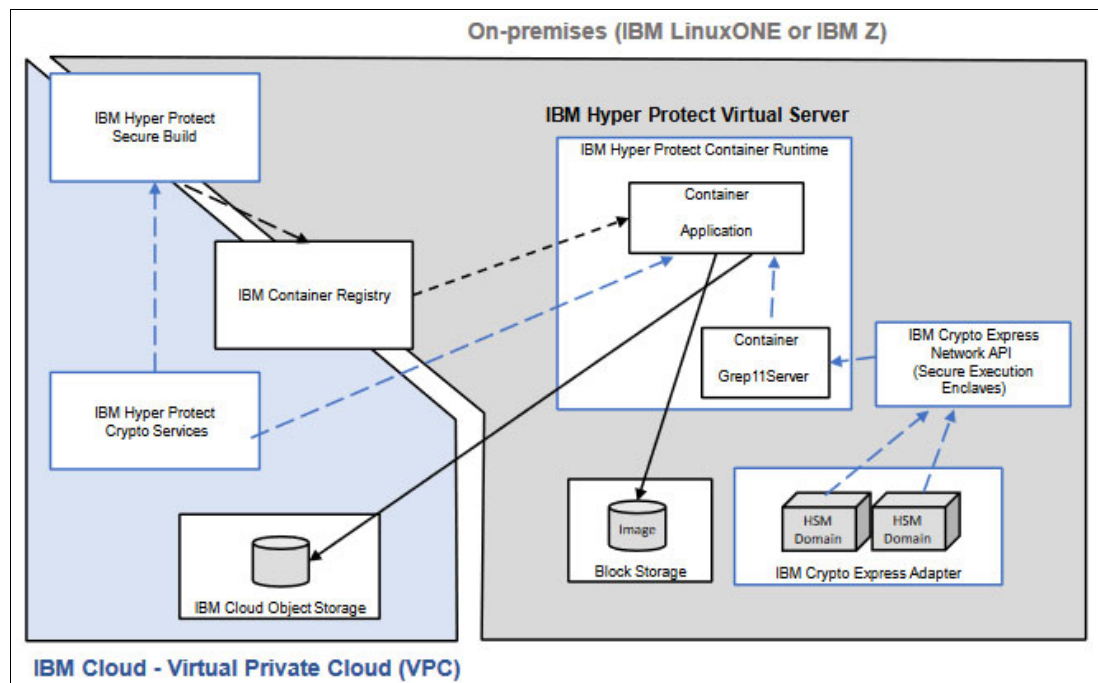


Figure 2-1 Secure hybrid cloud solution example

The IBM Cloud Virtual Private Cloud (VPC) domain includes the following components:

- ▶ Hyper Protect Secure Build. A trusted container image within a secure enclave that is provided by Hyper Protect Virtual Server.
- ▶ IBM Container Registry (ICR). A highly available, scalable, and encrypted private image registry. A storage and distribution service with public or private images that are used to create containers. The ICR is hosted and managed by IBM.
- ▶ Hyper Protect Crypto Services. A single-tenant key management service with which you can create, import, rotate, and manage keys with standardized APIs.
- ▶ IBM Cloud Object Storage (COS). A flexible storage for unstructured data, with additional services like secure encryption, backup and recovery, data archiving, and fast data transfer.

¹ IBM Secure Execution for Linux allows you to create a confidential computing environment that has encrypted Linux images running on a public, private, or hybrid cloud with data in-use protection. It requires feature codes [0115](#) and [3863](#).

The IBM LinuxONE or IBM Z on-premises cloud domain includes the following components:

- ▶ Hyper Protect Virtual Server. Deploy isolated workloads, protected by IBM Secure Execution for Linux, which is also known as confidential computing or secure enclave.
- ▶ Hyper Protect Container Runtime. Consists of different components or services that use certain sections in the [contract](#) to ensure data protection. The HPCR has container runtime support, and the image is not SSH enabled. It is a locked-down image.
- ▶ Block storage. Allows for the creation of storage volumes to which Linux on IBM Z images can connect. The storage volumes can be encrypted to protect data at rest.
- ▶ Crypto Express adapters. Tamper sensing and responding FIPS 140-2 level 4 hardware security modules (HSMs) that can perform advanced symmetric and asymmetric cryptographic operations and can securely store encryption keys.
- ▶ Crypto Express Network API for Secure Execution Enclaves. Runs in an IBM Secure Service Container LPAR and provides a REST API for application access to the Crypto Express adapters and domains.

The following Hyper Protect components can be deployed either on-premises or in IBM Cloud:

- ▶ Hyper Protect Virtual Server (HPVS)²
- ▶ Hyper Protect Secure Build (HPSB)
- ▶ IBM Container Registry (ICR)

There are advantages to using these components on IBM Cloud. Because the infrastructure and management of IBM Cloud are the responsibility of IBM, no on-premises staff need to be trained to operate these resources. On IBM Cloud, the resources are ready to be used. No resource provisioning or acquisition is necessary, and no time needs to be spent on design, install, and setup of the infrastructure.

There are also disadvantages to running these components on IBM Cloud. Thought must be given to the availability of these components, so redundant internet links or direct links might be needed.

Consideration should also be given to where the developer staff resides or works. If the staff works remotely, then IBM Cloud might be more suitable for HPSB and ICR as it can be more easily accessible to the development staff.

If the development staff is on-site, then an on-premises deployment might make more sense as it is more easily secured from outside tampering or access violation and can be more readily available as there are no dependencies on external network links.

Note: HPSB has documentation for both [IBM Cloud deployment](#) and [on-premises deployment](#).

The services and components that make up the secure hybrid cloud solution are described in subsequent sections.

² On-premises offerings include HPVS for IBM LinuxONE or IBM Z, and the IBM Cloud offering includes HPVS for VPC.

2.2 IBM Cloud Virtual Private Cloud

IBM Cloud Virtual Private Cloud (VPC) is a public cloud offering with which you create a private cloud computing environment on a shared public cloud infrastructure. By using an [IBM Cloud VPC](#) you can define and control a highly resilient virtual network that is logically isolated from all other public cloud tenants. Hence, creating a private, secure place on the public cloud.

Like with any other public cloud offering, you can choose the required compute, storage, and networking resources to be provisioned with maximum availability and scalability, plus various cost-effective options.

IBM Cloud VPC is purpose-built with inherent security to meet regulatory compliance standards, and multiple hardware and software solutions for IaaS, PaaS, and hybrid cloud needs. IBM Cloud VPC has a global network of multizone regions and availability zones for quick access, low-cost migration, low latency, and certified security. It supports hybrid or multicloud platforms.

A VPC can be created in IBM Cloud by following the instructions in [Creating and configuring a VPC](#).

2.2.1 IBM Cloud virtual server instance on IBM LinuxONE

A virtual server instance on IBM LinuxONE is an IBM Cloud VPC option that can be used to help migrate or recompile components to the s390x instruction set architecture (ISA). This option can also be used to test container-based workloads and solutions before their deployment within an HPVS as described in 2.3, “Hyper Protect Virtual Server” on page 18. All this with security and compliance in mind.

The IBM Cloud Virtual Private Cloud (VPC) user interface (UI) can be used to create virtual server instances with simple steps. For more information, see [Virtual server for VPC](#).

The available [s390x instance profiles](#) and [s390x virtual server images](#) can be selected based on your requirements.

2.3 Hyper Protect Virtual Server

An HPVS takes advantage of IBM Secure Execution for Linux to create a secure boundary around each workload, which ensures unauthorized users do not have access to the workload or data. Workloads are locked down by individual, instance-level, secure enclaves.

Figure 2-2 on page 19 illustrates the components that make up the Hyper Protect Platform. For example, the Hyper Protect Container Runtime (HPCR) includes the Bootloader and Hyper Protect Services to validate the authenticity and trust of the workload. This section provides an outline of the architectural components such as contracts with separation of duty, volume encryption, attestation, and Hyper Protect services.

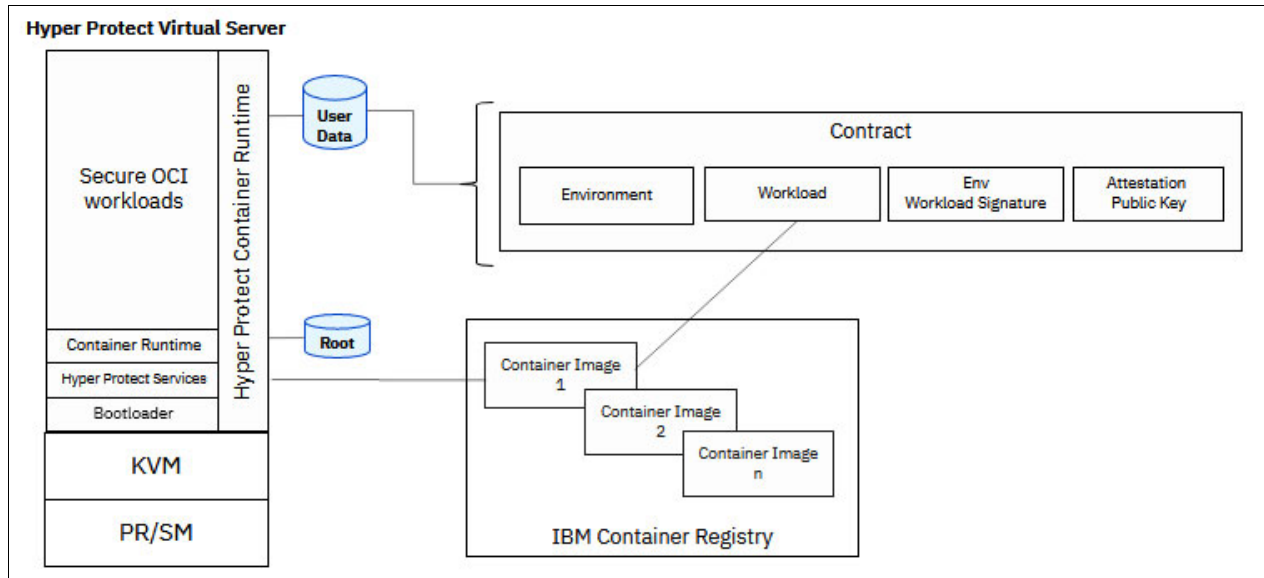


Figure 2-2 Technical assurance - Hyper Protect Platform

2.3.1 Bootloader

The bootloader is the first part of the Linux start-up process that brings up a KVM virtual machine.

The bootloader is responsible for the following:

- ▶ Setting up the LUKS encrypted boot partition.
- ▶ Writing the Operating System into the boot partition.
- ▶ Decrypting the user provided contract and placing a decrypted copy in a known file system location to be processed by the other components after the boot completes. The bootloader destroys the private key after decryption.
- ▶ Writing a cryptographically signed attestation document of various checksums of important components.

All steps are done at every boot. That means that the root volume file system is not persistent across reboots. All remote access such as SSH or login through a serial console are disabled by default.

2.3.2 Volume encryption

Both the root volume and user data volumes in the HPVS instance are automatically encrypted with Linux Unified Key Setup (LUKS) encryption. The root volume is re-created and encrypted during every boot, with the original content provided with the HPCR image. A passphrase for the root volume is not stored.

User data volume encryption is configured by using the seed that is provided in the workload and env sections of the contract. During the HPVS instance initiation, the volumes are attached and encrypted by using the seed to create a LUKS passphrase. If the seed information or the user data volume is not configured, the HPVS instance fails to initiate.

Each hour the volume encryption status check daemon examines the crypto headers of the root volume and user data volumes that are attached to the HPVS instance, and then writes the information messages about volume encryption status into the log.

2.3.3 Description of the contract

The implementation of the Hyper Protect Platform includes, in part, an encrypted secure execution image, the HPCR image. The contract is passed into an HPCR image as user data through the *cloud-init* process³. To protect the contract from any type of intrusion, a public and private key pair⁴ is created by an IBM internal build pipeline for the HPCR image. This key pair allows for the encryption of the contract sections. The public X509 certificate of the contract encryption public key is published by IBM and can be validated by any persona with the 3rd party certificate authority root key.

Each persona, like the Workload Provider or the Workload Deployer can independently encrypt their contract section by using this contract encryption public key. The contract encryption private key exists only inside the secure execution-encrypted HPCR image. This image can be decrypted by using only secure execution and keys that are rooted in hardware of an IBM LinuxONE or IBM Z platform. During deployment of an HPVS instance, this key is used by the Bootloader component to decrypt the contract.

Contract enforcement within the HPCR image ensures the following conditions:

- ▶ The contract cannot be deployed outside of the HPCR image.
- ▶ Secrets contained inside the contract cannot be retrieved outside the HPCR image.
- ▶ Only containers that are specified by the Workload Provider can run in the HPVS instance.
- ▶ The Workload Provider and Workload Deployer can independently provide their section of the contract, sharing only the contract section after it is encrypted.
- ▶ User data volumes are encrypted based on the secret seeds that are contained in the contract.
- ▶ Data volumes can be reattached to a new HPVS instance if the same secret seeds are defined within the contracts.

This concept of contract enforcement allows the Workload Provider, the Workload Deployer, and the Auditor personas to cooperate and ensure confidentiality and integrity of the information:

- ▶ The Workload Provider creates and encrypts the workload section and passes it securely to the Workload Deployer.
- ▶ The Auditor creates the *attestationPublicKey* and passes it securely to the Workload Deployer.
- ▶ The Workload Deployer creates the *env* section for the contract.
- ▶ The Workload Deployer combines the *env workload* and *attestationPublicKey* sections, and calculates and adds the signature across the *env* and workload sections.
- ▶ The Workload Deployer provides the contract in its *user data* section at provision time.

³ This software package (*cloud-init*) automates the initialization of virtual server instances during system boot.

⁴ A public key is part of the owner's digital certificate and is available for anyone to use. However, a private key is protected by and available only to the owner of the key.

2.3.4 The attestation record

The attestation record is provided to the workload in the file system in the directory `/var/hyperprotect`. The workload can then make this record available outside the HPVS instance and ultimately to the Auditor persona. The Auditor can verify the record and verify that deployment has occurred into an HPVS instance before any data is accessible to the workload.

Refer to 3.4, “Attestation” on page 54 for more information.

2.3.5 Logging

All components log to syslog as the central place for logs. Log output is forwarded to the ingestion backend defined in the contract. The components do not log any sensitive pieces of information, such as PI data or credentials. The detail level of logging can differ from component to component, but the guiding principle is to provide enough information to identify problems without flooding the logs.

Errors are logged once, by using the error log level. Each error features a unique identifier for the issuing component and the error situation. These identifiers are part of the API of HPCR, so they can be used in automation components to react on error situations. Identifiers are kept stable across semantic releases.

2.3.6 Hyper Protect layer services

The Hyper Protect Platform uses layer services within the HPCR image that validate a user contract signature, validate the authentication of registries, validate the integrity of OCI images being brought up within the HPVS instance, encrypt the volumes, and ensure the confidentiality by not allowing access to the HPVS instance.

The Hyper Protect layer consists of the following services:

- ▶ Logging service
- ▶ Contract validation service
- ▶ Registry authentication service
- ▶ Image service
- ▶ Signature service
- ▶ Storage service
- ▶ LUKS passphrase service
- ▶ Container service
- ▶ Catch service

Logging service

The logging component is responsible for the setup of the logging configuration. This configuration is available in the contract and allows a Workload Deployer to configure a logging backend, such as [Mezmo](#), or a custom backend compatible with the rsyslog protocol. The logging component validates the configuration and transforms it into a configuration for the rsyslog component that is used as a log forwarder.

If the logging configuration is invalid, this is indicated on the serial console, and the start of the virtual server instance fails.

Contract validation service

The contract validation component validates the contract syntactically and semantically against a JSON schema. If validation fails, so does the start of the virtual server instance.

Registry authentication service

The registry authentication component assembles all credentials in the contract that are required to authenticate against a remote container registry and transforms them into the configuration format required by the OCI runtime.

Image service

The image service assembles all information in the contract related to the validation of OCI images, for example, Red Hat signatures, and converts them into the format that is required by the OCI runtime.

Signature service

The signature service verifies the optional contract signature.

Storage service

The storage service initializes attached storage volumes according to their configuration in the contract. It creates necessary partitions, encrypts them through LUKS2 encryption or opens an existing LUKS2 encrypted layer. After the successful execution of the storage service, storage is mounted to the file system ready to be consumed by OCI images.

LUKS passphrase service

The LUKS passphrase service computes the passphrase that is used for LUKS2 encryption of the storage volumes.

Container service

The container service is responsible for starting the OCI images by using the OCI runtime of choice. After the successful execution of this service, the configured containers are running.

Depending on the selection within the workload section of the contract, a different container service is used to run the container. When specifying the compose subsection within the contract, docker compose will be used. When specifying the play subsection within the contract, podman kube will be used. Refer to 3.1.1, “The workload section” on page 39 for configuration details.

Catch service

The catch service monitors the successful start of the other services. If there is failure, it will automatically shut down the VSI after a grace period.

2.3.7 Hyper Protect Virtual Server for VPC

The HPVS for VPC is built in the IBM Cloud Virtual Private Cloud (VPC) infrastructure for extra network security. You can choose from various profile sizes and grow as needed to protect containerized applications, and you can pay-as-you-go on an hourly basis. You can also use existing or common [network security groups](#) and [logging](#) infrastructure.

When using IBM Cloud services, there is no need to understand the intricacies of their physical deployment of the infrastructure. Deployment of the services happen automatically as part of the managed-service offering. The provisioning process itself depends on a contract as input in the user-data field in the IBM Cloud UI for HPVS for VPC.

HPVS for VPC with Secure Execution can be provisioned from the IBM Cloud portal Virtual Private Infrastructure by a registered IBM cloud subscriber. The IBM Cloud command line interface, `ibmcloud`, can be used to create an instance of HPVS for VPC by selecting the [ibm-hyper-protect-container-runtime](#) (HPCR) stock image and the appropriate Secure Execution Profile. The HPCR stock image that is associated with the Hyper Protect operating system is periodically updated to provide security fixes and new functionality. To be able to use an updated stock image, deploy a new instance of HPVS for VPC using your contract and the current version of the HPCR image.

2.3.8 Hyper Protect Virtual Server for IBM LinuxONE and IBM Z

The HPVS for IBM LinuxONE and IBM Z is a private, on-premises, cloud deployment solution where containers can be initiated as KVM guests that run an HPCR image, as shown in Figure 2-2 on page 19.

The Secure Execution on Linux feature ([0115](#)) and CP Assist for Cryptographic Function (CPACF) feature ([3863](#)) are required to enable confidentiality and integrity by protecting and isolating the data at the virtual machine (KVM guest) level.

To learn how to setup a KVM host LPAR and start KVM guest images, see [Introducing IBM Secure Execution for Linux](#).

2.3.9 Considerations when deploying workloads in HPVS instances

It is generally accepted that the adoption of cloud-native principles in a hybrid cloud environment can enable workloads to become more cost-effective and convenient to run as separate components by using a microservices approach. Microservices typically have their own technology stack, including the database and data management, and communicate with other microservices over REST APIs.

Multiple HPVS instances can be used for each microservice with the infrastructure providing the necessary capabilities to secure networking communication between them. Even in the absence of cloud native orchestration such as Kubernetes, independent scaling of services and independent upgrades or updates of services can be achieved.

For example, you can use automation tools and cloud services to deploy multiple HPVS instances and implement failover, load balancing, and scaling and eliminate single points of failure.

Typically, with state-of-the-art cloud services and automation tools no human intervention is required during failure events as the requests can be automatically routed around failures. Therefore, do not start multiple independent workloads or microservices within the same HPVS instance.

Each instance should be a single unified workload or microservice, which can be composed of multiple containers. The containers that are specified in a contract are assumed to be mutually dependent, so if deployment of one container fails then the workload stops.

As an expected practice, HPVS instances are intended to be deleted and re-created during the lifetime of a workload. Changes to the root volume do not persist after reboot, so be aware of the following conditions and behaviors:

- ▶ A contract is set when an HPVS instance is deployed and cannot be changed.
- ▶ HPVS instances cannot be upgraded. Therefore, snapshots of the attached boot volumes are not recommended.

- ▶ Updated OCI container images require a new contract, and a new HPVS instance must be deployed.
- ▶ Updated API keys, passwords, and so on, require a new contract and a new HPVS instance deployment.
- ▶ An updated HPCR image requires a new HPVS instance. The previous contract can be reused if the encryption keys for the updated HPCR image are unchanged from the prior image.
- ▶ Reboot re-pulls container images although root volume changes are not persistent. This means that contract details relating to pulling images must remain valid through all HPVS instance reboots after creation.
- ▶ No workload data should be held on root or boot volumes. The workload data is deleted by a reboot or update.
- ▶ Externally attached data volumes have an independent lifecycle and can be reattached to the HPVS instance assuming that contract sections relating to storage remain consistent.

Tools such as Terraform can help make generating new contracts and replacing HPVS instances straightforward. For more information, see 4.1.9, “Deployment automation - Terraform” on page 70.

2.4 Hyper Protect Secure Build

According to [OWASP-SAMM](#), the Secure Build practice must emphasize the importance of building software in a standardized, repeatable manner, and doing so by using secure components.

The goal is to reach a Software Assurance Maturity Model (SAMM) maturity-level of three in the build process to help with the following tasks:

- ▶ Prevent known defects from entering the production environment
- ▶ Define mandatory security checks and ensure that building non-compliant artifacts fails
- ▶ Analyze the dependencies used for security issues

Security should be effective and at the same time it should not be cumbersome to implement best practices for the following personnel that participate in the secure build process:

- ▶ Cloud administrators. Create the Secure Build Server and register the repository for the application developer
- ▶ Application developers. Use the Secure Build Server to build and deploy applications from a GitHub repository

By using HPSB, a trusted container image can be built within a secure enclave that is provided by IBM HPVS. The enclave is highly isolated, where developers can access the container only by using a specific API and the cloud administrator cannot access the contents of the container. Therefore, the image that is built can be highly trusted. Specifically, the build server cryptographically signs the image, and a manifest for audit purposes. The manifest is a collection of materials that are used during the build. Because the enclave protects the signing keys within the enclave, the signatures can be used to verify that the image and manifest files are from the HPSB.

The HPSB can be used to securely build source code from a GitHub repository, publish the built image to a repository like Docker hub or [IBM Container Registry](#) and deploy the built image as an HPVS instance.

For more details and sample configuration files, see [IBM Hyper Protect Secure Build](#).

2.5 Cryptographic agility is the key to SecDevOps

Cryptographic algorithms can break or become weak or obsolete over time. Recommendations and regulations on which cryptography to use often change. Therefore, businesses and organizations must replace the underlying cryptography that they use today, and it is not the only time such a change will be required. For more information, see [NIST Cryptographic Algorithm Validation Program](#).

With change comes also an opportunity to rethink how applications use complex cryptography such that future enhancements, updates, and patches are simpler to apply. For that reason, the ability to rapidly change cryptographic algorithms and key strengths used to encrypt and sign data becomes essential.

When thinking about cryptographic agility, consider the following requirements:

- ▶ Build a cryptographic inventory and create a roadmap:
 - Build a cryptographic inventory as a reusable security asset where crypto is used
 - Perform a risk assessment and gap analysis
 - Evaluate vendor products
 - Develop plans for use of stronger cryptography
 - Understand the open source effect
 - Use a bottom-up approach
- ▶ Design and run with cryptographic agility in mind:
 - Manage internal and external dependencies
 - Make it simple to change underlying algorithms, methods, or protocols
 - Verify changes by automating as much as possible
 - Prepare for future changes

Develop new applications to be as flexible as possible to react to new situations.

2.6 Hyper Protect Crypto Services

Hyper Protect Crypto Services (HPCS) is a dedicated key management service (KMS) and hardware security module (HSM) that is built on the technical assurance of the Hyper Protect Platform (see Figure 2-3 on page 26).

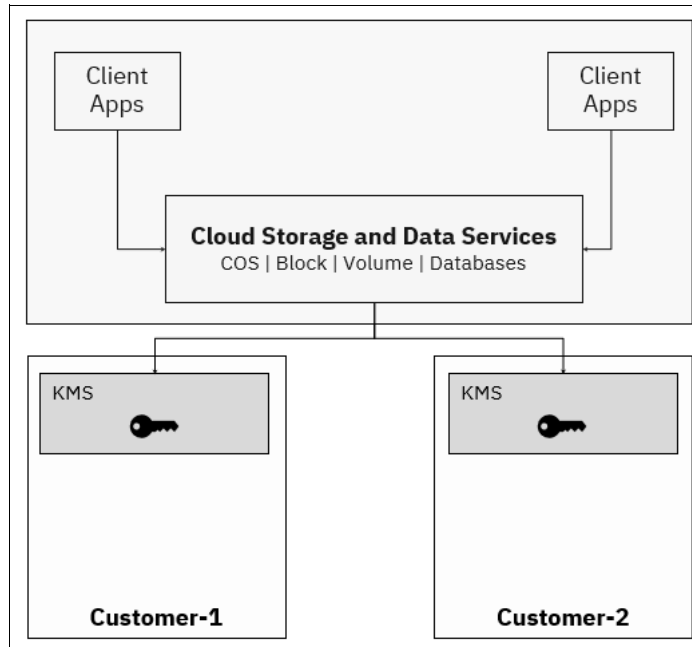


Figure 2-3 HPCS - high-level view

HPCS is built on a FIPS 140-2 Level 4-certified HSM. With HPCS, you have complete control of your encryption keys for cryptographic operations service that is known as keep your own key (KYOK). This is in contrast to bring your own key (BYOK) based services. For a detailed comparison on why KYOK is better than BYOK, refer to [How is Keep Your Own Key different from Bring Your Own Key?](#)

With the Unified Key Orchestrator (UKO) function you can perform encryption key lifecycle management from a single pane of glass on IBM Cloud into other cloud environments like AWS, Azure, and Google Cloud. UKO also doubles up as a repository for all encryption keys that can be synced and pushed to multiple keystores with the push of a button.

HPCS allocates physical crypto-units to every instance provisioned on IBM Cloud in accordance with MZR Load Balancing and High Availability standards. In some geographies, HPCS also provides cross-regional availability to protect the customers from regional disasters.

For more information, refer to [Hyper Protect Crypto Services](#) and [Hyper Protect Crypto Services General FAQs](#).

2.6.1 Accessing cryptographic services with HPCS

HPCS supports multiple APIs and programming models that are integrated with other services in the IBM Cloud. HPCS services are accessible from the IBM Cloud UI and programmatically over public and private endpoints. A whole suite of middleware, such as object storage, open data foundation (ODF) clusters, Kubernetes clusters, databases, load balancers, and so on, can integrate seamlessly with HPCS.

Authentication and authorization of access to the resources in IBM Cloud is done by [IBM Cloud Identity and Access Management](#). Applications accessing HPCS through the endpoints must do so with an API-Key, which is part of a Service-ID, which in-turn, is a part of an Access Group.

HPCS offers the following application programming interfaces (APIs):

- ▶ Key Protect API
- ▶ GREP11 API
- ▶ PKCS#11 API

Key Protect API

HPCS supports Key Lifecycle Management activities like creating, maintaining, protecting, and deleting cryptographic keys along with other Key Actions like:

- ▶ Wrap a Key. Use a root key to wrap or encrypt a data encryption key (DEK)
- ▶ Unwrap a Key. Use a root key to unwrap or decrypt a data encryption key
- ▶ Rewrap a Key. Use a root key to rewrap or re-encrypt a data encryption key
- ▶ Rotate a Key. Create a new version of a root key
- ▶ Set a key for deletion
- ▶ Unset a key for deletion
- ▶ Enable a key
- ▶ Disable a key

For more details on how to perform the previously listed actions, see [IBM Cloud Key Protect Methods](#).

GREP11 API

The FIPS 140-2 Level 4 HSM that HPCS uses is built on the IBM Crypto Express features operating in EP11 mode. EP11 is a stateless API⁵, which provides functionality similar to the industry-standard Public-Key Cryptography Standards (PKCS) #11 API. PKCS #11 API defines a platform-independent API to cryptographic tokens, such as hardware security modules (HSM) and smart cards. Existing applications can use PKCS #11 to benefit from enhanced security with secure key cryptography and from the stateless programming model, which makes the cryptographic operations much more efficient and scalable.

For more information about EP11 capabilities and extensions, see [Enterprise PKCS #11 introduction](#).

HPCS provides a set of EP11 over gRPC (GREP11) API calls, with which the cryptographic functions are run in the cloud HSM of HPCS. The GREP11 API is a stateless interface for cryptographic operations on the cloud that is based on EP11 and gRPC.

An application deployed to HPVS can use a stateless cryptographic API like GREP11 to create and use cryptographic keys. With GREP11, cryptographic material like keys is created in the HSM, but is stored outside of the HSM by the client application.

The application can store the keys in the boundary of the HPVS instance on attached devices and use volume encryption and protection of memory and data in use.

You can use the features that are provided by HPVS, such as separation of duty, with features of HPCS to design and implement advanced cryptographic mechanisms for deriving keys from combined seeds. The combined seeds are a combination of different seeds that are owned by the Workload Provider and Workload Deployer, and defined in the respective sections of the contract.

⁵ Stateful services keep track of sessions or information about previous or pending interactions or transactions and thus react differently to the same inputs based on that information. Stateless services do not maintain such information on the side of the service, but rely on clients to maintain “state” information about sessions or previous interactions.

Refer to “How encryption keys are created using GREP11” on page 124 for more information. The topic describes using the stateless capabilities of the EP11 interface by keeping state information like encryption keys, seed-parts and backup-key parts inside secure enclaves created for an HPVS.

PKCS#11 API

HPCS also provides a PKCS#11 library that allows user applications to interact directly with the FIPS 140-2 Level 4 HSM through the PKCS#11 API. PKCS11 is a standard that specifies an API, called Cryptoki, to perform cryptographic operations. The Cryptoki API follows a simple object-based approach by presenting to applications a common, logical view of the HSM, called a cryptographic token.

Cryptoki isolates an application from the details of the cryptographic device, which makes the application portable and usable with any cryptographic device that supports the PKCS#11 standard. For more information on cryptographic functions supported by HPCS, refer to: [Introducing PKCS#11](#).

2.7 Crypto Express Network API for Secure Execution Enclaves

With Crypto Express Network API for Secure Execution Enclaves, REST API is used to configure the c16 server, which provides gRPC API to access Crypto Express domains that are assigned to the IBM Z or IBM LinuxONE LPAR. After the configuration, your applications can access the c16 gRPC API through the IBM GREP11 interface, provided by the GREP11 server, to securely connect from a Secure Execution for Linux LPAR to the Crypto Express domains. It is possible to enable the c16 server to send logs to a configured Rsyslog server to view logs.

Figure 2-4 illustrates an architecture overview of Crypto Express Network API for Secure Execution for Linux.

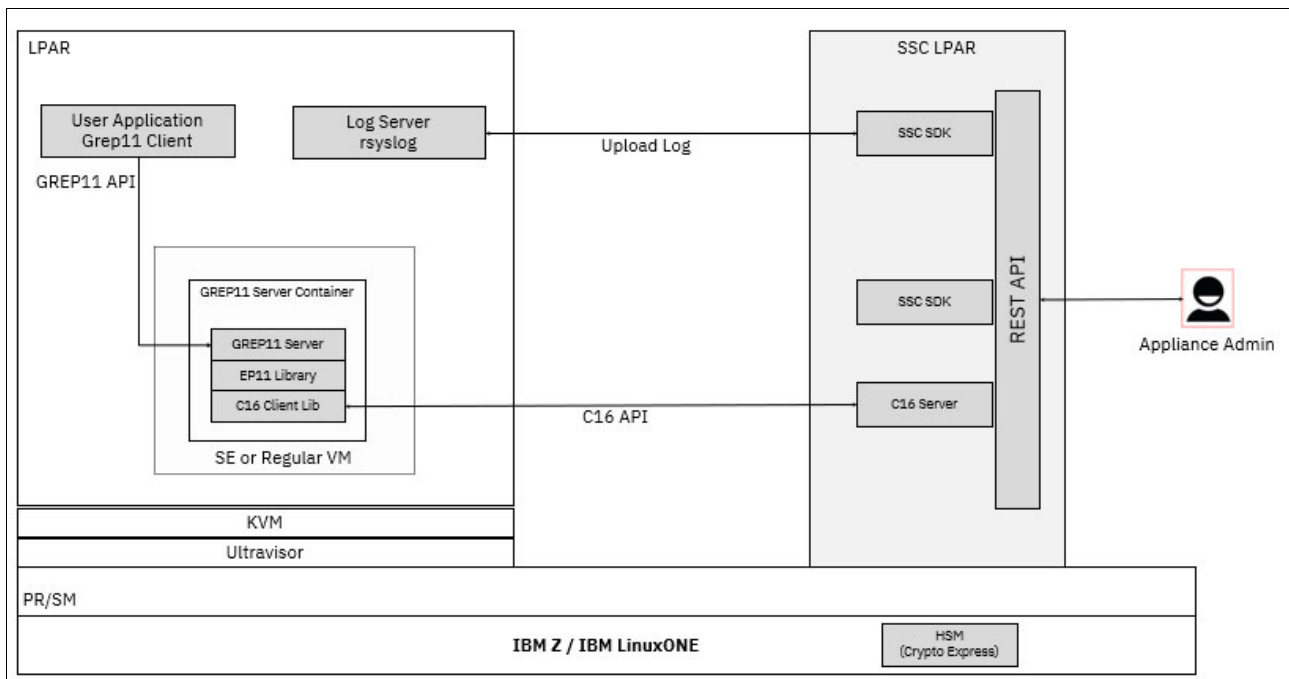


Figure 2-4 Architecture overview of Crypto Express Network API

Note: The GREP11 client and the GREP11 server are shown in the same KVM host LPAR. However, the GREP11 client can run anywhere that has connectivity to the GREP11 server if the environment is trusted.

2.7.1 Security considerations

The Crypto Express Network API for Secure Execution Enclaves' REST API is critical for configuring and managing the c16 server, so it must never be accessed by any untrusted person or system. Users are responsible for controlling access to the API to keep it secure.

The c16 API invokes crypto operations on Crypto Express domains that are made accessible by the Crypto Express Network API for Secure Execution Enclaves. Therefore, the c16 API must be kept secure, and it is not recommended to expose the c16 API over a public network. The c16 API is protected through mTLS with the certificate authority (CA) configured through the Crypto Express Network API. Anyone with a valid certificate that the CA issues can access the API. Users are responsible for controlling access to the CA and issuing client certificates only to trusted clients.

For more information, see [Crypto Express Network API for Secure Execution Enclaves](#).

2.8 Storage and repositories in the cloud

Data storage requirements can vary by use case. Applications at the front end of interaction with clients need to be responsive. Any data that is handled by these applications must be accessible to meet the response time requirements of the application. Because of the distributed nature of applications running in the cloud, the application design can be simpler with a design that is independent of storage. For example, the application developer might decide to outsource the data availability requirements to a data service. In such cases, the infrastructure provider must provide several options for data storage, based on latency and access pattern. The provider must also allow data to be moved seamlessly from one storage platform to another. Also, data protection concerns, which are inherent to a cloud environment for data at rest, might require that all storage services be connected to an available key management system.

Data in IBM Cloud storage services, such as cloud object storage, block storage, and file storage, is encrypted by default using randomly generated keys. For added security, you can also use HPCS to create and keep your own root keys⁶ to protect your data through envelope encryption.⁷

The following subsections describe the different storage options in a cloud environment.

2.8.1 Cloud object storage

IBM Cloud Object Storage (COS) is an industry-leading cloud service that is ideal for storing large volumes of data. It provides best-in-class security and data durability at near-infinite scalability, complemented with immutable data retention, audit controls and continuous compliance, which is ideal for meeting the demands of business and regulatory requirements.

⁶ The root key is generated by using AES 256

⁷ Envelope encryption is the practice of encrypting data with a data encryption key (DEK) and then wrapping the DEK with a root key that is kept inside HPCS, which you can fully manage.

COS has built-in integration with other services in IBM Cloud and security services. Access to COS is available for applications and services from inside IBM Cloud, other cloud services and on-premises through RESTful APIs that are exposed on public and private network endpoints. Data in COS can be protected with encryption keys that are obtained from key management systems like HPCS.

Other cloud service providers offer object storage like S3 from AWS and Azure Blob Storage from Azure. For more information, see [IBM Cloud Object Storage](#).

2.8.2 Block storage

Block storage, sometimes referred to as block-level storage, is a technology that is used to store data files on storage area networks (SANs) or cloud-based storage environments. Block storage divides data into blocks and then stores those blocks as separate pieces, each with a unique identifier. The SAN places those blocks of data wherever it is most efficient. That means it can store those blocks across different systems and each block can be configured or partitioned to work with different operating systems.

Block storage allows for the creation of raw storage volumes to which server-based operating systems can connect. Block storage also decouples data from user environments, allowing that data to be spread across multiple environments. This creates multiple paths to the data and allows the user to retrieve it quickly. When a user or application requests data from a block storage system, the underlying storage system reassembles the data blocks and presents the data to the user or application.

Block Storage services in IBM Cloud are available in the VPC environments. For more information, see [What is block storage?](#).

2.8.3 File storage

File storage, which is also referred to as file-level or file-based storage, is normally associated with Network Attached Storage (NAS) technology. NAS presents storage to users and applications by using the same ideology as a traditional network file system. In other words, the user or application receives data through directory trees, folders, and individual files. This functions similarly to a local hard disk. However, NAS or the Network Operating System (NOS) handle access rights, file sharing, file locking, and other controls.

File storage can be easy to configure, but access to data is constrained by a single path to the data, which can impact performance compared to block or object storage. File storage also operates only with common file-level protocols, such as a New Technology File System (NTFS) for Windows or a Network File System (NFS) for Linux. This might limit usability across dissimilar systems. For more information, see: [What is file storage?](#).

2.8.4 On-premises storage

On-premises storage is categorized according to principles of storage tiering that assigns data to various categories of storage media based on requirements such as cost, availability, performance, and recovery objectives. Recovery objectives are usually stated as Recovery Time Objective (RTO) and Recovery Point Objective (RPO) policies, which are crucial components of an organization's data life cycle strategy.

Storage tiering is a component of Information Lifecycle Management (ILM) that helps organizations minimize storage costs and ensures performance and compliance. Storage tiering is applied to the following data classes:

- ▶ Mission critical. For data that should not experience any downtime and is usually assigned to the fastest storage devices like Flash and electronic storage devices.
- ▶ Hot. For data that needs to be accessed frequently to support business-critical applications such as Customer Relationship Management (CRM) and Enterprise Resource Planning (ERP). This category also requires the fastest storage.
- ▶ Warm. For data that is accessed less frequently. This kind of data can be stored on medium-speed storage devices like disk storage
- ▶ Cold. For data that might never be accessed again but must be retained for the organization to meet regulatory requirements. Such data is usually saved on tape and other archival storage devices.

LUKS encryption with envelope encryption of the root keys in HPCS can be used to protect data for on-premises storage. For a tutorial, see [Protect LUKS encryption keys with IBM cloud Hyper Protect Crypto Services and IBM Key Protect](#).

For more information on various storage devices and solutions, see [IBM Storage](#).

2.9 Common usages

In general, any application that either uses sensitive data or secrets, or comprises sensitive code, or needs to conform to compliance and regulatory requirements can take advantage of confidential computing with the Hyper Protect Platform.

This section discusses multiple exemplary use cases and explains how the Hyper Protect Platform can be used to implement it. In particular, these use cases are discussed:

- ▶ Securely bring applications to hybrid cloud
- ▶ Digital assets infrastructure
- ▶ Confidential AI
- ▶ Secure multi-party computation
- ▶ Secure distributed cloud

2.9.1 Securely bring applications to hybrid cloud

Enterprises are moving their applications to cloud to reduce cost, simplify and consolidate their IT environment, and take advantage the flexibility of hybrid cloud. However, these applications contain many types of sensitive, sometimes regulated data that needs to be protected. Examples for this are applications in the financial services, healthcare, government, and nonprofit domains.

The biggest barrier for bringing sensitive applications to the cloud is that the cloud infrastructure is owned and operated by the cloud provider. There are well-established methods to protect data-at-rest and data-in-motion. However, there was always a concern about data-in-use, until Confidential Computing with the Hyper Protect Platform became available in the cloud. The Hyper Protect Platform protects the code of the application and the data-in-use from a malicious system administrator or other privileged actors.

It thus enables taking advantage of hybrid cloud for application modernization and design of cloud native applications and ensures that compliance with regulations, data sovereignty, and data protection requirements are met. The Hyper Protect Platform also ensures protection of trade secrets and intellectual property that can be part of the application, such as proprietary business logic, private AI training data, and AI models.

The following list includes features of HPVS that are relevant in the context of application modernization:

- ▶ Secrets can be used to establish instant trust to a newly deployed HPVS instance as Zero Knowledge Proofs are possible. The instance can be integrated and authenticated in an existing flow. Even if the new instance is in a remote data center or cloud.
- ▶ HPVS provides LUKS volume encryption for attached storage devices. The LUKS passphrase can be derived from two seeds, which are defined by and known to separate personas. This ensures that no individual persona can reconstruct the passphrase.
- ▶ HPVS provides Attestation with which an Auditor persona can validate the HPVS instance.
- ▶ Secure Build can be used to build the application OCI images in HPVS, thus protecting the build process, the built OCI images, the build manifest and signing keys for signing the images.

2.9.2 Digital assets infrastructure

In a digital assets infrastructure, maintaining control over private keys is extremely challenging and poses a major risk, especially when managing thousands if not millions of wallets. Loss of control of an account's private key through cyberattack or mishandling can result in irreversible asset loss.⁸

The Hyper Protect Platform provides a trusted platform for reliable and scalable digital custody applications, for managing, transferring, and storing high value digital assets in highly secure wallets.

To provide a secure Digital Assets Infrastructure, you can use features of the Hyper Protect Platform:

- ▶ Trusted Container Runtime
- ▶ Memory protection
- ▶ Data in use protection
- ▶ Volume encryption
- ▶ Attestation
- ▶ Contract mechanism
- ▶ Separation of duty

In addition, key creation, key derivation, and further cryptographic operations such as encryption and signing are done by using HPCS on FIPS 140-2 Level 4 certified hardware that is provided by IBM LinuxONE and IBM Z. Cryptographic keys and access tokens can be kept inside the HPVS instance and cannot be accessed even by privileged actors.

To protect high-value transactions and comply with industry regulation, many applications are isolated, and communications with other applications are restricted. A key regulation that has emerged is the requirement of cold storage or alike for security purposes. However, today's cold storage solutions have several limitations that include protection of keys and assets, manual operations, and the inability to scale due to the manual process.

The [IBM Hyper Protect Offline Signing Orchestrator](#) is an alternative approach to cold storage. It is designed to broker communications between two different applications that are designed not to communicate directly. Hyper Protect Offline Signing Orchestrator is deployed in HPVS instances on IBM LinuxONE and IBM Z. Hyper Protect Offline Signing Orchestrator helps protect high-value transactions by offering additional security layers that include disconnected network operations, time-based security, and electronic transaction approval by multiple stakeholders. Offline Signing Orchestrator also changes the digital asset transaction

⁸ Reference: [Microsoft mitigates China-based threat actor](#)

signing process from a manual operation to an automated and policy-driven one, by eliminating the operational involvement without eliminating the human control.

2.9.3 Confidential AI

Data that is used for training and testing AI and ML models can be valuable, sensitive, and regulated. Also, AI and ML models can constitute intellectual property or trade secrets, be sensitive, and require confidentiality.

The Hyper Protect Platform can protect AI processes, like training, test, and inference by using features like memory protection, code and data confidentiality, code and data integrity, volume encryption, attestation, and the contract mechanism with separation of duty.

For example, the Hyper Protect Platform can be used to build confidential AI as a service in a hybrid cloud environment. Separation of duty is possible to allow a Workload Deployer persona to provision confidential AI in an HPVS instance and ensure this persona cannot access the AI model or related data.

Figure 2-5 shows an example of a complete end-to-end flow for securely training a machine learning model and using this model inference. This all occurs within a confidential computing environment with boundaries of protection around all the components.

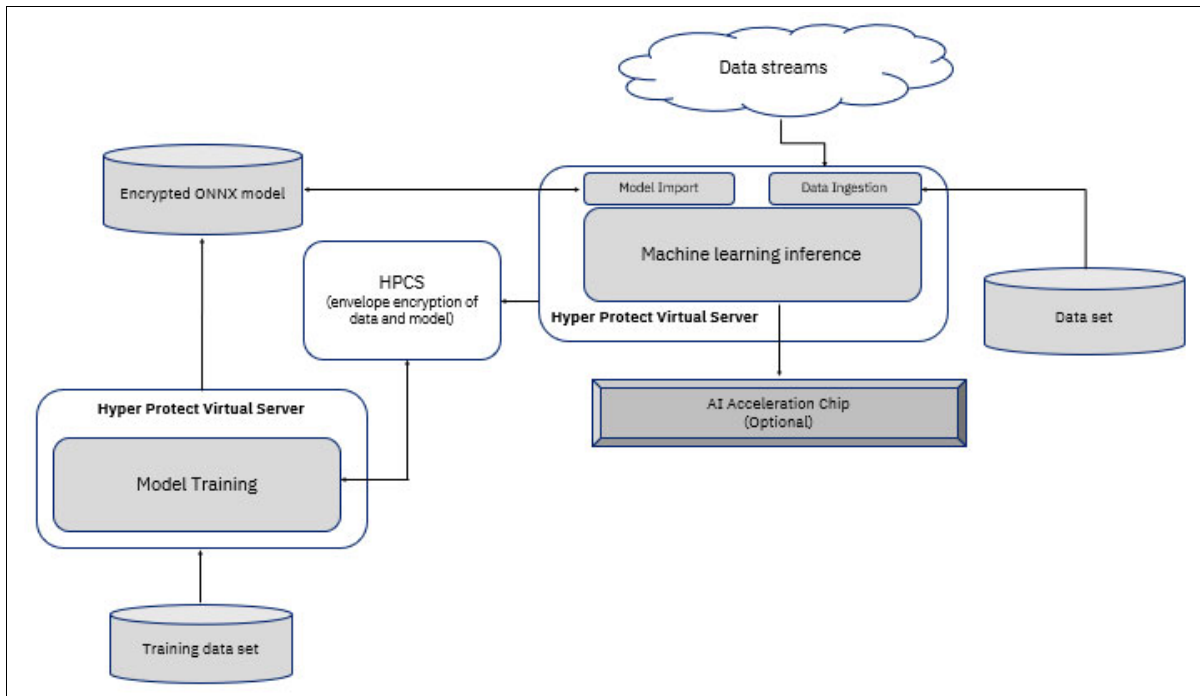


Figure 2-5 Confidential AI example

You can also add additional AI acceleration, such as by using the [IBM Telum processor](#) that contains on-chip acceleration for artificial intelligence, with the IBM LinuxONE and IBM Z platforms.

As another example, confidential, federated learning can be applied to cases in which multiple parties have private data to combine and analyze without exposing the underlying data or machine learning models to any of the other parties. This technology can be applied to preventing fraud in financial services, detecting or developing cures for diseases in the

healthcare industry, or gaining business insight. As an illustration, multiple hospitals might combine data to train a machine learning model to clinically analyze medical images.

2.9.4 Secure multi-party computation

Secure multi-party computation (SMPC) enables multiple parties, each holding their own private data, to collaborate in a computation without revealing any of the private data. The two important requirements on SMPC are privacy and correctness. The privacy requirement states that parties learn their computation output and nothing else. The correctness requirement states that each party receive its correct output. Therefore, no adversary must be able to cause the result of the computation to deviate from the function that the parties set out to compute.

A primary concern is how the confidentiality and integrity of the data can be preserved when calculations happen outside the party's direct control.

For example, banks can collaborate to find patterns of anti-money laundering. See Figure 2-6. Each bank brings encrypted financial-transactions data to a confidential computing enclave, such as an HPVS, where data can be safely decrypted. The multi-party collaboration (MPC) application runs fraud-detection algorithms in the confidential computing enclave. Malicious administrators or operators cannot see financial transaction data from any banks collaborating on the MPC platform.

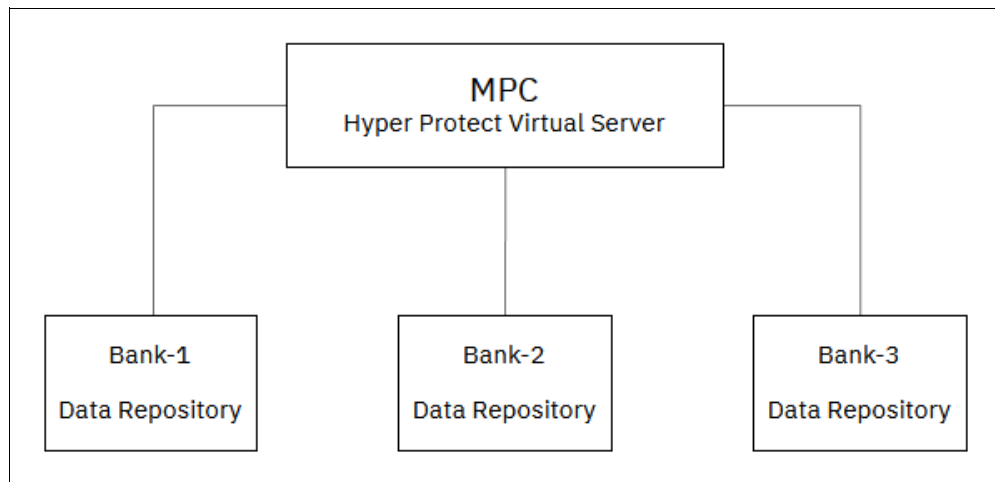


Figure 2-6 Banks collaborate to find patterns of anti-money laundering

By using the Hyper Protect Platform and confidential computing, organizations can now ensure that data is protected against tampering and compromise, and data sovereignty and privacy regulations can be fulfilled. This includes threats within the partnering organizations and validating the integrity of the code processing that data.

The data can be combined and analyzed and the results can be sent in an encrypted format back to each party. Data remains protected throughout the entire process: while in transit, in use, and at rest.

2.9.5 Secure distributed cloud

A secure distributed computing application can comprise a front end running in a TEE on mobile, personal computing, or point of sale (POS) devices and a back end in a hybrid cloud infrastructure. Such a type of application can be required for security critical applications like payment systems and in financial services, healthcare, or in other regulated industries.

The back-end infrastructure can use the Hyper Protect Platform to provide protection of the code and data in use by the back end. Data in transit between the back end and the front end is typically protected by mTLS.



Making the infrastructure secure

The previous chapters described the deficiencies of traditional infrastructures in the context of increased cyberthreats that are posed by malicious actors, whether internal or external. The chapters included strategies to mitigate risks by using infrastructures built on the Hyper Protect Platform and described the relevant qualities of the components and services that comprise the solution.

There are several distinct steps that are involved in configuring Hyper Protect Virtual Server (HPVS) instances, which can quickly become overwhelming without some guidance. This chapter provides step-by-step instructions with sample configuration files, where required, to deploy a secure infrastructure for confidential computing.

Before you start, you need an [IBM Cloud account](#) for a Virtual Private Cloud (VPC) or KVM to host LPARs on IBM LinuxONE or IBM Z to deploy the examples in this chapter. For more information, see [System requirements](#).

Refer to Appendix B, “Creating a Hyper Protect Virtual Server for VPC” on page 107 for detailed steps on deploying an HPVS instance by using the IBM Cloud Virtual Private Cloud (VPC) user interface (UI).

You must also complete [Setting up and configuring IBM Hyper Protect Virtual Servers](#) before you can deploy HPVS instances on IBM LinuxONE or IBM Z.

The deployment of HPVS consists of the following aspects:

- ▶ 3.1, “The contract” on page 38
- ▶ 3.2, “Contract encryption” on page 49
- ▶ 3.3, “Contract certificates” on page 53
- ▶ 3.4, “Attestation” on page 54
- ▶ 3.5, “Logging for HPVS instances” on page 56
- ▶ 3.6, “Encrypting data volumes” on page 63

3.1 The contract

The contract is a definition file in the YAML format that is specific to the Hyper Protect Virtual Servers (HPVS) instance, which is also called an image. This file must be created by the user as a prerequisite for creating an instance. After the file is created, it must be passed as input through the User Data field when an instance is created.

The HPCR image decrypts the contract, if it is encrypted, validates the contract schema, checks for contract signature, creates the passphrase to encrypt the disk device, and starts the container that is specified in the contract.

The contract has several sections, each one documenting how the HPVS instance should be built:

- ▶ workload (mandatory).
- ▶ env (mandatory).
- ▶ attestationPublicKey (optional). This section provides a public RSA key as part of the contract, which is used to encrypt the attestation document and the attribute must be named as attestationPublicKey.
- ▶ envWorkloadSignature (optional section). This section contains the signature of the other sections of the contract.

The workload and env sections are mandatory because the information that is provided in these sections defines how the HPVS instance is built. The contents of these sections come from two different personas, namely the Workload Provider and the Workload Deployer. See “Separation of duty” on page 11 for a description of the predefined personas.

The Workload Provider persona provides information about the container, or workload that needs to be started on the HPVS instance. It includes the following information:

- ▶ Name of the container
- ▶ Container registry and where it resides
- ▶ Credentials of the container registry
- ▶ Image digest
- ▶ Notary server information, which is required for image validation
- ▶ Volumes to be present and attached
- ▶ Environment variables that need to be passed to the container
- ▶ Docker compose file or Pod descriptors with the container information

Note: If a docker compose file is used, only one container is supported. However, pod descriptors, a podman construct, support one or multiple containers

The Workload Deployer persona works closely with the infrastructure staff. This persona receives the workload information, preferably an encrypted workload section, from the Workload Provider persona. The Workload Deployer then creates the env section of the contract. The env section has information that is specific to the environment. Usually, it is information that the Workload Provider persona does not have and does not need to know. An example is information about the “Logging” instance, which the Workload Deployer persona creates, before it adds information to the env section of the contract.

3.1.1 The workload section

This is one of the most important sections of the contract. The workload section can have multiple subsections and the purpose of the subsections is to provide information that is required for bringing up the workload. The workload section is defined with `type: workload`.

This section is mandatory and it encompasses the following subsections:

- ▶ `auths`. This subsection is optional and required only when authentication is needed to download or get the image file from the registry.
- ▶ `compose`. For a single container. `compose` subsection must exist if `play` subsection is not defined. The `compose` subsection and the `play` subsection are mutually exclusive.
- ▶ `play`. For multiple containers. `play` subsection must exist if the `compose` subsection is not defined. The `compose` subsection and the `play` subsection are mutually exclusive.
- ▶ `images`. This subsection is optional and used for validating the integrity of signed images.
- ▶ `volumes`. This subsection is optional and used only when a data volume is mounted on the HPVS guest.

Example 3-1 shows a high-level sample of the workload section of the contract. The minimum that a workload section needs is the `compose` section. The other sections can be added based on the requirement.

Example 3-1 High-level sample of the workload section

```
type: workload
auths:
  <registry url>:
    password: <password>
    username: <user name>
  <registry url>:
    password: <password>
    username: <user name>

compose:
  archive: <base64 encoded of tgz of docker-compose.yaml>
images:
  dct:
    <docker image name (without the tag, an example is
docker.io/redbookuser/s390x):>:
      notary: "<notary URL>"
      publicKey: <docker content trust signed public key>
    <docker image name>:
      notary: "<notary URL>"
      publicKey: <docker content trust signed public key>

volumes:
  <volume name>:
    mount: "<data volume mount path>"
    seed: "<Passphrase of the luks encryption>"
    filesystem: "ext4"
```

The auths subsection

The auths subsection consists of information about the container's registry. If a public image is used in the contract, you do not need the auths subsection because no credentials are required. The auths subsection is required only if the container images are private. This subsection does not have any image information, as shown in the following sample. This subsection needs to contain the name of the image registry and the credentials, such as username and password, for the same. The key must be the hostname of the container registry or the following string for the default docker registry:

```
https://index.docker.io/v1/
```

Example 3-2 shows an IBM Cloud Registry snippet. For more information about using the API key, see [Using client software to authenticate in automation](#).

Example 3-2 IBM Cloud registry

```
auths:
  us.icr.io:
    password: <apikey>
    username: iamapikey
```

The compose subsection

The compose subsection consists of an archive subsection. The archive subsection contains the Base64 encoded GNU compressed tar (TGZ) file archive of the docker-compose.yaml file with any other file such as certificates and configuration files. Because the HPCR image uses Docker Engine and Docker Compose to start the container, the information about the container must first be created by using a standard docker-compose file. This file is then archived and Base64 encoded, and the output of this is provided as the value to the archive subsection within the compose section. For more information, see [Docker Compose overview](#).

The mount points specified under the volumes information of the docker-compose file might be aligned with the volume mount point that is specified in the workload section of the contract.

Note: Running a build as part of a docker compose file is not supported. Make sure that your docker compose file does not have a build section.

Both “yaml” and “yml” formats support docker-compose files. See Example 3-3.

Example 3-3 Docker-compose file

```
version: '3'
services:
  nginx:
    image:
      nginx@sha256:e73ba8654ba7fd1834e78a3d4e9d72ffaaa3372d42996af5c34ba3e1abc293e8
    user: 0:0
    restart: always
    ports:
      - 80:80
```

Tip: There are use cases when the workload section is pre-encrypted in which the registry is not known. For example, when the Workload Provider wants to allow the Workload Deployer to use a registry mirror or a private container registry. In such a case, it is possible to dynamically override the registry and pull credentials. This is a coordinated effort between the Workload Provider and the Workload Deployer. For more information, see [Using a dynamic registry reference](#).

Use the `tar` command to get the Base64 encoded archive file. See Example 3-4. The Base64 output is available in the `compose.b64` file.

Example 3-4 Command to get the Base64 encoded archive file

```
tar czvf - -C <COMPOSE_FOLDER> . | base64 -w0 > compose.b64
```

Note: Make sure that the `compose.tgz` file contains only directories and regular files. Links or pipes are not supported and will result in an error during deployment.

Copy the content of `compose.b64` file as a value of the `compose -> archive` subsection. See Example 3-5.

Example 3-5 Copy of compose.b64 file content

```
compose:
  archive: <paste the content of compose.b64 >
```

For this example, the response is similar to the output in Example 3-6.

Example 3-6 Output of the compose -> archive

```
compose:
  archive:
H4sIAKOFmGIAA+2RTW6DMBBGs84pRuyB8Q8k+DIRwZOGtmBkkycvhnLVVV1EWkqhJv4ZHt8ednWZvqhW
xcmaYzjpKhed08HETmpQRfd3k2VeRhPpEJCUxymTPkIu0ALBOIG8DHq3zn4vrSj iqdLY/nsv+xb2w7nRZy
w1Pgo/4THNm3uiKntgCWd01aowmZnwLUTf1ECpwo8Jpu9NyZ2zvQgdADFEudoXyQzSu+fPPzseSvedo6qj
V7mDa2anZbdH8totL6somtU1vX8K4SjshDsFKU2NmFvAZuMc9U37wceey+Y6BI8Fi6+6vxK5RS+YFDh6R
Nu//tuV1ZwVJd4BcjKckQAIAAA=
```

The play subsection

In the play subsection, you can define the workload through [Pod descriptors](#). Each pod can contain one or more container definitions. Descriptors can be provided in one of the following ways:

- ▶ In plain YAML format in the resources subsection of play. This section is an array of descriptors and supports two types of descriptors: [Pods](#) and [ConfigMaps](#).

Example 3-7 illustrates how to define the resources in the play subsection.

Example 3-7 Resources in the play subsection

```
workload: |
  type: workload
  play:
    resources:
      - apiVersion: v1
        kind: Pod
```

```

metadata:
  name: busybox
spec:
  containers:
  - name: main
    image: ...
    command:
    - printenv
    envFrom:
    - configMapRef:
        name: contract.config.map
        optional: false
    restartPolicy: Never

```

- ▶ In the archive subsection of play. The archive is a Base64 encoded, compressed, tar file. The Pods or ConfigMaps are represented as YAML files, top level in this tar file. The file may also contain extra files and all files are extracted to the host file system before starting the Pods. The current working directory is the directory in which the files have been extracted, so it's possible to use a volume mount with a relative path to mount files or directories from the contract.
- ▶ In a template format in the templates subsection of play. This section is an array of descriptors in the YAML format. Pods or ConfigMaps may have points of variability (POV) that are not known at the time of writing the descriptors. These POVs may be represented as templates and the values are provided at deployment time from information in the contract. [Go templates](#) are used as the templating syntax, which is the same as used for [Helm charts](#), so templates can easily be exchanged with Kubernetes. The following Built-In objects are supported:
 - Environment: this object contains the environment variables as merged between the workload and the environment section. See Example 3-8. The object is available as `{{ .Env }}`.

Example 3-8 Environment variables

```

workload: |
  type: workload
  play:
    templates:
    - apiVersion: v1
      kind: Pod
      metadata:
        name: busybox
      spec:
        containers:
        - name: main
          image: "{{ .Env.REGISTRY
}}/hpse-docker-busybox-s390x@sha256:732efa374f1e6c964caeacab0bcb370385ee386041a14d
4a32176462e3f75c7b"
          command:
          - printenv
          envFrom:
          - configMapRef:
              name: contract.config.map
              optional: false
          restartPolicy: Never
env:

```

```
env:
  REGISTRY:
docker-eu-private.artifactory.swg-devops.com/sys-zaas-team-hpse-dev-docker-local/z
aas
```

Environment variables

In the contract, environment variables can be defined in the workload and env sections. Both sets of variables are merged together with workload taking precedence. Pods use the concept of a ConfigMap to define configuration, so HPCR represents the merged environment sections as a special ConfigMap named `contract.config.map`. Example 3-9 mounts all environment variables from the contract into the container.

Example 3-9 Environment variables in the contract

```
apiVersion: v1
kind: Pod
metadata:
  name: busybox
spec:
  containers:
  - name: main
    image: ...
    command:
    - printenv
    envFrom:
    - configMapRef:
        name: contract.config.map
        optional: false
  restartPolicy: Never
```

Pod communication

Pod communication can be container-to-container, pod-to-host, or pod-to-pod

► Container-to-container

Containers inside one Pod communicate to each other through the localhost. Each container needs to listen on a different port because, per design, they share the same IP address.

► Pod-to-host

Usually, a Pod needs to expose at least one of its containers to the host so that the exposed container is accessible through the IP address on the host through a mapped port. For this use case, use the `hostPort` feature on a container. This is not best practice when you use Kubernetes, for which a service would be used instead.

Important: Specify both `hostPort` and `containerPort` explicitly. If you specify only `containerPort`, ports are not bound.

See Example 3-10 for a snippet.

Example 3-10 Pod communication

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-with-busybox
```

```

spec:
  containers:
    - image: ...
      name: frontend
      ports:
        - containerPort: 80
          hostPort: 80
      volumeMounts:
        - mountPath: /etc/nginx
          name: local-frontend
          readOnly: true
    - command:
        - httpd
        - -vv
        - -f
        - -p
        - "8080"
        - -h
        - /www
      image: ...
      name: backend
      volumeMounts:
        - mountPath: /www
          name: local-backend
          readOnly: true
  volumes:
    - hostPath:
        path: ./www
        type: Directory
        name: local-backend
    - hostPath:
        path: ./nginx
        type: Directory
        name: local-frontend

```

► Pod-to-Pod

To reach from one Pod to another, expose a `hostPort` on the target Pod. The source Pod can then make a request to the host on the exposed port to get to the target Pod.

The source Pod can find the IP address of the host through the following command:

```
ip route | awk '/default/ { print $3 }'
```

Volumes

For HPCR, volumes are managed by the `volumes` section in the contract. Based on this information, HPCR will encrypt and mount external block devices on the host. To mount these volumes into the pod, use the `hostPath` mount option on the volume. See Example 3-11.

Example 3-11 Mount volume into pod

```

apiVersion: v1
kind: Pod
metadata:
  name: busybox
spec:

```

```

containers:
- name: main
  image: ...
  volumeMounts:
  - name: test-volume
    readOnly: true
    mountPath: /fromHost
volumes:
- name: test-volume
  hostPath:
    path: /var/hyperprotect
    type: Directory
restartPolicy: Never

```

Note: The volumes field here defines the data on the host to be mounted into the pod. It's different from volumes in the HPCR contract.

The images subsection

The images subsection is meant only for an image that is signed. See Example 3-12.

The following list includes aspects of images that are described by docker compose:

- ▶ The container image that is listed in the docker-compose file can be signed or not signed by using Docker Content Trust (DCT).
- ▶ The following example shows an image URL:

```
<container registry>/<username or namespace>/<image name>
```

An example with defined variables:

```
us.icr.io/mynamespace/my-haproxy:
```

- ▶ The following contents show an example of a notary URL:

```
notary: "https://notary.us.icr.io"
```

- ▶ The publicKey is the public key corresponding to the private key by which the images are signed using DCT. Use the following command to get the public key:

```
cat ~/.docker/trust/tuf/us.icr.io/<username>/<imagename>/metadata/root.json
```

Example 3-12 Image that is signed using DCT

```

images:
  dct:
    us.icr.io/mynamespace/my-haproxy:
      notary: "https://notary.us.icr.io"
      publicKey:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB0tLS0tCk1JSUJpRENDQVM2Z0F3SUJBZ01SQXVCMXBPY1pEQ1RRc0
9GSF1xazMzaWd3Q2dZSUtvWk16ajBFQXdJd0tqRW8KTUNZROExVUVBeE1mZFhNdWFXtn1MbWx2TDNcV1X
Sm9ZWFF4TWpNdmJYa3RhR0Z3Y205NGVUQWVGdzB5TWpBMApNVE14TURFd01ETmFGdzB6TWpBME1UQXhNRE
V3TUR0YU1Db3hLREftQmd0VkJBTVRIM1Z6TG1samNpNXBiET13CmNtRm1hR0YwTVRjEkwYMTVMV2hoY0hK
dmV1a3dXVEFUMdjcWhrak9QUU1CQmdncWhrak9QUU1CQnd0Q0FBU1AKWGsre1E2M1FZNjI3MMWQ1cTBMZH
Y3SGc3QzZkMGZOU1RsQmJXekh0WwFDZz1pU0piYnVNdjVBY0JmMj1qQi83eApqYzhzV1txMksyemtkTHV4
QWxGwM96VXdNekFPQmdOVkhROEJBZjhFQkFNQ0JhQXdFd11EV1IwbEJBD3dDZ11JCKt3WUJCUVVIQXdNd0
RBWURWUjBUQVFIL0JBSXdBREFLQmdncWhrak9QUVFEQWd0SUFkZBaU1zd0JTa0IxaXAKZHZZY1BMbFBM
S3RZT0hsYnZzU11Ka0FZM2hnY0xuNWhwQU1oQUt6cmhsU3p4K115bmdtMTB1ZVkyYFNCRmRgrawpMWHp6SF
kwaktTVzhyM1FhCi0tLS0tRU5EIEFUF1RJRk1DQVRFLS0tLS0k

```

For an image that is not signed, no entry is required in the images subsection. However, for unsigned images, a digest is required. Complete the following steps to get the digest:

1. Log in to the container registry dashboard.
2. Open the image.
3. Click Tag, and then click Digest.

After you get the digest, add this digest in the `docker-compose.yaml` file. The following entry is an example:

```
services:
  <imagename>:
    image:
      s390x/redis@sha256:db467ab5c53bdeef65762a7534e26fecb94a0f218bd38afd2eaba1a670c472b1
```

The following images are described by Pod descriptors:

- ▶ Container images that are described by Pod descriptors can be validated by Red Hat Simple Signing.
- ▶ If the image is referenced by a digest, the service allows its usage without additional checks.
- ▶ Images without a digest need a GPG key to be validated. The key is transferred in Base64 encoded binary format that can be created as shown in the following example:

```
gpg --export ${KEY_ID} | base64 -w0
```

This key is conveyed through the `rhs` subsection of the images section. This section is a map with the image identifier as the key and the GPG key in the `publicKey` field as shown in the following example:

```
images:
  rhs:
    OCI-image-identifier:
      publicKey: abcdef
```

3.1.2 The workload volumes subsection

The volumes subsection needs to be provided in the contract only if a data volume is attached to the instance at the time of creation. The information that is provided in this subsection is used to mount the attached data volume, which is provided by the user, and is later encrypted using the seeds that are provided in the workload and env sections. You can provide any path of your choice for the mount field. The path that is provided by the user is used internally to mount the data volume. The mount path that is provided in the contract must match the path provided under the volumes subsection of the `docker-compose.yaml` file, so that all the data associated with the container workload is stored in this data volume.

The volumes subsection has support for auto encryption of the data volume with user-provided seeds. If a data volume is attached to the HPVS instance, it is encrypted automatically with the seeds that are provided through the `seed` field in the volumes subsections of the contract. Thus, two seeds must be provided, one through the workload section, by the Workload Provider persona and the other through the env section by the Workload Deployer persona. These two seeds are internally converted to UTF8 sequences and then concatenated. Later, the hash (SHA256) of the concatenated sequence is computed as a hexdigest, which is used as the LUKS passphrase to encrypt the data volume.

You can use the following command to validate the hexdigest:

```
echo -n "seed1seed2" | sha256sum
```

This is how the seed can be provided in the workload section of the contract. For more information about how the seed input can be provided through the env section, see 3.1.3, “The env section” on page 47. It is mandatory to provide both the seeds for encryption. If only one of the seeds is provided then encryption fails and the instance shuts down.

Note that for deployments on IBM Cloud it is possible to add a higher level of encryption protection and control to the data-at-rest by integrating with Hyper Protect Crypto Services (HPCS). Starting with `ibm-hyper-protect-container-runtime-1-0-s390x-11` for HPVS for VPC instance or `ibm-hyper-protect-container-runtime-23.6.2-encrypt.crt` for HPVS instance on IBM LinuxONE or IBM Z version 23.6.2, HPCS can be used to generate a random value as the third seed and wrap it with the root key. The LUKS passphrase is generated by using three seeds: the seed in the metadata partition and the two seeds from the contract. For more information, see [Securing your data](#).

The following snippet is an example of the volumes subsection:

```
volumes:  
  
  test:  
  
    filesystem: ext4  
  
    mount: /mnt/data  
  
    seed: workload phrase
```

For more information on the volume subsection, see [HPVS for VPC: The workload - volumes subsection](#) and [HPVS on IBM LinuxONE and IBM Z: The workload - volumes subsection](#).

3.1.3 The env section

The env section is one of the most important sections in a contract and is mandatory. The env section of a contract deals with information that is specific to the cloud environment and is not known to the Workload Provider persona. This section is created by the Workload Deployer persona.

The env section is defined with `type: env`. The env section includes the following subsections:

- ▶ `logging`. This subsection is mandatory and tells the HPVS service where to send logging data.
- ▶ `volumes`. This subsection is optional and must be used only when a data volume is attached.
- ▶ `signingKey`. This subsection is optional and must be used only when a contract signature is used.
- ▶ `env`. This subsection is optional and used to specify values for env variables when defined by the Workload Provider, specifically.


```
crn:
"crn:v1:bluemix:public:hs-crypto:us-south:a/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:xxxxx
xxx-xxx-xxx-xxx-xxxxxxxxxxxx:key:xxxxxxxx-xxx-xxx-xxx-xxxxxxxxxxxx"

type: "private"

seed: "seed1"

kmsTimeout: 10
```

signingKey subsection

For information about how to use the signingKey, see “Contract signature” on page 52.

env subsection

If Pod descriptors are used in the workload section, see the example for the template format in “The play subsection” on page 41.

If a docker compose file is used in the workload section:

If the docker compose file has an environment section, use the following snippet as an example:

```
environment:
  KEY1: "${Value1}"
  KEY2: "${Value2}"
```

When the docker compose file has an environment section, as shown in the previous example, it is possible to pass the values in the env section of the Workload Deployer. The following example shows how to specify the values for the env variables:

```
env:
  value1: "abc"
  value2: "xyz"
```

For details about each section and subsection of the contract, see [IBM Hyper Protect Virtual Servers: About the Contract](#)

3.2 Contract encryption

When the HPVS instance boots, the bootloader decrypts the contract. It takes the contents of each of the sections in the contract and decrypts the sections that are encrypted.

Although the contract can be passed through the User Data field without encryption, the always encrypt the contract with an encryption certificate to protect all sensitive information.

It is possible to encrypt all the sections of the contract or just one. At a minimum, it is best practice to encrypt sections that have login credentials for container registries or LogDNA ingestion keys, for example.

Each encryption and attestation certificate is signed by the IBM intermediate certificate,¹ which in turn is signed by DigiCert Trusted Root G4. For more information about the certificates, see [DigiCert Trusted Root Authority Certificates](#).

¹ An intermediate certificate acts as a layer between the certificate authority and the end user's certificate.

Note: For illustration purposes only, the sample code in the examples throughout this chapter is unencrypted code.

Where to find the encryption certificate

For HPVS for VPC, and instance in IBM Cloud, do the following steps:

1. For instructions to download the encryption certificate, see [Downloading the encryption certificate and extracting the public key](#).
2. For instructions to validate the encryption certificate, see [Validating the contract encryption certificate](#).

For the HPVS instance on IBM LinuxONE or IBM Z do the following steps:

1. Follow the instructions from:
<https://www.ibm.com/docs/en/hpvs/2.1.x?topic=servers-downloading-hyper-protect-container-runtime-image>

Look for the file with name `ibm-hyper-protect-container-runtime-XX.YY.Z-encrypt.crt`
Where `XX.YY.Z` is the version being used.

The certificates are available in the `/config/certs` directory of the downloaded TAR.GZ package.

2. Validate the encryption certificate by using the instructions found in the download package.

Encryption of the workload section of the contract

To encrypt the workload section used in a contract, on an Ubuntu image, perform the following steps:

- ▶ Create the `docker-compose.yaml` file based on the workload requirements. For example:

```
services:
```

```
  redisnode01:
```

```
    image:
```

```
    s390x/redis@sha256:db467ab5c53bdeef65762a7534e26fecb94a0f218bd38afd2eaba1a670c472b1
```

```
    ports:
```

```
      - "6379:6379"
```

For more information, see [Docker Compose overview](#).

- ▶ Create the workload section of the contract and add the contents in the `workload.yaml` file. Do not include the top-level element "workload" of the workload section if it is encrypted. Example 3-1 on page 39 shows an example of a workload section in a format that can be encrypted.

For more information see the following documentation:

- HPVS for VPC: [IBM cloud: The workload section](#)
- HPVS on IBM LinuxONE or IBM Z: [IBM Hyper Protect Virtual Servers: The workload section](#)

- ▶ Export the complete path of the `workload.yaml` file and the public certificate (this can be `ibm-hyper-protect-container-runtime-1-0-s390x-11-encrypt.crt` for HPVS for VPC or the file in the `/config/certs` directory if you are using HPVS for IBM LinuxONE or IBM Z on-premises:

```
WORKLOAD="<PATH to workload.yaml>"
```

```
CONTRACT_KEY="<PATH to public certificate ...encrypt.crt>"
```

- ▶ Create a random password. The contract is encrypted through symmetric AES with a random PASSWORD):

```
PASSWORD_W="$(openssl rand 32 | base64 -w0)"
```

- ▶ Encrypt password with

```
ibm-hyper-protect-container-runtime-1-0-s390x-11-encrypt.crt:
```

```
ENCRYPTED_PASSWORD_W="$(echo -n "$PASSWORD_W" | base64 -d | openssl rsautl  
-encrypt -inkey $CONTRACT_KEY -certin | base64 -w0 )"
```

- ▶ Encrypt the workload.yaml file with a random password:

```
ENCRYPTED_WORKLOAD="$(echo -n "$PASSWORD_W" | base64 -d | openssl enc  
-aes-256-cbc -pbkdf2 -pass stdin -in "$WORKLOAD" | base64 -w0)"
```

- ▶ Get the encrypted section of the contract using the **echo** command:

```
echo "hyper-protect-basic.${ENCRYPTED_PASSWORD_W}.${ENCRYPTED_WORKLOAD}"
```

- ▶ Paste the encrypted section of the contract to the user-data.yaml file with the prefix "workload:" or use the **echo** command to start a new user-data.yaml file with an encrypted workload section:

```
echo "workload:  
hyper-protect-basic.${ENCRYPTED_PASSWORD_W}.${ENCRYPTED_WORKLOAD}" >  
user-data.yaml
```

It should be noted that the prefix hyper-protect-basic is mandatory.

Encryption of environment (env) section of the contract

We are using an Ubuntu image for our examples. Similar to the workload section, these are the steps to encrypt the env section used in a contract:

- ▶ Create the [env section](#) of the contract and add the contents in the env.yaml file. Do not include the top level element "env" in the environment subsection that is to be encrypted.

For more information see:

- HPVS for VPC:

https://cloud.ibm.com/docs/vpc?topic=vpc-about-contract_se#hpcr_contract_env

- HPVS on IBM LinuxONE or IBM Z:

https://www.ibm.com/docs/en/hpvs/2.1.x?topic=servers-about-contract#hpcr_contract_env

- ▶ Export the complete path of the env.yaml file and the public certificate. For HPVS for VPC, the public certificate can be `ibm-hyper-protect-container-runtime-1-0-s390x-11-encrypt.crt`. For HPVS for IBM LinuxONE or IBM Z on-premises, use the file in the `/config/certs` directory:

```
ENV="<PATH to env.yaml>"
```

```
CONTRACT_KEY="<PATH to public certificate ...encrypt.crt>"
```

- ▶ Create a random password. The contract is encrypted through symmetric AES with a random PASSWORD:

```
PASSWORD_E="$(openssl rand 32 | base64 -w0)"
```

- ▶ Encrypt password with public certificate:

```
ENCRYPTED_PASSWORD_E="$(echo -n "$PASSWORD_E" | base64 -d | openssl rsautl  
-encrypt -inkey $CONTRACT_KEY -certin | base64 -w0 )"
```

- ▶ Encrypt the `env.yaml` file with a random password:


```
ENCRYPTED_ENV="$(echo -n "$PASSWORD_E" | base64 -d | openssl enc -aes-256-cbc -pbkdf2 -pass stdin -in "$ENV" | base64 -w0)"
```
- ▶ Get the encrypted section of the contract by using the `echo` command:


```
echo "hyper-protect-basic.${ENCRYPTED_PASSWORD_E}.${ENCRYPTED_ENV}"
```
- ▶ Paste the encrypted section of the contract to the `user-data.yaml` file with the prefix `"env:"`, or use the `echo` command to append the encrypted section of the contract to the `user-data.yaml` file started in the previous workload section:


```
echo "env: hyper-protect-basic.${ENCRYPTED_PASSWORD_E}.${ENCRYPTED_ENV}" >> user-data.yaml
```

Contract signature

Contract signature is an optional feature that can be used to sign a contract before it is passed as input. Any contract can be signed regardless of it being encrypted (fully or partially) or in plain text. Validation of the contract signature is performed by the HPVS instance. The purpose of this signature feature is to ensure that the workload and `env` sections are always used together and are not tampered with by a third party. The signature of the workload and the `env` sections are added as the value to the `envWorkloadSignature` section of the contract. The following are two sections in a contract that are relevant while creating and adding a contract signature:

- ▶ `envWorkloadSignature`. This is a section where the signature of the other sections of the contract is added. This section is not required for a contract that is not signed.
- ▶ `signingKey`. This is a subsection that must be added to the `env` section of the contract. This holds the value to the user-generated public key, whose corresponding private key was used to create the contract signature.

Complete the following steps to create the contract signature. We are using an Ubuntu image for these examples:

- ▶ Use the following commands to generate a key pair to sign the contract. Note that `"CustomPassphrase"` is the passphrase to generate keys; you can use your own:


```
openssl genrsa -aes128 -passout pass:CustomPassphrase -out private.pem 4096
openssl rsa -in private.pem -passin pass:CustomPassphrase -pubout -out public.pem
```
- ▶ Use the following command to save the signing key from the `public.pem` certificate in `yaml` compatible format:


```
key=$(awk -vRS="\n" -vORS="\n" '1' public.pem)
echo "${key%\n}"
```

The key must be added to the `env` section of the contract with the prefix `"signingKey:"` before it is encrypted.
- ▶ To add the key, run the following command below in the same directory where the `env.yaml` file is located to append the signing key to the `env.yaml` file:


```
echo "${key%\n}" >> env.yaml
```

After appending the signing key, see “Encryption of environment (`env`) section of the contract” on page 51 to encrypt the `env.yaml` file with the signing key.
- ▶ Create the `contract.txt` file. Add the value of workload first then add the value of `env` from the `user-data.yaml` file. Ensure that there is no space or new line after workload and before `env`. Also, ensure that there is no new line or space at the end of the file. It is

recommended to cross-check the binary content of the `contract.txt` file with tools such as `hexdump`. In the binary file dump, make sure that there is no 0a ASCII value as the last entry. The `contract.txt` file should look similar to the following text:

```
hyper-protect-basic.js7TGt77EQ5bgkjhC0pViFTRHqWtn.....hyper-protect-ba
sic.VWg/5/SWE+9jLkjhr8q4i.....
```

Alternatively, if you defined the workload and `env` sections as discussed in 3.1.1, “The workload section” on page 39 and 3.1.3, “The `env` section” on page 47, then you can create the `contract.txt` file with the `echo` command:

```
echo
"hyper-protect-basic.${ENCRYPTED_PASSWORD_W}.${ENCRYPTED_WORKLOAD}hyper-protect
-basic.${ENCRYPTED_PASSWORD_E}.${ENCRYPTED_ENV}" > contract.txt
```

- ▶ A signature can then be generated with the `echo` command:

```
echo $( cat contract.txt | openssl dgst -sha256 -sign private.pem | openssl enc
-base64) | tr -d ' '
```

- ▶ This can then be added to the `user-data.yaml` contract with the prefix `"envWorkloadSignature:"` or use the `echo` command to append the signature directly in the `user-data.yaml` file that was started in the workload section and add to the `env` section:

```
echo "envWorkloadSignature: `echo $( cat contract.txt | openssl dgst -sha256
-sign private.pem -passin pass:CustomPassphrase | openssl enc -base64) | tr -d '
`" >> user-data.yaml
```

3.3 Contract certificates

To protect the contract a public and private key pair is created as part of the Hyper Protect Secure Build (HPSB) pipeline that is used to generate the HPCR image. This key pair is used to provide confidentiality for contract contents. The public X509 certificate of the Contract Encryption public key is published by IBM and can be validated with the 3rd party certificate authority root key by any persona out-of-band.

Each persona independently encrypts their contract section using this Contract Encryption public key. The contract encryption private key is randomized during image build and exists inside the Secure Execution encrypted HPCR image only. Such image can only be decrypted by using Secure Execution and keys rooted in hardware of the IBM Z or IBM LinuxONE platform. During deployment this key is used by the Bootloader to decrypt the Contract and the Bootloader ensures protection of this key from User space and the Workload.

Certificate Revocation List

A Certificate Revocation List (CRL) is a list of digital certificates that have been revoked by the Certificate Authority (CA) before their scheduled expiration date and should no longer be trusted. According to RFC 5280, a revoked certificate can be in one of two states:

1. Revoked. A certificate is irreversibly revoked if:
 - it is discovered that the CA improperly issued a certificate
 - the private-key for the certificate is compromised for some reason
2. Hold. A certificate can be put on hold if there is a chance that a private key is compromised. The hold can be removed if it is determined that the private key was not compromised.

The HPCR images used to build HPVS instances are accompanied by a CRL, so the user can verify that the certificates being used to validate contract encryption certificate and validate attestation certificate are in fact valid.

For more details, consult [IBM Hyper Protect Virtual Server for VPC: Validating the Certificates](#) and [IBM Hyper Protect Virtual Server: Validating the certificates](#)

3.4 Attestation

Attestation is the process with which a TEE or Hyper Protect Platform can provide evidence or measurements of its origin and current state. This evidence can be verified by another party, for example a party assuming the Auditor persona, who can then decide whether to trust the application running in the TEE or not. Typically, the attestation record is signed by a trusted key that is anchored in hardware that can be vouched for by a manufacturer. This is necessary to assure the Auditor that is representing the party validating the evidence was not created or altered by an unauthorized component or actor. The attestation record enables validation and proof by the Auditor persona with root of trust based in 3rd-party authority. See Figure 3-1.

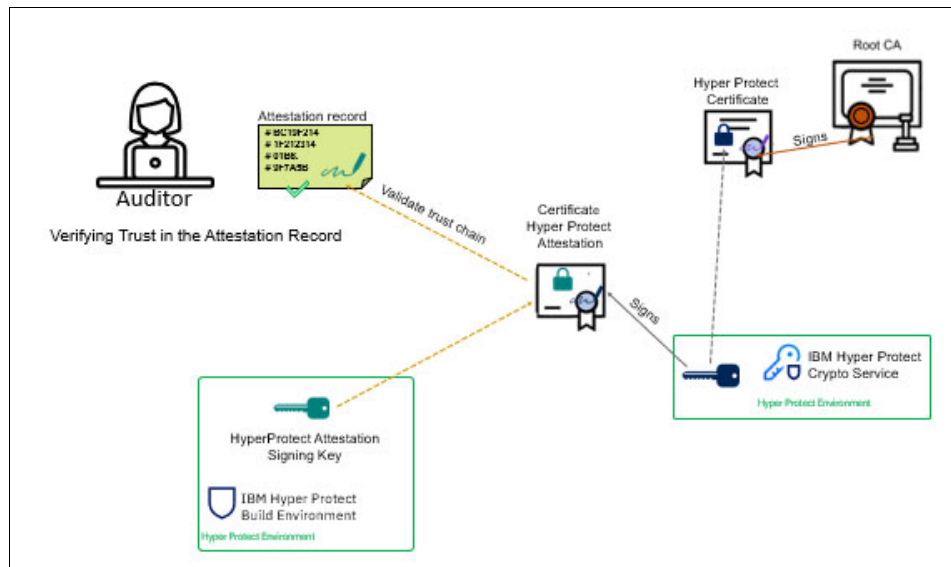


Figure 3-1 Verifying trust in an attestation record

The HPVS instance provides an *attestation record* that is securely generated and signed by the Bootloader during the instance deployment. The signing key is published as a X509 certificate and can be validated out-of-band to a 3rd-party certificate authority. The attestation record is available to the workload only if the `attestationPublicKey` section is provided in the contract.

The Workload Provider persona can implement means for providing the attestation record to the Auditor persona. The Auditor can then verify, out-of-band, the environment in which the workload was started.

Figure 3-2 on page 55 highlights the creation and management of the certificate hierarchy that is involved in signing the Attestation Record. The Attestation Record is signed by the Hyper Protect Attestation Signing Key (HPASK). The HPASK can be confirmed by the published intermediate certificate. The intermediate certificate is signed by a 3rd party

certificate authority, which is proven by the root certificate of that given certificate authority, thus completing the chain of trust.

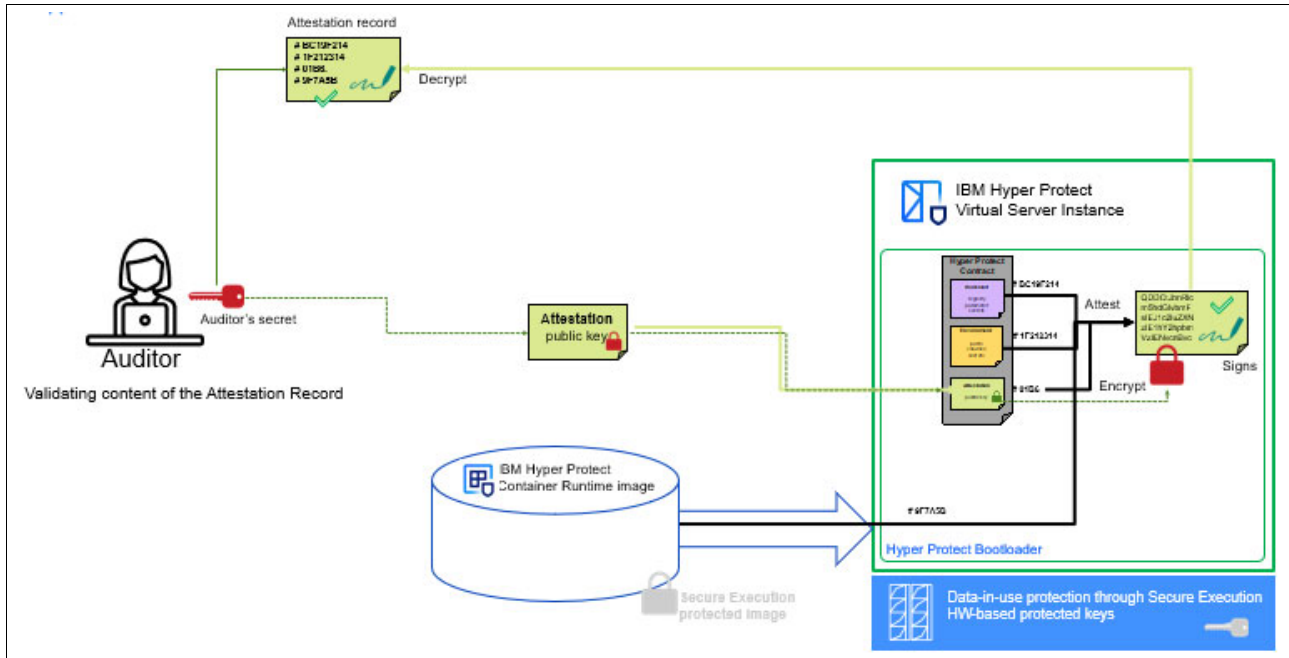


Figure 3-2 Validating content of an attestation record

An optional public key for encrypting the Attestation Record at the point of creation is provided by the Auditor persona. The public key is defined in the contract section: `attestationPublicKey`. The private part of this key is kept secret by the Auditor. The hash of this public key is added to the Attestation Record. By encrypting the Attestation Record and including the hash of the public key that is used for encryption within the Attestation Record, only the Auditor can decrypt the attestation record. Only the auditor can validate that the workload that is deployed in the enclave is the expected and untampered version of the workload that is expected to be deployed.

This attestation record contains measurements of what has been started and can be used to validate that the environment is the one deployed based on the following measurements:

- ▶ The original base image. The compressed root file system to be stored in the LUKS encrypted partition.
- ▶ The cloud initialization options, including the contract. The Hyper Protect Attestation Signing Key confirms with signature that the image got created in the HPSB pipeline.

The reference values for the measurements specific to the HPCR Image used are originated in the HPSB pipeline. Other measurements like the Cloud initialization options are dependent on the contract or can be used by the workload to provide insight about identifying individual instance, which enables the Auditor persona to validate that the deployment is as expected.

For more details, consult the following web pages:

- ▶ [IBM Cloud Hyper Protect for VPC: Attestation](#)
- ▶ [The Second Generation of Hyper Protect Platform](#)
- ▶ [Confidential Computing with SUSE Linux Enterprise Base Container Images Using the IBM Hyper Protect Platform](#)

3.5 Logging for HPVS instances

To start an HPVS instance, set up logging first by adding the logging configuration in the `env` section of the contract. The instance reads the configuration and configures logging. All other services start only after logging is configured. If the logging configuration is incorrect, the instance will not start and an error message is displayed in the serial console.

The logs include startup logs, service logs issued by the HPVS instance, and container logs.

Tip: For workloads that produce sensitive information, it is possible to encrypt log messages.

IBM Log Analysis in IBM Cloud

Logging to Log Analysis depends on the state and health of the Log Analysis service. Service outages might lead to a loss of log data. If you are logging data for audit purposes, consider using a logging service by performing the following steps:

1. Log in to your IBM Cloud account.
2. Provision a Log Analysis instance. Choose a plan according to your requirements.
3. After creating the Log Analysis instance, click it to open it and click **Open dashboard**.
4. Click the question mark icon at the lower left of the page to view Install Instructions.
5. On the Add Log Source page, under Via Syslog, click **rsyslog**.
6. Make a note of the ingestion key value at the upper right of the page, and the endpoint value. The endpoint value is contained in that starts with `*.*`. In Example 3-13, the endpoint value is `syslog-u.au-syd.logging.cloud.ibm.com`.

Example 3-13 Start log analysis

```
### START Log Analysis rsyslog logging directives ###

## TCP TLS only ##
$DefaultNetstreamDriverCAFile /etc/ld-root-ca.crt # trust these CAs
$ActionSendStreamDriver gtls # use gtls netstream driver
$ActionSendStreamDriverMode 1 # require TLS
$ActionSendStreamDriverAuthMode x509/name # authenticate by hostname
$ActionSendStreamDriverPermittedPeer *.au-syd.logging.cloud.ibm.com
## End TCP TLS only ##

$template LogDNAFormat,"<%PRI%>%protocol-version% %timestamp:::date-rfc3339%
%HOSTNAME% %app-name% %procid% %msgid% [logdna@48950
key=\"bc8a5ba9aa5c0c12b26c6c45089228a4\"] %msg%"

# Send messages to Log Analysis over TCP using the template.
*.* @syslog-u.au-syd.logging.cloud.ibm.com:6514;LogDNAFormat

### END Log Analysis rsyslog logging directives ##
```

7. Add these values in the `env` logging subsection of the contract:

```
env:
  logging:
    logDNA:
```



```
hostname: ${RSYSLOG_LOGDNA_HOSTNAME}
ingestionKey: ${LOGDNA_INGESTION_KEY}
```

For more information, see “The logging subsection” on page 48.

When the HPVS instance boots, it extracts the Log Analysis information from the contract and configures accordingly, so that all the logs are routed to the endpoint specified. The information can be seen on the console window of HPVS during boot up. After that happens, open the Log Analysis dashboard and view the logs from the virtual server instance.

syslog

Logging can be configured with a generic syslog backend such as an rsyslog server or a Logstash server. The HPVS instance uses TLS with mutual authentication to connect to the logging backend. Find the following pieces of information to configure logging:

- ▶ Syslog hostname and port.
- ▶ Certificate Authority (CA). The certificate used to verify the certificate chain both for client and server authentication. The same CA must be used for both the client and server certificates.
- ▶ Client certificate. Used to prove the client to the server, signed by the CA.
- ▶ Client key. A private key used by the virtual server instance to establish trust.

Complete the following parts of the contract with the information. The certificates and the key must be in PEM format²:

```
env: |
  type: env
  logging:
    syslog:
      hostname: ${HOSTNAME}
      port: 6514
      server: ${CA}
      cert: ${CLIENT_CERTIFICATE}
      key: ${CLIENT_PRIVATE_KEY}
```

Note: Make sure to use a strong digest algorithm for the certificates. Otherwise, the syslog server might reject the certificates.

Also, the port value can be changed to any valid TCP port number, however, 6514 is the default port that is used for secure logging. If you use the Crypto Express Network API, also called the Crypto Appliance, the default port must be used. A different port configuration is not supported by the Crypto Appliance.

² A container format that includes public certificate or the entire certificate chain (private key, public key, root certificates).

Preparation steps

You can use the following procedure to create the required certificates and keys. The example uses `openssl` and shows `bash` syntax.

1. Create a CA private key and a certificate signing request (CSR).

Prepare the `ca.cnf` configuration file:

```
[ req ]
default_bits = 2048
default_md = sha256
prompt = no
encrypt_key = no
distinguished_name = dn

[ dn ]
C = US
O = Logstash Test CA
CN = ca.example.org
```

Note: Make sure to update `dn` with your values. The actual values can be selected, and they do not play a role for the subsequent processing.

2. Create the key and certificate:

```
# create private key
openssl genrsa -out ca-key.pem 4096
# create CSR
openssl req -config ca.cnf -key ca-key.pem -new -out ca-req.csr
# create self-signed CA
openssl x509 -signkey ca-key.pem -in ca-req.csr -req -days 365 -out ca.crt
```

3. Create files used on the rsyslog server:

Prepare the `server.cnf` configuration file. It is important to set the `default_md` value to at least `sha256`. Make sure to complete the correct information for the `dn` field. It is preferred to use a domain name for `CN`, but an IP is also acceptable. For more information, see the OpenSSL documentation on Subject Alternative Name.

– Example that uses a hostname:

```
[ req ]
default_bits = 2048
default_md = sha256
prompt = no
encrypt_key = no
distinguished_name = dn

[ server ]
```

```
subjectAltName = DNS:${HOSTNAME}
extendedKeyUsage = serverAuth
```

```
[ dn ]
C = US
O = Rsyslog Test Server
CN = ${HOSTNAME}
```

- Example that uses an IP address:

```
[ req ]
default_bits = 2048
default_md = sha256
prompt = no
encrypt_key = no
distinguished_name = dn
```

```
[ server ]
subjectAltName = IP:${IP}
extendedKeyUsage = serverAuth
```

```
[ dn ]
C = US
O = Rsyslog Test Server
CN = ${IP_OR_HOSTNAME}
```

4. Create the key and certificate. Ensure the server certificate `server.crt` contains a SAN for the IP or the hostname, depending on whether the server is accessed through IP or hostname.

```
# create private key
openssl genrsa -out server-key.pem 4096
# create CSR for the server certificate
openssl req -config server.cnf -key server-key.pem -new -out server-req.csr
# have the CA created in (1) sign the certificate
openssl x509 -req -in server-req.csr -days 365 -CA ca.crt -CAkey ca-key.pem
-CACreateserial -extfile server.cnf -extensions server -out server.crt
```

5. Create files used on the client side for the HPVS instance.

Prepare the `client.cnf` configuration file:

```
[ req ]
default_bits = 2048
default_md = sha256
prompt = no
encrypt_key = no
```

```
distinguished_name = dn

[ dn ]
C = US
O = Logstash Test Client
CN = client.example.org
```

Note: Make sure to update dn with your values. Whether the actual values play a role depends on the `StreamDriver.Authmode` setting. In this example, we use the setting `StreamDriver.Authmode="x509/certvalid"`. In this case, the value of dn does not play a role because all valid client certificates are accepted. Adjust this according to your needs. For more information, see [StreamDriver.Authmode](#).

6. Create the key and certificate:

```
# create private key
openssl genrsa -out client-key.pem 4096

# create CSR for client auh
openssl req -config client.cnf -key client-key.pem -new -out client-req.csr

# have the CA created in (2) sign the certificate
openssl x509 -req -in client-req.csr -days 365 -CA ca.crt -CAkey ca-key.pem
-CAcreateserial -out client.crt

# export key to PKCS#8 format
openssl pkcs8 -topk8 -inform PEM -outform PEM -nocrypt -in client-key.pem
-out client-key-pkcs8.pem
```

Client setup steps

1. Configure the contract with the template in Example 3-14.

Example 3-14 Contract template

```
env: |
  type: env
  logging:
    syslog:
      hostname: ${HOSTNAME}
      port: 6514
      server: ${CA}
      cert: ${CLIENT_CERTIFICATE}
      key: ${CLIENT_PRIVATE_KEY}
```

2. Use the content of the following files in preparation to fill in the placeholders:

- The value for `${CA}` can be found in the `ca.crt` file. See Step 1 on page 58.
- The value for `${CLIENT_CERTIFICATE}` can be found in the `client.crt` file. See Step 5 on page 59.
- The value for `${CLIENT_PRIVATE_KEY}` can be found in the `client-key-pkcs8.pem` file. See Step 5 on page 59

- The values for `${HOSTNAME}`, `${CA}`, `${CLIENT_CERTIFICATE}`, and `${CLIENT_PRIVATE_KEY}` are strings without extra encoding or escaping. Regardless of their, ensure you use valid YAML. For more information, see [Scalars](#).

In Example 3-15, “|” (the pipe symbol) can be used to provide literal values, so the value of the certificates or keys can simply be pasted into the YAML file. The correct indentation must be observed. Note that the certificate values are truncated with “...”. For the complete example, see Appendix A, “Client contract setup sample files” on page 99.

Example 3-15 Client certificate - literal values

```
env: |
  type: env
  logging:
    syslog:
      hostname: ${HOSTIP} # eg 10.0.0.8 or ${HOSTNAME}
      port: 6514
      server: |
        -----BEGIN CERTIFICATE-----
        MII EuDCC AyCgAwI BAgI UBR9g6L5hivov7eNT00HSXW39oD0wDQYJKoZIhvcNAQEL
        BQAwXzELMAkGA1UEBhMCVVMxEzARBgNVBAGMCkNhbg1mb3JuaWExFDASBgNVBACM
        COxvcyBBbmd1bGVzMQwwCgYDVQQKDANJQk0xZzAVBgNVBAMMDmNhLmV4YW1wGUU
        ...
        -----END CERTIFICATE-----
      cert: |
        -----BEGIN CERTIFICATE-----
        MIID0zCCAjsCFFS5goaaDyhsJsUHv5WooqDg9ggGMAOGCSqGS Ib3DQEBcWUAMF8x
        CzAJBgNVBAYTA1VTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRQwEgYDVQQHDA1Mb3Mg
        QW5nZWxlczEMMAoGA1UECgwDSUJNMRCwFQYDVQQDDA5jYS5leGFtcGx1Lm9yZzAe
        ...
        -----END CERTIFICATE-----
      key: |
        -----BEGIN RSA PRIVATE KEY-----
        MII EogI BAAKCAQEA vQoaZ9z2ZU0sKCoJ+1TyzI7vN3Mhc2Q0sSYBwXrQIFUt4WW1
        pinXX0q1o4iPRnQsQPzhkN8b1ZrgI2Sfk1N8IdK8JHFc09yVWEKmnxNeIOgiOvj r
        k3nSTkDH7GZyZe0p0d+Dbk671P4cKoxi32JgSK2iFe1ZnYrgELiZFWbIZfkuy4Yz
        ...
        -----END RSA PRIVATE KEY-----
```

In Example 3-16 the new lines are replaced with `\n` and carriage returns are deleted to make sure the content fits in one line between the inverted commas. For more information, see [scalars in double-quoted style](#). The certificate values are truncated with “...”. For the complete example, see Appendix A, “Client contract setup sample files” on page 99.

Example 3-16 Client certificate - double-quoted

```
env: |
  type:env
  logging:
    syslog:
      hostname: ${HOSTIP} # eg 10.0.0.8 or ${HOSTNAME}
      port: 6514
      server: "-----BEGIN
CERTIFICATE-----\nMII FCTCCA vECFEp7wJLz4jNstIsVH2dUeHDN26ZyMAOGCSqGS Ib3DQEBcWUAMEE x
\nCzAJBgNVBAYTA1VTMRkwFYDVQQKDBBMB2dzdGFzaCBUZXR0IENBMRCwFQYDVQQD \nDA5jYS5leGFtcG
x1Lm9yZzAeFw0yMzAxMDUxNjU0MTNaFw0yNDAxMDUxNjU0MTNa...\n-----END CERTIFICATE-----\n"
```

```

cert: "-----BEGIN
CERTIFICATE-----\nMIIFETCCA\nkCFBhx5DuYtRzCxRx8Bo+WIS2LFI2uMAOGCSqGS\nIb3DQEB\nCwUAMEE\nX\nCzAJBgNVBAYTA1VTMRkwFwYDVQQKDBBmb2dzdGFzaCBUZXNOIENBMRCwFQYDVQ\nQD...\n-----END
CERTIFICATE-----\n"
key: "-----BEGIN PRIVATE
KEY-----\nMIIJQQIBADANBgkqhkiG9w0BAQEFAASCCS\nswggknAgEAAoICAQCtj437cgRishC1\n0w9PrE\nyqSxJLjeDb7jgR1iI82ic/YqMRR0b+DnsIGcg5pR9nK+DcVz1E4EyGphro...\n-----END PRIVATE

```

You can also use other valid YAML variations. Out of the two variations described, the literal variation is considered more user friendly as it allows for easier visualization of the complete YAML file. However, unless a YAML compatible editor is used, spaces must be added to the beginning of the pasted lines to match the correct indentation. In this case, it might be easier to use double-quoted variants and a simple script to output the certificate and key files in the correct form. See Appendix A, “Client contract setup sample files” on page 99 for sample script `yaml_doublequoted_input.sh`.

Server setup steps

There are many ways to set up a compatible server endpoint. The following steps provide a simple setup of an rsyslog server:

1. We are using an Ubuntu image for our examples. Install the required server packages:
 - `apt-get install rsyslog rsyslog-gnutls`
2. Get certificates and keys from the preparation steps:
 - `ca.crt` - from *Preparation Step 1 on page 58*, copy to `/certs/ca.crt`
 - `server.crt` - from *Preparation Step 3 on page 58*, copy to `/certs/server.crt`
 - `server-key.pem` - from *Preparation Step 3 on page 58*, copy to `/certs/server-key.pem`
3. Configure the rsyslog server in the `/etc/rsyslog.d/server.conf` file. See Example 3-17.

Example 3-17 rsyslog server config file

```

# make gtls driver the default and set certificate files
$DefaultNetstreamDriver "gtls"
$DefaultNetstreamDriverCAFile /certs/ca.crt
$DefaultNetstreamDriverCertFile /certs/server.crt
$DefaultNetstreamDriverKeyFile /certs/server-key.pem
# provides TCP syslog reception
module(load="imtcp"
        StreamDriver.Name="gtls"
        StreamDriver.Mode= "1"
        StreamDriver.Authmode="x509/certvalid" # Currently, CA does not support
#       StreamDriver.Authmode="anon" # Use this for CA server
)
input(type="imtcp" port="6514")
# Template for incoming logs
$template
incoming-remote-logs, "/var/log/remotelogs/%FROMHOST-IP%/%PROGRAMNAME%.log"
*.* ?incoming-remote-logs

```

For more information, see [rsyslog](#). The example config will log the received logs to the directory and file `/var/log/remotelogs/%FROMHOST-IP%/%PROGRAMNAME%.log`. So `host-ip` is the IP of the remote host that is sending logs and the program name is the running application that is sending logs. In a production configuration, you might want to forward the logs to a database, but this is outside of the scope of this documentation.

Note: The `gnutls` package imposes a requirement for the signatures for the client certificate. For more information, see [Digital signatures](#).

Also, in this configuration, logs are accepted from any client certificate that is signed by the certificate authority through the `x509/certvalid` mode. This might change depending on the `StreamDriver.Authmode` setting. For more information, see [StreamDriver.Authmode](#).

If you are setting up a logging server that will also be used by the Crypto Express Network API, then uncomment the line `StreamDriver.Authmode="anon"` and comment the previous line, `StreamDriver.Authmode="x509/certvalid"`, because the Crypto Appliance is not compatible with `StreamDriver.Authmode="x509/certvalid"`.

4. Restart the `syslog` service:

```
service syslog restart
```

3.6 Encrypting data volumes

The data volume that can be attached to an HPVS instance is protected by a Linux Unified Key Setup (LUKS) encryption passphrase that is derived from seeds provided during deployment.

For workloads on IBM Cloud, a higher level of encryption protection and control is possible by combining the seeds with an additional secret that is generated by, and held within, the HPCS. HPCS is backed by FIPS 140-2 Level 4-certified hardware, which is the highest offered by any cloud provider in the industry.

Before proceeding with this offering, thought should be given to the availability aspects of HPCS from the on-premises environment, such as additional physical internet connections.

How your data volume is encrypted

Without your own key, the data volume that is attached to the instance is encrypted automatically with two seeds that are provided in the `workload - volumes` and `env - volumes` sections of the contract. The seeds are internally converted to UTF8 sequences and then concatenated. The hash (SHA256) of the concatenated sequence is computed as a hexdigest, which is used as the LUKS passphrase to encrypt the data volume. For more information, see 3.1, “The contract” on page 38.

Protecting your sensitive data with your own key

Key management service (KMS) support is integrated in HPCS for HPVS for VPC with `ibm-hyper-protect-container-runtime-1-0-s390x-11` and for HPVS for IBM LinuxONE or IBM Z with product version 2.1.5.

HPCS generates a random value as the third seed and wraps it with the customer root key (CRK). The wrapped seed is stored in the metadata partition of your data volume. The LUKS passphrase is generated by using three seeds: the seed in the metadata partition, which is unwrapped first, and the two seeds from the contract. See Figure 3-3.

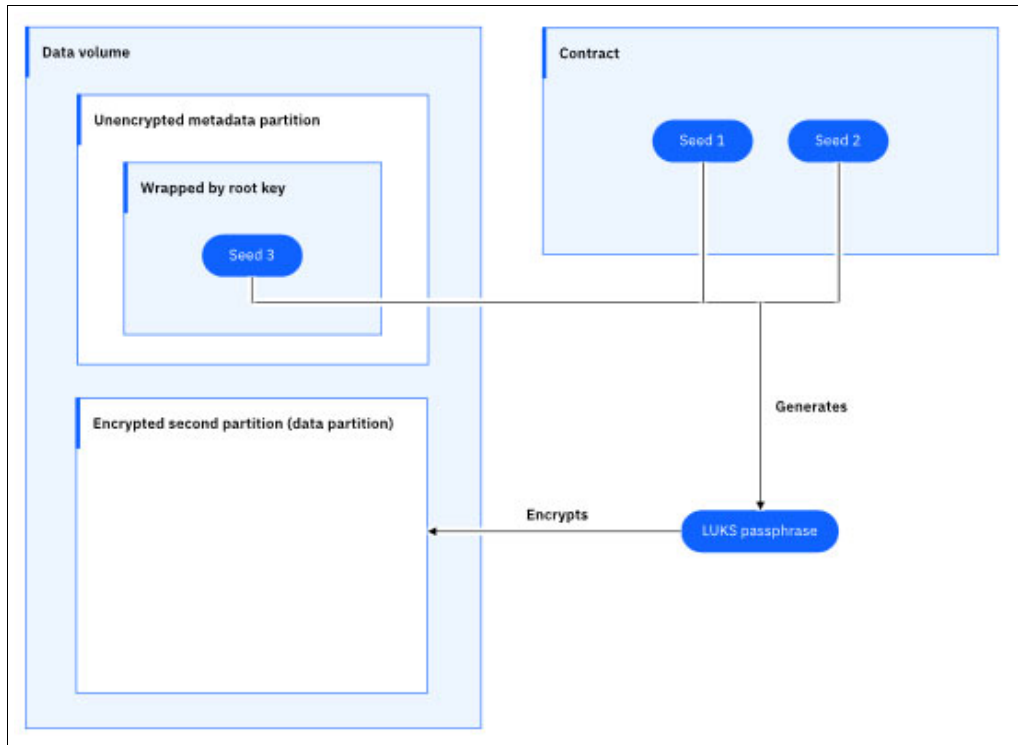


Figure 3-3 Integration with key management service

Note: For new HPVS instances, the data volume is partitioned into two parts. The first partition of 100 MiB is reserved for internal metadata only. It is not to be accessed by a workload. The second partition remains as the data volume for workload. Only new volumes are partitioned.



Application development in a trusted environment

This chapter describes how IBM Hyper Protect Services can be used to establish a trusted application development and deployment process. It shows how secure practices can be used to improve each part of the development process and different parts of deployed applications.

The chapter topics include improvements of the application lifecycle and application development, test, build, and release. It includes descriptions of initial and update deployment, and aid for secure implementation of the different steps is provided. Where feasible, multiple options are provided as alternatives. Less secure implementation or configuration options might be omitted.

The concepts are demonstrated with sample code that uses a combination of shell, makefiles, and Terraform commands to show how the resources can be deployed. Sample code of deployed applications is written in `golang` and `JavaScript`.

For more information about the source code from this chapter, see [IBM hyperprotect GitHub repository](#).

Required configuration values are supplied in the form of environment variables and Terraform variable files. For convenience, the environment variables can be set from `.env*` files located next to the Makefile files within the different directories. The required values are outlined and described in sample files.

In this chapter, the following topics are discussed:

- ▶ 4.1, “Securing the application lifecycle” on page 66
- ▶ 4.2, “Build container image by using Hyper Protect Secure Build” on page 71
- ▶ 4.3, “Zero knowledge proofs: TLS server certificates and wrapped secrets” on page 78
- ▶ 4.4, “Trust in-depth based on boot flow attestation” on page 85
- ▶ 4.5, “Data storage” on page 87
- ▶ 4.6, “Securing cloud native services” on page 92
- ▶ 4.7, “Secure supply chain with SLSA” on page 95

4.1 Securing the application lifecycle

The use cases that are described in this chapter are based on the application lifecycle of a secured application that presents an interface for making monetary payments. The application is called *SamplePaymentSystem* and shows how sensitive payment-related information, such as credit card data, is used in a secure way. A key requirement is to run this application in a confidential computing environment to ensure PII data that is in use is protected from malicious actors.

Figure 4-1 shows a high-level overview of the Hyper Protect components that are involved in the development lifecycle of the *SamplePaymentSystem* application.

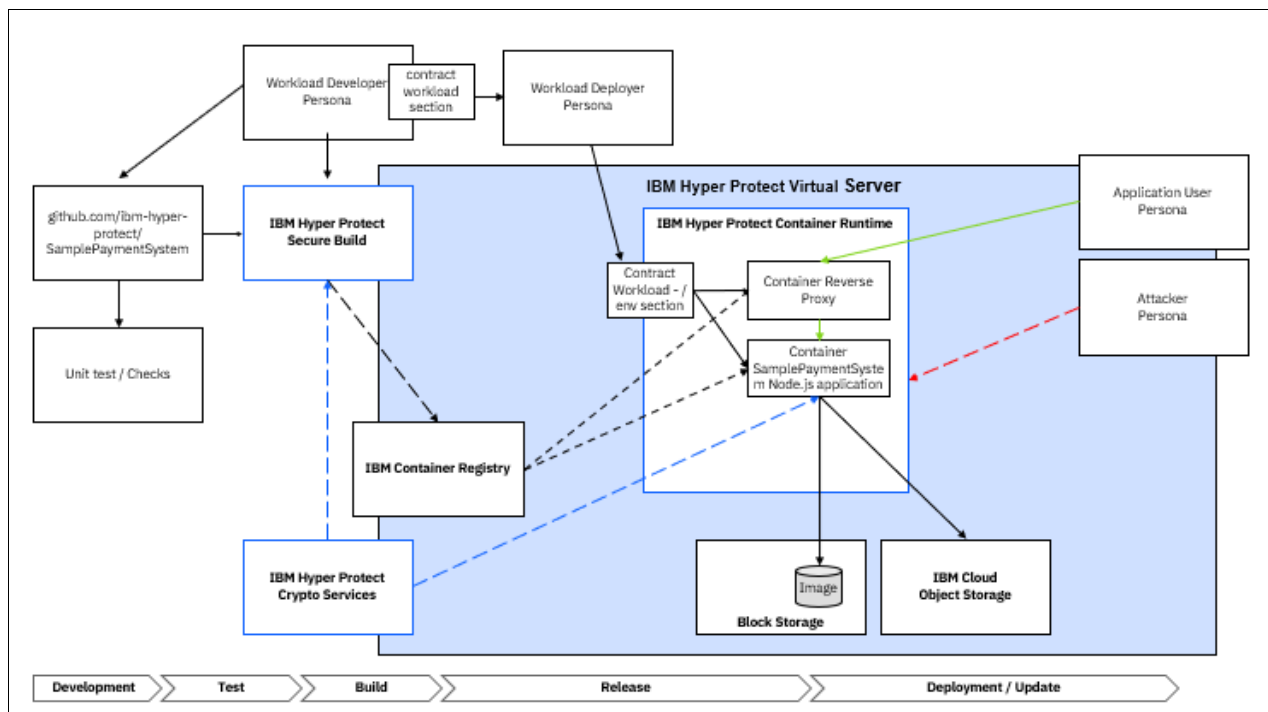


Figure 4-1 *SamplePaymentSystem* application lifecycle using Hyper Protect services

For an explanation of the different Hyper Protect components, refer to Chapter 2, “Understanding the solution” on page 15. For a description of the personas see, “Separation of duty” on page 11.

The subsequent section includes discussion of the development lifecycle phases:

- ▶ Development
- ▶ Test
- ▶ Build
- ▶ Release
- ▶ Deployment
- ▶ Update

4.1.1 Development

Code is stored and developed against a source-code management system, which is predominantly Git. The most prominent hosted Git service currently is GitHub. For more information, see github.com.

Code scanning and tests are run by a pipeline such as Jenkins, Tekton on Red Hat OpenShift Pipelines, GitHub Actions. Also, see 4.7, “Secure supply chain with SLSA” on page 95. You can use the pipeline model to run various checks on source code that is committed to a repository. These checks are typically run against the source code directly, but can also be run during the build steps, test cases, or ephemeral automatic test deployments. Not all checks that are run need to be blocking or critical. It is also possible to run checks that are not preventing integration of the code but are raising only informational findings. To distinguish this, the repository is configured with the required status checks. For more information, see [About protected branches](#).

These repository settings can also be used to force commit signatures to allow cryptographically secured tracking of the code authorship. Other source-code-hosting facilities typically provide similar functionality.

4.1.2 Test

The functionality is typically secured by tests on multiple test layers. While in development, the code is locally tested by using unit tests. These unit tests are designed to be portable and quick to run, and are often implemented against mocked services and static test data.

After a push into the source-code management, more complicated tests can be run. These tests can include integration tests against other services that are deployed within a test deployment.

Depending on the stage in the application lifecycle, the application is then either promoted for integration, acceptance testing, or for production deployment.

The steps that follow are the same for all cases and differ by only the environment in which the application is deployed.

4.1.3 Build

Depending on the specific implementation of the development pipeline and application, the developer can choose to push the developed and tested code forward into the integration test. To be able to run the application within a secured environment, it needs to be built into either a custom image or into a container to be started within the HPCR image.

With Hyper Protect Secure Build (HPSB), you can build a trusted container image within a secure enclave that is provided by an HPVS instance. The enclave is highly isolated, where developers can access the container only by using a specific hardened API and the cloud administrator cannot access the contents of the container. Therefore, the image that is built can be highly trusted. Specifically, the build server cryptographically signs the image and a manifest. The manifest is a collection of materials that are used during the build and can be used for audits. Because the enclave protects the signing keys within the enclave, the signatures can be used to verify whether the image and manifest are from the build server and not from elsewhere.

For more details on configuring and using HPSB in IBM Cloud VPC and IBM LinuxONE or IBM Z, see [Configuring and using Hyper Protect Secure Build in Hyper Protect Virtual Servers for VPC](#) and [Building your applications with Hyper Protect Secure Build](#)

4.1.4 Release

Releases can be targeted to internal or external deployment for different purposes ranging from integration testing to production use.

Along with the published container images the Workload Provider persona publishes the encrypted workload section of the contract.

For detailed information on the contract creation see 3.1, “The contract” on page 38 and [IBM Cloud VPC: The workload section](#) and [IBM Hyper Protect Virtual Servers: About the contract](#).

The workload section of the contract defines whether there is a single container through the compose subsection or multiple containers through the play subsection. For the SamplePaymentSystem application, we configure three containers in the play subsection. For more information, see [hyperprotect/redbook-samples/sf248555](#).

The workload section of the contract can optionally contain the references and login credentials to the container registry and required volumes. Because it is encrypted by the contract encryption key of the workload image, the workload section is not readable by the Workload Deployer persona or anyone else, such as a system administrator.

Note: The contract should always be encrypted with an encryption certificate to protect all sensitive information. See 3.2, “Contract encryption” on page 49 for details.

For illustration purposes only, we show unencrypted sample code throughout this chapter.

4.1.5 Deployment

To deploy the application, the Workload Deployer persona complements the workload section of the contract with the env section as needed for the current scenario.

For detailed information on the contract creation see 3.1.3, “The env section” on page 47 or [IBM Cloud: The env section](#) and [IBM Hyper Protect Virtual Servers: The env section](#).

4.1.6 Update

Updating a service that is deployed on a Hyper Protect Virtual Server (HPVS)¹ can affect multiple components. When a new Hyper Protect Container Runtime (HPCR) image is released, it is recommended to update to the new version. New revisions of the images are typically released every four to six weeks and contain fixes and new features.

Furthermore, the containers and services running within the runtime might need to be updated for the same reasons of applying fixes and new features.

Because the HPCR follows a container lifecycle, an update always requires a restart of the HPVS instance and is thus disruptive for the node. In cases where high availability (HA) capabilities are required for the overall service, it is necessary to stagger the rollout and

¹ An HPVS instance is also referred to as a Virtual Server Instance (VSI) as part of a Virtual Private Cloud within IBM Cloud.

update of each HPVS so that the overall service remains available during the update. All updates are triggered by updates to the contract.

Updating the HPCR image

When a new image is available within the cloud or on-premises, it can be selected when starting the HPVS. When a different image is selected, it might be necessary to re-encrypt the contract with HPCR-Contract-Encryption-Public-Key. When working with a separate Workload Provider, they must provide the encrypted workload section first.

To build a complete contract for HPVS on IBM LinuxONE or IBM Z, see [Downloading the IBM Hyper Protect Container Runtime image](#)

To build a complete contract for HPVS for VPC, the creation can be done by using the IBM Cloud virtual server for VPC User Interface. See [Virtual server for VPC](#). Also, refer to Appendix B, “Creating a Hyper Protect Virtual Server for VPC” on page 107 for the corresponding steps.

Updating service containers

When new container versions for the service are available, the HPVS instance must be restarted to pull the image and run the new service version.

When a floating tag with container signing is used to define the container versions, the change might not be visible on the contract, and is in effect only at runtime. Otherwise, the digest of the images needs to be updated.

Depending on the workload, the composition of containers might change.

4.1.7 Application and service development

Within the context of the zero-trust architecture, each service must be implemented with security in mind. This is true for services communicating over the public internet and within the seemingly protected intranet.

To implement services following this posture, it is useful to consider each microservice or service to be always under attack. Because of this, no connection should be trusted. Each connection must be protected and verified.

Furthermore, the development team must be prepared to fix and redeploy any service rapidly.

As previously outlined, there exist multiple attack vectors on any service that make it necessary to update the service code itself or its dependencies. See 1.1, “Identifying the threat” on page 2. To be able to develop, test, and deploy service updates, various best practices can be followed.

A good starting point is the 12-factor app. For more information, see [The Twelve-Factor App](#).

Developing services this way contributes to testability, portability, scalability, and security.

4.1.8 Working with the log

The log of the HPVS image does contain information that originated from multiple sources within the HPVS.

Upon connection to the log server, the boot log is replayed. The first item that is replayed is the kernel startup. After the initial kernel messages, the rest of the HPVS boot process is

visible. The boot and HPVS configuration is internally orchestrated using systemd. After the contract validation and mounting of eventual data volumes the log shows the start of the workload containers.

Different boot stages record their states into the log for audit purposes. Notable log tokens have the format HPL[0-9]{5}[IE]. Other tokens are logged during the boot and deployment process of an HPVS instance.

For example, a successful startup of the instance logs the special token HPL10001I. See Example 4-1.

Example 4-1 Successful startup example

```
hpcr-dnslookup: HPL14000I: Network connectivity check completed successfully.
hpcr-logging: HPL01010I: Logging has been setup successfully.
hpcr-disk-mount: HPL07003I: Mounting volumes done
hpcr-container-play: HPL15004I: The pod started successfully.
verify-disk-encryption: HPL13000I: Verify LUKS Encryption
verify-disk-encryption: HPL13003I: Checked for mount point /, LUKS encryption with
1 key slot found
verify-disk-encryption: HPL13001I: Boot volume and all the mounted data volumes
are encrypted
hpcr-catch-success: HPL10001I: Services succeeded -> systemd triggered
hpl-catch-success service
```

A failure to start the containers that are defined within the contract logs the special token HPL10000E and schedule a shutdown of the HPVS. See Example 4-2.

Example 4-2 Failed startup example

```
hpcr-logging: HPL01010I: Logging has been setup successfully.
hpcr-disk-mount: HPL07003I: Mounting volumes done
hpcr-catch-failure: VSI has failed to start!
hpcr-catch-failure: HPL10000E: One or more service failed -> systemd triggered
hpl-catch-failed service
hpcr-catch-failure: Shutdown scheduled, use 'shutdown -c' to cancel.
systemd: Finished Trigger Catch failed service and shutdown.
```

The workload containers are expected to log to stdout and stderr. These outputs are gathered into the journal of the HPVS. From the journal, the messages are forwarded to the configured log server.

Log configuration

The forwarding of the log to a remote server is mandatory and is configured within the env section of the contract. For details on the log configuration see 3.5, “Logging for HPVS instances” on page 56.

A single logDNA instance can be used to capture the logs of multiple HPVSs and other servers.

4.1.9 Deployment automation - Terraform

To easily deploy and manage applications based on HPVS, a certain degree of automation is encouraged. The primary mode to drive the deployment and configuration of cloud infrastructure and workload is Terraform. For more information, see [Terraform](#).

The IBM-Cloud/ibm Terraform plug-in is required to manage all deployments on IBM Cloud. IBM provides a separate ibm-hyper-protect/hpcr Terraform plugin specifically to simplify the handling of the HPVS contract.

For more information, see the following websites:

- ▶ [Sample Terraform templates for IBM Cloud](#)
- ▶ [Terraform samples for Hyper Protect Virtual Servers for VPC](#)
- ▶ [ibm-hyper-protect/linuxone-vsi-automation-samples](#)

4.2 Build container image by using Hyper Protect Secure Build

You can build a trusted container image within a secure enclave that is provided by an IBM HPVS. See Figure 4-2. The enclave is highly isolated. Developers can access the secure build server by using a specific API, and the administrator cannot access the contents of the secure build server. Therefore, the image that is built can be highly trusted. Specifically, the build server cryptographically signs the image, and a manifest. The manifest, a collection of materials that are used during the build, is for audit purposes. Because the enclave protects the signing keys within the enclave, the signatures can be used to verify that the image and manifest are from the secure build server.

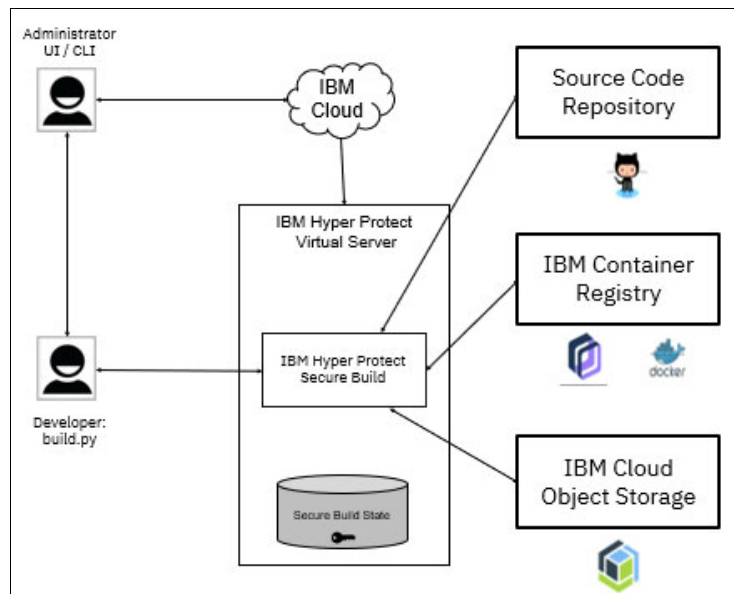


Figure 4-2 Trusted container image

To securely build container images, follow the process in the subsequent sections:

- ▶ “Determine readiness” on page 71
- ▶ “Install the secure build CLI” on page 72
- ▶ “Create client and server certificates for secure build” on page 72
- ▶ “Prepare user_data.yaml” on page 73
- ▶ “Create the Hyper Protect Secure Build instance” on page 74
- ▶ “Configure the HPSB client with the HPVS IP address” on page 75

4.2.1 Determine readiness

Ensure that you meet the following hardware or software prerequisites:

- ▶ Linux management server from where you can run the build CLI tool
- ▶ Recommended 2 CPUs/4 GB memory or more
- ▶ KVM hosts in IBM Secure Execution mode are supported by these distributions:
 - Red Hat Enterprise Linux 9.0 with service
 - Red Hat Enterprise Linux 8.4 with service
 - SUSE Linux Enterprise Server 15 SP3 with service
 - Ubuntu Server 20.04 LTS with service
- ▶ KVM guests in IBM Secure Execution mode are supported by these distributions:
 - Red Hat Enterprise Linux 9.0 with service
 - Red Hat Enterprise Linux 8.4 with service
 - Red Hat Enterprise Linux 7.9 with service
 - SUSE Linux Enterprise Server 15 SP3 with service
 - SUSE Linux Enterprise Server 12 SP5 with service
 - Ubuntu Server 20.04 LTS with service
- ▶ Attestation is available on IBM z16™ and IBM LinuxONE 4 by these distributions:
 - Red Hat Enterprise Linux 9.1 with service
 - Red Hat Enterprise Linux 8.7 with service
- ▶ Python 3.8 (Python 2.x is not supported)
- ▶ Access to GitHub, for hosting the source code
- ▶ Dockerfile (everything that you need to build your container image)
- ▶ Access to IBM Cloud Registry
- ▶ (Optional) Access to IBM Cloud Object Storage (COS) Service
- ▶ Access to IBM HPVS for VPC
- ▶ Get the encrypted workload section of the contract file of Secure Build from [Configuring and using Hyper Protect Secure Build in Hyper Protect virtual server for VPC](#)

4.2.2 Install the secure build CLI

Use the following steps to install the secure build command line interface (CLI):

1. On the client machine where Linux is installed, verify the OS version. See Example 4-3.

Example 4-3 Verify the OS version

```
$ cat /etc/os-release | grep 20.04
VERSION="20.04.6 LTS (Focal Fossa)"
PRETTY_NAME="Ubuntu 20.04.6 LTS"
VERSION_ID="20.04"
```

2. Download the secure build CLI code to the client machine where linux is installed. See Example 4-4.

Example 4-4 Download secure build CLI

```
$ git clone git@github.com:ibm-hyper-protect/secure-build-cli.git
Cloning into 'secure-build-cli'...
```

4.2.3 Create client and server certificates for secure build

Use the following steps to create certificates for the secure build CLI and secure build communication:

1. Create a client certificate for the secure build CLI and secure build communication. See Example 4-5 on page 73.

Example 4-5 Create client certificate

```
$ ./build.py create-client-cert --env sbs-config.json
INFO:__main__:parameter file sbs-config.json renamed to
sbs-config.json.2023-07-21_14-35-53.197898
INFO:root:client_certificate: generating client CA and certificate
```

2. Create a server certificate for the secure build CLI and secure build communication. See Example 4-6.

Example 4-6 Create server certificate

```
$ ./build.py create-server-cert --env sbs-config.json
INFO:root:server_certificate: using supplied pem files
cert_directory=.HPSBContainer-2e6ce21c-4c05-4101-abea-564f3adbd8e4
capath=./.HPSBContainer-2e6ce21c-4c05-4101-abea-564f3adbd8e4.d/client-ca.pem
cakeypath=./.HPSBContainer-2e6ce21c-4c05-4101-abea-564f3adbd8e4.d/client-ca-key.pem
INFO:root:server_certificate: Generating server certificate
INFO:root:server_certificate: Successfully generated server CSR
INFO:root:server_certificate: Successfully generated server certificate
```

3. Get the environment key value pair to be used in instance-create command. See Example 4-7.

Example 4-7 Get environment key value pair

```
$ ./build.py instance-env --env sbs-config.json
INFO:root:client_certificate: using supplied pem files
client_cert_key=.HPSBContainer-2e6ce21c-4c05-4101-abea-564f3adbd8e4
capath=./.HPSBContainer-2e6ce21c-4c05-4101-abea-564f3adbd8e4.d/client-ca.pem
cakeypath=./.HPSBContainer-2e6ce21c-4c05-4101-abea-564f3adbd8e4.d/client-ca-key.pem
WARNING:gnupg:pgp returned a non-zero error code: 2
INFO:__main__:

***** Copy below environment variables and use in env contract as environment
variables. *****

CLIENT_CERT: "LS0tLS1CRUdJTiBD.....RCBDRVJUSUZJQ0FURS0tLS0tCg=="
CLIENT_CA: "LS0tLS1CRUdJTiB.....5EIENFU1RJRk1DQVRFLS0tLS0k"
SERVER_CERT: "LS0tLS1CRUdJTiBDRVJUSUZJ.....ZC0tLS0tRU5EIENFU1RJRk1DQVRFLS0tLS0k"
SERVER_KEY: "LS0tLS1CRUdJTiBQR1Ag.....U1NBROUtLS0tLQo="
```

4.2.4 Prepare user_data.yaml

Use the following steps to prepare the encrypted env section of the contract:

1. Compose a plain text contract. See Example 4-8.

Example 4-8 Plain text contract

```
env: |
  type: env
  logging:
    logDNA:
      hostname: syslog-a.us-***.ibm.com
```

```
    ingestionKey: *****
    port: 6514
volumes:
  hpsb:
    seed: "*****"
env:
  CLIENT_CRT: LS0tLS1CRUdJTiBD.....RCBDRVJUSUZJQ0FURS0tLS0tCg==
  CLIENT_CA: LS0tLS1CRUdJTiB.....5EIENFU1RJRk1DQVRFLS0tLS0K
  SERVER_CRT: LS0tLS1CRUdJTiBDRVJUSUZJ.....ZCi0tLS0tRU5EIENFU1RJRk1DQVRFLS0tLS0K
  SERVER_KEY: LS0tLS1CRUdJTiBQR1Ag.....U1NBROUtLS0tLQo=
```

2. Encrypt the env section of the contract (see Example 4-9).

Example 4-9 Encrypted env section

```
env: hyper-protect-basic.ItMcZ+CaxWp4YbUMs2eVF6o7hiaRDMhgWpWaTWChg2a/.
/7cwKUEthQ1ww=
```

3. Get the encrypted secure build workload of the contract. See Example 4-10.

Example 4-10 Encrypted workload section

```
workload:
hyper-protect-basic.JNNGRfeic/H4j5Xc0LMMCA2KrAUu5+gc05.....ax4E2mCzHZMUuHVk3U
```

4. Add the encrypted content from the env and workload sections of the contract to prepare the user_data.yaml. See Example 4-11.

Example 4-11 Combined encrypted env and workload sections

```
env: hyper-protect-basic.ItMcZ+CaxWp4YbUMs2eVF6o7hiaRDMhgWpWaTWChg2a/.
/7cwKUEthQ1ww=
workload:
hyper-protect-basic.JNNGRfeic/H4j5Xc0LMMCA2KrAUu5+gc05.....ax4E2mCzHZMUuHVk3U
```

4.2.5 Create the Hyper Protect Secure Build instance

The HPSB instance can run in an HPVS for VPC instance and in HPVS on IBM LinuxONE or IBM Z running in a KVM host LPAR:

- ▶ To create an HPVS for VPC instance for the HPSB using IBM Cloud VPC UI, see Appendix B, “Creating a Hyper Protect Virtual Server for VPC” on page 107 for a setup example.

Also see [Creating a Hyper Protect Virtual Server for VPC instance](#).

A virtual private cloud (VPC) can be created in IBM Cloud by following the instructions in [Creating and configuring a VPC](#).

- ▶ To create the HPVS in a KVM host LPAR, follow the instructions in [Example of bringing up IBM Hyper Protect Virtual Servers on a KVM host by using the virsh utility](#). You must provide the combined env and workload section obtained from the previous step, and use them as the content of the user-data file for deployment.

To configure your HPSB instance see [Bringing up the Hyper Protect Secure Build on the KVM LPAR](#).

4.2.6 Configure the HPSB client with the HPVS IP address

Use the following steps to configure the HPSB client with the HPVS IP address in `/etc/hosts`:

1. Ensure the floating IP address of the HPSB server is mapped to the hostname in the `/etc/hosts` file, which is given during the certificate creation. See Example 4-12.

Example 4-12 Verify the floating IP address

```
$ cat sbs-config.json | grep HOSTNAME
"HOSTNAME": "test.xyz.com",
```

```
$ cat /etc/hosts | grep test.xyz.com
150.239.221.33 test.xyz.com
```

2. Check that the HPSB client and HPSB server are able to communicate. See Example 4-13.

Example 4-13 Verify communications

```
$ ./build.py status --env sbs-config.json
INFO:__main__:status: response={
  "status": ""
}
```

3. Update the `sbs-config.json` with the GitHub repo where the source code and dockerfile is present. Include the registry details for where the built container image needs to be pushed. See Example 4-14.

Example 4-14 HPSB configuration details

```
$ cat sbs-config.json
{
  "HOSTNAME": "test.xyz.com",
  "CICD_PORT": "443",
  "IMAGE_TAG": "",
  "CONTAINER_NAME": "HPSBContainer",
  "GITHUB_KEY_FILE": "~/ssh/id_rsa",
  "GITHUB_URL": "https://github.com/ibm-hyper-protect/paynow-website",
  "GITHUB_BRANCH": "main",
  "DOCKER_REPO": "devuser/samplepaymentsystem",
  "DOCKER_USER": "devuser",
  "DOCKER_PASSWORD": "*****",
  "IMAGE_TAG_PREFIX": "v3",
  "DOCKER_CONTENT_TRUST_BASE": "False",
  "DOCKER_CONTENT_TRUST_BASE_SERVER": "",
  "DOCKER_RO_USER": "devuser",
  "DOCKER_RO_PASSWORD": "*****",
  "RUNTIME_TYPE": "vpc"
}
```

4. Initialize the HPSB server with configuration. See Example 4-15.

Example 4-15 Initialize the HPSB server

```
$ ./build.py init --env sbs-config.json
INFO:__main__:init: response={
  "status": "OK"
```

```
}
```

5. Initiate the build for the HPSB server. See Example 4-16.

Example 4-16 Build for the HPSB server

```
$ ./build.py build --env sbs-config.json
INFO:__main__:build: response={
  "status": "OK: async build started"
}
```

6. Check the progress status of the HPSB server build. See Example 4-17.

Example 4-17 Check the status of the HPSB server build

```
$ ./build.py status --env sbs-config.json
INFO:__main__:status: response={
  "build_image_tag": "1.3.0.11",
  "build_name": "",
  "image_tag": "",
  "manifest_key_gen": "",
  "manifest_public_key": "",
  "status": "github cloned"
}
```

7. To check the final status of the build, run the command shown in Example 4-18.

Example 4-18 Check the status after the HPSB build

```
$ ./build.py status --env sbs-config.json
INFO:__main__:status: response={
  "build_image_tag": "1.3.0.11",
  "build_name":
"docker.io.devuser.samplepaymentsystem.v3-f29b1ab.2023-07-25_09-01-40.401144",
  "image_tag": "v3-f29b1ab",
  "manifest_key_gen": "soft_crypto",
  "manifest_public_key":
"manifest.docker.io.devuser.samplepaymentsystem.v3-f29b1ab.2023-07-25_09-01-40.401
144-public.pem",
  "status": "success"
}
```

8. Run the command in Example 4-19 to see the build logs.

Example 4-19 HPSB logs

```
$ ./build.py log --log build --env sbs-config.json
INFO:__main__:2023-07-25 08:59:36,446 build_task
INFO    starting a build
...
```

The full build log can be found in the Example C-1 on page 116.

9. Get digest for the built image. See Example 4-20.

Example 4-20 Digest for the build image

```
$ ./build.py get-digest --env sbs-config.json
```

Digest value of the built image:
docker.io/devuser/samplepaymentsystem@sha256:d10e26e72a2f83a3fdf8a6a79da5b88f1b6747ce0af9309749afc55295973bd8

10. To get the signed public key, use the command in Example 4-21.

Example 4-21 Signed public key

```
$ ./build.py get-signed-image-publickey --env sbs-config.json
INFO: __main__: Downloaded signed image public key to file
docker.io-devuser-samplepaymentsystem-public.key
```

```
$ cat docker.io-devuser-samplepaymentsystem-public.key
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSOtLS0tCk1JSUJsVENDQVR1Z0F3SUJBZ01RV2RmQStQcm9tVjRwd2
RVS01KYnFqREFLQmdncWhrak9QUVFEQWpBeE1TOHcKTFZRFZRUURFeVprYjJ0c1pYSXVhVzh2WVdKb2FY
SmhiV3N2YzJGdGNHeGxjROyY1Y1dWdWRITjVjM1JsY1RBZQpGdzB5TXpBM01qVXdPVEF4TXpoYUZ3MHPNek
EzTWpJd09UQUXhNemhhTURFeEx6QXRRCz05WQkFNVEptUnZZMnRsCmNpNXBieTlowW1ocGNtRnRheTl6WVcx
d2JHVndZWGx0W1c1MGMzbHpkR1Z0TUZrd0V3WUhlb1pJemowQ0FRWUkKS29aSXpqMERBUWNEUWdBRWp1Mm
FPYTVPYUE4UHJFUThHNTgybTZxWm1JaEFvYWo2bDZaaThsaDFwM01VbTBLNAP4TVBJcytZNHA2TzVzeE9t
RFpFaW9SbmJ0eU1NRDJrN05zNXVLYU0xTURNdORnWURWUjBQQVFIL0JBUURBZ1dnCk1CTUdBMVVsK1FRTU
1Bb0dDQ3NHQVFRk1J3TURNQXdhQTFVZEV3RUIvd1FDTUFBd0NnWU1Lb1pJemowRUF3SUQKU0FBd1JRSWdZ
UnZ0bW1nRkg1dTBSSn1ENUhxcDFtcW9zM2k5cVczbWxRWV1JN2oyZXJVQ01RRFZGNjhXbXo0RQp1UXExeV
JvaHIwZXPzck520Eh4eXQvUS9CMB1QUY1Nz1RPTOKLS0tLS1FTkQgQ0VSVE1GSUNBVEUtLS0tLQo=>
```

11. Get the manifest files, which consist of files needed for auditor to check on the build. See Example 4-22.

Example 4-22 Manifest files

```
$ ./build.py get-manifest --env sbs-config.json
INFO: __main__: get-manifest manifest_name:
manifest.docker.io.devuser.samplepaymentsystem.v3-f29b1ab.2023-07-25_09-01-40.401144
```

12. Use the command in Example 4-23 to verify the manifest files.

Example 4-23 Verifying the manifest files

```
$ ./build.py get-manifest --env sbs-config.json --verify-manifest
INFO: __main__: get-manifest manifest_name:
manifest.docker.io.devuser.samplepaymentsystem.v3-f29b1ab.2023-07-25_09-01-40.401144
INFO: __main__: verify_manifest:
manifest_name=manifest.docker.io.devuser.samplepaymentsystem.v3-f29b1ab.2023-07-25_09-01-40.401144 test=0
INFO: __main__: verify=OK
```

13. Get the state image, which can be used later to bring up the HPSB instance with the same set of GitHub and registry configuration. It is important to safely store this state image. This image consists of keys that are needed to push the image into the registry.

Example 4-24 Get state image

```
$ ./build.py get-state-image --env sbs-config.json
INFO: __main__: state:name:
docker.io.devuser.samplepaymentsystem.v3-f29b1ab.2023-07-25_09-01-40.401144
```

4.3 Zero knowledge proofs: TLS server certificates and wrapped secrets

Within the context of the zero knowledge architecture, the communication between different components requires authentication of the components against each other. These authentications depend on secrets that are generated by the components and depend on certificates that are used to exchange PINs and secret keys.

This section explains how secrets can be generated safely and securely, how secret keys and certificates can be injected into components, and how some common communication components, such as reverse proxies, can be configured and hardened against attacks.

4.3.1 Passing secrets into a secure HPVS

To establish verified and trusted connections to other parts of the IT landscape, it is required to pass information to HPVS instances. The mechanism used to pass the information must be checked for integrity and confidentiality.

With such a unidirectional communication channel, it is possible to distribute shared information and even shared secrets. The following list includes examples of shared secrets:

- ▶ CA and server certificates to establish connections to only trusted servers or peers
- ▶ Secret keys used for identification of an instance
- ▶ Credentials used to log in to other services

In the HPVS instance this unidirectional communication channel is provided by the contract. For more information, see [IBM Cloud: About the contract](#).

4.3.2 Certificate benefits

Certificates are generally used to support the distribution of public keys. Embedding a public key within a certificate does provide three main benefits:

1. The certificate is signed by a certificate authority called issuer, relaying trust to the contained public key. This allows building a verifiable key hierarchy of which only the public root key must be distributed.
2. The validity of the certificate can be defined. This is done by definition of a validity time frame and the option to embed revocation information that can be checked additionally.
3. Each certificate can restrict the intended usage of the public key.

For a complete definition, see [Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List \(CRL\) Profile](#).

Within the context of securing a container running in HPVS, certificates are useful in many ways. Certificates are used to simplify the verification of keys that are used to encrypt and sign the contract and to allow attestation of the started virtual server instances.

Certificates can be used by the Workload Provider and the Workload Deployer to introduce trusted keys within the contract. These can also be used to extend the trust into keys introduced within the workload. This is either being done by introducing an owned certificate authority (CA) or preparing a new CA for this purpose. It is also possible to generate and use a key within HPCS for the CA.

For example guidance on manual generation of self-signed certificates, see [Generate root CA key and certificate](#).

4.3.3 Importing server certificate from contract

A common function of an HPVS is to provide a communication end point in the form of a secure web server. Apart from the recommended hardening of the web server configuration, configure the server to use TLS / HTTPS. For more information, see [SSL/TLS Best Practices for 2023](#).

The simplest way to achieve this as the Workload Provider is to add the required files to the workload directly. This has the downside that all instances of the workload would be using the same secret. A better option is to let the Workload Deployer supply the secrets and dependent information for each instance. In the current example, the Workload Deployer pre-generates a key and a certificate for the server to use. Certificate and key are then set as environment variables within the compose or play file.

It is a best practice to encode keys and certificates in Base64 when handing them through the contract. How this is introduced within the container differs from the composition method that is used to orchestrate the containers.

Secret environment definition in compose

When you use the compose option to define the workload, the environment attribute can be defined to declare variables. For more information, see [docker docs: Use the environment attribute](#).

The value of the variables can then be deferred to the env section. See Example 4-25.

Example 4-25 Defer variables to the env section

```
# set by the workload provider in compose
environment:
  HTTPS_CERT: "${HTTPS_CERT_VALUE}"
  HTTPS_KEY: "${HTTPS_KEY_VALUE}"
```

Secret environment definition in play with templates

When using the play option to define the workload, the env attribute can be defined to declare variables. For more information, see [kubernetes: Define an environment variable for a container](#).

The value of the variables can then be deferred to the env section using templates. See Example 4-26.

Example 4-26 Defer variables to the env section using templates

```
workload: |
  type: workload
  play:
    templates:
      - apiVersion: v1
    ...
  env:
    - name: HTTPS_CERT
      value: {{ .Env.HTTPS_CERT_VALUE }}
    - name: HTTPS_KEY
```

```
value: {{ .Env.HTTPS_KEY_VALUE }}
```

Secret environment definition using config map

In simpler workload setups, it might be feasible to not use a template. Instead, you can set all variables from the env section as environment variables of a container by using the special content map file, `contract.config.map`. In the `contract.config.map` file, the Workload Deployer adds secrets in the env section of the contract. See Example 4-27.

Example 4-27 Variables settings in the env section

```
# set by deployer in env section
env: |
  type: env
  HTTPS_CERT_VALUE: LS0tLS1CRUdJTiBDRVJUS...0tLS0tCg==
  HTTPS_KEY_VALUE: HVkp4BMwFunWgmay...0BwM0w==
```

Tip: When using this method, use only an encrypted contract to protect the secret key. This can be encoded with `base64 -w 0 server.crt`. The application can then read and apply these values on startup.

4.3.4 Random number generation

An important part of each application is the generation of random numbers. Random numbers are needed in cryptographic calculations, simulation, and other usages. IBM z/Architecture®, also known as s390x, of the IBM LinuxONE and IBM Z platforms provides special support for these functions.

Generating a new random key

The HPCS offering provides multiple improvements to generate strong keys over other environments. This extension is provided by the IBM LinuxONE and IBM Z cryptographic coprocessors (Crypto Express adapter in CCA coprocessor mode or in EP11 mode).

Linux users might already be familiar with the default kernel interface to the random number generator. `/dev/urandom` is used for non-blocking random numbers, and `/dev/random` is used for entropy based random numbers.

When you run Linux on Z, the additional pseudo random number generator, (PRNG) device, `/dev/prandom`, and true random number generator (TRNG) device, `/dev/trng`, are provided for enhanced random number generation. See Table 4-1.

Table 4-1 Pseudo random and random number generator devices

	Linux	Linux on Z
Pseudo Random	<code>/dev/urandom</code>	<code>/dev/prandom</code>
Random	<code>/dev/random</code>	<code>/dev/trng</code>

For more information, see [Device Drivers, Features, and Commands](#).

Generating a new random key using HPCS

In addition to the random sources built into every IBM LinuxONE and IBM Z, the cryptographic accelerator hardware or hardware security module (HSM) can be used to

generate keys within the tamper resistant HSM alone without depending on or being influenced by the VM.

IBM provides a `golang_grep11` library to interface with the HSM provided by the HPCS. For more information, see [IBM-Cloud/hpcs-grep11-go](#).

The library does include coding examples to perform simple key operations, such as generate, encrypt, decrypt, sign, verify, and more.

CA backed by HPCS

To further improve the protection for keys used by a workload, it is possible to store and use the key within an HSM. This section explains the usage of the HSM provided by the HPCS service to protect a CA root key and to use it for the signing of certificates.

To be able to use the HPCS service, the access information must be passed into the virtual server instance. This can be done through the workload section of the contract by the Workload Provider or through the `env` section of the contract by the Workload Deployer.

By using the `play` section within the contract, the HPCS access information is made available to the instance. See Example 4-28.

Example 4-28 HPCS access information

```
workload: |
  type: workload
  play:
    templates:
      - apiVersion: v1
        kind: Pod
        metadata:
          name: samplepaymentsystem
        spec:
          containers:
            - name: backend
              image:
icr.io/ibm/samplepaymentsystem@sha256:aa921f4009b33b926aeae931fef2b0536514e7a62ae0
13cee6c345b1ac7f11ba
...
    env:
      - name: HPCS_ADDRESS
        value: {{ .Env.HPCS_ADDRESS }}
      - name: HPCS_KEY
        value: {{ .Env.HPCS_KEY }}
...
env:
...
  env:
    HPCS_ADDRESS: ep11.us-south.hs-crypto.cloud.ibm.com:8082
    HPCS_KEY: SFBDU19BUE1fSOVZCg==
```

These environment variables `HPCS_ADDRESS` and `HPCS_KEY` can now be used within the workload. See Example 4-29.

Example 4-29 Environment variables

```
// Read the HPCS configuration from the environment
var (
```

```

Address      = os.Getenv("HPCS_ADDRESS")
APIKey       = os.Getenv("HPCS_KEY")
IAMEndpoint  = "https://iam.cloud.ibm.com"
)

```

With the newly defined HPCS_ADDRESS and HPCS_KEY variables, you can generate a new RSA keypair to be used within the certificate. See Example 4-30.

Example 4-30 RSA keypair

```

var callOpts = []grpc.DialOption{
    grpc.WithTransportCredentials(credentials.NewTLS(&tls.Config{MinVersion:
        tls.VersionTLS12})),
    grpc.WithPerRPCCredentials(&util.IAMPerRPCCredentials{
        APIKey: APIKey,
        Endpoint: IAMEndpoint,
    }),
}

func main() {
    context := context.Background()
    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)
    _, _, _ = GenerateKeyPair(context, cryptoClient)
}

// generate a new 4096 bit RSA key pair
func GenerateKeyPair(context context.Context, cryptoClient pb.CryptoClient)
([]byte, []byte, error) {
    generateKeyPairResponse, err := cryptoClient.GenerateKeyPair(context,
    generateRSA4096KeyPairRequest())
    if err != nil {
        panic(fmt.Errorf("generate RSA key pair error: %w", err))
    }

    return generateKeyPairResponse.PubKeyBytes, generateKeyPairResponse.PrivKeyBytes,
    nil
}

func generateRSA4096KeyPairRequest() *pb.GenerateKeyPairRequest {
    return &pb.GenerateKeyPairRequest{
        Mech: &pb.Mechanism{
            Mechanism: ep11.CKM_RSA_PKCS_KEY_PAIR_GEN,
        },
        PubKeyTemplate: util.AttributeMap(ep11.EP11Attributes{
            ep11.CKA_ENCRYPT: true,
            ep11.CKA_VERIFY: true,
            ep11.CKA_MODULUS_BITS: 4096,
            ep11.CKA_PUBLIC_EXPONENT: 65537,
        }),
        PrivKeyTemplate: util.AttributeMap(ep11.EP11Attributes{

```

```
ep11.CKA_PRIVATE:    true,
ep11.CKA_SENSITIVE:  true,
ep11.CKA_VERIFY:     false,
ep11.CKA_ENCRYPT:     false,
ep11.CKA_DECRYPT:     false,
ep11.CKA_SIGN:       true,
ep11.CKA_EXTRACTABLE: false,
}),
}
}
```

The resulting key pair can be used to create a new certificate. Although it is possible to create a CA this way if that is required for the use case, it is recommended to not use a CA with a key in the clear like this. This method should rather be used to generate keys that are required to be in the clear.

After this generation, the key is open in the clear and available for use as a TLS certificate within a reverse proxy or web server.

In our `SamplePaymentSystem` example, we use the generated RSA key as a key for a certificate.

Note: Note: For more examples on how to use the `grep11` go library, see [IBM-Cloud/hpcs-grep11-go/blob/master/examples/server_test.go](https://github.com/IBM-Cloud/hpcs-grep11-go/blob/master/examples/server_test.go).

4.3.5 Reverse proxy

To adhere to the zero trust architecture, all communication between different services must be authenticated and encrypted.

The communication between the reverse proxy and internal services might or might not be encrypted, based on security needs. Internal communication does not need to be encrypted.

In case any inter-service communication leaves a single pod,² the communication should be authenticated and encrypted because pods might be moved into other scopes.

A reverse proxy within the pod can be used for various purposes, such as SSL/TLS termination, caching, content serving, load balancing, and authentication.

Existing proxies SSL certificates can be set up between the reverse proxy and backend or other services to check a JSON Web Token (JWT) authentication. See Example 4-31.

Example 4-31 SSL certificate checks

```
containers:
  - name: frontend
    image:
      docker.io/library/nginx@sha256:67f9a4f10d147a6e04629340e6493c9703300ca23a2f7f3aa56
      fe615d75d31ca
    ports:
      - containerPort: 80
        hostPort: 80
      - containerPort: 443
        hostPort: 443
```

² A pod encapsulates one or more applications or containers.

```
volumeMounts:
  - mountPath: /etc/nginx
    name: contract-nginx
    readOnly: true
  - mountPath: /www
    name: contract-www
    readOnly: true
```

The nginx server can be configured. See Example 4-32.

Example 4-32 nginx server configuration

```
error_log /dev/stderr info;

events {}
http {
  access_log /dev/stdout;
  server {
    # reverse proxy with tls termination
    listen 443 ssl;
    ...
    location ~ .ico {
      # serve static content
      root /www;
    }
    location / {
      # serve dynamic content from backend
      index index.html;
      proxy_pass https://localhost:8443/;
      proxy_ssl_trusted_certificate /etc/nginx/backend.crt;
      proxy_ssl_session_reuse on;
      proxy_ssl_verify on;
    }
  }
}
```

No logs are written into the containers file system. Instead, all logs are written to stdout to be surfaced on the container host. On the host, the logs are further integrated into the journal and forwarded to the configured remote log server.

When possible, container volumes should be read-only to prevent any modification and potential compromise of other containers that use the same volume mounts for configuration.

4.3.6 Basic web server (nginx) hardening

The use of a reverse proxy like nginx allows the protection of secondary services within or outside of the deployed pod. To maximize this protection, the configuration should be hardened to allow only the minimum required for the function of the services. This includes the used protocols, encryption, timeouts, and enforcement of certain headers. See Example 4-33 on page 85.

For more information about nginx security configuration, see [Security Controls](#).

For more information, see [SSL/TLS Best Practices for 2023](#).

```
http {
...
    server {
        # reverse proxy with tls termination
        listen 443 ssl;
        ssl_certificate sample.test.crt;
        ssl_certificate_key sample.test.key;
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_ciphers
ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-S
HA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20
-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384;
        ssl_prefer_server_ciphers on;
        ssl_session_cache shared:SSL:50m;
        ssl_session_tickets off;

        # set default headers
        add_header X-Frame-Options SAMEORIGIN;
        add_header X-Content-Type-Options nosniff;
        add_header X-XSS-Protection "1; mode=block";
        add_header Content-Security-Policy "default-src 'self'; script-src 'self'
'unsafe-inline' 'unsafe-eval' https://code.jquery.com/, img-src 'self'; style-src
'self' 'unsafe-inline'; font-src 'self'; object-src 'none'";
        add_header Strict-Transport-Security "max-age=31536000; includeSubdomains;
preload";
    }
...
}
```

4.3.7 Offloading NGINX TLS to HPCS

With the nginx server running as a reverse proxy within the application pod, it is also possible to further offload the TLS handling completely to HPCS. For more information, see [Use IBM Cloud Hyper Protect Crypto Services to offload NGINX TLS](#).

4.4 Trust in-depth based on boot flow attestation

After successful deployment of a service, the Workload Deployer should verify that the service has been deployed as specified. This includes the running containers as specified within the contract as well as the underlying HPCR image.

During deployment of the virtual server instance in the cloud, an attestation record is created. It contains hashes of the following items:

- ▶ The original base image
- ▶ The root partition at the moment of the first boot
- ▶ The root partition at build time
- ▶ The cloud initialization options

The attestation record is signed by the attestation key.

As an extra protection layer, you can provide a public key within the contract as `attestationPublicKey` (see Example 4-34 on page 86). When the public key is provided, the

attestation record (se-checksums.txt) is encrypted and generated (se-checksums.txt.enc). Only one of the files is present depending on if a public key was provided or not.

Example 4-34 Public key provided with the attestation record

```
env: |
...
attestationPublicKey: |-
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEA6v4drcQa2Rm6+Gdp3+xG
InhWhTG1TVN8kzCAZmZfwiYxc6REOTVUAqEwcXxo01A4Qp8rGRpR5RFsdGVry95
yachZk4ut0/Od6BHDEG306hQVYMZYQmj8GX7IFHbGv/eqdvZJIWm0m30c+NLJRX
pU9XdAY31J4tzxRQwyM1wsDt/3u7TWGi5P7cZMfjYeoP1CyWJujvCTWGNH8550cU
BoF3833NcJSKpN3L0o/v71IMp3H4Frq50Juek/QQIkhPN9muN1yUBa0ujp5WDWac
1Hwwqd2dNRx6Z0X/X4nLI30Y8rrScFcU+TnZKHaVGQ2eFTMdGDLJehTUGOV0euvb
+70QRCg+VP2Ayo1sU53Q20Dr+Aer49qjdxN+8Pjw94AVdtC5dvTPmG08p9bB9IHx
3ycMH2RngxixsBmtZ+iu4vX0zw9zIynQhsUQoKE9prJtLI7011+JfpMvaJtkYaUa
zb2wnsj740CwhucP8q2wtqyKbaz8xj8wXXRGNCvkerZYa0YzcRQb9t+IPb5iDD30
eq7cA+uM5n3CXB7cpD1uq1SPIpWmUztCbvcWmI0HPcpP1Pa+98t3izAHpk1/90Td
qcZSox81iEMg4RUzxLVpVM7FR6+CN6nVSck0P/8mhf1A1dzoGeLN2UGR1TfpDmDU
1cRQ95+m1xdGk7sSwZpvRAOCAwEAAQ==
-----END PUBLIC KEY-----
```

The hash of this public key is added to the attestation record to ensure that the record can be viewed only by the compliance authority, and the expected authority can be easily identified through that hash.

Within the env volume section of the compose contract, add /var/hyperprotect. See Example 4-35.

Example 4-35 Volume section

```
volumes:
- "/var/hyperprotect/:/var/hyperprotect/:ro"
```

The attestation record is signed by the attestation signing key, and the signature is a provided separate file se-signature.bin.

The attestation signing key can be confirmed by the IBM intermediate certificate. The IBM intermediate certificate is signed by DigiCert, which is proven by the root certificate of DigiCert, thus completing the chain of trust.

The encryption and attestation certificates are signed by the IBM intermediate certificate and this has been signed by the IBM DigiCert intermediate cert (which in turn is signed by DigiCert Trusted Root G4). For more information about the certificates, see DigiCert Trusted Root Authority Certificates.

To validate the attestation record and hashes, obtain the attestation record se-checksums.txt and the signature file se-signature.bin from your HPVS instance. To do so, you can implement your container to provide the attestation record and the signature file. The attestation record and the signature file are made available to your container in the /var/hyperprotect directory.

Sample node.js code snippet implementation to get attestation .zip files. See Example 4-36 on page 87.

Example 4-36 Get attestation .zip files

```
app.get('/api/v1/attestation', function(req, res) {
  console.log('GET ' + req.path);
  const fileName = 'attestation.zip';
  const fileType = 'application/zip';
  var zip = new admzip();
  try {
    zip.addLocalFile("/var/hyperprotect/se-checksums.txt.enc");
  }
  catch (e) {
    zip.addLocalFile("/var/hyperprotect/se-checksums.txt");
  }
  zip.addLocalFile("/var/hyperprotect/se-signature.bin");
  var zipFileContents = zip.toBuffer();
  res.writeHead(200, {
    'Content-Disposition': `attachment; filename="${fileName}"`,
    'Content-Type': fileType,
  })
  res.end(zipFileContents);
});

app.get('/api/v1/attestationdocument', function(req, res) {
  console.log('GET ' + req.path);
  res.type('txt').sendFile('/var/hyperprotect/se-checksums.txt');
});
```

For the sample code that we used in our environment, see [IBM/hyperprotect/redbook-samples/sg248555](#).

You can decrypt the encrypted attestation by using the provided script For more information for HPVS for VPC, see [IBM Cloud: Decrypting the attestation document](#). For more information for IBM LinuxONE or IBM Z, see [IBM Hyper Protect Virtual Servers: Decrypting the attestation document](#).

4.5 Data storage

HPVS does not support persistent storage over reboot on the root partition. Therefore, the root volume is deployed in the same state on each boot of the HPVS. This means that all data must be stored on user data volumes that are attached to the HPVS.

To provide effective data at rest protection, all encryption must happen within the trusted execution environment (TEE) before any data blocks or objects are written back to external storage.

The two main storage classes are block storage and object storage.

4.5.1 Encrypting block storage

Linux Unified Key Setup (LUKS) is the standard for Linux volume encryption. It enables secure management of multiple user passwords. LUKS stores all necessary setup information in the partition header, which enables users to transport or migrate data seamlessly.

The user data volumes of a HPVS are encrypted using LUKS by default. To open and mount a LUKS protected partition, a password is required. This password is derived from the password seeds that are supplied within the workload and env sections of the contract.

4.5.2 Encryption state

HPVS periodically logs the encryption state of the attached block storage devices to the internal journal. Because the journal is forwarded to a remote log server, this output can be used to audit the encryption state of the data at rest.

In case the default cadence is not sufficient, the workload can request additional audit output by logging the special token HPL13000I. See Example 4-37.

Example 4-37 Request additional audit output

```
workload: |
...
  containers:
    - name: verify-disk-encryption-trigger
      image: docker.io/library/ubuntu:22.04
      command: echo
  args:
    - HPL13000I
...
```

Logging the token to stdout of any of the containers triggers the verify-disk-encryption service, which adds information about the encryption state of the attached block devices to the log. See Example 4-38.

Example 4-38 Verify disk (volume) encryption

```
common[991]: HPL13000I
...
verify-disk-encryption: HPL13003I: Checked for mount point /, LUKS encryption with
1 key slot found
verify-disk-encryption: HPL13001I: Boot volume and all the mounted data volumes
are encrypted
```

A complete example of the log is displayed in the Example C-2 on page 118.

For more information on disk (volume) encryption verification, see [IBM Cloud: Verifying disk encryption status](#).

HPCS user data volume encryption

An HPVS instance can also be configured to use an additional HPCS seed for the LUKS password. Because this secret is tied to the HPCS instance, it can also be remotely controlled.

When this option is used, the key state on the HPCS instance is monitored to lock the partition when the HPCS key is disabled or removed.

The service can be configured within the env part of the contract, which allows the use of a different HPCS instance in each deployed HPVS instance and volume. See Example 4-39 on page 89.

Example 4-39 HPVS volume encryption

```
env: |
  type: env
  volumes:
    payment-data-volume:
      seed: "secret-env-seed1"
      kms:
        - apiKey: "${var.ibmcloud_api_key}"
          crn: "${var.hpcs_crn}"
          type: "private"
      kmsTimeout: 10
      signingKey: "xxxxxxxxx"
  ...
```

When it is possible, give precedence to the private API endpoint before the public endpoint. When configured this way, the communication does not need to leave the VPC network. If only the private API endpoints of the HPCS service are used, it is possible to completely disable the public API endpoint on the service. This further reduces the attack surface.

The additional protection by an HPCS secret can be added to an existing volume. To do so, the workload and env seeds still must match to allow access to the volume. This is reflected in the HPVS log with an entry that is similar to the following message:

```
hpcr-disk-mount[699]: HPL07011I: Migration started from non-BYOK to BYOK for
volume [/dev/vdd]
```

The KMS connection can be changed to another KMS but cannot be removed as this is indistinguishable from a downgrade attack.

In case the data needs to be exported, this must be done on the workload level.

If the KMS connection is lost or the KMS key is locked at any time, the HPVS restarts. After the restart, the HPVS starts polling for an enabled key for a limited time to re-open the volume and re-start the workload. This can be retried anytime by restarting the HPVS.

The required crn can be easily obtained by using the `ibmcloud` CLI command as shown in Example 4-40.

Example 4-40 Retrieve service instance

```
$ ibmcloud resource service-instance "sample-hpcs"
Retrieving service instance sample-hpcs in all resource groups under account XXX's
Account as xxx@ibm.com...
OK

Name:                sample-hpcs
ID:
crn:v1:bluemix:public:hs-crypto:us-south:a/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:xxxxxx
xx-xxxx-xxxx-xxxx-xxxxxxxxxxxx:::
GUID:                xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
Location:            us-south
Service Name:        hs-crypto
Service Plan Name:   standard
Resource Group Name: xxx
State:               active
Type:                service_instance
Sub Type:            kms
```

```
Locked:                false
Created at:            2023-07-30T14:16:13Z
Created by:            IBMid-xxxxxxxxxx
Updated at:            2023-08-17T08:30:13Z
Last Operation:        ...
```

When you use Terraform to manage HPCS, the information can also be acquired from the Terraform state. See Example 4-41.

Example 4-41 Manage Terraform information

```
$ terraform show
# ibm_hpcs.hpcs:
resource "ibm_hpcs" "hpcs" {
  created_at      = "2023-07-30T14:16:13Z"
  created_by      = "IBMid-xxxxxxxxxx"
  crn              =
"crn:v1:bluemix:public:hs-crypto:us-south:a/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:xxxx
xxx-xxx-xxx-xxx-xxxxxxxxxxxx:"
  ...
  extensions      = {
    "allowed_network" = "public-and-private"
    "endpoints.private" =
"https://api.private.us-south.hs-crypto.cloud.ibm.com:8080"
    "endpoints.privateGrep11" =
"https://ep11.private.us-south.hs-crypto.cloud.ibm.com:8080"
    "endpoints.public" =
"https://api.us-south.hs-crypto.cloud.ibm.com:8080"
    "endpoints.publicGrep11" =
"https://ep11.us-south.hs-crypto.cloud.ibm.com:8080"
  }
  ...
  id              =
"crn:v1:bluemix:public:hs-crypto:us-south:a/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:xxxx
xxx-xxx-xxx-xxx-xxxxxxxxxxxx:"
  location        = "us-south"
  name            = "redbook-sample"
  plan            = "standard"
  service         = "hs-crypto"
  service_endpoints = "public-and-private"
  state           = "active"
  status          = "active"
  update_at       = "2023-08-17T08:30:13Z"
  ...
}
```

The results of the output is the crn of the service:

```
"crn:v1:bluemix:public:hs-crypto:us-south:a/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:xxxx
xxx-xxx-xxx-xxx-xxxxxxxxxxxx:".
```

To secure a disk (volume) with this, the key id must be specified in the end of the crn. The output is similar to the following example:

```
"crn:v1:bluemix:public:hs-crypto:us-south:a/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:xxxx
xxx-xxx-xxx-xxx-xxxxxxxxxxxx:key:xxxxxxxx-xxx-xxx-xxxxxxxxxxxxxxxx".
```

Note: Create a separate API key to provide within the contract instead of using the same API key within the contract and to deploy the contract. Using different keys does provide the benefit that the key can be restricted and follow its own lifecycle.

4.5.3 Upgrade, backup, and disaster recovery

Although all volumes can be backed up by taking a snapshot, this is not necessary for HPVS root volumes. Because the root volume is cleanly prepared on each boot, it will always be in good shape after a reboot.

All valuable application data must be stored on data volumes. The data volumes can be snapshot. Snapshots can be restored into new data volumes that can be attached to the same or another HPVS instance. When attempting this, the following criteria must be met:

- ▶ The workload volume seed within the contract must be the same
- ▶ The env volume seed within the contract must be the same
- ▶ The HPCS instance attached to the data volume must provide the same key

It is possible to change the workload containers and the provided environment data between snapshot and restore. This means, if it is needed, that it is possible to move data volumes to another region and update the workload in one step.

For details about snapshots of data volumes, refer to the following documentation:

- ▶ For creating snapshots, see [Creating snapshots](#).
- ▶ For restoring a volume, see [Restoring a volume from a snapshot](#).
- ▶ For managing a snapshot, see [Managing snapshots](#).

When creating snapshots while an HPVS is running, it is possible that the snapshot does contain inconsistent data that cannot be read by the application. Therefore, it is recommended that the HPVS is rebooted or restarted before a snapshot is requested. The reboot or restart flushes all data from memory to disks (volumes). Alternatively, the workload containers might provide modes that allow for safe snapshots. This is highly dependent on the workload and cannot be controlled by the IBM infrastructure or container runtime image.

4.5.4 High Availability

HPVS does not provide any services in support of a high availability (HA) environment. The containers run as defined within the contract. The Workload Deployer persona of the application is responsible for providing enough instances and a configuration for the overall service to perform properly.

For more dynamic applications, including scaling and workload migration, Kubernetes³ is the industry standard.

Deployment of pods that are protected by Secure Execution for Linux is allowed within the Cloud Native Community: Confidential Container Project. For more information, see [confidential-containers](#).

4.6 Securing cloud native services

Kubernetes is an open source container orchestration platform that is extensively used in the industry for managing, scaling, and automating cloud native services.

Kubernetes has four key underlying infrastructure layers to protect, which include the public, private, or hybrid cloud infrastructure, clusters, containers, and code. From a security perspective, each layer relies on the next layer. For example, the code layer benefits from strong security at the public, private or hybrid cloud infrastructure, cluster, and container layers. You cannot safeguard against poor security standards at the other layers by addressing security at only the code layer.

Depending on the attack surface of the service, focusing on specific aspects of security is important. For instance, if a critical service, *Service A*, is in a chain of other resources with a separate service, *Service B*, which is exposed to a resource exhaustion or vulnerable API attack, then the risk of compromising Service A is high.

If services are run in the same cluster with a shared set of resources and common base where a privilege escalation can be achieved through one service (Service B), then potentially, Service B can be used to attack or exploit Service

There are three areas of concern for securing Kubernetes:

1. Cluster components like kubelet⁴ that are configurable
2. Applications that run in the cluster
3. Appropriate level of protection for APIs

4.6.1 Confidential cluster

With the ability to protect virtual machines through confidential computing, it is also possible to create confidential clusters. For more information, see [Red Hat OpenShift Container Platform for IBM Z and LinuxONE 4.13](#). Virtual machines, by using confidential computing, protect the control plane and worker or compute nodes from the underlying infrastructure. Although such clusters address several attack vectors, one key aspect to consider is the protection boundary. Kubernetes requires multiple APIs to operate a cluster, and with possible vulnerability of those APIs, the cluster might be infiltrated. At that point, a cyberattacker is within the protection boundary with administrator privileges at the control node. Therefore, the compute nodes and workloads that are running within the cluster are considered compromised.

A secondary aspect to consider is that clusters are managed more and more by other parties. This can be either by cloud providers offering managed Kubernetes clusters or by a service partner managing a cluster in a data center. Either way, confidential clusters need to protect a

³ If a pod (or the node it runs on) should fail, Kubernetes can automatically create a new replica of that pod to continue operations

⁴ Kubelet is an agent that runs on each node in a cluster to ensure the container or containers are running in a pod.

larger attack surface and must rely on network-level segmentation for security to protect against lateral movement⁵ if one part of the cluster is compromised.

Therefore, a more enhanced security approach that uses confidential containers is needed, particularly when considering the principles of zero trust.

4.6.2 Confidential containers

Confidential containers is an open source community working to enable cloud-native confidential computing by using TEE to protect containers and data. The community has the following goals:

- ▶ Allow cloud-native application owners to enforce application security requirements
- ▶ Transparent deployment of unmodified containers
- ▶ Support for multiple TEE and hardware platforms
- ▶ Create a trust model that separates Cloud Service Providers (CSPs) from guest applications

Thanks to the confidential containers community, there is now an operator to deploy confidential containers runtime and required configurations on a Kubernetes cluster. The confidential containers operator provides a means to deploy and manage confidential containers runtime on Kubernetes clusters. The primary resource describes runtime details such as installation type, source, and nodes to deploy.

Confidential containers work by embedding a Kubernetes pod inside a virtual machine (VM) together with an engine called the enclave software stack. There is a one-to-one mapping between a Kubernetes pod and a VM-based TEE or enclave. The container images are kept inside the enclave and can be either signed or encrypted.

The enclave software stack is measured, which means that a trusted cryptographic algorithm is used to authenticate its content. It contains the enclave agent, which is responsible for initiating attestation and for fetching the secrets from the key management service.

The supporting components for the solution are the container image registry and the relying party, which combines the attestation service and key management service.

After the VM has been started, the flow can be summarized in the following four steps:

1. A request is sent to the attestation service through the enclave agent. In response to this, the attestation service sends a cryptographic challenge for the agent to prove the workload's identity using the measurement of the enclave. If the enclave agent effectively solves this challenge, the attestation service informs the key management service that it can proceed with delivering the secrets.
2. After the workload's authorization to run is confirmed, the key management service locates the secrets that are associated with the workload and sends them to the agent. Decryption keys are among the necessary secrets for the volumes to be used.
3. The image management service inside the enclave downloads container images from the container images registry, verifies them, and decrypts them locally to encrypted storage. At that point, the container images become usable by the enclave.
4. The enclave software stack creates a pod and containers inside the virtual machine and starts running the containers. All containers within the pod are secured.

⁵ Techniques used by cyberattackers to move deeper into a network after gaining access to obtain increased privileges.

Confidential containers go beyond prior efforts of isolating the container from only the infrastructure administrator, by also isolating the containers from the Kubernetes administrator. The tenant can fully use the abstraction of a managed Kubernetes to develop-once-deploy-anywhere. The tenant can deploy data and workloads with technical assurance into a fully private and isolated enclave even if the latter is hosted and managed on third-party infrastructure.

Moreover, confidential containers reduce the set of components that interact with protected workload and data so that most components within the Kubernetes cluster can operate unchanged to address the risk of compromising data. This approach fosters speed of innovation and prioritizes data protection.

Another key component to consider in a Kubernetes based environment is the kubelet. Not only in cloud environments, this component can be delivered and managed by a 3rd party.

The advantage of confidential containers is that if a service exploitation leads to a privilege escalation to attack, vulnerable kubelet or APIs at the cluster level are addressed. See Figure 4-3.

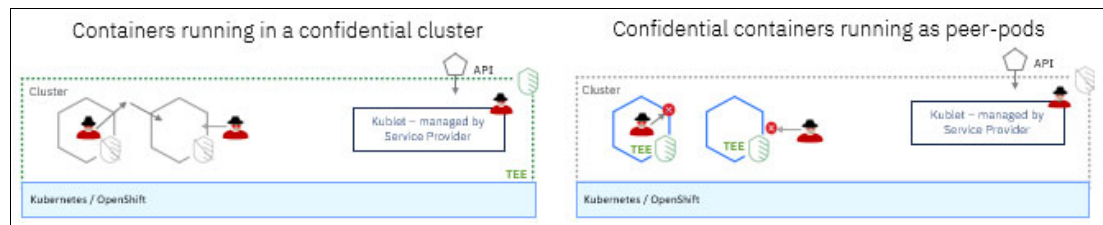


Figure 4-3 Confidential clusters versus confidential containers

Data and operation protection is enforced at the pod runtime level. This zero trust model bolsters security by strictly validating and authorizing the communication and access requests, enhancing the overall security posture of containerized environments.

4.6.3 Confidential service platform

For data security based on technical assurance through zero trust policy, enforcement is essential to have the security within the moment of deployment. This fosters workloads confidentiality with deployment, and 3rd-party attestation can be done for validation. Hence, data and workload can be protected from the very start, based on technical measures rather than trust in a third-party service.

Secure Execution offers the ability to have secrets being injected as part of the encrypted image. Such secrets can be used for zero knowledge proofs and to decrypt artifacts within the TEE.

The notion of user-controlled policy enforcement at deployment for confidential containers can be combined with Hyper Protect encrypted contracts. Therefore, the protection of data with technical assurance throughout all stages of the data lifecycle ensures data sovereignty where the complete control over the actual data lies with the cloud user and not the cloud provider.

This concept can be expanded to build a confidential service platform in which key aspects, like identify and access management (IAM), key management, attestation of workloads, policy enforcement and audit/logging records are deployed. They are deployed by using confidential containers and other workloads built on a container platform that inherits the premier zero trust value proposition. For this, the confidential container projects need to

further mature. Not only regulation needs to be defined, but also corresponding certification needs to be established by the industry and acknowledged by the regulatory parties.

4.7 Secure supply chain with SLSA

Supply chain levels for software artifacts (SLSA) is a security framework. It is a checklist of standards and controls that prevent tampering, improve integrity, and secure both packages and infrastructure. By using this framework, you can ensure that every link in the chain is as resilient as possible, as illustrated in Figure 4-4.

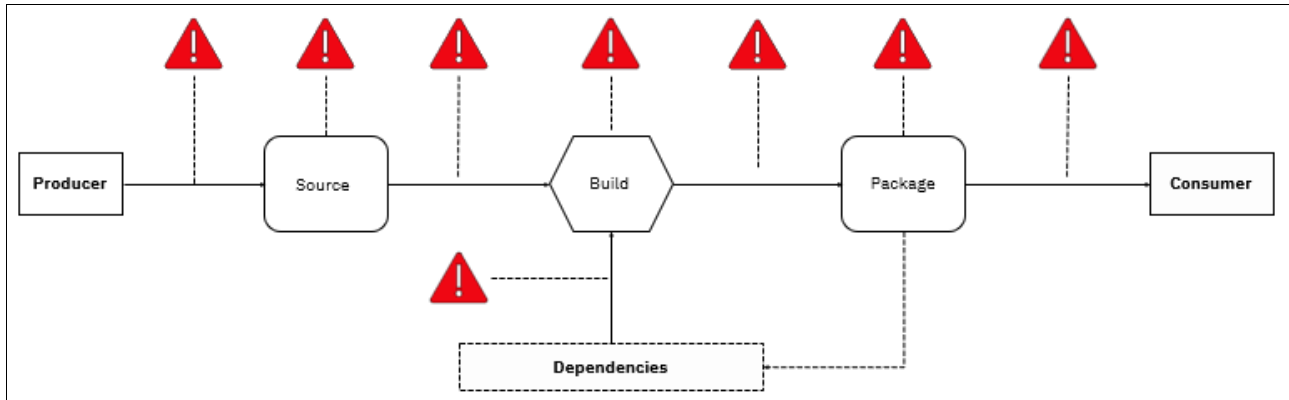


Figure 4-4 Secure supply chain - high-level overview

Industry consensus established SLSA as a set of guidelines for supply chain security that can be adopted incrementally. SLSA specifications are useful for consumers and producers of software. Consumers can use the SLSA framework to decide whether to trust a software package or not. Producers can adhere to the SLSA guidelines to add security to their supply chain.

SLSA provides the following benefits:

- ▶ A standardized terminology for discussing software supply chain security
- ▶ A method to enhance the security of your incoming supply chain by assessing the reliability of the artifacts you use
- ▶ A practical checklist to enhance the security of your own software
- ▶ A method to track your progress toward complying with the [Executive Order](#) standards in the Secure Software Development Framework (SSDF)

SLSA is designed to automate tracking of code handling, spanning from source to binary, by safeguarding against tampering, regardless of the complexity of the software supply chain. This approach can provide greater confidence that the analysis and review conducted on the source code remains valid for the binary that is consumed after the build and distribution process.

SLSA protects against the following possible security vulnerabilities:

- ▶ Modification of code by providing a “seal” to code after source control to ensure it is not modified.
- ▶ Artifacts that have been uploaded but were not built using the anticipated CI/CD platform. Artifacts are “stamped” with their build platform when created.
- ▶ Build platform threats.

SLSA is only one part of a complete approach to supply chain security. Because of this, there are several areas that sit outside of SLSA's current framework that still need to be considered along with SLSA. Areas outside of SLSA's current framework include the following examples:

- ▶ Quality of code. SLSA cannot determine if the source code that was written follows secure coding practices.
- ▶ Producer trust. SLSA's Build Track protects from tampering after or during the build. However, it cannot address organizations that purposefully create malicious software.
- ▶ Transitive trust for dependencies. An artifact's SLSA level does not depend on the level of its dependencies. At the time of writing, there is no SLSA level that can refer to both an artifact and its transitive dependencies.

4.7.1 Jenkins

Jenkins is an open source web server that is built for automation, and allows developers to build, test and deploy their software. This ensures that developers are able to implement continuous integration and continuous delivery (CI/CD). Jenkins interacts with various servers and components, meaning security is a crucial consideration. Because of the way Jenkins plug-ins and their dependencies function, providing a basic authentication mechanism is not sufficient to fully secure the complete pipeline. There are multiple options for configuration within Jenkins settings to enable, customize, or disable various security features.

The Jenkins Security Advisory process is where Jenkins will constantly review and update plug-in vulnerabilities. The Security Advisory is a list of vulnerabilities and security issues that are identified and highlighted in a report within Jenkins. The report includes a description of the vulnerability, security risks posed, severities, versions it affects, workarounds, and any possible solutions.

The eight key forms of security for Jenkins fall under three categories:

1. Basic setup:
 - Controller Isolation
 - Access Control
2. Build Behavior:
 - Access control for builds
 - Securing builds
 - Handling environment variables
3. User Interface:
 - Cross-Site Request Forgery (CSRF) Protection
 - Markup formatter
 - Rendering user content

4.7.2 Source-to-image (S2I)

The Source-to-image (S2I) framework makes it easier to write images that take application source code as an input and output a new image that runs the assembled application as output.

Using S2I for constructing consistent container images offers a primary benefit in terms of developer convenience. Build image authors need to understand two key concepts to get the best performance out of S2I - The build process and S2I scripts.

S2I creates images that are ready-to-run by injecting the source code into a specific container that prepares it for execution.

S2I can be used to control what permissions and privileges are available to the builder image because the build is launched in a single container. In parallel with platforms like Red Hat OpenShift, S2I can enable admins to tightly control what privileges developers have at build time.

4.7.3 GitHub Actions

GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows developers to automate their build, test, and deployment pipeline. Workflows are created that build and test every pull request to the repository or deploy merged pull requests to production.

GitHub Actions can be configured into a workflow to be triggered when an event occurs in a repository, such as a pull request being opened, or an issue being created. The workflow contains one or more jobs, which can run in sequential order or in parallel. Each job runs inside its own virtual machine runner, or inside a container. Each job has one or more steps that either run a script that you define or run an action, which is a reusable extension that can simplify the workflow.

A workflow is a configurable automated process that can run multiple jobs. Workflows are defined by a YAML file that is checked in to a repository and will run when triggered by an event, manual intervention, or at a defined schedule.

Store sensitive values as secrets and never as plain text in workflow files. Secrets can be configured at the organization, repository, or environment level, and can allow storage of sensitive information in GitHub.

Secrets use [Libsodium sealed boxes](#) so that they are encrypted before reaching GitHub. This occurs when the secret is submitted by using the UI or through the REST API. This client-side encryption helps minimize the risks that are related to accidental logging such as exception logs and request logs, within GitHub's infrastructure. After the secret is uploaded, GitHub can then decrypt it, so that it can be injected into the workflow runtime.

There are certain proactive steps and good practices to follow to help ensure that secrets are redacted and to limit other risks associated with secrets:

- ▶ Never use structured data as a secret.
- ▶ Register all secrets used within workflows.
- ▶ Audit how secrets are handled.
- ▶ Use credentials that are minimally scoped.
- ▶ Audit and rotate registered secrets.
- ▶ Consider requiring review for access to secrets.



A

Client contract setup sample files

This appendix contains the following sample files from snippets shown in “Client setup steps” on page 60.

- ▶ “Sample YAML file with literal scalars” on page 100
- ▶ “Sample YAML file with double-quoted scalars” on page 101
- ▶ “Sample script for certificate or key files” on page 103

Sample YAML file with literal scalars

See Example A-1 for a complete example of YAML file with literal (l) scalars.

Example A-1 *YAML file with literal (l) scalars*

```
env: |
  type: env
  logging:
    syslog:
      hostname: ${HOSTIP} # eg 10.0.0.8 or ${HOSTNAME}
      port: 6514
      server: |
        -----BEGIN CERTIFICATE-----
        MII EuDCC AyCgAwI BAgI UBR9g6L5hivov7eNT00HSXW39oD0wDQYJKoZIhvcNAQEL
        BQAwXzELMAkGA1UEBhMCVVMxEzARBgNVBAgMCkNhbG1mb3JuaWExFDASBgNVBAcM
        COxvcyBBbmd1bGVzMQwwCgYDVQQKDANJQk0xZzAVBgNVBAMDMNhLmV4YW1wbGUu
        b3JnMB4XDTEzMDUyNzE4MzgzMFoXDTE1MDE4NzE4MzgzMFowXzELMAkGA1UEBhMC
        VVMxEzARBgNVBAgMCkNhbG1mb3JuaWExFDASBgNVBAcMCOxvcyBBbmd1bGVzMQww
        CgYDVQQKDANJQk0xZzAVBgNVBAMDMNhLmV4YW1wbGUuUub3JnMIIBoJANBgkqhkiG
        9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsgnjL9ZxtDlQf1IAwEwkcPXQEHA47LY03B07
        c5jai9vRSGxa16hj6SU0idzKj4lK7w9REhxnaihk1P8zff9E1d/FvMKANpdani0
        JUFT/FXzlwG1c+R+h3ZTv4LiEi9SzWwkbNB0hMN5YM7nCOWNxQBhptisNvdRqb6V
        idQfneU0aesmmu61IDeB54wpsZXiRXB7M2fpx3c/9ppFYSCGWPc0Bibu12r6kg85
        EykR4zjEqiVg4U9m4zchQE73Lj1+pDHaGuJlM1fKgPKwAdmPEGK8IBctb3WbuBff
        ojmger1AEajDlDD0IQE4s+dX/F+bZCZiJrpfHz8CAs6i/GPqVR1U2jP7zaAjggs
        KNfWm0CXcLdsdbxgGNU8ECzVKw9pWPcG4A4bxiznZpgWxwoBhmbqriZe1KEAjRM
        fGCGJI3xyODCsT/64zGIwX1ajpn9b6cWnAGoxrj0c5HSXnPGyirDc/7jmmSxcOKd
        Cd3SaQu/ahNHXjr5X1dQ7XrzgY/1AgMBAAGjbDBqMA4GA1UdDwEB/wQEAwIChDAP
        BgNVHRMBAf8EBTADAQH/MecGA1UdEQRAMD6CEEnlcnZlci5leGFtcGx1Lm9yZ4IX
        c2VydmlvYmV4YW1wbGUuUub3JnOjY1MTSCCTkuMjAuNy45MocECRQHxDANBgkqhkiG
        9w0BAQsFAAOCAQEAFldjI2j0XbVqbZcQNBIPbyL1fhsNX0iOnvPRU9tpA1M41+Ur
        dCFNQKivai7jKsATzDZZVG0Adprki7YFqUocZ/NpwC6GHFETrTCICaVrjUuF2ecg
        iwXLZiHwV+q0AbycG9VoUw7eRVKarfmd31YwZ1i127e44V1mjVd4gdJvV6a4H6
        5l3fobogin9535lnJuZjHQNQ7cbN1hrK0kNrvZpSfe51Z6EFUZHk3S/Wdux31rz
        1cQQAhpfJL84KdVm/cBISKGDutgAADcb9jtH8q+ow19n7R1ff10r4/9G7CA9mv6
        1lJf8+P8Y22CGDvez3YPs3Dt7AnZe7bEEpQg1ED1gVq+WzbyEemmaVwJcj5o7ann
        ifkUIrmJOiActD1klQXYIiZcyg8zflIcFYkf+PHpvtYF8c5Av4DS2YFPwtc50oLz
        xUOI54V6pJZBc60aL9vNyxBAKj/cQEQRltUGr1vjPTOfycNW5yQAaHDL6o6E+3S
        uTEC1Fsd2QMaCJuQ
        -----END CERTIFICATE-----
      cert: |
        -----BEGIN CERTIFICATE-----
        MIID0zCCAjsCFFS5goaaDyhsJsUHv5WooqDg9gqGMA0GCSqGSIB3DQEBcWUAMF8x
        CzAJBgNVBAYTA1VTMRwEQYDVQQIDApDYWpZm9ybmlhMRQwEgYDVQQHDA1Mb3Mg
        QW5nZWxlczEMMAoGA1UECgwDSUJNMRCwFQYDVQQDDA5jYS5leGFtcGx1Lm9yZzAe
        Fw0yMzAxMzAxNjIzMDFaFw0yNTAxMjIxMDFaMG0xZzAVBgNVBAYTA1VTMsw
        CQYDVQQIDAjDQTEUMBIGA1UEBwwLTG9zIEFuZ2VsZXN0ZDAkBgNVBAoMA01CTTEQ
        MA4GA1UECwwHU3lzdG9zIEFuZ2VsZXN0ZDAkBgNVBAoMA01CTTEQMA4GA1UECwwHU3lzdG9zIEFuZ2VsZXN0ZDAkBgNVBAoMA01CTTEQ
        IjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvQoaZ9z2ZU0sKCoJ+lTyzI7v
        N3Mhc2Q0sSYBwXrQIFut4WW1pinXX0q1o4iPrNqsQPzhkN8blZrgI2SFk1N8IdK8
        JHFc09yVWEKmnxNeIOgi0vjrk3nSTkDH7GZyZe0p0d+Dbk671P4cKoxi32JgSK2i
        Fe1ZnYrgELiZFWbIZfKuy4YzWFOBSIPN2GZfp2IFmjzAyDpasc5ucL4u1I8jqBVd
        FXzpygkbnadWNrmD57LyVgUK+aez0+JXkSOBL9XiDjIDvSNFuSuzUdZqHBBJsDNI
        I2AF+e1a6JRgisK0A0U5m6Jfemnu6e+oHToY07vEUiRtueWg99Y1C0+zdnsgwID
```

```
AQABMAOGCSqGS1b3DQEBcWUAA4IBgQBN8StXpYaFIHj6E7Kn1q30kn50IUriZSez
/XUm1wi bmqVV/oh5YSQhMYrhJZ1yX6onR11sj2BHxQ7HDnZpChwo8RvYvTQE6EF2
uhAQxAdOV+pQ7U/oMTF5JzFK8YnhXWsTqJ11SI/FR5A1dLWne9B6y8FcBzEnATHV
Ly8aJJCsi7zFwWzfQZn7WL1SKDKy8f7sZaWJAfch9auoqQH+AVvoQCK7sT26dzpI
hskWkj0cwrSgmtU2XSERS09mPCyVBCFeX08Igl3XSyKb1h4qb++zeUsS8xb4EN27
hL/Y0oBH3nVcpWw5EhVU5RJfbStrp1kGT7a6CR14LyVSDNzPX0zEtUt958fdoXNV
rkPh/xre3lCxV8pTC7BG681kCMfDqHdKkwcP8HJpJfXc/aQ220vh2JevXNSFQtIP
2idwz4A+jn+FEVWAKB6j0S4XBoEdNh/8nSbiJLJ6bIOc4Rz2KLprKNZ4y43toI4
+nQ1PgTmc+x6hmiSZJuV00nGVEDGxA0=
-----END CERTIFICATE-----
```

key: |

```
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEA vQoaZ9z2ZU0sKCoJ+lTyZ17vN3Mhc2Q0sSYBwXrQIFUt4WW1
pinXX0q1o4iPRnQsQPzhkN8blZrgI2SFk1N8IdK8JHFc09yVWEKmnxNeIOgiOvj r
k3nSTkDH7GZyZeOp0d+Dbk671P4cKoxi32JgSK2iFe1ZnYrgELiZfWbIzFkuy4Yz
WFOBSIPN2GZfp2IFmjzAyDpasc5ucL4ulI8jqBVdFXzpygkbnadWnrmD57LyVgUK
+aez0+JXkSOBL9xiidjIDvSNFusZudZqHBBJsdNII2AF+e1a6Jrgi sKOA0U5m6Jf
emnu6e+oHToY07vEUiRtueWg99Y1C0+zdnsdgwIDAQABAoIBAGU1l19iEUxehHPV
GoWhE1C1YfWvWoSdUuciSLNwg0/pg6UIgMsptBv5SstZFCBEZLFZHXAgkGfz+He3
f2k20EJguV5ecVvUt1JsRy7yc/jze+1F5vZ/xjEspEwu/KLg5PFwKgy5Hmhw2W
tAiGYLJChw1n9BVBi2Ym/3HeDvFMBSv4JkgVwZvj1Qdd1pP21jMiZdKjblvdU45
6QUYsv0kz2WYQP01b23B5Yy5cUmRwutYacFgqkPbTmdjbpAAwr1nYi8uIVqYn47h
72dwV9RfiqKUGkb05UmfwonQYwHecpqFIG4j1kKgjY5MeIcecgYYaq9WpoCUgk5h
BaOfT+ECgYEA670o5rd1E4kGV6b4XvtIyWAsKGD50favyxj9VjCSElb10UC4vUp3
kTu42EJ1UVGN8TYkpz3sAqkT+tVTrC663KLD3bdRwF0fP6BkvHTshWP03nnWwjTN
RNjAroojjDKfSLI/Qn2BPAQm2QsJFRa5ZYM17WMAjuZu+V0z1z+uphsCgYEAzVG2
zTZ5668AU0FU7eHjJfDdBYhiEzIjgme3zFKg3tINv1yStyLKURkxtKUm1iw8Tdhk
4as3YUu0ky0Dz+XopN8IBhwBkwGizXMHP9BYfxrIy7yOVj9Lp2WSORvjwoB7nGxJ
1Wwyrrer202tNKMX6sGynJvfGsfh9AvTddl dwfLkCgYAtDnsLH6PCyD7e1pz4CzEu
zaOjVGZQHkgcmvpSr5ZQXS1Ahw7JoKKasL/1Fz81/FEV+y6uKbgkCg43t0/5yjU0
WE7440I54Z1HoHGhVp1hxynYHZJgZfPwV+T/fQtqL+qNejB3RwHTQpGavyzBVE
wn1Nk89XgdVU9Bs82n+YYQKBgBuwn1y5SfvUn9CacN+bpMxLDSNf3woDqvI9FnZB
d1xwK3B0f6Ke2HXTVfashuWd1YxR8FjWhCNk2Dc4zNjOgm8pfkWRUoNcG0QZBvg
9u49KHMBPJi49HTgp7V342EpfoH7wHiChMGDFc1LLR6hZLJCFNCwGPKArGsnpm39
ILhRAoGARwIM8o27pBymXcoyQDSqDgobXwKs9qqDxgYqX0JudPm8F39AG/Ww3FCx
f3t+UjvqAQWEL5bt+kGeBlrIbidgr2zuQ5mlEHCRchE1GI4QcTmGSBqW7KZJpGy2
BAsiTpe4uyh4+Vk7gXqt9+pfdf7aFiWphDERYJVFUfoizytAYjY=
-----END RSA PRIVATE KEY-----
```

Sample YAML file with double-quoted scalars

See Example A-2 for a complete example of YAML file with double-quoted (") scalars.

Example A-2 *YAML file with double-quoted (") scalars*

```
env: |
  type: env
  logging:
    syslog:
      hostname: ${HOSTIP} # eg 10.0.0.8 or ${HOSTNAME}
      port: 6514
      server: "-----BEGIN
CERTIFICATE-----\nMIIFCTCCA vECFEp7wJLz4jNstIsVH2dUeHDN26ZyMAOGCSqGS1b3DQEBcWUAMEEx
\nCzAJBgNVBAYTA1VTMRkwFwYDVoQKDBBMB2dzdGZaCBUZxNOIENBMRcwFQYDVoQD\nDA5jYS51eGftcG
```



```
xf+i0VpabwUKYgPLP5kqb9iBpfhQIDAQABAoICACsovIzfgHmuf/dMcc1FM1dS\njnb0eDeGC7ox47FCniw
T3GUfNqri4jx2nk6PKDPIR9ju0sfaztDPzkFNTK81ioeqA\nnabs97Ue7vWfNjiBqHySvyF5fmRFqQGIIHVH
N5GfeJ3Aru4914/lqxaAVnMKNMttK3\nDf6DEMIxI3JfPwi6qQSVJiDezK+oyNsWvAk0+gqHP6XPu3XIUB
tRLHs12Q3kA4tW\nSCH7q6I+huWZOANKqs4jObctJ1XUMyihSvZVjHlHwm1XQc/KTkfXQIyMsF349XA0V\n
wccvt4gA3jaZwWPL5LaIKkJ212tI9NaH7BiYZ64XUs1YGdvQ7130M1EztA1z70e\n9M4tkvdELLcvyByE
sY3JaObWe/N/RPk3vom4EP/XF+dTnIXR00nLDP5Kwzj1Cpwb\nRh7Jp6dmfOpBMLKtb5iEKUROpjGJT+jK
ORhWaTwo4zmjqj6EHp1Z2cUeZdvZomQWM\nnb75xNoJyBKooLAAfdGW00ADR1nbK56+RpbF07/xHHdWR1SuJ
E6vppkQ33W0sLMJb\nCo141AG+5NRPe5bn5KH9KrgsJTNPplhNfq5KGE+xb+gfIH71KuxMgHafBp2Ng432
\njGr4ZfBJy/w8cS0jLWrzAPEvz1ZmhIEGNeiOs78Q06efeT4fCAohw7qQur23K8YB\nTyVfDUaq63ndFq
3kqBgTAoIBAQDZPs86SyDwvLttalLgpe8z+XgxLRWRwPCQ+yv\nAhJkaehxyavApkp+k5f1EaAL9g2OZC
zMA3N8imsEe8zvbrDuR/xBCIUgppmQnD/E\nnCUQk4Un63znNZdJ+h7cn/kysi7D0od9oHgYxI3oj0VhPNk
dwm4GSJ81vXrL7RD/f\nncXFkrzI0mdkOJY1DydtRAS772MKXVURxaFZ3kePFETlqqBIAkt1gw9uSrtk
g8\nnecW3NVQDI521Q/uNYQaqA2AKGmLXC9Cg4GcfxseaI LxWaEsd+c0AzzoU8v+8nPR0\ntVdKPEg//b0T
JARxh4ZQV0xogLVbgW2JoY9gsas1qZpQaRbLaoIBAQMhCBVG8v0\nnqsmqv2qV0+251KLt5Y2hfU5k300N
sIAQ11a7sK0Y4eXiVpKzT3J/emZeLsQmHnw2\nRdIXqjaVjH5jNADV9tHsQZ2KA0qv2J0/VK7fQtjv8XIa
Hh/gIAXXerSyLdh7rdHp\ng+xfHaDB1kKUmZR6MtB87h1Q6ngB149/7xgJReTee2oj4sN10ALVi51XyNf
Z+FV\nQu8D2VP6ssn70qvempzLaP70Znj2eES7KK8XEm7x7StvOptZ17n+E+OqZ1i40VcF\n7UCRBCrmDY
WYCLmajmR04zyBeppJfSRBh3y3mXBeNwm1FqmQz5NVNDg/YkcPbM40\nnakCX7ZXP0jvAoIBAFhZx/NgLI
RbbSpAxet8x0107sepGzib/fZao30yRPgIfGUS\nnbIwhiTBTHCCpy1ox9j7f4qwr1zzWG1HXe3AAp2ow0n
EsYtVimd+K/A/g6NrK2Mhz\nu1GrUGDvFtjn/gzKPuwuj0oOE9yWHg1FDVihTaoi5B4pmi116PpxNjzMKR
QDjisL\n/I9ZTEs+Y7hc79uyuuJk36vzj/700UAEJrzwaQUUxdFG1PghRckadpWd8dbo9An\nAvN+M2a7
B/fKqZtSQdJNVsqmlgVE1Va0t3G4tpv5QL45CNkOP11Zw7h1z4s8WvHT\nYrrPFLhHsqMm9oJFnfwZ9g9c
KjBb8Uu+3yhGpOMCggEAEHzfZwReGB2zev0TvLrC\nlTS426/dtWKN2YvsHt/5Qkz2EtKoPnvuo13fRPWr
/X/NaPTiJ5bUFGEjuT9ustu2\n5ZiQWXz0BNxPBCyWNxsFR7WK5AqM1dWNI+fVXYNgDabDZyjtHue0n35R
tWNN/opM\na6Diw07ItqDK10nacs1RU8w2e9gkUirAoQoXoIrlTyIJCqoeu6kGLEw1y72v5MSJ\nni+p7yE
OL1aXAdZgn3WPTEf0Q2uXIKIXRh6oqTSi+sP1kqVIDCVz2cI5p+ETdRPR4\nnXK3fmjdg5Rwt4pwo6VxpG6
m8Hqmtck04UeK8+IvhP1PpQyYRuPuf1Qxxi0+zvbb+\nTwKCAQAtxUAS8r+AP3Uufi9DvujI5z3+mWqZiM
5Mxg80JqQnPE8V6gfrSspEgDt\nnHWF8TUNoATWLCcAk1u9ImBqiPZMH9WfRxaLSofrFJsVTft+5ZeT6QM
nc0RnBZakL\nnvJMX9rKkb987eIRfCwz1nbQ84IFM41e0F15+853aIibpBAI7BEftvJ8Eg/m20w1H\nnrPRP
1j6GyhpkaIm2+TVx6DFY/J06JM1i0tzHv7zihSeji01wBMKJ7MOTRX1dJeR\nn3GsD1D7mKwLVaBBKQ1Ux
h1zYbiaUzVst1S2Wdvt13f89IV4Mmuq2v1Uz4je7pDB\nnhJITxResgCTR2aD0nMzF8egEKJoY\n-----E
ND PRIVATE KEY-----\n"
```

Sample script for certificate or key files

This simple sample script can be used to output certificate or key files into double-quoted scalar values for YAML files. See Example A-3.

Example A-3 Script for certificate or key files

```
~# cat yaml_doublequoted_input.sh
#!/bin/bash

sed 's/$/\n/' $1 | tr -d '\n'
echo ""
```

See Example A-4 for how to use the script.

Example A-4 Making use of the script

```
~# cat ca.pem
-----BEGIN CERTIFICATE-----
MIIEuDCCAyCgAwIBAgIUBR9g6L5hivov7eNT00HSXW39oD0wDQYJKoZIhvcNAQEL
BQAuXzELMAkGA1UEBhMCVVMxEzARBgNVBAgMCKNhbg1mb3JuaWEeFDASBgNVBAcM
```


double-quote (") in a YAML file as can be seen in the double-quoted scalar example in Example A-2 on page 101.



B

Creating a Hyper Protect Virtual Server for VPC

This appendix provides a walkthrough that shows how to create a Hyper Protect Virtual Server (HPVS) instance by using the IBM Cloud Virtual Private Cloud (VPC) user interface (UI).

Using the IBM Cloud VPC UI

In this section we show the steps that are needed to create a HPVS instance using the IBM Cloud VPC UI.

1. Login to IBM Cloud at <https://cloud.ibm.com/login>.
2. From the icons in the left panel, select VPC infrastructure. See Figure B-1.

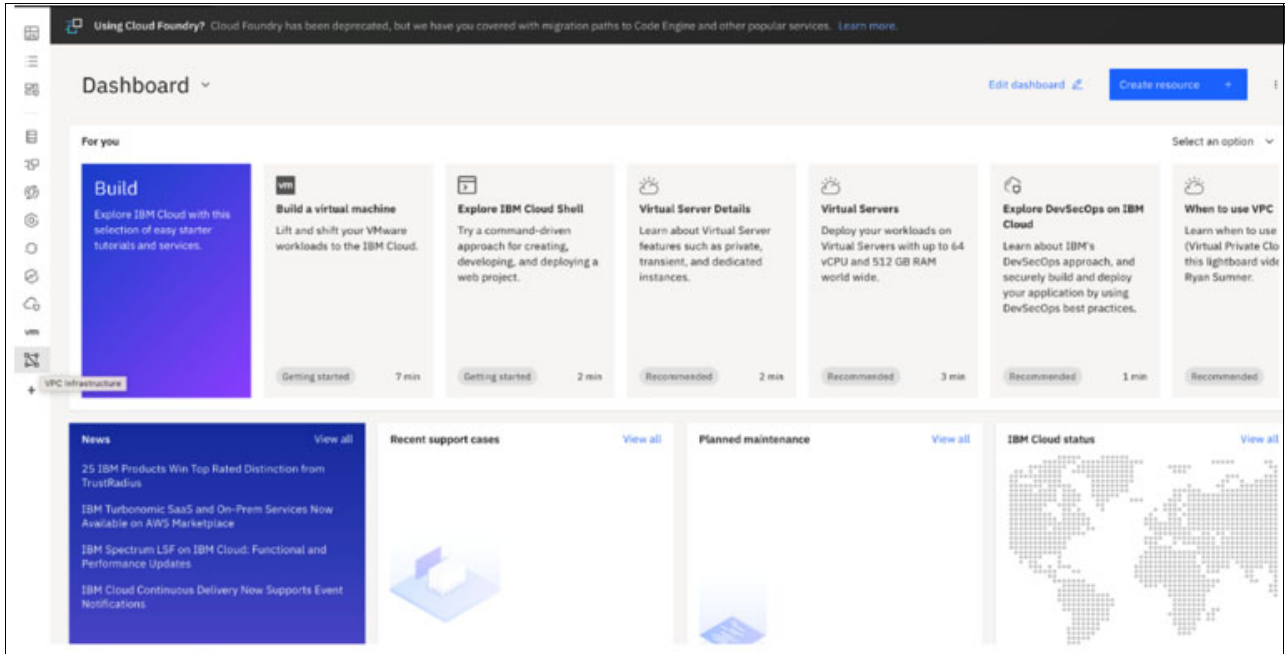


Figure B-1 Login to IBM Cloud

3. Click **Virtual server instances** and click **Create**. See Figure B-2.

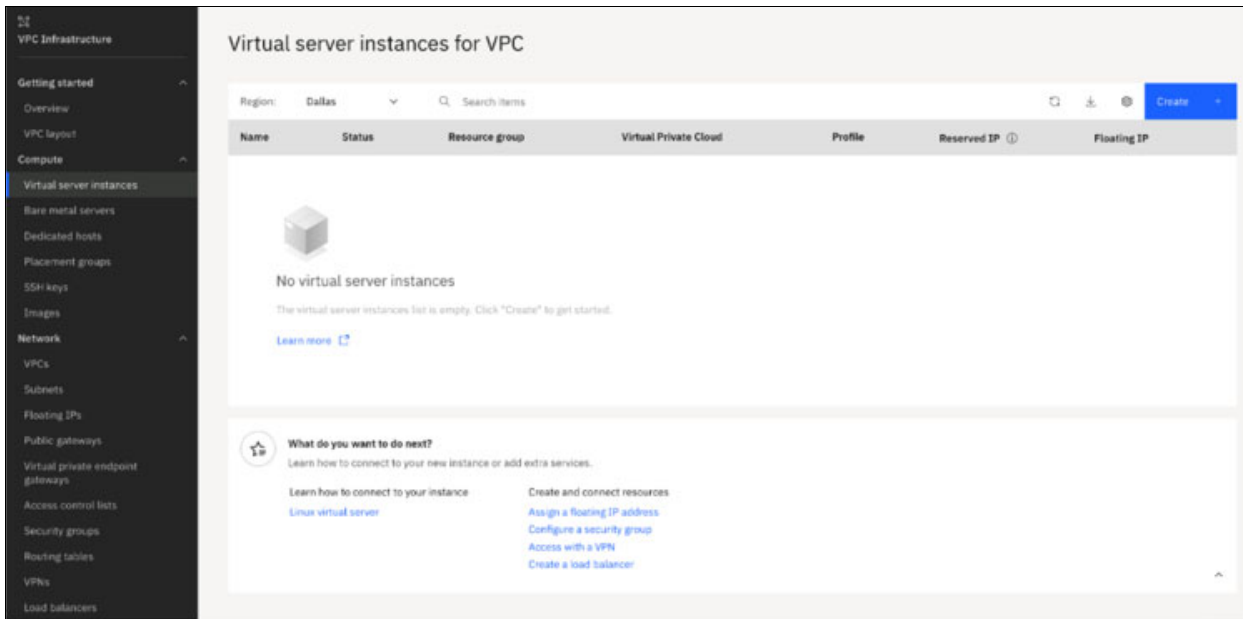


Figure B-2 Virtual server instance for VPC

4. Select the location of your choice, then provide a name for your virtual server. See Figure B-3.

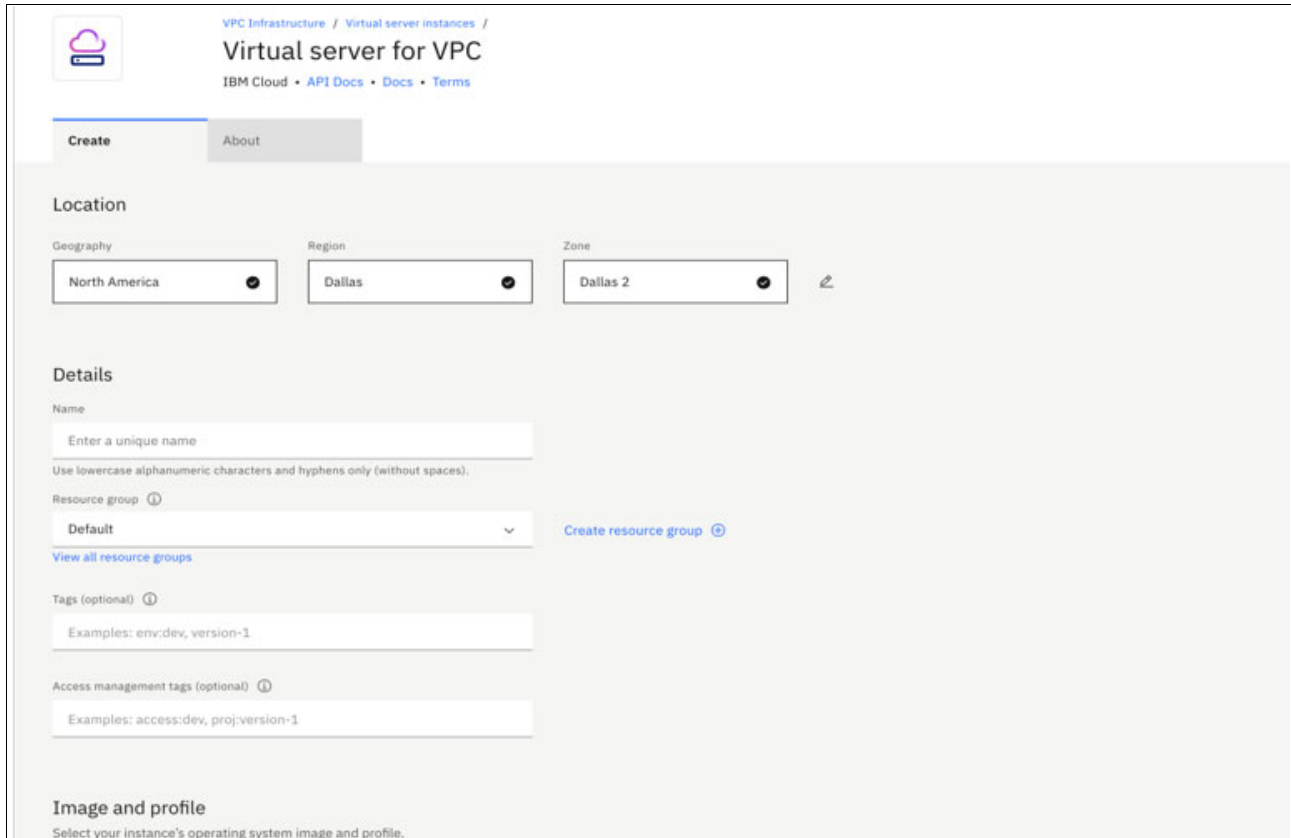


Figure B-3 Select a location

5. Under the Image and Profile heading, for the CentOS image, click **Change image**. See Figure B-4.

Image and profile
Select your instance's operating system image and profile.

Image ⓘ


 **CentOS** [Change image](#)

Image type	Name	Version	Architecture
Stock image	ibm-centos-7-9-minimal-amd64-11	7.x - Minimal Install	amd64

[Change image](#)

Profile ⓘ

Balanced | bx2-2x8 [Change profile](#)

vCPUs	RAM	Volume bandwidth	Network bandwidth
2	8 GiB RAM	1 Gbps	3 Gbps

SSH keys ⓘ
Add one or more SSH keys to your instance

SSH keys

SSH keys [Create an SSH key](#)

Storage

Boot volume

Name	Size	Max IOPS	Bandwidth ⓘ	Encryption	Tags	Auto-delete ⓘ	
boot-volume	100 GB	3000	393 Mbps	Provider managed	-	Enabled	↗

Figure B-4 Change image

6. On the next screen select **IBM Z, LinuxONE, s390 architecture**. See Figure B-5.
7. Select the switch under Confidential computing.
8. Select the desired image.

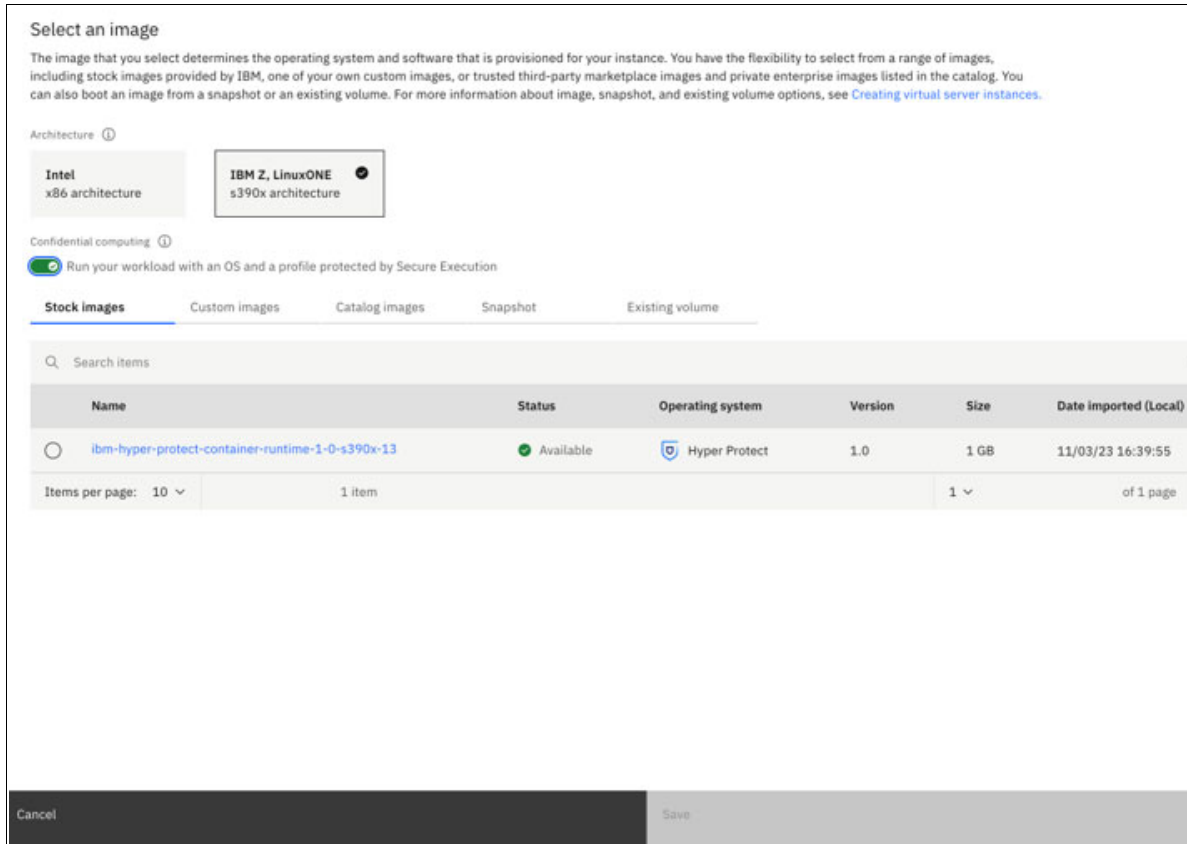


Figure B-5 Select architecture and confidential computing

9. Click **Save**.

10. Select the profile and under the Data volumes heading, select **Create**. See Figure B-6.

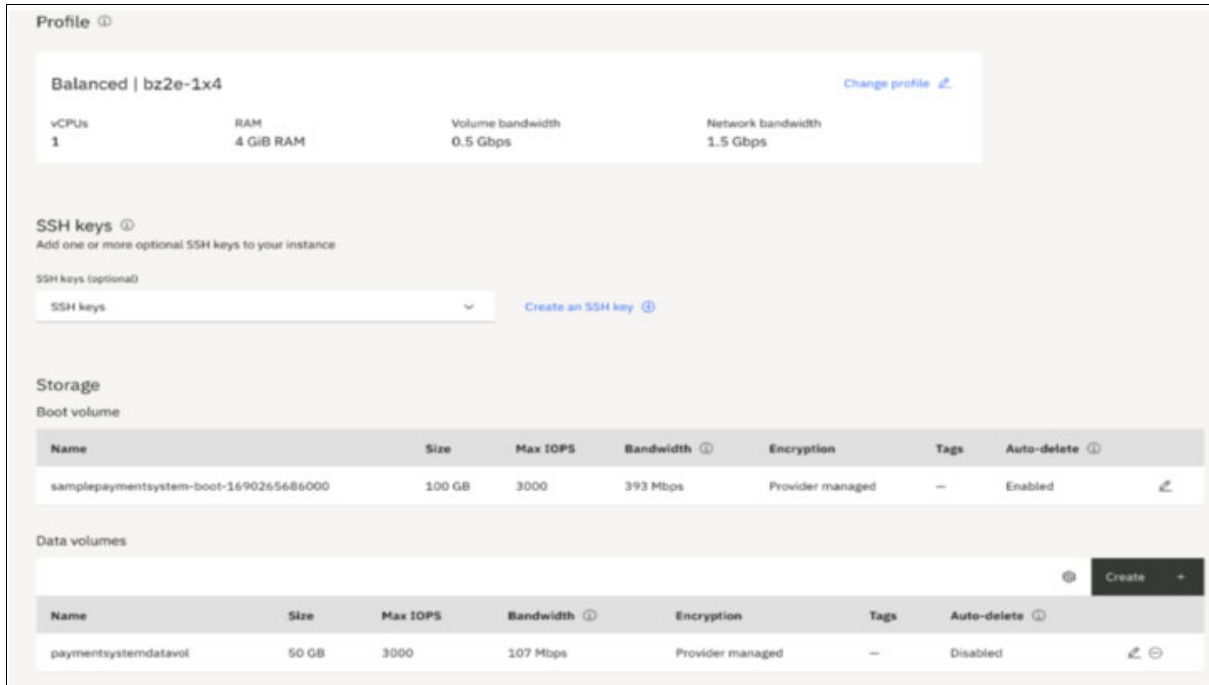


Figure B-6 Select a profile and storage

11. Input the contents of `user_data.yaml` prepared in 4.2.4, “Prepare `user_data.yaml`” on page 73 to “User data” field. See Figure B-7.

12. Click **Create virtual server**.

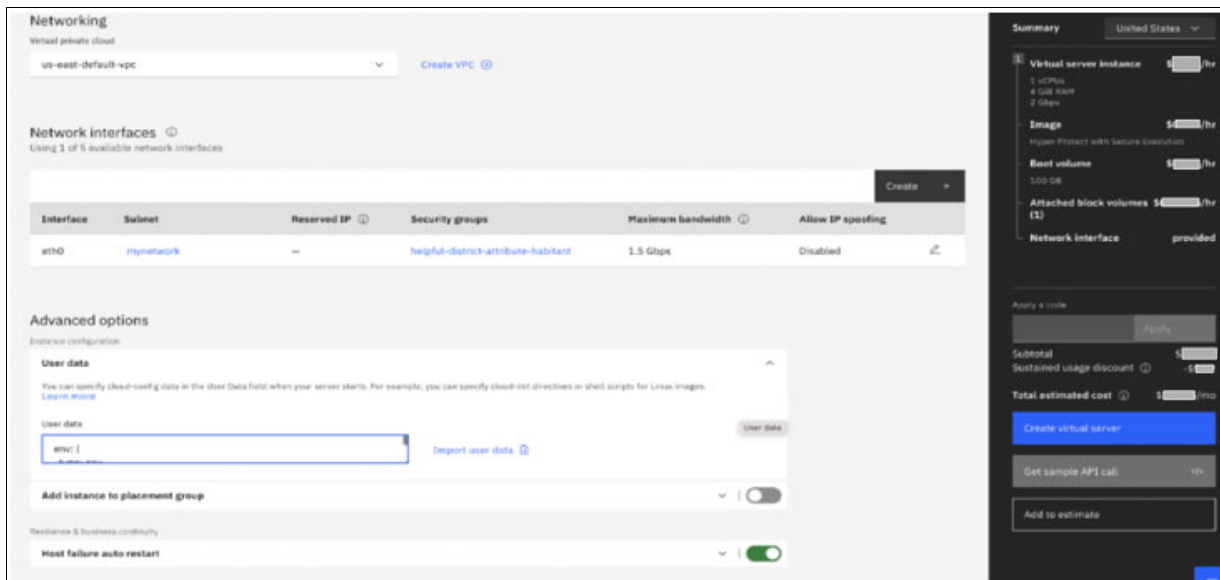


Figure B-7 Create a virtual server

13. Monitor the serial console to see the boot process of the virtual server instance. See Figure B-8.

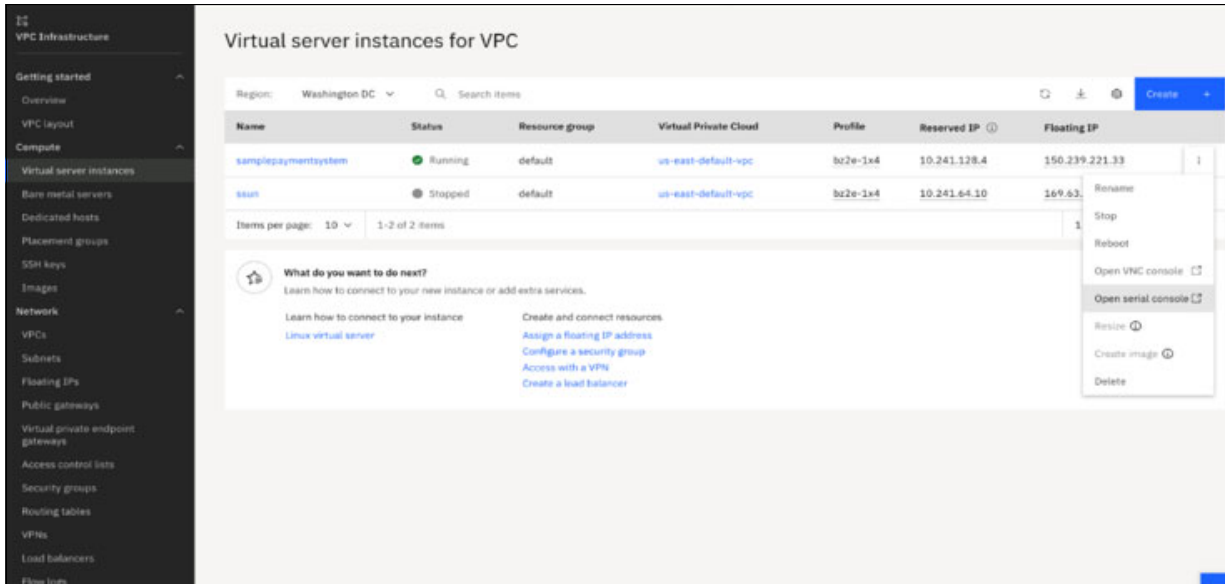


Figure B-8 Open serial console

14. View the messages in the serial console. See Figure B-9. If there are any errors, the virtual server instance shuts down.

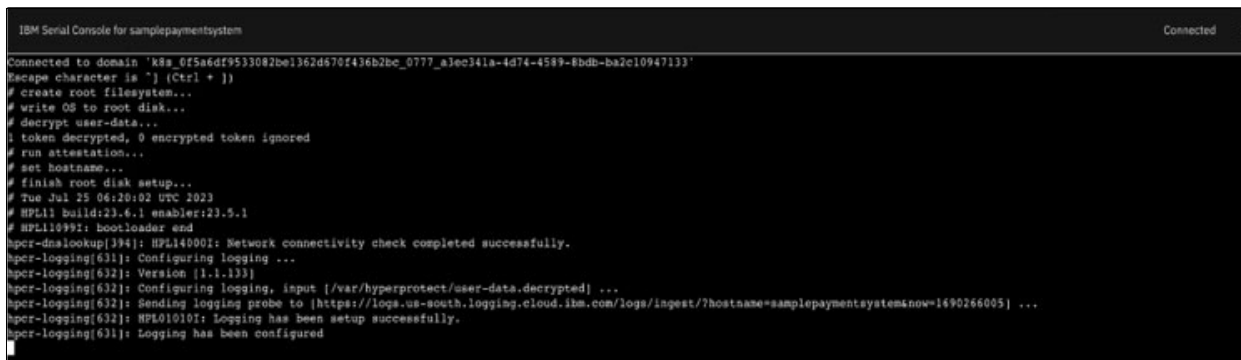


Figure B-9 Virtual server boot

15. Monitor IBM Log Analysis to see if the container workload is started successfully. See Figure B-10.

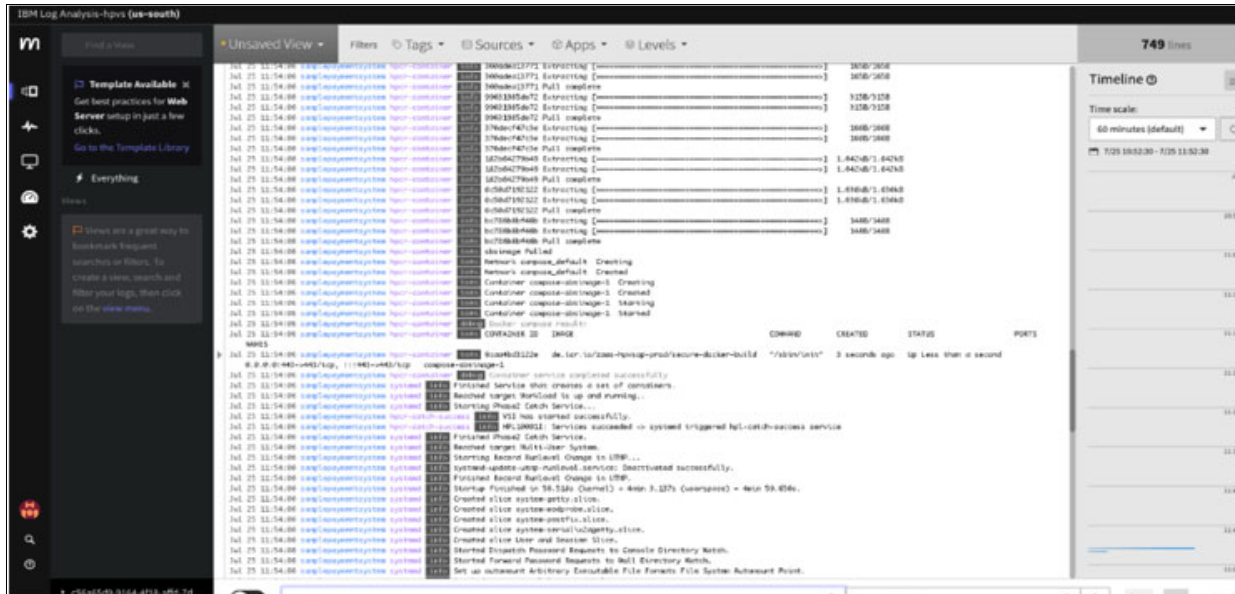


Figure B-10 Check the container workload



Additional examples for HPSB and HPVS

This appendix contains the complete examples of the Hyper Protect Secure Build (HPSB) process from Example 4-19 on page 76 and HPVS instance verifying disk (volume) encryption from Example 4-38 on page 88.

- ▶ “Hyper Protect Secure Build log” on page 116
- ▶ “How to verify disk (volume) encryption with HPL13000!” on page 118

Hyper Protect Secure Build log

The `./build.py log --log build --env` command can be used to view the HPSB log. See Example C-1.

Example C-1 HPSB log

```
$ ./build.py log --log build --env sbs-config.json
INFO:__main__:2023-07-25 08:59:36,446 build_task INFO starting a
build
INFO:__main__:2023-07-25 08:59:36,447 build_task INFO cleaning
up the local github repo and the github access credential
INFO:__main__:2023-07-25 08:59:36,447 clean_up INFO
github_dir=paynow-website
INFO:__main__:2023-07-25 08:59:36,448 build_task INFO cloning a
github repo
INFO:__main__:2023-07-25 08:59:36,448 clone_github_repo INFO
github_host=github.com
INFO:__main__:2023-07-25 08:59:36,448 clone_github_repo INFO
github_dir=paynow-website
INFO:__main__:2023-07-25 08:59:36,626 run INFO run:
Cloning into 'paynow-website'...
INFO:__main__:2023-07-25 08:59:37,010 clone_github_repo INFO
image_tag=v3-f29b1ab
INFO:__main__:2023-07-25 08:59:37,011 build_task INFO building a
docker container
INFO:__main__:2023-07-25 08:59:37,011 build_docker_image INFO
github_dir=paynow-website
INFO:__main__:2023-07-25 08:59:37,025 run INFO run:
environment-variable.
INFO:__main__:2023-07-25 08:59:37,025 run INFO run:
INFO:__main__:2023-07-25 08:59:37,129 run INFO run:
Sending build context to Docker daemon 2.842MB
INFO:__main__:2023-07-25 08:59:37,131 run INFO run: Step
1/7 : FROM node:19
INFO run: 5ac921848b31: Waiting
INFO:__main__:2023-07-25 08:59:37,393 run INFO run:
86b7a0ecd4be: Waiting
INFO:__main__:2023-07-25 08:59:37,717 run INFO run:
e4000487deec: Verifying Checksum
INFO:__main__:2023-07-25 08:59:37,719 run INFO run:
e4000487deec: Download complete
INFO:__main__:2023-07-25 09:00:12,822 run INFO run:
e4000487deec: Pull complete
INFO:__main__:2023-07-25 09:00:18,052 run INFO run:
Digest: sha256:92f06fc13bcc09f1ddc51f6ebf1aa3d21a6532b74f076f224f188bc6b9317570
INFO:__main__:2023-07-25 09:01:16,173 run INFO run:
Status: Downloaded newer image for node:19
INFO:__main__:2023-07-25 09:01:16,175 run INFO run: --->
f2e8386523b1
INFO:__main__:2023-07-25 09:01:16,175 run INFO run: Step
2/7 : WORKDIR /app
INFO:__main__:2023-07-25 09:01:29,718 run INFO run: Step
3/7 : COPY app/package*.json ./
```

```

INFO:__main__:2023-07-25 09:01:29,878 run INFO run: Step
4/7 : RUN npm install
INFO:__main__:2023-07-25 09:01:33,744 run INFO run: added
65 packages, and audited 66 packages in 3s
INFO:__main__:2023-07-25 09:01:35,142 run INFO run: Step
5/7 : COPY app/ .
INFO:__main__:2023-07-25 09:01:35,450 run INFO run: --->
1ed7cc739746
INFO:__main__:2023-07-25 09:01:35,450 run INFO run: Step
6/7 : EXPOSE 8443
INFO:__main__:2023-07-25 09:01:35,572 run INFO run: Step
7/7 : CMD npm start
INFO:__main__:2023-07-25 09:01:35,594 run INFO run: --->
Running in fcba8cd7f8ac
INFO:__main__:2023-07-25 09:01:35,678 run INFO run:
Removing intermediate container fcba8cd7f8ac
INFO:__main__:2023-07-25 09:01:35,679 run INFO run:
Successfully built 9cae137191b0
INFO:__main__:2023-07-25 09:01:35,683 run INFO run:
Successfully tagged devuser/samplepaymentsystem:latest
INFO:__main__:2023-07-25 09:01:35,684 run INFO run:
return code = 0
INFO:__main__:2023-07-25 09:01:35,748 run INFO run:
return code = 0
INFO:__main__:2023-07-25 09:01:35,749 build_task INFO pushing a
container to a docker hub
INFO:__main__:2023-07-25 09:01:35,860 run INFO run:
INFO:__main__:2023-07-25 09:01:35,860 run INFO run: Login
Succeeded
INFO:__main__:2023-07-25 09:01:35,861 run INFO run:
return code = 0
INFO:__main__:2023-07-25 09:01:35,910 run INFO run: The
push refers to repository [docker.io/devuser/samplepaymentsystem]
INFO:__main__:2023-07-25 09:01:35,922 run INFO run:
fb4fa3257dd1: Preparing
INFO:__main__:2023-07-25 09:01:35,923 run INFO run:
fda0660b571f: Waiting
INFO:__main__:2023-07-25 09:01:36,422 run INFO run:
ef13dc0a223f: Mounted from library/node
INFO:__main__:2023-07-25 09:01:37,211 run INFO run:
ef13dc0a223f: Pushed
INFO:__main__:2023-07-25 09:01:38,885 run INFO run:
v3-f29b1ab: digest:
sha256:d10e26e72a2f83a3fdf8a6a79da5b88f1b6747ce0af9309749afc55295973bd8 size: 2839
INFO:__main__:2023-07-25 09:01:38,888 run INFO run:
Signing and pushing trust metadata
INFO:__main__:2023-07-25 09:01:39,289 run INFO run:
Finished initializing "docker.io/devuser/samplepaymentsystem"
INFO:__main__:2023-07-25 09:01:39,388 run INFO run:
Successfully signed docker.io/devuser/samplepaymentsystem:v3-f29b1ab
INFO:__main__:2023-07-25 09:01:39,390 run INFO run:
return code = 0
INFO:__main__:2023-07-25 09:01:39,448 run INFO run: The
push refers to repository [docker.io/devuser/samplepaymentsystem]
INFO run: ef13dc0a223f: Layer already exists

```

```

INFO:__main__:2023-07-25 09:01:40,152 run INFO run:
latest: digest:
sha256:d10e26e72a2f83a3fdf8a6a79da5b88f1b6747ce0af9309749afc55295973bd8 size: 2839
INFO:__main__:2023-07-25 09:01:40,154 run INFO run:
Signing and pushing trust metadata
INFO:__main__:2023-07-25 09:01:40,398 run INFO run:
Successfully signed docker.io/devuser/samplepaymentsystem:latest
INFO:__main__:2023-07-25 09:01:40,399 run INFO run:
return code = 0
INFO:__main__:2023-07-25 09:01:40,400 build_task INFO extracting
an image keyid and key
INFO:__main__:2023-07-25 09:01:40,400 build_task INFO docker
contrust value:https://notary.docker.io
INFO:__main__:2023-07-25 09:01:40,400 build_task INFO entered
dct
INFO:__main__:2023-07-25 09:01:40,400 extract_image_key_id INFO
keyid=60340916db8868c4db38f99a9928829332b341ca5313f548ec25cc015102c140
INFO:__main__:2023-07-25 09:01:40,400 extract_image_key_id INFO
publickey=LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUJsVENDQVR1Z0F3SUJBZ01RV2RmQStQ
cm9tVjRwd2RVS01KYnFqREFLQmdncWhrak9QUVFEQWpBeE1TOHcKTFZFRZRUURFeVprYjJ0c1pYSXVhVz
h2WVdKb2FYSmhiV3N2YzJGdGNHeGxjROY1Y1dWdWRITjVjM1JsY1RBZQpGdzB5TXpBM01qVXdPVEF4TXpo
YUZ3MHpNekEzTWpJd09UQXhNemhhTURFeEx6QXRCZ05WQkFNVEptUnZZMnRsCmNpNXBiE1lowW1ocGNtRn
RheTl6WVcxzd2JHVndZWGx0W1c1MGMzbHpkR1Z0TUZrd0V3WUhlb1pJemowQ0FRWUkKS29aSXpqMERBUWNE
UmdBRWp1MmFPYVpYUE4UHJFUThHNTgybTZxWm1JaEFvYWo2bDZaaThsaDFwM01VbTBLNAP4TVBJcytZNH
A2TzVzeE9tRFpFaw9SbmJ0eU1NRDJrNO5zNXVLYU0xTURNd0RnWURWUjBQVFILOJBUURBZ1dnCk1CTUdB
MVVks1FRTU1BbOdDQ3NHQVfVRk3JTURNQXdhQTfVZEV3RUIvd1FDTUFbd0NnwU1Lb1pJemowRUF3SUQKU0
FBd1JRSWdZUnZObW1nRkg1dTBSNn1ENUhxcDFTcW9zM2k5cVczbWxRWV1JN2oyZXJVQ01RRFZGjHxXo0
RQp1UXExeVJvaHIwZXpZck520Eh4eXqVUS9CMBD1QUY1Nz1RPTOKLS0tLS1FTkQgQ0VSVE1GSUNBVEUtLS
0tLQo=
INFO:__main__:2023-07-25 09:01:40,400 build_task INFO generating
a config file
INFO:__main__:2023-07-25 09:01:40,903 digest INFO
digest=636a6f000b503de6ac9f65c9c226b20d6ee72d513ed578547433f3a093061df0
INFO:__main__:2023-07-25 09:01:41,284 build_task INFO completed
a build
INFO:__main__:

```

How to verify disk (volume) encryption with HPL13000I

User data volumes in the HPVS instance are encrypted with Linux Unified Key Setup (LUKS) encryption. The encryption status can be verified by checking the messages in the log. Example C-2 shows verification of disk (volume) encryption.

Example C-2 Verifying disk (volume) encryption

```

common[991]: HPL13000I
...
verify-disk-encryption: Return value for disk-encrypt: 0
verify-disk-encryption: Executed cmd: ('lsblk', '-b', '-n', '-o', 'NAME,SIZE')
verify-disk-encryption: Return value: 0
verify-disk-encryption: Stdout: vda
107374182400
verify-disk-encryption: ??vda1 4292870144
verify-disk-encryption: ??vda2 103079215104

```

```
verify-disk-encryption: ??luks-4089f10e-284b-43aa-ac29-37203fb046c9 103062437888
verify-disk-encryption: vdb 391168
verify-disk-encryption: vdc 45056
verify-disk-encryption: List of volumes greater than or equal to 10GB are:
['/dev/vda']
verify-disk-encryption: Updated Volumes list: ['/dev/vda2']
verify-disk-encryption: Executed cmd: ('lsblk', '/dev/vda2', '-b', '-n', '-o',
'NAME,MOUNTPOINT')
verify-disk-encryption: Return value: 0
verify-disk-encryption: Stdout: vda2
verify-disk-encryption: ??luks-4089f10e-284b-43aa-ac29-37203fb046c9 /
verify-disk-encryption: Boot volume is /dev/vda2
verify-disk-encryption: Volume /dev/vda2 has mount point /
verify-disk-encryption: List of mounted volumes are: ['/dev/vda2']
verify-disk-encryption: Verifying the boot disk /dev/vda2 is encrypted or not
verify-disk-encryption: Executed cmd: ('lsblk', '/dev/vda2', '-b', '-n', '-o',
'NAME,TYPE')
verify-disk-encryption: Return value: 0
verify-disk-encryption: Stdout: vda2 part
verify-disk-encryption: ??luks-4089f10e-284b-43aa-ac29-37203fb046c9 crypt
verify-disk-encryption: Executed cmd: ('cryptsetup', 'isLuks', '/dev/vda2')
verify-disk-encryption: Return value: 0
verify-disk-encryption: Executed cmd: ('cryptsetup', 'luksDump', '/dev/vda2')
verify-disk-encryption: Return value: 0
verify-disk-encryption: HPL13003I: Checked for mount point /, LUKS encryption with
1 key slot found
verify-disk-encryption: HPL13001I: Boot volume and all the mounted data volumes
are encrypted
```



Encryption keys explained

Key generation is the process of creating encryption keys that can be used to encrypt and decrypt sensitive data that needs to be protected from unauthorized access. The kind of keys that need to be generated depends on the algorithm used for encryption. Symmetric-key algorithms, such as AES, use a single key for encryption and decryption. Asymmetric- or Public-Key algorithms, such as ECC, use a public-key to encrypt and the corresponding private-key to decrypt. The keys can be used to generate and verify signatures over data to prove their originality. This is especially useful with asymmetric keys in which the private key is used to generate the signatures and the public key is sufficient to verify the signatures.

The keys are created with algorithms designed to ensure that each key is unique and unpredictable. The starting point is usually a seed that is used to generate a series of random numbers to increase the statistical randomness of the algorithm used. A seed can be generated in software, but it can be embedded in hardware, which makes it secure and helps Hardware Security Modules (HSMs) generate random numbers. The generated random numbers can be used to generate encryption keys.

The same seed, if it is used with the same algorithm, generates the same encryption key. So, it is vitally important to protect the seed.

For more details on how seeds are used in confidential computing with Hyper Protect Virtual Server (HPVS), see [IBM Cloud: Securing your data](#).

For further information on random number generation see 4.3.4, “Random number generation” on page 80.

This appendix covers the following topics:

- ▶ “What is a master key (MK)” on page 122
- ▶ “What are data encryption keys (DEKs)” on page 122
- ▶ “What are key encryption keys (KEKs)” on page 123
- ▶ “Using and protecting keys” on page 123
- ▶ “How encryption keys are created using GREP11” on page 124

What is a master key (MK)

A Hardware Security Module (HSM) is a physical device that is used to protect and manage cryptographic keys. Before an HSM can be deployed, it must be initialized in a secure environment with a master key. Because all other keys generated by the HSM are derived from the Master Key, it is important to ensure that the master key is not compromised by any one individual. Hence, the master key is often divided into master key-parts with each part being owned by different individuals within an organization. See Figure D-1.

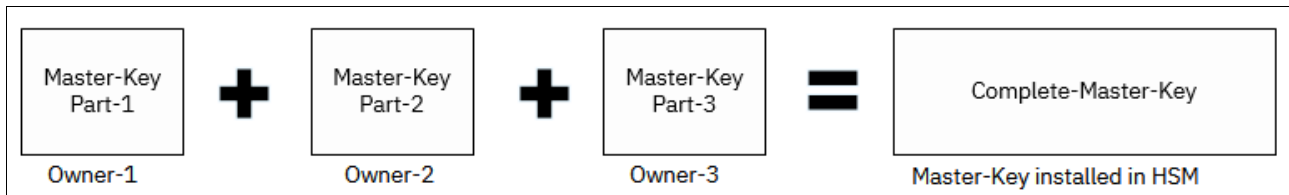


Figure D-1 Multiple master key parts

After the master key is stored in an HSM through the initialization process, the HSM is ready to derive a Data Encryption Key (DEK) or Key Encryption Keys (KEKs). See Figure D-2.

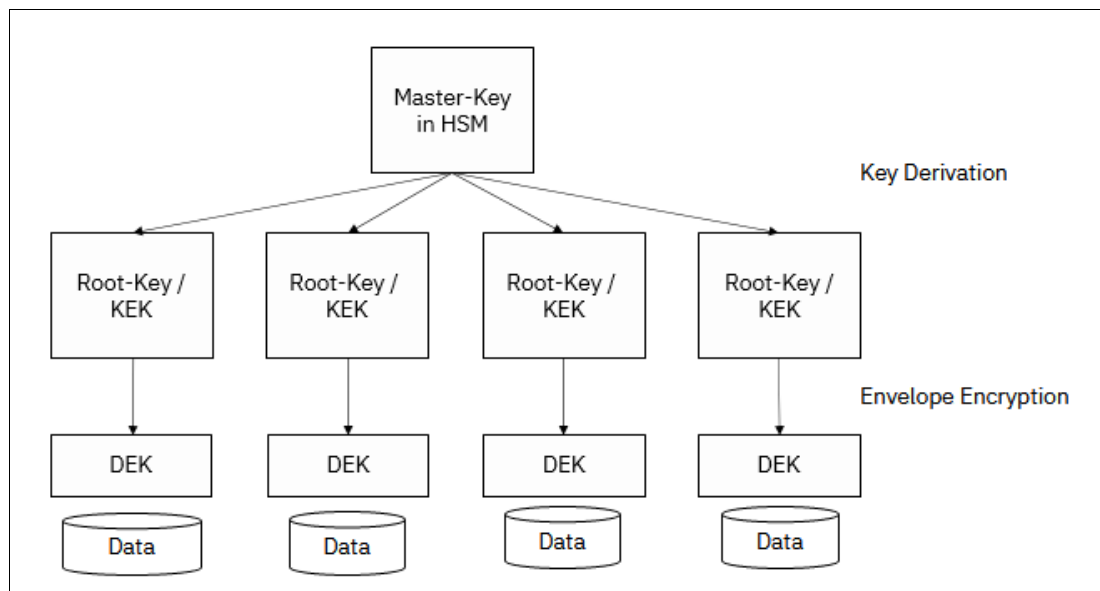


Figure D-2 HSM initialization process

What are data encryption keys (DEKs)

A data encryption key is a type of key that is designed to encrypt and decrypt data. DEKs are generally provided by the application that owns the data being encrypted. Data is encrypted and decrypted with the help of the same DEK. Hence, a DEK must be stored for at least a specified duration for decrypting the generated cypher text.

There are four levels in a DEK lifecycle:

1. The key is created using the crypto module of the encryption engine.
2. The key is then provided to a key vault and to various other encryption engines.
3. The key is used for encrypting and decrypting data.
4. The key is then suspended, terminated, or destroyed.

A DEK can be customized to expire during a particular time frame to prevent data from being compromised. Under such circumstances, it should be used once more for decrypting the data and then the resulting clear text is re-keyed, which means it is encrypted by using a new key.

What are key encryption keys (KEKs)

As the name suggests, a key encryption key is used with *envelope encryption* to protect a DEK from unauthorized use. Envelope encryption is the practice of encrypting plain text data with a data key and then encrypting the data key under another key.

Envelope encryption reduces the network load for the application or the cloud service as only the request and fulfillment of the much smaller data key through KMS must go over the network.

Using and protecting keys

If the key-generation is performed within an HSM, the seed stays in the HSM at all times. If the seed is generated within a confidential computing enclave or Trusted Execution Environment (TEE), encryption keys generated can be used without exposing them to external threats. To achieve the best security, the technologies can be combined.

With a Hyper Protect Crypto Services (HPCS) instance connected to an HPVS instance that is protected with Secure Execution for Linux, a simple yet secure signing setup can be established. This setup can be used to securely generate and use a certificate authority (CA).

The whole process can be built into containers running within the HPVS instance. First the HPCS instance is used to privately generate a new intermediate key pair. This is wrapped within the HSM and exported into the HPVS, where it is stored into an encrypted data volume. This wrapped key can be used only with the HSM backing the HPCS instance. To use this secure key, the container generates a CA certificate signing request (CSR). This is then in turn signed using an external signer with a well-known root certificate. Alternatively the CSR can be self-signed to produce a new root certificate. The resulting certificate can be used to establish a trusted certificate chain on the leaf certificates to be generated next.

In the next step, the intermediate key pair can be used to sign CSRs by itself. This further expands the certificate chain to leaf certificates to be used for means such as TLS, code signing, and so on. Each of these leaf certificate keys can either be generated in the application where it is used or within the secure environment.

How encryption keys are created using GREP11

Secrets like seeds, encryption keys, and certificates are required for applications to perform tasks that they are designed to do. Also, the secrets must be protected from unauthorized use.

When you work with a confidential computing enclave like HPVS, the stateless nature of the GREP11 (EP11) service offered by HPCS can be used effectively to keep the secrets in a tamper-proof secure enclave of HPVS. This is illustrated in Figure D-3.

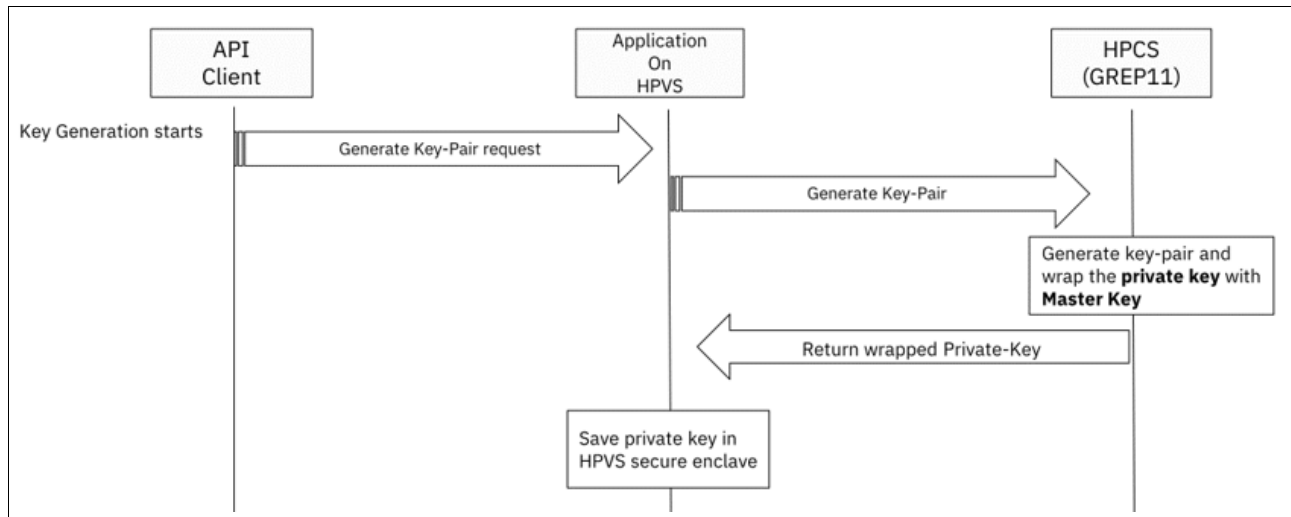


Figure D-3 Tamper-proof secure enclave of HPVS



Redbooks

IBM Hyper Protect Platform: Applying Data Protection and

SG24-8555-00

ISBN DocISBN

(1.5" spine)

1.5" <-> 1.998"

789 <-> 1051 pages



Redbooks

IBM Hyper Protect Platform: Applying Data Protection and

SG24-8555-00

ISBN DocISBN

(1.0" spine)

0.875" <-> 1.498"

460 <-> 788 pages



Redbooks

Applying Data Protection and Confidentiality in a Hybrid Cloud

SG24-8555-00

ISBN DocISBN

(0.5" spine)

0.475" <-> 0.873"

250 <-> 459 pages



Redbooks

Applying Data Protection and Confidentiality in a Hybrid Cloud Environment

(0.2" spine)

0.17" <-> 0.473"

90 <-> 249 pages

(0.1" spine)

0.1" <-> 0.169"

53 <-> 89 pages



IBM Hyper Protect Platform:

SG24-8555-00

ISBN DocISBN

(2.5" spine)
2.5" <-> mmm.n"
1315 <-> mmm pages



IBM Hyper Protect Platform: Applying Data Protection and Confidentiality in a Hybrid Cloud

SG24-8555-00

ISBN DocISBN

(2.0" spine)
2.0" <-> 2.498"
1052 <-> 1314 pages





SG24-8555-00

ISBN 0738461490

Printed in U.S.A.

Get connected

