Protegrity Data Security Gateway (DSG) Container Dynamic Policy Update User Guide 3.1.0.5

Created on: Nov 19, 2024

Copyright

Table of Contents

Chapter 1

Introduction to this Guide

The Data Security Gateway (DSG) Dynamic Policy Update User Guide discusses how to deploy the Protegrity DSG for protecting, unprotecting, and reprotecting data on a cloud platform.

Chapter 2

Overview of the Data Security Gateway (DSG)

The Data Security Gateway applies data security operations on the network and uses the CoP Profiles to configure a data security solution.

For information about the DSG, refer to the *DSG User Guide*.

The DSG Container offers a cloud standard form factor that can be deployed on Kubernetes.

For more information about the DSG Container, refer to the *DSG Immutable Policy User Guide* for the respective cloud platform.

The DSG Container supports autoscaling with dynamic policy updates.

2.1 Business Problem

This section identifies the problems that a company faces while protecting data in a containerized application on Cloud:

- *Dynamic policy updates* - The policy should be made available to the application containers as and when there is any update to the Protegrity security policy.
- *No downtime of application containers* - The Protegrity solution needs to ensure that there is no downtime for updating the policy. In case of a traditional Blue Green deployment, when you update the policy using the green deployment, you first have to shut down all the application containers that are using the old policy. You then have to bring up the new application containers that will use the updated policy.
- *Autoscaling* - The Protegrity solution must support autoscalability. When the application containers that are processing the data scale, the Protegrity data security components need to leverage the autoscaling capabilities of Kubernetes to autoscale along with the application containers.

2.2 Protegrity Solution

Protegrity offers a metadata based solution for deploying the DSG container. In this solution, the Protegrity PEP server is deployed in a containerized format. It downloads the latest policy package file from the ESA. A metadata file is created that identifies the latest policy package. Whenever there is a change to the policy package, the Protegrity solution compares the new policy package with the one identified by the metadata. The latest policy package file is then uploaded to the shared memory, which can be used by the DSG container.

This solution also ensures auto scaling. Whenever a new DSG container comes up, the Protegrity solution ensures that the Protegrity security policy is made available to this container.

For more information about the architecture of the Protegrity solution, refer to the section *General Architecture*.

Chapter 3

General Architecture

This section describes the basic architecture of the DSG Container on the Kubernetes cluster. The DSG Container on the Kubernetes cluster uses a new architecture to leverage the existing containerized form factor solution for the DSG.

For more information about the DSG, refer to the *DSG User Guide 3.1.0.5*.

For more information about the DSG Container, refer to the *DSG Immutable Policy User Guide 3.1.0.0* for the respective Cloud platform.

The DSG Metaserver-based deployment contains the following components:

Table 3-1: Components of the DSG Metaserver-based Deployment on the Kubernetes

Components	Pod	Function
PEP server (*pty-publisher-pepserver*) container	PTY-Publisher	The PEP server container runs the PEP server process, which connects to the ESA to fetch the deployed policy, encrypts it, and then stores it in the shared storage. The PEP server container also creates a *metadata.json* file corresponding to the ESA policy package stored in the *tmpfs* volume. The *metadata.json* file contains the metadata for the policy package. The PEP server container is part of the Publisher pod.
Meta server (*pty-publisher-metaserver*) container	PTY-Publisher	The Meta server container maintains the information about the latest policy package on the storage. The Meta server container reads the *metadata.json* file and responds to requests from the Subscriber-Sidecar container about updating the policy. The Meta server container is part of the Publisher pod.
Subscriber-Sidecar (*dsg-subscriber-sidecar*) container	DSG	The Subscriber-Sidecar container queries the Meta server container for the latest policy package information. Based on this information, it reads the policy from the storage. It decrypts the downloaded policy package and populates the shared memory. The policy is then used to perform the URP (Unprotect, Reprotect, and Protect) operation.
DSG Container	DSG	The DSG Container reads the policy from the shared memory to perform the URP operation.

Components	Pod	Function
		The DSG Container also reads the COP from the storage.

The following figure shows the general architecture of the DSG container on the Kubernetes cluster.

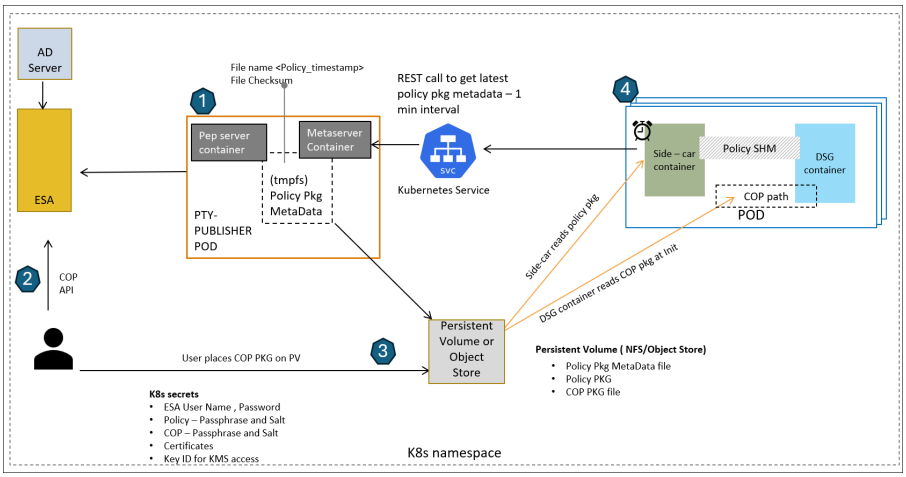


Figure 3-1: General Architecture - DSG Container on the Kubernetes Cluster

The following steps describe the general working of the DSG Container the Kubernetes cluster.

1. The user deploys the PTY-Publisher pod and adds the Kubernetes secrets for encrypting the policy package. The *pty-publisher-pepserver* container fetches the policy from the ESA using the *pty-secret-esa-cred* secret. It encrypts the package using the *pty-secret-pbe* or the Cloud KMS ID, and stores the package on persistent volume or object store.

   The *pty-publisher-pepserver* creates the *metadata.json* file using the latest policy from the ESA, and it is stored in *tmpfs*.

2. The user runs the COP API to create the encrypted COP package.
3. Copy the COP package to the persistent volume or object store.
4. Add the Password Based Encryption (PBE) secrets for COP decryption and deploy the DSG Container. The *pty-subscriber-sidecar* container is initialized.

   After a default interval of one minute, which is customizable, the *pty-subscriber-sidecar* container polls the *pty-publisher-metaserver* container using the endpoint of the Publisher service. This is used to fetch the latest metadata for the policy package from the storage.

   The *pty-subscriber-sidecar* container reads the latest ESA policy from the persistent volume or object store.

   The *pty-subscriber-sidecar* container decrypts the ESA policy using the *pty-secret-pbe* or the KMS Key ID and pushes it into the shared memory. This shared memory is accessed by the DSG Container.

   The decrypted ESA policy is used to perform the URP (Unprotect, Reprotect, and Protect) operations.

The following table describes the secrets used by the DSG Metaserver-based deployment on the Kubernetes cluster.

Table 3-2: Kubernetes Secrets

Parameter in the Values.yaml File	Description
pty-secret-esa-cred	Stores the credentials for the ESA user with the *Export Certificates* and *Appliance CLI Viewer* permissions.
pty-secret-pbe	Stores the salt and passphrase to perform the encrypt and decrypt operations on the ESA policy. It is also used to encrypt and decrypt the COP.
pty-secret-registry-cred	Stores the credentials for the Container registry to download the Container images.

Parameter in the Values.yaml File	Description
	This is Optional if the container registries are pre-integrated with the Kubernetes cluster.
pty-secret-tls-publisher	Stores the TLS server certificate for the Publisher pod. This certificate is used by the Meta server container in the Publisher pod to communicate with the Protegrity Sidecar container.
pty-secret-ca-cert	Stores the CA certificate of the Publisher pod. This certificate is used as the client certificate by the Protegrity Subscriber-Sidecar container to authenticate its communication with the Meta server container.

3.1 Workflow of the PEP Server Container

This section describes the workflow of the *pty-publisher-pepserver* container. The following figure shows the workflow of the *pty-publisher-pepserver* container.

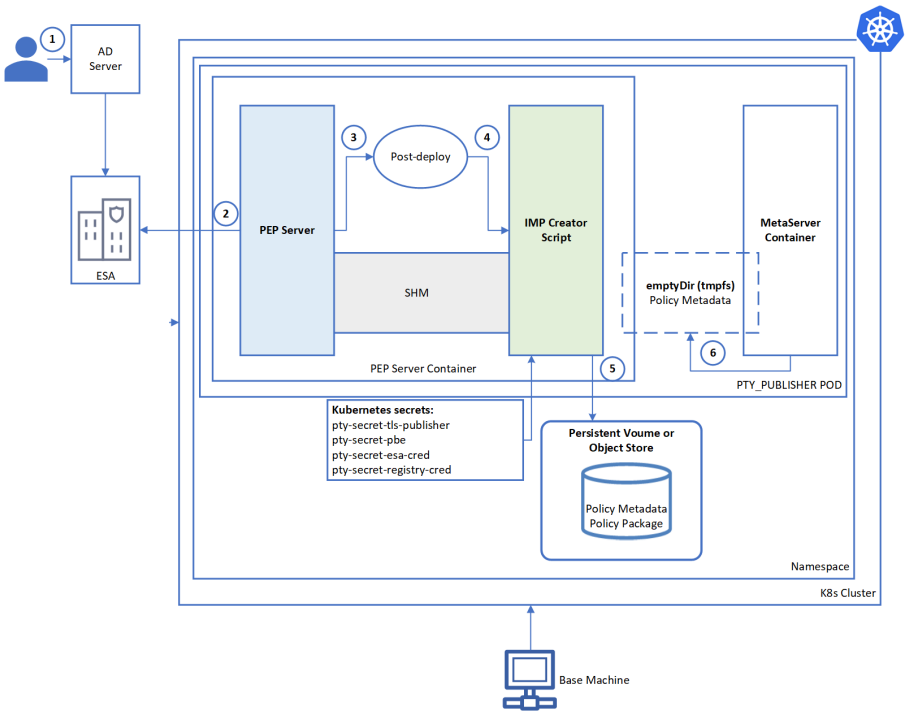


Figure 3-2: Workflow of the PEP Server Container

The *pty_publisher* pod handles the encrypted policy package creation process. It contains the following two containers:

- PEP Server (*pty-publisher-pepserver*) Container
- MetaServer (*pty-publisher-metaserver*) Container

The *pty-publisher-pepserver* container runs the standard PEP server process and connects it to the ESA.

The following steps describe the workflow of the *pty-publisher-pepserver* container.

1. The user creates the policy datastore in the ESA and specifies the IP address of the Node, where the Publisher pod is deployed, in the list of allowed servers.

   For more information about adding allowed servers to the data store, refer to the section *Adding Allowed Servers for the Data Store* in the *Policy Management Guide 9.1.0.5*.

2. The user adds the changes, such as changes to users in the policy, to the ESA.
3. The PEP server polls the ESA at an interval of 60 seconds. If the policy has been updated, then the PEP server pulls the policy from the ESA.

4. The PEP server updates the shared memory and calls the post-deploy script.

> **Note:** A new policy package is created every time the post-deploy script is triggered. The post-deploy script is triggered for every shared memory update.

5. The post-deploy script calls the IMP Creator script, which creates an encrypted policy using the *pty-secret-pbe* or the KMS secret. The PBE secret uses the PKCS5 Passphrase-Based Encryption (PBE) algorithm.
6. The encrypted policy is stored in the persistent volume or object store.
7. The post-deploy script also updates the *metadata.json* file as per the latest encrypted policy. A backup of the *metadata.json* file is also stored on the persistent volume or the object store.

3.2 Workflow of the Meta Server Container

This section describes the workflow of the *pty-publisher-metaserver* container. The following figure shows the workflow of the *pty-publisher-metaserver* container.

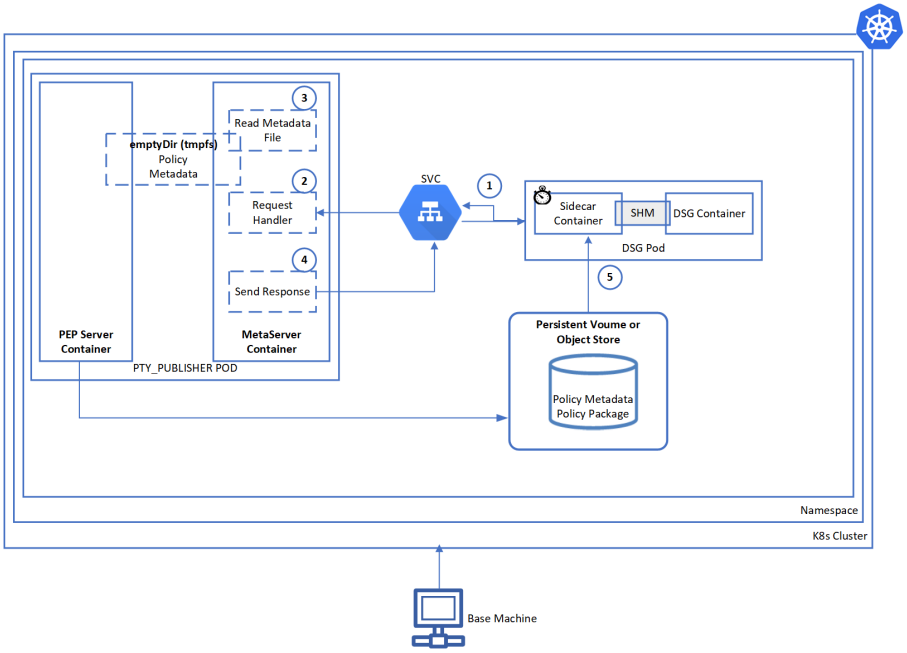


Figure 3-3: Workflow of the Meta server Container

The following steps describe the workflow of the *pty-publisher-metaserver* container.

1. The *pty-subscriber-sidecar* container initiates the following request to the *pty-publisher-metaserver* container.

```
curl https://<<service-name>>:<port number>/status --cacert certificate.pem
```

Example:

```
curl https://publisher-pty-publisher.test-demo:11443/status --cacert certificate.pem
```

> **Note:** The endpoint certificates are provided as a secret named *pty-secret-tls-publisher*.

2. The request handler receives the request.
3. The *pty-publisher-metaserver* container reads the *metadata.json* file from the emptyDir volume *tmpfs*.

4. The *pty-publisher-metaserver* container sends the response as content from the *metadata.json* file to the *pty-subscriber-sidecar* container.

```
{
  "version": "<metadata document version>",
  "name": "<policy package filename>",
  "volume": "<volume mount point where policy package is store>",
  "path": "<policy package path in volume mount>",
  "timestamp": "<policy package creation timestamp>",
  "hash_type": "<check sum alogorithm>",
  "hash": "<policy package checksum>",
  "kek": "<type of padding>"
}
```

Example:

```
{
    "version": "1.0.0",
    "name": "policy_2023_10_10_09_39_54.tgz",
    "volume": "/var/data/policy",
    "path": "demo/policy",
    "timestamp": "1696930795",
    "hash_type": "SHA256",
    "hash": "5dc449126a3f473f9e80240e8b1b008f55aa0d424caf1030a219b76855a6b2b7",
    "kek": "pkcs5"
}
```

The following table describes the parameters specified in the *metadata.json* file.

Table 3-3: Parameters in the metadata.json file

Parameter	Description
version	Version of the metadata document.
name	File name of the policy package.
volume	Mount point on the volume where the policy package is stored.
path	Path of the policy package on the volume mount.
timestamp	Timestamp when the policy package was created.
hash_type	Check sum algorithm for the encrypted policy package.
hash	Value of the policy package checksum.
kek	Encryption algorithm used for encrypting the policy package.

5. The *pty-subscriber-sidecar* container loads the latest ESA policy from the persistent volume or the object store, if the policy package is updated.

3.3 Workflow of the Subscriber-Sidecar Container

This section describes the workflow of the *dsg-subscriber-sidecar* container. The following figure shows the workflow of the *dsg-subscriber-sidecar* container.

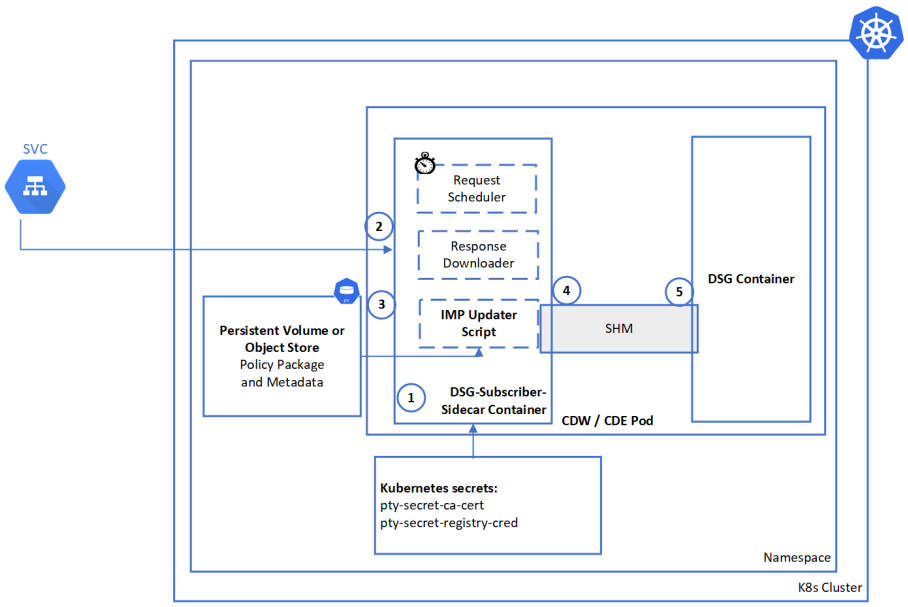


Figure 3-4: Workflow of the Subscriber-Sidecar Container

The following steps describes the workflow of the *dsg-subscriber-sidecar* container.

1. The *dsg-subscriber-sidecar* container is deployed in the target DSG pod when you deploy the DSG container. The *dsg-subscriber-sidecar* container performs the following tasks:
   - Queries the Meta server container in the publisher pod using the certificates specified in the secret *pty-secret-tls-publisher* at a specified *ping interval*.
   - Reads the JSON response of the policy package metadata and decrypts the latest policy package from the persistent volume or object store.
2. The *dsg-subscriber-sidecar* container invokes a REST API at regular intervals to get the policy status from the Meta server container in the *pty_publisher* pod. The *dsg-subscriber-sidecar* container stores the response from the Meta server container in its memory.
3. If the REST API is invoked for the first time, then the *dsg-subscriber-sidecar* retrieves the latest policy package based on the metadata JSON response.

   For subsequent API requests, the *dsg-subscriber-sidecar* container compares the newly obtained response with an existing response that it has fetched earlier. If both the responses are identical, then the *dsg-subscriber-sidecar* retains the existing policy package and does not retrieve a new policy package from the persistent volume or object store. If both the JSON responses are not identical, then the *dsg-subscriber-sidecar* container retrieves the latest policy package from the persistent volume or object store based on the new metadata JSON response.

> **Important:**
> If the metadata JSON is missing, then the *dsg-subscriber-sidecar* container cannot retrieve the latest policy.

4. The IMP Updater script decrypts the policy package from the persistent volume or object store using the *pty-secret-pbe* secret or the KMS and loads it to the shared memory of the DSG pod.

> **Important:**
> Before updating the policy package in the shared memory, the *dsg-subscriber-sidecar* container first calculates the checksum of the policy package. It then compares the checksum of the policy package with the checksum present in the metadata JSON response. If both the checksums do not match, then the *dsg-subscriber-sidecar* container does not update the policy package in the memory and instead returns an error. This error appears in the log file.

> If an existing policy is present in the shared memory, then the DSG Container continue to use this policy. The *dsg-subscriber-sidecar* container then waits for the next interval to invoke the REST API to get the policy status from the Meta server container.

5. The DSG Container executes the URP operations using this decrypted policy.

3.4 Workflow of the DSG Container Loading the COP

This section describes how the COP is loaded using the DSG Container.

The loading of the COP consists of the following two stages:

- Stage 1: Creating CoP and uploading it to an object storage or a persistent volume.
- Stage 2: Deploying the DSG Container Application on a Kubernetes cluster, which uses the CoP stored in stage 1.

The following figures illustrates the process for creating a CoP and uploading it to an object storage or a persistent volume.

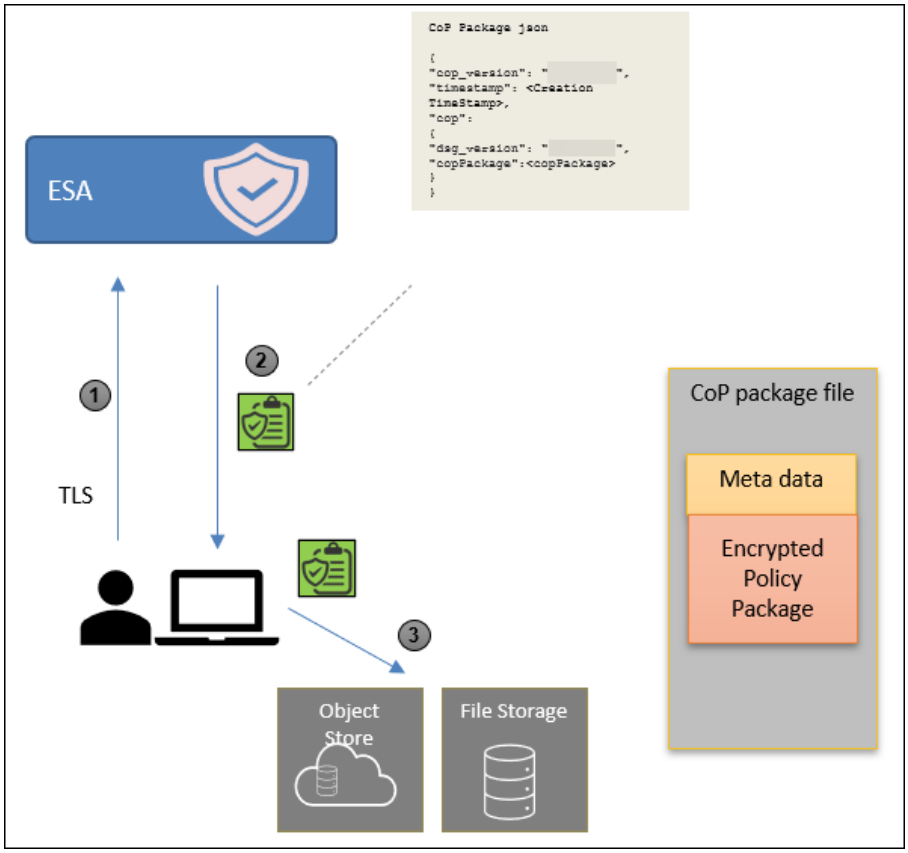


Figure 3-5: Creating CoP and uploading it to an object storage or persistent volume

Stage 1

1. The user first creates the CoP on the ESA.
2. The user then exports the CoP using the export API. The CoP package is encrypted only using the Password-based encryption (PBE), where a passphrase and salt is used to encrypt the CoP package.
3. The user copies the encrypted CoP package and policy to an object storage or a persistent volume.

Stage 2

The following steps describe the workflow of loading the COP to the DSG container on a Kubernetes cluster.

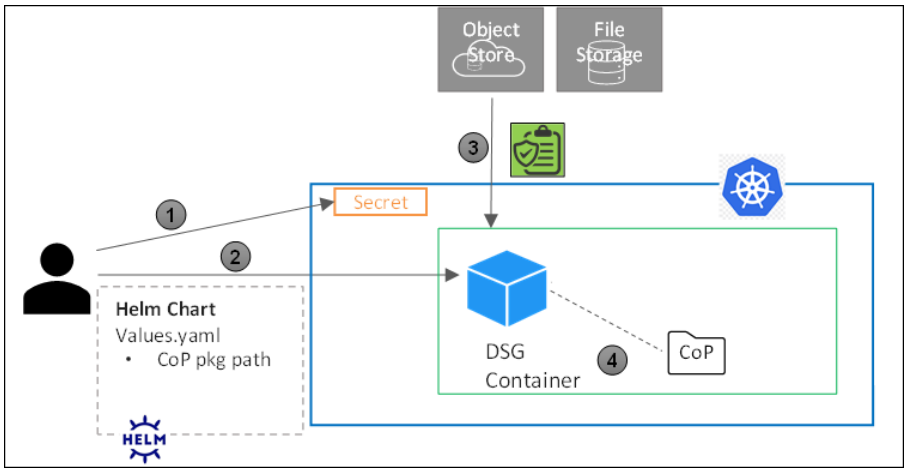


Figure 3-6: Loading the COP to the DSG Container

1. The user creates a Kubernetes secret that is used to decrypt the CoP package. This Kubernetes secret contains the Passphrase and salt for the CoP package:
2. The user runs the Helm chart to deploys the DSG container to the Kubernetes cluster.
3. The DSG container retrieves the CoP object storage or persistent volume, and decrypts the package using the secrets provided in the Kubernetes secrets.
4. The CoP package is extracted and the contents are stored in a specific directory on the Pod. The DSG service is started and it uses the CoP that is available in the Pod.

Chapter 4

Installing the DSG Container on Amazon Elastic Kubernetes Service (EKS)

This section describes the procedures to install the DSG Container on Amazon EKS.

The DSG Container will be used in combination with the Enterprise Security Administrator (ESA).

4.1 Verifying the Prerequisites

Ensure that the following prerequisites are met:

1. Verify the reference solution deployment
2. Verify the prerequisites for the reference application
3. Verify the prerequisites that are required on AWS

4.1.1 Verifying the Prerequisites for Reference Solution Deployment

Reference Solution Deployment

- *ESA* – Protegrity provides an AWS image *ESA_PAP-ALL-64_x86-64_AWS_9.1.0.5.xxxx*. Use this image to launch an ESA instance on AWS.
  - After launching the ESA instance, you must install the *ESA_PAP-ALL-64_x86-64_9.1.0.5.<Build_Number>.DSGUP.pty* patch to extend the DSG Web UI on the ESA.
  - Install the *DSG_PAP-ALL-64_x86-64_3.1.0.5.<Build_Number>.FE-1* patch to extend the DSG Web UI on the ESA.

  For more information about installing the DSG 3.1.0.5 FE-1 patch, refer to section *Installing the DSG* in the *Data Security Gateway Guide 3.1.0.5*.

- *DSG_RHUBI-ALL-64_x86-64_Generic.K8s_<Version>.tgz* – *Installation package* for the Protegrity Reference Deployment.

4.1.2 Verifying the Prerequisites for Reference Application

Reference Application

This section describes the prerequisites for using the reference application:

- Policy – Ensure that you have defined the security policy on the ESA. In addition, ensure that you fulfill either of the following requirements:
  - You must link the policy to the default data store in the ESA.
  - You must add the Kubernetes node or pod IP address ranges as allowed servers on the data store to which the policy is linked.
- Configuration over Programming (CoP) – The Protegrity Reference Deployment uses the default REST API Example as the CoP.
- Client application – For example, the banking application, which contains the customer data that you want to protect using the DSG application.

4.1.3 Verifying Prerequisites for AWS

The following additional prerequisites for AWS are required:

Table 4-1: AWS Prerequisites

Prerequisite	Description
Linux Instance	This instance can be used to communicate with the Kubernetes cluster. This instance can be on-premise or on AWS. Ensure that Helm is installed on this Linux instance. You can also choose to install Docker on this Linux instance to communicate with the Container Registry, where you want to upload the Docker images.
AWS account	Access to an AWS account
EFS-CSI	If you are using AWS EFS as the persistent volume, then install the latest version of the *EFS-CSI* driver. For more information about installing the EFS-CSI driver, refer to the *Amazon EFS CSI driver* documentation.
AWS Elastic File System (EFS) or AWS S3 bucket	This is the storage service where you upload the CoP package and the policy snapshot. **Important:** Ensure that the default encryption is enabled for this AWS S3 bucket.
Permissions	Ensure that the following permissions are available: • Creating a Kubernetes cluster • Creating a persistent volume and persistent volume claim in the Kubernetes cluster • Creating a bucket in AWS S3 • Creating an IAM policy • Custom IAM policy with AWS S3 read and write permissions
AmazonEKSClusterAutoscalerPolicy	This is a custom policy by AWS that must be created if you want to deploy the Cluster Autoscaler. For more information about this policy, refer to the section *Cluster Autoscaler* in the AWS documentation.

Prerequisite	Description
AmazonEKS_EFS_CSI_Driver_Policy	This is a custom policy by AWS that must be created if you want to use AWS EFS to store the policy. For more information about this policy, refer to the section *Amazon EFS CSI driver* in the AWS Documentation.
KMSDecryptAccess	This is a custom policy that allows the user to decrypt the policy packages that have been encrypted using the AWS Customer Master Key. The following action must be permitted on the IAM service: `kms:Decrypt`

IAM User 1	This is required to create the Kubernetes cluster. This user requires the following permissions:
	• *AmazonSSMFullAccess* - This is a managed policy on AWS
	• *AmazonEC2FullAccess* - This is a managed policy on AWS
	• *AmazonEKSClusterPolicy* - This is a managed policy on AWS
	• *AmazonEKSServicePolicy* - This is a managed policy on AWS
	• *AWSCloudFormationFullAccess* - This is a managed policy on AWS
	• Custom policy that allows the user to create a new role and an instance profile, retrieve information regarding a role and an instance profile, attach a policy to the specified IAM role, and so on. The following actions must be permitted on the IAM service:
	  • GetInstanceProfile
	  • GetRole
	  • AddRoleToInstanceProfile
	  • CreateInstanceProfile
	  • CreateRole
	  • PassRole
	  • AttachRolePolicy
	  • TagOpenIDConnectProvider
	  • TagRole
	• Custom policy that allows the user to delete a role and an instance profile, detach a policy from a specified role, delete a policy from the specified role, remove an IAM role from the specified EC2 instance profile, and so on. The following actions must be permitted on the IAM service:
	  • GetOpenIDConnectProvider
	  • CreateOpenIDConnectProvider
	  • DeleteInstanceProfile
	  • DeleteRole
	  • RemoveRoleFromInstanceProfile
	  • DeleteRolePolicy
	  • DetachRolePolicy
	  • PutRolePolicy
	• Custom policy that allows the user to manage EKS clusters. The following actions must be permitted on the EKS service:
	  • ListClusters
	  • ListNodegroups
	  • ListTagsForResource

Prerequisite	Description
	• ListUpdates • DescribeCluster • DescribeNodegroup • DescribeUpdate • CreateCluster • CreateNodegroup • DeleteCluster • DeleteNodegroup • TagResource • UpdateClusterConfig • UpdateClusterVersion • UpdateNodegroupConfig • UpdateNodegroupVersion • UntagResource
IAM user 2	This is required to upload the CoP and the policy packages to the S3 bucket. This user requires the *S3_PutObject* permissions, which is a managed policy by AWS, to write data into the S3 bucket.
IAM user 3	This is required to create an AWS Cognito User Pool. For more information about the permissions required to create an AWS Cognito User Pool, refer to the section *Resource Permissions* in the AWS Cognito documentation.
Amazon Elastic Kubernetes Service (EKS)	This is required to provide access to Amazon Elastic Kubernetes Service (EKS) to create a Kubernetes cluster.
AWS Key Management Service (KMS)	This is required to provide access to AWS Key Management Service to create a key used in the encryption and decryption operations.
AWS Elastic Container Registry (ECR)	This is required provide access to AWS Elastic Container Registry (ECR) to upload the DSG Container image.

4.1.4 Preliminary Steps

The section describes the basic steps required to set up the DSG container deployment on the EKS platform.

To begin:

1. Create a role with *Export Certificates* and *Appliance CLI Viewer* permissions.
2. Create an ESA user with the given role.

> **Note:** The user credentials are used by the Policy Enforcement Server to connect to the ESA and fetch the policy.

> **Note:** For more information about creating a ESA user with the *Export Certificates* and *Appliance CLI Viewer* permissions, refer to the section *Managing Roles* in the *Appliances Overview Guide 9.1.0.5*.

The following parameters are configured for the DSG container on the EKS Platform.

- Linux instance with connectivity to the EKS Cluster
- EKS Cluster configuration
- Amazon EFS, NFS server creation, or AWS S3 Bucket configuration

4.1.4.1 Initializing the Linux Instance

After downloading the installation packages, you must initialize the Linux instance. This involves installing the tools and utilities in the order as shown in the following table.

Table 4-2:

Order	Tools Utilities	Description	Notes	References
1	AWS CLI	Provides a set of command line tools for the AWS Cloud Platform.	Configure AWS CLI on your machine by running the following command. `aws configure` You must specify the credentials of IAM User 1 to allow the user to create the Kubernetes cluster. For more information about the IAM User 1, refer to the section *Verifying Prerequisites on AWS*.	For more information about installing AWS CLI on the Linux instance, refer to *https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html*.
2	*eksctl*	Command line utility to create and manage Kubernetes clusters on Amazon Elastic Kubernetes Service (Amazon EKS).		For more information about installing *eksctl* on the Linux instance, refer to *Using eksctl*.
3	*kubectl*	Command line interface for Kubernetes. Kubectl enables you to run commands from the Linux instance so that you can communicate with the Kubernetes cluster.		For more information about installing *kubectl*, refer to the section *Installing kubectl*.
4	*aws-iam-authenticator*	Tool that uses your IAM credentials to authenticate to your Kubernetes cluster.		For more information about installing *aws-iam-authenticator*, refer to the *Installing aws-iam-authenticator* section in the AWS documentation.
5	Helm setup		Run the following commands sequentially to install the Helm client on the Linux instance. 1. `curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3`	For more information about installing a Helm client, refer to *https://helm.sh/docs/using_helm/#installing-helm*. **Important:** Verify the version of the Helm client that must be used from the Readme provided with this build.

Order	Tools Utilities	Description	Notes	References
			2. `chmod 700 get_helm.sh` 3. `./get_helm.sh`	

4.1.4.2 Logging in to the AWS Environment

This section describes how you can login to the AWS environment.

➤ To login to the AWS environment:

1. Access AWS at the following URL:
   https://aws.amazon.com/

   The AWS home screen appears.

2. Click **Sign In to the Console**.
   The *Sign in* screen appears.

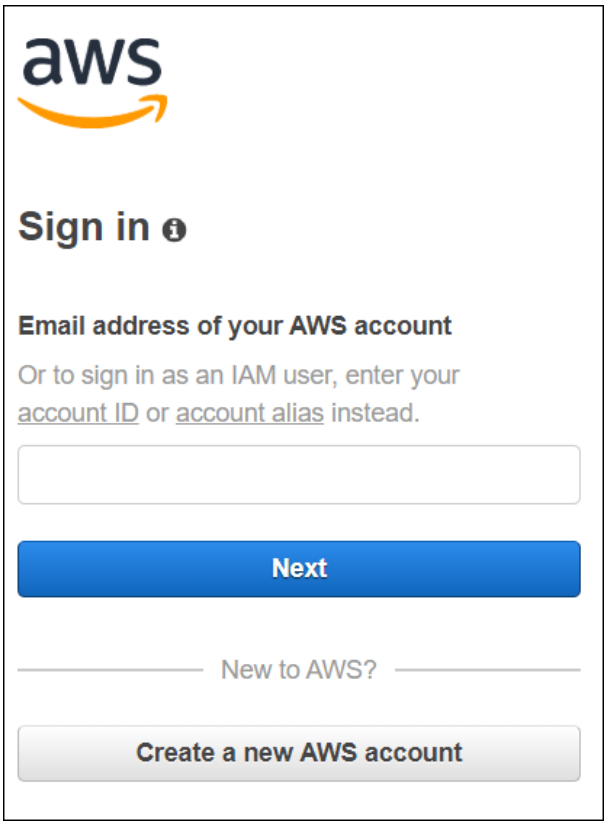


Figure 4-1: AWS Sign in screen

3. Enter the account ID or alias that has been provided to you by your administrator, and then click **Next**.
   A screen appears that prompts you to enter the IAM user credentials for signing into the AWS Management Console.

4. Enter the following details:
   - Identity and Access Management (IAM) user name
   - Password

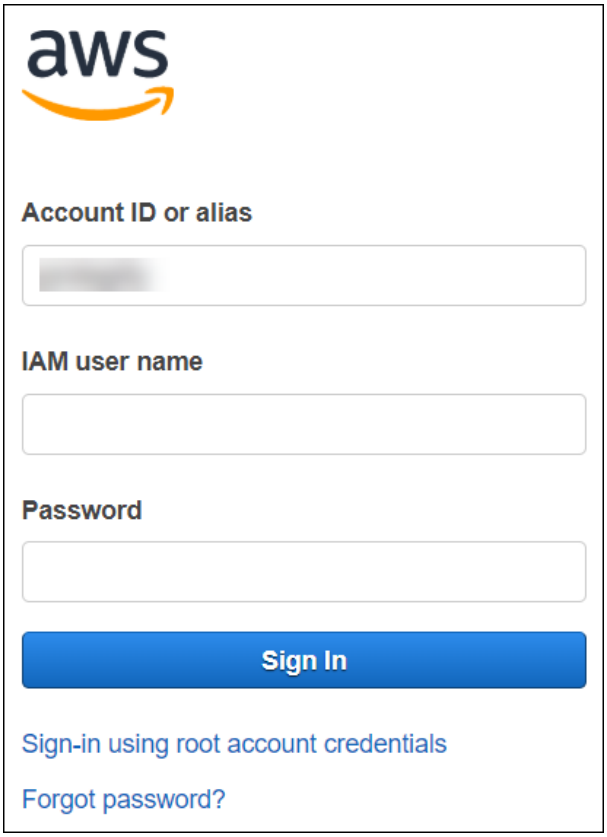


Figure 4-2: AWS login screen

5. Click **Sign in**.

   After successful authentication, the *AWS Management Console* screen appears.

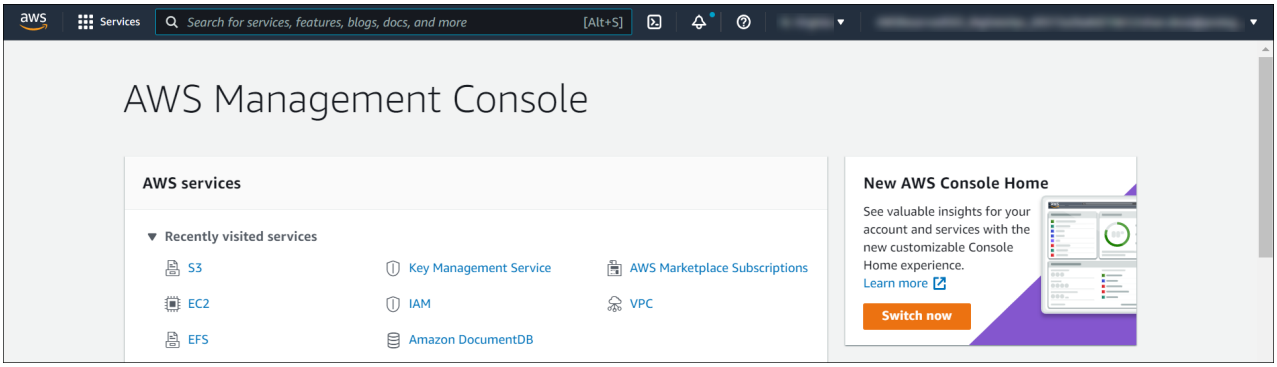


Figure 4-3: AWS Management Console Screen

4.1.4.3 Creating a Key Pair for the EC2 Instance

This section describes how to create a key pair for the EC2 instance on which you want to create the Kubernetes cluster.

➤ To create a key pair for the EC2 instance:

1. Login to the AWS environment.
   For more information about logging in to the AWS environment, refer to the section *Logging in to the AWS Environment*.
2. Navigate to **Services.**
   A list of AWS services appears.
3. In **Compute**, click **EC2**.
   The **EC2** console opens. By default, the **EC2 Dashboard** screen appears.
4. On the left pane, click **NETWORK & SECURITY** > **Key Pairs**.
   The **Key pairs** screen appears.
5. Click **Create key pair**.
   The **Create key pair** screen appears.
6. In the **Name** field, enter a name for the key pair.
   After the key pair is created, you need to specify the key pair name in the *publicKeyName* field of the *Sample_App_EKSCreateCluster.yaml* file, for creating a Kubernetes cluster.

   For more information about creating a cluster, refer to the section *Creating a Kubernetes Cluster*.
7. In the **Key pair type** field, select *RSA*.
8. In the **Private key file format** field, specify the format as *pem* for use with OpenSSH.
9. Click **Create key pair**.
   The **Key pairs** screen appears, and the following message appears on the screen.

```
Successfully created key pair
```

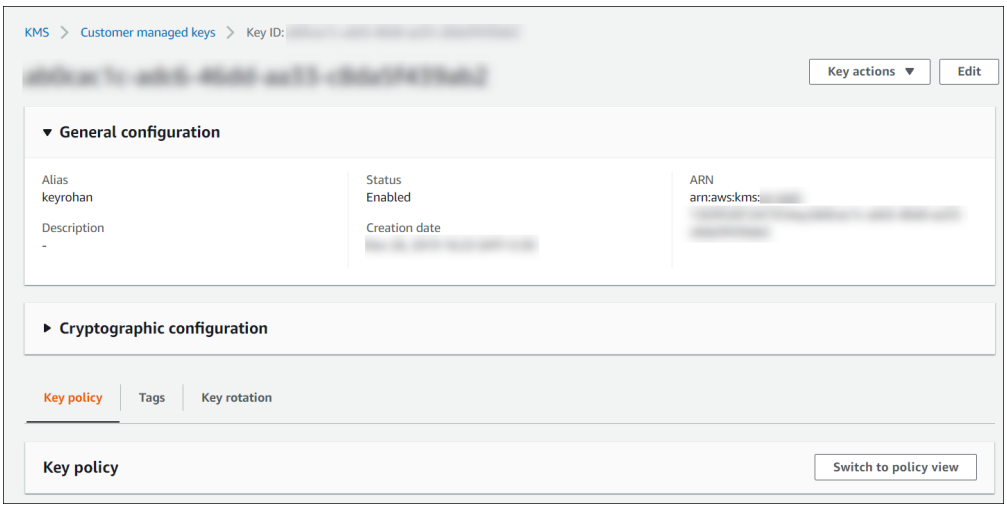
4.1.4.4 Creating an AWS Customer Master Key

This section describes how to create the customer master AWS key.

> **Note:** This section is applicable only if you want to use the AWS key to encrypt the policy package. Alternatively, you can use *PBE* to encrypt the policy package.

➤ To create an AWS customer master key:

1. Create an AWS Customer Master Key in the AWS KMS consolde.
   For more information regarding creating an AWS Customer Master Key, refer to the section *Creating an Asymmetric KMS key* in the AWS documentation.
2. After creating Customer Master Key, click the key alias.
   A screen specifying the configuration for the selected key appears.



3. In the **General Configuration** section, copy the value specified in the **ARN** field, and save it on your local machine.

   You need to attach the key to the *KMS-encrypt-decrypt* policy. You also need to specify this ARN value in the command for creating a Kubernetes secret for the key.

4. Navigate to **Services** > **IAM**.
5. Click **Policies**.

   The **Policies** screen appears.
6. Select the **KMSDecryptAccess** policy.

   The **Permissions** tab appears.
7. Click **Edit policy** to edit the policy in JSON format.
8. Modify the policy to add the ARN of the key that you have copied in *step 15* to the Resource parameter.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:Encrypt"
            ],
            "Resource": [
                "<ARN of the AWS Customer Master Key>"
            ]
        }
    ]
}
```

9. Click **Review policy**, and then click **Save changes** to save the changes to the policy.

4.1.4.5 Creating a NFS Server

This section describes the steps to create a NFS server. This procedure is optional and is required only if you want to use NFS server for storing the policy package, instead of AWS S3.

➤ To create an NFS server, perform the following steps:

1. Create an EC2 instance on the AWS with connectivity to the EKS.

   > **Note:** For more information about creating an EC2 instance on the AWS, refer to the *Amazon documentation*

2. Update the apt repository using the following command.

   ```
   sudo apt update
   ```

3. Install the NFS server using the following command.

   ```
   sudo apt install nfs-kernel-server
   ```

4. Create a directory using the following command.

   ```
   sudo mkdir -p /demo/policy
   ```

   > **Note:** The directory is used as an NFS share.

5. Edit the */etc/exports* file using the following command.

   ```
   sudo nano /etc/exports
   ```

6. Set the directory as a shared directory in the NFS server using the following step.

   ```
   /demo/policy *(rw,root_squash)
   ```

7. Save and exit the */etc/exports* file.
8. Restart the NFS Kernel server using the following command.

   ```
   sudo systemctl restart nfs-kernel-server
   ```

9. Export the *demo/policy* file using the following command.

   ```
   sudo exportfs -v
   ```

To validate whether the folder is mounted on all the exported mount location, use the following command.

```
showmount -e <IPaddress>
```

The following output appears when you execute the command.

```
Export list for <IPaddress>: /demo/policy *
```

> **Note:**
> The parameters mentioned in the <> are generic variables and you must provide a valid input for the variables.

4.1.4.6 Creating an AWS Elastic File System (EFS)

This section describes how to create an AWS Elastic File System (EFS).

> **Important:** This procedure is optional and is required only if you want to use AWS EFS for storing the policy snapshot, instead of AWS S3.

To create an AWS EFS:

1. Login to the AWS environment.
   For more information about logging in to the AWS environment, refer to the section *Logging in to the AWS Environment*.
2. Navigate to **Services.**
   A list of AWS services appears.
3. In **Storage**, click **EFS**.
   The **File Systems** screen appears.
4. Click **Create file system**.
   The **Configure network access** screen appears.
5. In the **VPC** list, select the VPC where you will be creating the Kubernetes cluster.
6. Click **Next Step**.
   The **Configure file system settings** screen appears.
7. Click **Next Step**.
   The **Configure client access** screen appears.
8. Click **Next Step**.
   The **Review and create** screen appears.
9. Click **Create File System**.
   The file system is created.

   Note the value in the **File System ID** column. You need to specify this value as the value of the *volumeHandle* parameter in the *pv.yaml* file in *step 1c* of the section *Deploying a Release on the Kubernetes Cluster*.

4.1.4.7 Creating an AWS S3 Bucket

This section describes how to create an AWS S3 bucket.

Important: This procedure is optional and is required only if you want to use AWS S3 for storing the policy package, instead of NFS Server.

To create an AWS S3 bucket:

1. Login to the AWS environment.
   For more information about logging in to the AWS environment, refer to the *Amazon documentation*.
2. Navigate to **Services.**
   A list of AWS services appears.
3. In **Storage**, click **S3**.
   The **S3 buckets** screen appears.
4. Click **Create bucket**.
   The **Create bucket** screen appears.
5. In the **General configuration** screen, specify the following details.
   a. In the **Bucket name** field, enter a unique name for the bucket.

b. In the **AWS Region** field, choose the same region in which you want to create your EC2 instance.

If you want to configure your bucket or set any specific permissions, then you can specify the required values in the remaining sections of the screen. Otherwise, you can directly go to the next step to create a bucket.

6. Click **Create bucket**.

The bucket is created.

4.1.4.8 Creating a Kubernetes Cluster

This section describes how to create a Kubernetes Cluster on Amazon Elastic Kubernetes Service (EKS) using *eksctl*, which is a command line tool for creating clusters.

Note: The steps listed in this section for creating a Kubernetes cluster are for reference use. If you have an existing Kubernetes cluster or want to create a Kubernetes cluster based on your own requirements, then you can directly navigate to *step 3* to connect your Kubernetes cluster and the Linux instance. However, you must ensure that your DSG service port is enabled on the Network Security group of your VPC.

Important: If you have an existing Kubernetes cluster or want to create a Kubernetes cluster using a different method, then you must install the Kubernetes Metrics Server and Cluster Autoscaler before deploying the Release.

➤ To create a Kubernetes cluster:

1. Login to the Linux instance, and use the file named *Sample_App_EKSCreateCluster.yaml extracted from the DSG-Samples_Linux-ALL-ALL_x86-64_<Version>.tgz package* to specify the configurations for creating the Kubernetes cluster.

   Modify the following *Sample_App_EKSCreateCluster.yaml* snippet.

```
 apiVersion: eksctl.io/v1alpha5
 kind: ClusterConfig
 metadata:
   name: <Name of your Kubernetes cluster>
   region: <Region where you want to deploy your Kubernetes cluster>
   version: "<Kubernetes Version>"
 vpc:
   id: "<ID of the VPC where you want to deploy the Kubernetes cluster>"
   subnets:
     private:
       <Availability zone for the region where you want to deploy your Kubernetes
cluster>:
         id:  "<Subnet ID>"
       <Availability zone for the region where you want to deploy your Kubernetes
cluster>:
         id:  "<Subnet ID>"
 nodeGroups:
   - name: <Name of your Node Group>
     instanceType: m5.large
     minSize: 1
     maxSize: 3
     tags:
       k8s.io/cluster-autoscaler/enabled: "true"
       k8s.io/cluster-autoscaler/<Name of your Kubernetes cluster>: "owned"
     privateNetworking: true
     securityGroups:
       withShared: true
       withLocal: true
       attachIDs: ['<Security group linked to your VPC>']
     ssh:
      publicKeyName: '<EC2 keypair>'
     iam:
```

```
      attachPolicyARNs:
        - "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
      withAddonPolicies:
        albIngress: true
        autoScaler: true
```

Note: If you want to copy the contents of the *Sample_App_EKSCreateCluster.yaml* file, then ensure that you indent the file as per YAML requirements.

For more information about the sample configuration file used to create a Kubernetes cluster, refer to the section *Using Config Files* in the eksctl documentation.

In the *ssh/publicKeyName* parameter, you must specify the value of the key pair that you have created in the section *Creating a Key Pair for the EC2 Instance*.

In the *iam/attachPolicyARNs* parameter, you must specify the following policy ARNs:

- ARN of the *AmazonEKS_CNI_Policy* policy - This is a default AWS policy that enables the Amazon VPC CNI Plugin to modify the IP address configuration on your EKS nodes.

  For more information about this policy, refer to the *AWS documentation*.

  You need to sign in to your AWS account to access the AWS documentation for this policy.

The content snippet displays the reference configuration required to create a Kubernetes cluster using a private VPC. If you want to use a different configuration for creating your Kubernetes cluster, then you need to refer to the section *Creating and managing clusters* in the eksctl documentation.

For more information about creating a configuration file to create a Kubernetes cluster, refer to the section *Creating and managing clusters* in the eksctl documentation.

2. Run the following command to create a Kubernetes cluster.

   ```
   eksctl create cluster -f ./Sample_App_EKSCreateCluster.yaml
   ```

   Important: IAM User 1, who creates the Kubernetes cluster, is automatically assigned the *cluster-admin* role in Kubernetes.

3. Run the following command to connect your Linux instance to the Kubernetes cluster.

   ```
   aws eks update-kubeconfig --name <Name of Kubernetes cluster>
   ```

4. Validate whether the cluster is up by running the following command.

   ```
   kubectl get nodes
   ```

   The command lists the Kubernetes nodes available in your cluster that you have created using the *eksctl* command.

5. Deploy the Cluster Autoscaler component to enable the autoscaling of nodes in the EKS cluster.

   For more information about deploying the Cluster Autoscaler, refer to the section *Deploy the Cluster Autoscaler* in the Amazon EKS documentation.

6. Install the Metrics Server to enable the horizontal autoscaling of pods in the Kubernetes cluster.

   For more information about installing the Metrics Server, refer to the section *Horizontal Pod Autoscaler* in the *Amazon EKS* documentation.

7. If you are using a private VPC, then you need to perform the following steps to use the Metrics Server. The Metric Server is used to communicate with the pods on their internal IP addresses and internal DNS to retrieve the pod metrics.

   a. Run the following command.

   ```
   kubectl edit deployment metrics-server -n kube-system
   ```

b. Scroll down to the container arguments and update the value of the *kubelet-preferred-address-type* argument.

```
- --kubelet-preferred-address-
types=InternalIP,Hostname,InternalDNS,ExternalDNS,ExternalIP
```

The following snippet shows the modified *kubec-let-preferred-address-type* argument.

```
spec:
      containers:
      - args:
        - --cert-dir=/tmp
        - --secure-port=4443
        - --kubelet-preferred-address-
types=InternalIP,Hostname,InternalDNS,ExternalDNS,ExternalIP
        - --kubelet-use-node-status-port
```

8. Run following commands to tag the cluster subnets to ensure that the Elastic load balancer can discover them.
   - ***aws ec2 create-tags --tags Key=kubernetes.io/cluster/<Cluster Name>,Value=shared --resources <Subnet ID>***
   - ***aws ec2 create-tags --tags Key=kubernetes.io/role/internal-elb,Value=1 --resources <Subnet ID>***
   - ***aws ec2 create-tags --tags Key=kubernetes.io/role/elb,Value=1 --resources <Subnet ID>***

   For example:

   - ***aws ec2 create-tags --tags Key=kubernetes.io/cluster/new-test,Value=shared --resources subnet-0b7db183ae643a743***
   - ***aws ec2 create-tags --tags Key=kubernetes.io/role/internal-elb,Value=1 --resources subnet-0b7db183ae643a743***
   - ***aws ec2 create-tags --tags Key=kubernetes.io/role/elb,Value=1 --resources subnet-0b7db183ae643a743***

   Repeat this step for all the cluster subnets.

4.2 Extracting the Package

This section provides information about extracting the installation package to verify that the individual components that are essential for the DSG Container installation are available.

1. Create a directory to download the package using the following command.

```
mkdir packaging
```

2. Navigate to the directory *packaging* using the following command.

```
cd packaging
```

3. Download the *DSG_RHUBI-ALL-64_x86-64_Generic.K8s_<Version>.tgz* package on the base machine.
4. Extract the files from the package using the following command.

```
tar -xvf DSG_RHUBI-ALL-64_x86-64_Generic.K8s_<Version>.tgz
```

The following is the list of the packages that are extracted:

- Helm charts and container images
- Additional packages and directories

Table 4-3: List of the Helm Charts and Container Images

List of Helm Charts and Docker Images	Description	Package Name
DSG container image	It is used to create the DSG container.	*DSG_K8S-ALL-64_x86-64_<Version>.tar.gz*
Sidecar image	It is used to create the Sidecar container.	*SUBSCRIBER-SIDECAR_RHUBI-8-64_x86-64_K8S_<Version>.tar.gz*
Helm Charts for the DSG Container	It is a package which contains the Helm Charts for deploying the Webhook container.	*DSG-HELM-DYNAMIC_ALL-ALL-ALL_x86-64_K8S_<Version>.tgz*
Meta server image	It is used to create the MetaServer container.	*PUBLISHER-METASERVER_RHUBI-8-64_x86-64_K8S_<Version>.tar.gz*
PEP server image	It is used to create the PEP Server container.	*PUBLISHER-PEPSERVER_RHUBI-8-64_x86-64_K8S_<Version>.tar.gz*
Helm Charts for the Meta server and PEP server	It is a package that contains the Helm Charts used to deploy the MetaServer and PEP Server containers.	*PUBLISHER-HELM_ALL-ALL-ALL_x86-64_K8S_<Version>.tgz*
Source package for the PEP server container	It is a package containing the Dockerfile that can be used to create a custom image for the PEP server container and the associated binary files.	*PUBLISHER-PEPSERVER_SRC_<Version>.tgz* For more information about building a custom image using the Dockerfile, refer to the section *Creating Custom Images Using Dockerfile*.
Source package for the Meta server container	It is a package containing the Dockerfile that can be used to create a custom image for the Meta server container and the associated binary files.	*PUBLISHER_METASERVER_SRC_<Version>.tgz* For more information about building a custom image using the Dockerfile, refer to the section *Creating Custom Images Using Dockerfile*.
Source package for the DSG containers	It is a package containing the Dockerfiles that can be used to create a custom image for the DSG containers and the associated binary files.	*DSG_K8S-ALL-64_x86-64_SRC_<Version>.tgz* For more information about building a custom image using the Dockerfile, refer to the section *Creating Custom Images Using Dockerfile*.
Source package for the Sidecar containers	It is a package containing the Dockerfiles that can be used to create a custom image for the Sidecar containers and the associated binary files.	*SIDECAR_SRC_<Version>.tgz* For more information about building a custom image using the Dockerfile, refer to the section *Creating Custom Images Using Dockerfile*.
HOW-TO-BUILD-DOCKER-IMAGES	Text file specifying how to build the Docker images.	

Table 4-4: List of the Additional Packages and Directories

List of Additional Packages and Directories	Description
DSG-Samples_Linux-ALL-ALL_x86-64_<Version>.tgz	It includes the following files and directories: • *Sample_App_COP_V5.zip* - Sample COP package. • *Sample_App_Policy_V5.tgz* - Sample policy. • *Sample_App_EKSCreateCluster.yaml* - Sample YAML file used to create the Kubernetes cluster.

List of Additional Packages and Directories	Description
certs	Contains the *cnf* file to create the certificates required for secure communication.

4.3 Uploading the Images to the Container Repository

This section describes how you can upload the PEP server, Meta server, DSG, and Sidecar container images to the Container Repository.

Before you begin
Ensure that you have set up your Container Registry.

> **Note:** The steps listed in this section for uploading the container images to Amazon Elastic Container Registry (ECR) are for reference use. You can choose to use a different Container Registry for uploading the container images.

➤ To upload the PEP server, Meta server, DSG, and Sidecar container images:

1. Verify whether Docker is up and running on the Linux instance by running the following command.
   systemctl status docker
   If Docker is stopped, then run the following command to start it.
   systemctl start docker
2. Run the following command to authenticate your Docker client to Amazon ECR.
   aws ecr get-login-password --region ${AWS_REGION} | docker login --username AWS --password-stdin ${AWS_ECR_REGISTRY_ADDRESS}
   For example:

```
export AWS_REGION=us-east-1
export AWS_ECR_REGISTRY_ADDRESS=<aws_account_id>.dkr.ecr.${AWS_REGION}.amazonaws.com

aws ecr get-login-password --region ${AWS_REGION} | docker login --username AWS --
password-stdin ${AWS_ECR_REGISTRY_ADDRESS}
```

   The following output appears.

```
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

   The Base64 encoded Docker credentials are stored in the *config.json* file.

   Note down the path where you save the *config.json* file. You need to specify the path in *Step c > 3*, where you create the Kubernetes Secret to store the credentials needed to pull the Docker images from your image repository.

   For more information about authenticating your Docker client to Amazon ECR, refer to the *AWS CLI Command Reference* documentation.

3. Upload the PEP Server container image to Amazon ECR by performing the following steps.
   a. Load the docker image using the following command.

```
docker load -i PUBLISHER-PEPSERVER_RHUBI-9-64_x86-64_K8S_<<Version>>.tar.gz
```

b. Tag the image to the Amazon ECR by running the following command.

```
docker tag <Container image>:<Tag> <Container registry path>/<Container image>:<Tag>
```

For example:

```
export AWS_REGION=us-east-1
export AWS_ECR_REGISTRY_ADDRESS=<aws_account_id>.dkr.ecr.${AWS_REGION}.amazonaws.com

docker tag <<LOADED_IMAGE>>:<<TAG>> ${AWS_ECR_REGISTRY_ADDRESS}/demo/publisher-pepserver:<<TAG>>
```

c. Push the tagged image to the Amazon ECR by running the following command.

```
docker push <Container registry path>/<Container image>:<Tag>
```

For example:

```
docker push <aws_account_id>.dkr.ecr.us-east-1.amazonaws.com/demo/publisher-pepserver:<<TAG>>
```

For more information about pushing a tagged image, refer to the *AWS* documentation.

4. Repeat *step 3* for pushing the Meta server, DSG, and Sidecar container images to the Amazon ECR.
5. Navigate to the directory where you have extracted the Helm charts packages.
6. In the *values.yaml* file in the *pty-publisher* directory, update the required repository path and tag for the following entries:
   - PEP server image in the *pepServerImage* setting.
   - Meta server image in the *metaServerImage* setting.
7. Navigate to the directory where you have extracted the Helm charts packages for DSG container.
8. In the *values.yaml* file in the *dsg*, update the required repository path and tag for the following entries:
   - DSG image in the *dsgImage* setting.
   - Sidecar image in the *sidecarImage* setting.

4.4 Setting Up the Certificates

This section describes how you can set up the CA and TLS certificates for self-signed secure communication and communication between the Metserver and Sidecar.

To set up certificates for secure communication:

1. Navigate to the *packaging/certs* directory using the following command.

```
cd packaging/certs
```

The certs directory contains the following files:

- *example_ca.cnf* - File used to create the CA certificate
- *example_svc.conf* - File used to create the TLS certificate
- *README* - Readme file listing the commands to be executed for generating the CA and TLS certificates

2. Run the following command to create the CA certificate.

```
openssl req -nodes -new -x509 -sha256 -newkey rsa:4096 -keyout ca_key.pem -out ca.pem \ -days 1825 -config example_ca.cnf
```

The following files are created in the *certs* directory.

- *ca.pem*
- *ca_key.pem*

The CA certificate is used to generate the TLS certificate in *step 3c*.

The CA certificate is also used by the sidecar container to communicate with the PTY-Publisher pod.

3. Perform the following steps to generate the TLS certificate that can be used in both the Publisher deployments.
   a. Create a copy of the *example_svc.conf* file using the following command.
   ```
   cp example_svc.conf example_svc_1.conf
   ```
   b. Edit the *example_svc_1.conf* file to update the hostname of the Publisher service that you want to deploy.
   ```
   [alt_names]
   DNS.1 = $PUBLISHER_METASERVER_SVC_HOSTNAME
   ```

   Important:
   The Publisher Service Hostname is not available until you deploy the Publisher Helm chart. It is formed from the following three components.

   - The Helm installation name.
   - Helm chart name.
   - The Kubernetes namespace, where you want to install the Helm chart.

   In this scenario, the TLS certificate is created before the Publisher Helm chart is deployed or the Publisher Service Hostname is available.

   You can obtain the host name using the following method:

   Hostname = ${HELM_INSTALLATION_NAME}-${HELM_CHART_NAME}.${NAMESPACE}.svc

   Hostname = ${SERVICE_NAME}.${NAMESPACE}.svc

   Note:
   You need to identify the name that you are planning to use for the Publisher deployments, before the actual deployment.

   For example:

   ```
   export PTY_PUBLISHER_HELM_INSTALL_NAME=publisher
   export PTY_PUBLISHER_HELM_CHART_NAME=pty-publisher
   export WAREHOUSE_NAMESPACE=test_demo
   export PUBLISHER_METASERVER_SVC_HOSTNAME = ${PTY_PUBLISHER_HELM_INSTALL_NAME}-$
   {PTY_PUBLISHER_HELM_CHART_NAME}.${WAREHOUSE_NAMESPACE}.svc
   ```

   ```
   echo $PUBLISHER_METASERVER_SVC_HOSTNAME

   publisher-pty-publisher.test_demo.svc
   ```

   c. Run the following command to create the TLS certificate.
   ```
   openssl req -x509 -nodes -newkey rsa:4096 -days 1825 -CA ca.pem -CAkey ca_key.pem
   \ -keyout key.pem -out certificate.pem -config example_svc_1.cnf
   ```

The *certificate.pem* and *key.pem* files are available in the *certs* directory.

4.5 Creating and Uploading the CoP to AWS Storage

This section describes how you can create a Ruleset on the ESA. In addition, this section describes how you can export a CoP from the ESA.

Important: Disable the Learn Mode on the DSG as it is not applicable to the containerized application.

4.5.1 Creating and Exporting the Ruleset

This section also describes how you can create a Ruleset on the ESA.

To create a Ruleset:

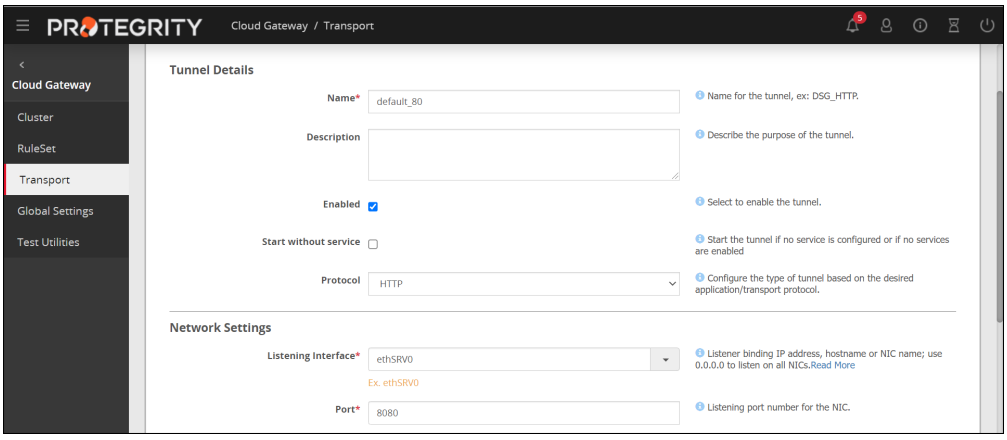
1. On the ESA Web UI, create a Configuration Over Programming (CoP) Ruleset.
   By default, the ESA provides you with default ruleset *REST API Examples* for testing the DSG functionality.

   For more information about creating a CoP Ruleset, refer to the section *Ruleset Menu* in the *Data Security Gateway User Guide 3.1.0.5*.

2. Navigate to **ESA Web UI** > **Cloud Gateway** > **Transport** > **Tunnels** > **Create Tunnel**.
   For more information about creating a tunnel, refer to the section *Tunnels tab* in the *Data Security Gateway User Guide 3.1.0.5*.

   Important:
   Create a tunnel with port number greater than *1024* as the Docker containers do not allow any external communication on port numbers lower than *1024*.

   The following figure shows a sample tunnel that uses the HTTP protocol on port *8080* using the service interface *ethSRV0*.



Note:

Ensure that any other tunnel that uses a port number lower than *1024* is disabled.

Note:
The NFS/CIFS tunnel and GPG transformations are not supported in the DSG containers.

Important:
Ensure that the port number that you specify in the **Port** field matches the value that you specify in the *targetPort* parameter in the *values.yaml* file under the **dsgService** > **tunnels** section.

For more information about the *values.yaml* file parameters, refer to the section *Values.yaml*.

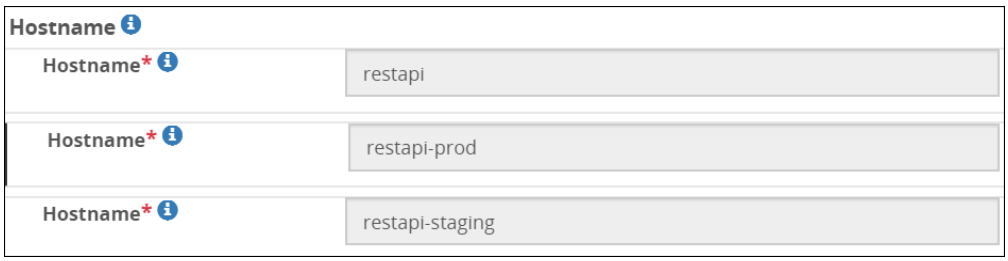
3. If you want to use the transaction metrics, then in the **Advanced Settings** section, specify the following value to support the X-Forwarded-For (XFF) header in the REST API requests sent to the DSG instances.

```
{
"xheaders": true
}
```

Figure 4-4: Advanced Settings

The XFF header is used to identify the client machine from which the REST API requests originate.

4. Click **Create** to create the tunnel.
5. On the ESA Web UI, navigate to **Cloud Gateway** > **RuleSet**.
   By default, the DSG provides you default rulesets for testing the DSG functionality.
6. If you want to specify multiple hostnames for the same ruleset, then perform the following steps.
   a. On the right-hand side screen, click **Edit**.
   b. Click the icon next to the **Hostname** field.
   c. Specify the required *hostname* in the **Hostname** fields.



You can use the multiple hostname option to run production URLs on one hostname and staging URLs on another hostname. If you want to use multiple hostnames, then you must add the same hostnames in the **hosts** > **host** field of the *values.yaml* file.

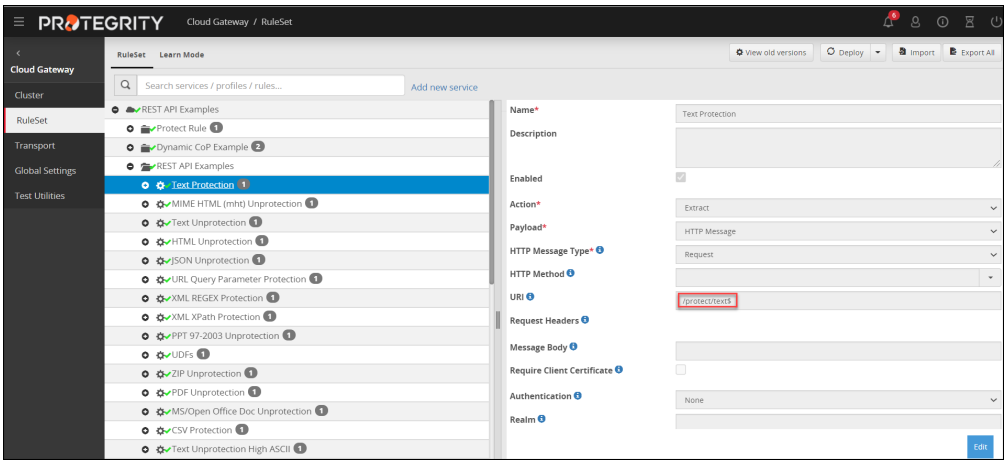
For more information about the *values.yaml* file parameters, refer to the section *Values.yaml*.

7. Navigate to **REST API Examples** > **Protect Rule** > **Data Protection**.

8. On the right-hand side screen, click **Edit** and specify the data element name in the **DataElement Name** field.

> **Note:**
> Ensure that this data element is part of the security policy that you have created.

9. Click **Save**.
10. Expand REST API Examples and navigate to the Text Protection ruleset. Note the URI.
    The Postman client is used to send a request to the following location:
    http://<hostname>/protect/text



> **Note:** The Text Protection rule is used to check the health of the pods. Leave the **HTTP Method** text box blank or enter the **GET** method to perform this check.

11. Perform the following steps to enable the basic authentication functionality, which ensures that the user who is performing the security operations, is authenticated as part of an existing LDAP. The username and password of the user are sent as part of the REST API request to authenticate the user.

> **Important:** Ensure that the user used for authentication is added to the Active Directory, and assigned a role that includes the *Cloud Gateway Auth* permissions.
> You can also use the ESA LDAP for adding users.
>
> For more information about managing users in the ESA, refer to the section *Managing Users* in the *Appliances Overview Guide 9.1.0.5*.
>
> For more information about assigning permissions to user roles in the ESA, refer to the section *Managing Roles* in the *Appliances Overview Guide 9.1.0.5*.

For more information about the basic authentication functionality, refer to the section *Basic Authentication* in the *Data Security Gateway Guide 3.1.0.5*.

a. Navigate to the **Cloud Gateway** > **RuleSet** tab.
b. Select the required rule.
c. In the **Authorization** list, select **Basic**.
   For more information about enabling the basic authentication functionality, refer to the section *Enabling REST API Authentication* in the *Data Security Gateway Guide 3.1.0.5*.

d. Save the changes.

e. Click **Deploy** to push the configurations to all the DSG nodes.

> **Note:** The user can either click **Deploy** or **Deploy to Node Groups** on the Ruleset page. For more information about the deploy functionality, refer to the section *RuleSet Tab* in the *Data Security Gateway User Guide 3.1.0.5*.

12. To export the CoP package from the ESA and encrypt the policy package using a passphrase and a salt, run the following command.

CURL request syntax for exporting CoP

```
curl -v -k
https://10.36.11.119/exportGatewayConfigFiles
-H "Authorization: Basic
YWRtaW46YWRtaW4xMjM=" -H "Content-Type:
text/plain" --data-raw '{"nodeGroup":"lob1:dsg1",
"kek":{"pkcs5":{"passphrase":"SECured#Thor@Data2019",
"salt":"SECured#Thor@Data2019"}}}' -o cop_node.json
```

Sample CURL request for exporting CoP

```
curl --location --request POST 'https://xxx.xxx.xxx.xxx/exportGatewayConfigFiles' \
--header 'Authorization: Basic <Base64 encoded administrator credentials>' \
--header 'Content-Type: text/plain' \
--data-raw '{
"kek":{
"pkcs5":{
"passphrase": "xxxxxxxxxx",
"salt": "salt"
}
}
}' -k -o cop_demo.json
```

> **Note:**
> Ensure that the passphrase contains a minimum of 10 characters, an uppercase, lowercase, a numeric, and a special character.

> **Note:** The response for the CURL request will be stored in *cop_demo.json* file.

> **Note:** For more information about exporting a CoP, refer to the section *Appendix H: CoP Export API for deploying the CoP (Containers only)* in the *Data Security Gateway User Guide 3.1.0.5*.

13. If you want to upload the CoP package to an AWS S3 bucket, then perform the following steps.
    a. Login to the AWS environment.
    b. Create an AWS S3 bucket.
    c. Upload the CoP package to an AWS S3 bucket.
14. If you want upload the CoP package to an AWS EFS, then perform the following steps.
    a. Login to the AWS environment.
    b. Create an AWS EFS instance.

    For more information about creating an AWS EFS instance, refer to the section *Creating an AWS EFS*.

    After you have created a file system on the AWS EFS instance, you need to mount this file system to the Linux instance that has been created in the AWS environment.

c. On the Linux instance, create a mount point for the file system by running the following command.

```
mkdir /<Mount point>
```

For example:

```
mkdir /test
```

This command creates a mount point *test* on the file system.

d. Run the following mount command to mount the file system from the AWS EFS instance on the mount point created in *step 13c*.

```
sudo mount ip-address:/file-system-id mount-point
```

For example:

```
sudo mount ip-address:/<file-system-id> test
```

Ensure that you set the value of the *file-system-id* parameter to the value of the **File System ID** field that you have specified in the section *Creating an AWS EFS*.

For more information about mounting a file system from the AWS EFS instance, refer to the section *Mounting EFS File Systems* in the AWS EFS documentation.

e. Create a mount directory by running the following command.

```
mkdir <Mount point>/<Mount directory>
```

For example:

```
mkdir test/xyz
```

This command creates a mount directory *xyz* on the file system.

f. Run the following command to copy the CoP package from your local machine to the mount directory on the Linux instance.

```
cp <local_folder>/cop_demo.json /test/xyz/
```

4.6 Working with the Publisher Deployment

The *pty-publisher* pod hosts two containers - PEP server and Meta server.

The PEP server service fetches the policy from the ESA. After the post-deploy script gets triggered, the IMP Creator utility creates the policy package by encrypting the policy. It then stores package on the object store or to the persistent volume that is attached to the pod running the PEP server and Meta server services.

The policy package is encrypted in memory using Passphrase-Based Encryption or Key Management Service (KMS), and the encrypted package is stored in the persistent volume or object store. The latest copy of the policy package (n) and an earlier copy of the policy package (n-1) are stored on the persistent volume or the object store.

In addition, a metadata JSON file containing the policy related information is created in the *tmpfs* volume mounted on the pod. The volume is accessible to the Meta server service. The Meta server service reads the metadata information and serves information through an endpoint over an *https* request.

A backup of the metadata file is also stored on the persistent volume or the object store.

For more information about the PEP server and Meta server containers, refer to the section *Workflow of the PEP Server Container* and *Workflow of the Meta Server Container*.

4.6.1 Configuring the Publisher Helm Chart

Extract the *PUBLISHER-HELM_ALL-ALL-ALL_x86-64_K8S_DSG_<Version>.tgz* package to create a folder directory *pty-publisher* with the Helm Charts using the following command.

```
tar -xvf PUBLISHER-HELM_ALL-ALL-ALL_x86-64_K8S_DSG_<Version>.tgz
```

The following output appears in the CLI:

```
pty-publisher/Chart.yaml
pty-publisher/values.yaml
pty-publisher/credentials.json
pty-publisher/nfs-pv.yaml
pty-publisher/nfs-pvc.yaml
pty-publisher/templates/_helpers.tpl
pty-publisher/templates/deployment.yaml
pty-publisher/templates/configmap-encryption.yaml
pty-publisher/templates/deployment.yaml
pty-publisher/templates/service.yaml
```

Before you begin
Ensure that the following prerequisites are complete.

a. Create the TLS certificate secret.
   a. Navigate to the *packaging/certs/* directory.

      ```
      cd packaging/certs/
      ```

      The *certificate.pem* and *key.pem* files are available in the *certs* directory.

      For more information about creating a TLS certificate, refer to *step 3c* in the section *Setting Up the Certificates*.

   b. Create the TLS certificate secret (*pty-secret-tls-publisher*) for the Publisher Meta server using the following command.

      ```
      kubectl create -n $DSG_NAMESPACE secret tls $PUBLISHER_SERVER_CERT_SECRET --cert=./
      certs/certificate.pem --key=./certs/key.pem
      ```

      Example:

      ```
      export DSG_NAMESPACE=test-demo
      export PUBLISHER_SERVER_CERT_SECRET=pty-secret-tls-publisher

      kubectl create -n $DSG_NAMESPACE secret tls $PUBLISHER_SERVER_CERT_SECRET --cert=./
      certs/certificate.pem --key=./certs/key.pem
      ```

      > **Note:** The secret is to be created from the *packaging/certs* directory.

b. Create a Kubernetes Secret in the namespace -

a. Create the ESA credentials secret (*pty-secret-esa-cred*) using the following command.

```
kubectl -n $DSG_NAMESPACE create secret generic $ESA_CREDENTIAL_SECRET --from-
literal=esa_user=$ESA_USER --from-literal=esa_pass=$ESA_PASS
```

Example:

```
export DSG_NAMESPACE=test-demo
export ESA_CREDENTIAL_SECRET=pty-secret-esa-cred
export ESA_User=imp
export ESA_PASS=Pwd$123#

kubectl -n $DSG_NAMESPACE create secret generic $ESA_CREDENTIAL_SECRET --from-
literal=esa_user=$ESA_USER --from-literal=esa_pass=$ESA_PASS
```

b. If you are using PBE to encrypt the policy package, then create the password-based encryption secret (*pty-secret-pbe*) using the following command.

```
kubectl -n $DSG_NAMESPACE create secret generic $PBE_SECRET --from-
literal=passphrase=$PASSPHRASE_PASSWORD --from-literal=salt=$SALT_PASSWORD
```

Example:

```
export DSG_NAMESPACE=test-demo
export PBE_SECRET=pty-secret-pbe
export PASSPHRASE_PASSWORD=<Passphrase>
export SALT_PASSWORD=<Salt_Password>

kubectl -n $DSG_NAMESPACE create secret generic $PBE_SECRET --from-
literal=passphrase=$PASSPHRASE_PASSWORD --from-literal=salt=$SALT_PASSWORD
```

Important: Ensure that the passphrase has a minimum length of 12 characters, which includes at least the following:

- 1 uppercase letter.
- 2 digits.
- 2 special characters.
- 2 non-letters, such as, digits or special characters.

The passphrase must also have a minimum value of entropybits as *30* and a minimum strength of *0.66*.

Entropybits defines the randomness of the password, and the strength defines the complexity of the password.

For more information about the password strength, refer to the section *Password Strength*.

c. Create a Kubernetes Secret to store the credentials needed to pull the Docker images from your image repository in the same namespace where you will deploy the PTY-Publisher pod.

```
kubectl -n $DSG_NAMESPACE create secret generic $IMAGE_REGISTRY_CREDS_SECRET --from-
file=.dockerconfigjson=$PATH_TO_DOCKER_CONFIG --type=kubernetes.io/dockerconfigjson
```

Example:

```
export DSG_NAMESPACE=test-demo
export IMAGE_REGISTRY_CREDS_SECRET=pty-secret-registry-cred
export PATH_TO_DOCKER_CONFIG=/etc/docker/config.json
```

```
kubectl -n $DSG_NAMESPACE create secret generic $IMAGE_REGISTRY_CREDS_SECRET --from-
file=.dockerconfigjson=$PATH_TO_DOCKER_CONFIG --type=kubernetes.io/dockerconfigjson
```

The *dockerconfigjson* file is generated when you set up the image container registry and perform the docker login command. This file stores the Docker credentials.

> **Note:**
> Create the Docker credentials only if required.

Edit the *imagePullSecrets* tag if using credentials to pull the Docker image.

d. If you are using Azure, then create the storage access secret (*storageAccessSecrets*) using the following command.

```
kubectl -n $DSG_NAMESPACE create secret generic $STORAGE_SECRET --from-
file=credentials=$HOME/pty-publisher/credentials.json --kubeconfig="${KUBECONFIG}"
```

You can also use service accounts to provide access to cloud services.

e. Run the following command to verify that the secrets have been created.

```
kubectl get secrets -n $DSG_NAMESPACE
```

For example:

```
export DSG_NAMESPACE=test-demo

kubectl get secrets -n $DSG_NAMESPACE
```

The following snippet shows the secrets that have been created.

```
NAME
TYPE                             DATA   AGE
pty-secret-esa-cred
Opaque                           2      34m
pty-secret-pbe
Opaque                           2      5d15h
pty-secret-tls-publisher
kubernetes.io/tls                2      5d15h
pty-secret-registry-cred                                              kubernetes.io/
dockerconfigjson   1      34m
```

> **Note:**
> Before editing the *values.yaml* file, ensure that the secrets are created in the same namespace where the Publisher and the DSG pods have been deployed.

➤ To configure the Publisher Helm chart, perform the following steps.

1. Navigate to the *packaging/pty-publisher* directory using the following command.

```
cd packaging/pty-publisher
```

2. Open the *values.yaml* file using the following command.

```
vi values.yaml
```

3. If you want to use AWS S3 for storing the CoP and policy snapshots, instead of AWS EFS, then perform the following steps to create service accounts in the Kubernetes cluster. These steps ensure that only the pod on which the DSG container has been deployed has write access to the AWS S3 bucket, which enables the DSG container to upload the immutable policy snapshot to the AWS S3 bucket.

   a. Run the following command to create an OpenID Connect (OIDC) identity provider in the IAM console.

```
eksctl utils associate-iam-oidc-provider --name <Name of the Kubernetes cluster> --
approve
```

   OIDC is an authentication protocol. The OIDC identity provider enables you to map AWS IAM roles to Kubernetes service accounts.

   b. Run the following command to create an IAM service account for the Sample Application container.

```
eksctl create iamserviceaccount --name <Service_account_name>
--namespace <Namespace> --cluster <Kubernetes_cluster_name> --attach-
policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess --attach-policy-arn
arn:aws:iam::829528124735:policy/KMSDecryptAccess --approve
```

   This command contains the following policies:

   - *AmazonS3WriteOnlyAccess* is a default AWS policy, which is attached to the required service account. This policy allows the service account to upload the encrypted policy package to the S3 bucket.
   - *KMSEncryptAccess* is a custom AWS policy, which is attached to the required service account. This policy allows the service account to encrypt the policy package that has been encrypted using an AWS Customer key.

   For example:

```
eksctl create iamserviceaccount --name dsg-sa --namespace dsg --cluster cluster-name
--attach-policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess --attach-policy-arn
arn:aws:iam::829528124735:policy/KMSDecryptAccess --approve
```

   Important: Ensure that the name of the service account is *dsg-sa*, which has been defined in the *pty-psp.yaml* file.

4. Edit the *imagePullSecrets* tag.

```
# specify the image registry credential secret name
# in case if the registry is accessible without any credentials uncomment
# square brackets and comment name field
imagePullSecrets: # []
  - name: $IMAGE_REGISTRY_CREDS_SECRET
```

   Replace the *$IMAGE_REGISTRY_CREDS_SECRET* variable with the value of the *pty-secret-registry-cred* secret that you have created in the *prerequisites*.

5. Edit the *pepServerImage* tag.

```
pepServerImage:
  repository: "$PEPSERVER_IMAGE_REPOSITORY"
  tag: "$PEPSERVER_IMAGE_TAG"
  pullPolicy: IfNotPresent
  debug: false  # change debug to true to get debug logs on STDOUT.
```

   Example:

```
pepServerImage:
  repository: <aws_account_id>.dkr.ecr.us-east-1.amazonaws.com/demo/publisher-pepserver
  tag: PUBLISHER_PEP_RHUBI-9-64_x86-64_K8S_9.1.0.0.xx
  pullPolicy: IfNotPresent
  debug: false  # change debug to true to get debug logs on STDOUT.
```

6. Edit the *metaServerImage* tag.

```
metaServerImage:
  repository: "$METASERVER_IMAGE_REPOSITORY"
  tag: "$METASERVER_IMAGE_TAG"
  pullPolicy: IfNotPresent
  debug: false  # change debug to true to get debug logs on STDOUT.
```

Example:

```
metaServerImage:
  repository: <aws_account_id>.dkr.ecr.us-east-1.amazonaws.com/demo/pty_metaserver
  tag: PUBLISHER_META_RHUBI-9-64_x86-64_K8S_9.1.0.0.xx
  pullPolicy: IfNotPresent
  debug: false  # change debug to true to get debug logs on STDOUT.
```

7. Edit the *publisherServerCertSecret* tag, which is used to specify the secret of the TLS certificate for the Publisher server.

```
## publisher server TLS certificate secret
## pty sidecar container will verify these certificates
# eg. kubectl command:
#     kubectl -n $DSG_NAMESPACE create secret tls pty-certs \
#     --cert=./certs/certificate.pem --key=./certs/key.pem
publisherServerCertSecret: $PUBLISHER_SERVER_CERT_SECRET
```

In the *publisherServerCertSecret* tag, specify the value of the *pty-secret-tls-publisher* that you have created in the *prerequisites*.

8. Edit the *esaCredSecrets* tag.

```
## k8s secret for storing ESA credentials
# eg. kubectl command:
#     kubectl -n $DSG_NAMESPACE create secret generic pty-esa \
#     --from-literal=esa_user=<esa_user> --from-literal=esa_pass='<esa_user_password>'
esaCredSecrets: $ESA_CREDENTIAL_SECRET
```

In the *esaCredSecrets* tag, specify the value of the *pty-secret-esa-cred* that you have created in the *prerequisites*.

9. If you are using PBE to encrypt the policy package, then edit the *pbeSecrets* tag.

```
## k8s secret containing passphrase for encrypting policy
# eg. kubectl command:
#     kubectl -n $DSG_NAMESPACE create secret generic pty-pbe \
#     --from-literal='passphrase=<complex chars>' --from-literal='salt=<complex chars>'
pbeSecrets: $PBE_SECRET
```

In the *pbeSecrets* tag, specify the value of the *pty-secret-pbe* that you have created in the *prerequisites*.

10. Specify one of the following options in the *securityDonfigDestination* tag for storing the policy package and COP:
    - VolumeMount
    - AWS
    - AZURE
11. If you are using Cloud KMS to encrypt the policy package, then edit the *privateKeySource* tag.

```
## If not using PBE, Choose your private key helper.
## Currently we support [AWS_KMS, AZ_VAULT]
privateKeySource: $KEY_SOURCE

privateKeyId: $KEY_ID
```

Specify one of the following values for the *privateKeySource* tag:

- *AWS_KMS* - Use AWS KMS
- *AZ_VAULT* - Use Azure Key Vault as the KMS

Also, specify the value of the *privateKeyID* tag.

Example 1 - AWS

```
## If not using PBE, Choose your private key helper.
## Currently we support [AWS_KMS, AZ_VAULT]
privateKeySource: "AWS_KMS"

privateKeyId: "a47844b7-ce4f-4650-8ce5-146875db5339"
```

Example 2 - Azure

```
## If not using PBE, Choose your private key helper.
## Currently we support [AWS_KMS, AZ_VAULT]
privateKeySource: "AZ_VAULT"

privateKeyId: "https://automation-cntrs.vault.azure.net/keys/dsg/
154e3a367e9240d5958cda0af44265f2"
```

12. Edit the *pvcName* tag only if you are using Persistent Volume for storage.

```
## specify the name of persistent volume claim to be used for this deployment.
pvcName: $PVC_NAME
```

Example:

```
## specify the name of persistent volume claim to be used for this deployment.
pvcName: esa-policy-store
```

13. Edit the *policy* tag.

```
policy:
  esaIP: $ESA_IP
  ## absolute path in PV where the policy dump file will be stored. <eg. /test/xyz/ >
  filePath: $ABSOLUTE_POLICY_PATH
```

Example:

```
policy:
  esaIP: 10.21.0.99
  filePath: demo/policy
```

> **Note:** Do not include the trailing '/' in the file path. For example; *test/xyz/policy*. The policy package file is placed in this path on the persistent volume.

4.6.1.1 Additional Configuration

The other values that you can edit from the *packaging/pty-publisher/values.yaml* file are mentioned in the following table.

Table 4-5: Additional Tags in values.yaml file

Tags	Default Configuration	Description
pepserverConfig[*1]	```## pepserver configurations that can be changed. ## emptystring: Set the output behavior for empty strings ## null = (default) null value is returned for empty input string ## empty = empty value is returned for empty input string ## encrypt = encrypted values is returned for empty input string ## level: Specifies the logging level```	It changes the default behavior of the PEP server settings. For more information about the PEP server configurations, refer to the section *PEP Server Configuration File* in the

Tags	Default Configuration	Description
	for pepserver ## OFF (no logging) ## SEVERE ## WARNING ## INFO ## CONFIG ## ALL (default) ## caseSensitive: Specifies how policy users are checked against policy ## yes = (The default) policy users are treated in case sensitive manner ## no = policy users are treated in case insensitive manner. pepserverConfig: emptyString: "null" logLevel: "ALL" caseSensitive: "yes"	*Protegrity Installation Guide 9.1.0.5*. **Note:** If the input data is less than or equal to zero, then DSG processes the input as it is. Hence, in such cases, even if the *pepserverConfig*/emptyString parameter is set to **Null** or **Encrypt**, an Empty output is returned.
pepServerResources[*1]	pepServerResources: # below allocation covers most of the use-cases which requires wide range of policy size. limits: cpu: 500m memory: 4000Mi requests: cpu: 200m memory: 256Mi	It configures the resource requirements for the PEP server container.
metaServerResources[*2]	metaServerResources: # below allocation covers most of the use-cases which requires wide range of policy size. limits: cpu: 500m memory: 512Mi requests: cpu: 200m memory: 256Mi	It configures the resource requirements for the Meta server container.
serviceAccount[*2]	## pod service account to be used ## leave the field empty if not applicable ## e.g. serviceAccount: user1 serviceAccount: name: ""	It is a service account to run the pod. Currently, it set to default. Customers can configure the service account as per their requirement.
podSecurityContext[*2]	## set the Pod's security context object podSecurityContext: runAsUser: 1000 runAsGroup: 1000 fsGroup: 1000	It changes the default security context of the *pty-publisher* pod. **Important:** At least one of the three parameters must be set to *1000*.
containerSecurityContext[*2]	## set the Container's security context object containerSecurityContext: capabilities: drop: - ALL allowPrivilegeEscalation: false privileged : false	It changes the default security context of the containers.

Tags	Default Configuration	Description
	`runAsNonRoot : true` `readOnlyRootFilesystem: true`	

livenessProbe[*2]	`## Configure the delays for Liveness Probe here`
	`livenessProbe:`
	`initialDelaySeconds: 15`
	`periodSeconds: 20`
		It changes the default delay interval of the liveness probe, which performs a diagnostic check to confirm if the containers are healthy.

readinessProbe[*2]	`## Configure the delays for Readiness Probe here`
	`readinessProbe:`
	`initialDelaySeconds: 15`
	`periodSeconds: 20`
		It changes the default delay interval of the readiness probe, which checks whether the containers are ready to accept the incoming traffic.
		If files with larger size are being processed, then the container might be busy in processing the file. This might lead to a probe failure message in the pod description. As a result, the pod changes its state from Ready to Not Ready.

Note:

[*1] - PEP server parameters

[*2] - Optional parameters

4.6.2 Deploying the Publisher Helm Chart

After performing the prerequisites and steps, you can deploy the Publisher Helm Chart.

1. Navigate to the *packaging* directory using the following command.

   ```
   cd packaging
   ```

2. Deploy the Publisher Helm Chart using the following command.

   ```
   helm install $PTY_PUBLISHER_HELM_INSTALL_NAME $PTY_PUBLISHER_HELM_CHART_LOCATION -n
   $DSG_NAMESPACE
   ```

   Example:

   ```
   export DSG_NAMESPACE=test-demo
   export PTY_PUBLISHER_HELM_INSTALL_NAME=publisher
   export PTY_PUBLISHER_HELM_CHART_LOCATION=pty_publisher

   helm install $PTY_PUBLISHER_HELM_INSTALL_NAME $PTY_PUBLISHER_HELM_CHART_LOCATION -n
   $DSG_NAMESPACE
   ```

   The following output appears if you have successfully installed the Publisher Helm Chart.

   ```
   WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /
   home/ec2-user/kubeconfig
   ```

```
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /
home/ec2-user/kubeconfig
NAME: publisher
LAST DEPLOYED: Mon Apr  1 06:49:42 2024
NAMESPACE: test-demo
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

If you are an advanced user of Kubernetes, then you can also deploy the Helm charts by substituting the environment variables or using the *set* argument.

For more information about deploying a Helm chart by substituting the environment variables, refer to the Appendix *Deploying the Helm Charts by Substituting the Environment Variables*.

For more information about deploying a Helm chart by using the *set* argument, refer to the Appendix *Deploying the Helm Charts by Using the Set Argument*.

3. Run the following command to check the status of the pods.

```
kubectl get pods -n $DSG_NAMESPACE
```

For example:

```
export DSG_NAMESPACE=test-demo

kubectl get pods -n $DSG_NAMESPACE
```

4. Run the following command to check the logs.

```
kubectl get logs -f <Pod_name> -n $DSG_NAMESPACE
```

For example:

```
export DSG_NAMESPACE=test-demo

kubectl logs -f publisher-pty-publisher-7f596d88d5-jq4jq -n $DSG_NAMESPACE
```

For more information about the output of this command, refer to the Appendix *Publisher Pod Log Files*.

If you are unable to download the policy from the ESA even after specifying the correct Node IP, then verify whether the logs contain the following snippet.

```
2024-04-16 04:37:28 (WARNING) Node registration failed, please check IP/Range of data
store or set default data store.
2024-04-16 04:37:28 (INFO) This node has been registered as node ID 16
2024-04-16 04:37:28 (WARNING) Internal error - Failed to register node
```

This error indicates that the ESA is unable to register the node.

To troubleshoot this issue, perform the following steps:

a. Navigate to the ESA and specify the **Set as Default Data Store** option to **Yes**.
b. Click **Save** and then deploy the policy.
c. Validate whether the policy deployment is successful.
   The following snippet of the Publisher pod log shows that the policy has been downloaded successfully.

```
....
....
....
2024-04-16 04:43:21,441:imp_creator:INFO:Done.
```

```
2024-04-16 04:43:21 (INFO) Post deploy application done!
2024-04-16 04:43:21 (INFO) Done with downloading policy files
```

d. In the **Nodes** tab of the ESA, verify the IP address of the registered node.

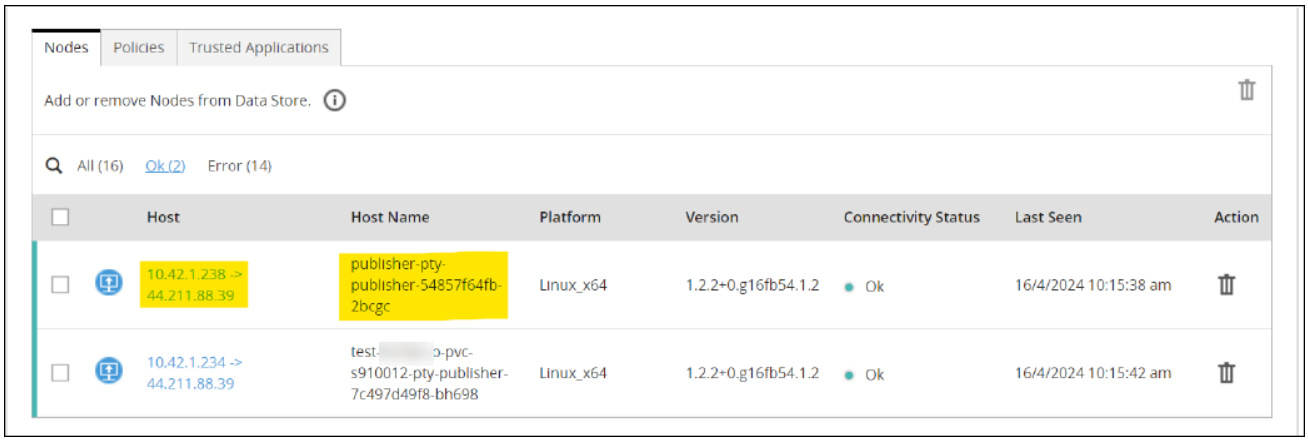


Figure 4-5: Nodes tab

In the **Host** column, the first IP address indicates the IP of the Publisher pod, while the second IP address indicates the IP of the Node where the Publisher pod has been deployed.

```
<IP address of Publisher Pod> --> <IP address of Node where Publisher Pod is deployed>
```

e. Configure the datastore in the ESA with the IP address of the Node that you have obtained in *step 4e*.

f. Specify the **Set as Default Data Store** option to **No**.

4.7 Deploying a Release on the Kubernetes Cluster

This section describes how to deploy a Release on the Kubernetes cluster by installing the Helm charts.

> **Note:** The procedures performed in this section require the user to have the *cluster-admin* role.

➤ To deploy a release on the Kubernetes cluster:

1. Perform the following steps if you want to use a persistent volume for storing the policy and CoP package instead of the AWS S3 bucket.
   a. Create a file named *storage_class.yaml* for creating an AWS EFS storage class.
   The following snippet shows the contents of the *storage_class.yaml* file.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: efs-sc
provisioner: efs.csi.aws.com
```

> **Important:**

> If you want to copy the contents of the *storage_class.yaml* file, then ensure that you indent the file as per YAML requirements.

b. Run the following command to provision the AWS EFS using the *storage_class.yaml* file.

```
kubectl apply -f storage_class.yaml
```

An AWS EFS storage class is provisioned.

c. Create a file named *pv.yaml* for creating a persistent volume resource.

The following snippet shows the contents of the *pv.yaml* file.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: efs-pv1
  labels:
    purpose: policy-store
spec:
  capacity:
    storage: 1Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: efs-sc
  #mountOptions:
  #  - tls
  #  - accesspoint=fsap-09081079ccfa471d8
  csi:
    driver: efs.csi.aws.com
    volumeHandle: <EFS file system ID>
```

> **Important:**
> If you want to copy the contents of the *pv.yaml* file, then ensure that you indent the file as per YAML requirements.

This persistent volume resource is associated with the AWS EFS storage class that you have created in *step 3b*.

In the *storageClassName* parameter, ensure that you specify the same name for the storage class that you specified in the *storage_class.yaml* file in step *3a*.

For example, specify *efs-sc* as the value of the *storageClassName* parameter.

d. Run the following command to create the persistent volume resource.

```
kubectl apply -f pv.yaml
```

A persistent volume resource is created.

e. Create a file named *pvc.yaml* for creating a claim on the persistent volume that you have created in *step 3d*.

The following snippet shows the contents of the *pvc.yaml* file.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: efs-claim1
spec:
  selector:
    matchLabels:
      purpose: "policy-store"
```

```
  accessModes:
    - ReadWriteMany
  storageClassName: efs-sc
  resources:
    requests:
      storage: 1Gi
```

> **Important:**
> If you want to copy the contents of the *pvc.yaml* file, then ensure that you indent the file as per YAML requirements.

This persistent volume claim is associated with the AWS EFS storage class that you have created in *step 3b*. The value of the *storage* parameter in the *pvc.yaml* file defines the storage that is available for saving the policy dump.

In the *storageClassName* parameter, ensure that you specify the same name for the storage class that you specified in the *storage_class.yaml* file in *step 3a*.

For example, specify *efs-sc* as the value of the *storageClassName* parameter.

f. Run the following command to create the persistent volume claim.

```
kubectl apply -f pvc.yaml -n <Namespace>
```

For example:

```
kubectl apply -f pvc.yaml -n dsg
```

A persistent volume claim is created.

g. On the Linux instance, create a mount point for the AWS EFS by running the following command.

```
mkdir /efs
```

This command creates a mount point *efs* on the file system.

h. Run the following mount command to mount the AWS EFS on the directory created in *step 3g*.

```
sudo mount -t nfs -o
nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,noresvport <file-
system-id>.efs.<aws-region>.amazonaws.com:/ /efs
```

For example:

```
sudo mount -t nfs -o
nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,noresvport
fs-618248e2.efs.<aws-region>.amazonaws.com:/ /efs
```

Ensure that you set the value of the *<file-system-id>* parameter to the value of the *volumeHandle* parameter, as specified in the *pv.yaml* file in *step 3c*.

> **Note:** This step is needed to validate the policy creation on AWS EFS.

For more information about the permissions required for mounting an AWS EFS, refer to the section *Working with Users, Groups, and Permissions at the Network File System (NFS) Level* in the AWS documentation.

2. If you want to use AWS S3 for storing the CoP and policy snapshots, instead of the persistent volume, then perform the following steps to create service accounts in the Kubernetes cluster. These steps ensure that only the pod on which the

DSG container has been deployed has write access to the AWS S3 bucket, which enables the DSG container to upload the immutable policy snapshot to the AWS S3 bucket.

a. Run the following command to create an OpenID Connect (OIDC) identity provider in the IAM console.

```
eksctl utils associate-iam-oidc-provider --name <Name of the Kubernetes cluster> --
approve
```

OIDC is an authentication protocol. The OIDC identity provider enables you to map AWS IAM roles to Kubernetes service accounts.

b. Run the following command to create an IAM service account for the Sample Application container.

```
eksctl create iamserviceaccount --name <Service_account_name>
--namespace <Namespace> --cluster <Kubernetes_cluster_name> --attach-
policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess --attach-policy-arn
arn:aws:iam::829528124735:policy/KMSDecryptAccess --approve
```

This command contains the following policies:

- *AmazonS3ReadOnlyAccess* is a default AWS policy, which is attached to the required service account. This policy allows the service account to download the encrypted policy package from the S3 bucket.
- *KMSDecryptAccess* is a custom AWS policy, which is attached to the required service account. This policy allows the service account to decrypt the policy package that has been encrypted using an AWS Customer key.

For example:

```
eksctl create iamserviceaccount --name dsg-sa --namespace dsg --cluster cluster-name
--attach-policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess --attach-policy-arn
arn:aws:iam::829528124735:policy/KMSDecryptAccess --approve
```

> **Important:** Ensure that the name of the service account is *dsg-sa*, which has been defined in the *pty-psp.yaml* file.

3. Run the following command to create the *cop-secret* secret for the CoP package.

```
kubectl create secret generic cop-secret --from-literal='passphrase=<COP_PASS>' --from-
literal='salt=<COP_SALT>' -n <SAME_AS_OF_CHART_DEPLOYMENT>
```

For example,

```
kubectl create secret generic cop-secret --from-literal='passphrase=Passphrase123@' --
from-literal='salt=salt' -n dsg
```

> **Note:**
> As the value of the *<COP_PASS>* and *<COP_SALT>* parameters, you must specify the passwords that you created the section *Creating a Ruleset*, when you export the CoP package from the ESA.

4. If you want to authenticate the communication between the Kubernetes cluster and the Amazon ECR by using your existing Docker credentials, then perform the following steps.

a. Run the following command to login to Docker.

```
docker login
```

The login process creates a *config.json* file, which contains an authorization token.

b. Run the following command to specify a secret for pulling the DSG container images from the Amazon ECR to Kubernetes:

```
kubectl create secret generic <Secret_name> --from-file=.dockerconfigjson=/<Path to the
config.json file> --type=kubernetes.io/dockerconfigjson --namespace dsg
```

For example:

```
kubectl create secret generic regcrd --from-file=.dockerconfigjson=/home/<user>/.docker/
config.json --type=kubernetes.io/dockerconfigjson --namespace dsg
```

For more information about creating secrets, refer to *https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/*.

> **Note:**
> Ensure that you specify the name of the secret as the value of the *imagePullSecrets/name* field in the *values.yaml* file.

For more information about the *imagePullSecrets* field, refer to the section *Image Pull Secrets*.

5. Create a secret for the Certificate Authority (CA) certificate that you have generated for the PTY-Publisher pod.
   a. Navigate to the *certs* directory.
      The *ca.pem* files are available in the *certs* directory.
      For more information about creating a CA certificate, refer to step 2 in the section Setting Up the Certificates.
   b. Run the following command to create a Certificate Authority (CA) certificate for the PTY-Publisher pod. This CA certificate is used by the sidecar container to communicate with the PTY-Publisher pod.

```
kubectl create -n $DSG_NAMESPACE create secret generic $PUBLISHER_CA_CERT_SECRET --
fromfile=ca.pem=./certs/ca.pem
```

Example:

```
export DSG_NAMESPACE=test-demo
export PUBLISHER_CA_CERT_SECRET=pty-secret-ca-cert

kubectl create -n $WAREHOUSE_NAMESPACE create secret generic $CA_CERT_NAME --
fromfile=ca.pem=./certs/ca.pem
```

6. If you want to authenticate the user using basic authentication, then run the following command to create an LDAP secret, which is used by the DSG container to connect to the LDAP.

```
kubectl create secret generic <LDAP_secret_name> -n <Namespace> --from-
literal=self.xml='<perldata> <hashref> <item key="LDAP_URI">ldap://{LDAP IP}:{port}/</
item> <item key="LDAP_BASEDN">{BASE DN}</item> <item key="LDAP_USERSDN">{USER DN}</item>
<item key="LDAP_BINDDN">{BIND DN}</item> <item key="LDAP_BINDPW">{BIND Password}</item>
<item key="LDAP_STARTTLS">1</item> </hashref> </perldata>'
```

For example, the following snippet shows the LDAP parameters that are used to connect to the ESA LDAP:

```
kubectl create secret generic cmdldap -n dsg
--from-literal=self.xml='<perldata> <hashref> <item key="LDAP_URI">ldap://
10.120.0.58:389/</item> <item key="LDAP_BASEDN">dc=esa,dc=protegrity,dc=com</item>
<item key="LDAP_USERSDN">ou=people,dc=esa,dc=protegrity,dc=com</item> <item
key="LDAP_BINDDN">cn=ldap_bind_user,ou=people,dc=esa,dc=protegrity,dc=com</item> <item
```

```
key="LDAP_BINDPW">{BIND_Password}</item> <item key="LDAP_STARTTLS">1</item> </hashref> </
perldata>'
```

In this command, you need to specify the value for these LDAP parameters:

Table 4-6: LDAP Parameters

Parameter	Description
LDAP_URI	The IP address and port number used to connect to the LDAP server.
LDAP_BASEDN	Distinguished name, which uniquely identifies the base location that is used to search for a user in an LDAP directory. For example, if you are using the ESA LDAP and your domain name is *protegrity.com*, then your Base DN might be: `dc=esa,dc=protegrity,dc=com`
LDAP_USERSDN	Distinguished name, which uniquely identifies the user location that is used to search for a user in an LDAP directory. It is a subset of the Base DN. For example, if you are using the ESA LDAP, your domain name is *protegrity.com*, and your users are part of an organizational unit named *people*, then your Users DN might be: `ou=people, dc=esa,dc=protegrity,dc=com`
LDAP_BINDDN	Distinguished name of the user that is used to connect to the LDAP server. For example: `cn=ldap_bind_user,ou=people,dc=esa,dc=protegrity,dc=com`
LDAP_BINDPW	Password of the user that is used to connect to the LDAP server.
LDAP_STARTTLS	Set the value of this parameter to *1* to use the TLS protocol for encrypting the communication with the LDAP server.

You need to specify the LDAP secret name as the value of the *ldapXmlSecrets* parameter in the *values.yaml* file in *step 6*.

Note: If you are using the ESA LDAP, then you can obtain the values for the LDAP parameters from the *self.xml* file.

To access the *self.xml* file, perform the following steps.

1. Login to the CLI Manager using the administrator credentials.
2. Navigate to **Administration** > **OS Console**.
3. Enter the *root* password and press **OK**.
4. Navigate to the `/etc/ksa/conf/` directory to access the *self.xml* file.

Important: If you are using a third-party LDAP server, then ensure that you add the following attribute to the user with corresponding value in the LDAP server.

Attribute	Value
businessCategory	pty_role:cloud_gateway_auth

For more information about adding an attribute to the LDAP server, refer to the LDAP server documentation.

> If you have connected the LDAP server to the ESA, and you add a user who has been assigned a role that includes the *Cloud Gateway Auth* permissions, then the *businessCategory* attribute is automatically added to the LDAP server.
>
> For more information about connecting an LDAP server to the ESA using the CLI Manager, then refer to the section *Managing LDAP* in the *Appliances Overview Guide 9.1.0.5*.
>
> For more information about authenticating and authorizing external LDAP users from the ESA, refer to the section *Working with Proxy Authentication* in the *Enterprise Security Administrator Guide 9.1.0.5*.

7. On the Linux instance, navigate to the location where you have extracted the Helm charts.
   For more information about the extracted Helm charts, refer to the *step 4* of the section *Extracting the Package*.

   The *values.yaml* file contains the default configuration values for deploying the DSG application on the Kubernetes cluster.

8. Modify the default values in the *values.yaml* file as required.
   For more information about Helm charts and their default values, refer to the section *Appendix A: DSG Container Helm Chart Details*.

Field	Description
pvcName	Specify the name of the Persistent Volume Claim where you want to store the policy snapshot. Leave the value of this field blank if you want to use AWS S3 buckets for storing the policy snapshot. If you are storing the policy package in persistent volume, then specify the name of the Persistent Volume Claim that you have specified in the *pv.yaml* file. Leave the value of this field blank if you want to use the AWS S3 bucket for storing the policy package. **Important:** This field is required if you want to store the policy package in the persistent volume.
privateKeySource	Specify one of the following methods used to encrypt the policy package: • *AZ_VAULT* - Azure Vault for encrypting the policy package. • *AWS_KMS* - Use AWS Customer Master Key for encrypting the policy package.
privateKeyId	Specify the private key from AWS KMS
securityConfigDestination	Specify one of the following values for storing the policy package and COP: • VolumeMount • AWS • Azure
copSecrets	Specify the Kubernetes secret that contains the passphrase and the salt that were used to generate key, which encrypted the CoP. The DSG Application container uses the value specified in the *CopSecrets* parameter to decrypt the CoP package.
serviceAccount	Specify the name of the service account for the pod that you have created in step 3.

Field	Description
sidecarObject/pingInterval	Specify the interval in seconds after which the Subscriber-Sidecar container checks for the policy update information.
sidecarObject/publisher/host	Specify the hostname of the service hosting the policy update information.
sidecarObject/publisher/port	Specify the port number of the service hosting the policy update information. BY default, this value is set to 443.
sidecarObject/publisher/tls/caCertSecret	Specify the CA secret that you have created in *step 6*.
pbeSecrets	Specify the Kubernetes secret that contains the passphrase and the salt that were used to generate key, which encrypted the policy. The DSG Application container uses the value specified in the *pbeSecrets* parameter to decrypt the policy. By default, the Passphrase-Based Encryption is used to encrypt the policy and CoP package. If you want to use the AWS KMS to decrypt the policy package, then you must comment out this entry or leave the value blank. **Note:** If you specify a value for *pbeSecrets* field, then Passphrase-Based Encryption is used instead of the AWS KMS. **Important:** Ensure that the passphrase has a minimum length of 12 characters, with at least 1 uppercase letter, 2 digits, 2 special characters, and 2 non-letters (such as, digits or special characters). The passphrase must also have a minimum value of entropybits as *30* and a minimum strength of *0.66*. Entropybits defines the randomness of the password and the strength defines the complexity of the password. For more information about the password strength, refer to the section *Password Strength*.
ldapXmlSecrets	Specify the value of the LDAP secret that you have created in *step 7*.
podSecurityContext	Specify the privilege and access control settings for the pod. The default values are set as follows: • *runAsUser* - 1000 • *runAsGroup* - 1000 • *fsGroup* - 1000 • *supplementalGroups* - 1000 **Note:** If you want to use AWS S3 for storing the policy and CoP, then set the value of the *fsGroup* parameter to a non-root value.

Field	Description
	For example, you can set the value of the *fsGroup* parameter to any value between *1* and *65534*. **Note:** Ensure that at any given point of time, the value of at least one of the three parameters must be *1000*. This ensures that the pod has the required permissions to access the *pty* directories in the containers. For more information about the Pod Security Context, refer to the section *Configure a Security Context for a Pod or Container* in the Kubernetes documentation. For more information about the parameters, refer to the section *PodSecurityContext v1 Core* in the Kubernetes API documentation.
cop/filepath	Specify the path in the persistent volume or the object store where you have stored the CoP package. For example, if you are using AWS EFS as the persistent volume, then you can specify the file path as *file://var/data/ptypolicy/pathInVolumeTocop*. If you have stored the IMP in an Object Store, such as an AWS S3 bucket, then you can specify the bucket name as the first name in the *filepath* field. For example, you can specify the value of the *filepath* field, as *s3:/test/LOB/cop.json*, where *test* is the bucket name, *LOB* is the directory inside the bucket, and *cop.json* is the name of the CoP package. In this example, the bucket name is specified as the first name of the file path. **Important:** Use only uppercase and lowercase letters, numbers, dot, and underscore in the file path. Do not use special characters in the file path. **Note:** This value is case-sensitive.
dsgService/type	Specify whether you want to use one of the following service types: • *LoadBalancer* - An external IP is created. You need to send a request to the IP address of the AWS Load Balancer for running the security operations. • *ClusterIP* - A cluster IP is created. You need to send a request to the cluster IP, which is a unique IP address assigned to the Kubernetes service, for running the security operations. If you specify the service type, then you also need to uncomment the annotations for the corresponding service type.

Field	Description
dsgService/healthCheckRestService/host	Specify the host name of the heath check rule defined in the CoP. By default, this value is set to *restapi*.
dsgService/healthCheckRestService/port	Specify the port number configured in the DSG CoP ruleset for the health check rule. By default, this value is set to *&healthCheckPort 8080*.
dsgService/healthCheckRestService/path	Specify the URI of the health check rule configured in the DSG CoP ruleset. By default, this value is set to */protect/text*.
dsgService/tunnels/name	Specify a name for the tunnel to distinguish between ports.
dsgService/tunnels/port	Specify the port number on which you want to expose the Kubernetes service externally.
dsgService/tunnels/targetPort	Specify the port number configured while creating a tunnel on the ESA. By default, this value is set to **healthCheckPort*, which indicates the value specified in the *dsgService/healthCheckRestService/port* field. For more information about creating a tunnel, refer to the section *Tunnels tab* in the *Data Security Gateway User Guide 3.1.0.5*.

9. Run the following command to deploy DSG on the Kubernetes cluster:

```
helm install <Release Name> <Location of the directory that contains the Helm charts>
--namespace <Namespace>
```

For example:

```
helm install dsg-demo dsg --namespace dsg
```

10. Perform the following steps to check the status of the Kubernetes resources.
    a. Run the following command to check the status of the pods.

```
kubectl get pods -n <namespace>
```

For example:

```
kubectl get pods -n dsg
```

4.8 Verifying the Operations for the DSG Container Deployment

This section specifies the commands to verify the operations performed on the DSG Container deployment.

4.8.1 Generic Commands to Validate the Cluster

This section describes the generic commands to validate the cluster.

- To fetch the project list, execute the following command.

```
kubectl projects list
```

- To describe the pod specification, execute the following command.

```
kubectl -n <namespace> describe <pod name>
```

- To describe the pod specification in *yaml*, execute the following command.

```
kubectl -n <namespace> describe <pod name> -o yaml
```

- To check the pods list, execute the following command.

```
kubectl -n <namespace> get pods -n <namespace>
```

- To check the service name of the deployed containers, execute the following command.

```
kubectl -n <namespace> get svc -n <namespace>
```

- To check the logs of the containers, execute the following command.

```
kubectl -n <namespace> logs <pod name> -c <container name>
```

- To check the logs of the pod, execute the following command.

```
kubectl -n <namespace> logs <pod name>
```

4.8.2 Validating the Software Version Installed on the Base Machine

This section describes the versions to be validated for the software or client installed on the base machine.

- To validate the Kubectl version installed on the base machine, execute the following command.

```
kubectl version
```

The following output appears.

```
kubectl version
Client Version: v1.31.2
Kustomize Version: v5.4.2
Server Version: v1.30
```

- To validate the Docker version installed on the base machine, execute the following command.

```
docker version
```

The following output appears.

```
Client:
 Version:           24.0.7
 API version:       1.43
 Go version:        go1.22.2
 Git commit:        24.0.7-0ubuntu4.1
 Built:             Fri Aug  9 02:33:20 2024
 OS/Arch:           linux/amd64
 Context:           default

Server:
 Engine:
  Version:          24.0.7
  API version:      1.43 (minimum version 1.12)
  Go version:       go1.22.2
  Git commit:       24.0.7-0ubuntu4.1
  Built:            Fri Aug  9 02:33:20 2024
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.7.12
  GitCommit:
 runc:
  Version:          1.1.12-0ubuntu3.1
  GitCommit:
 docker-init:
```

```
  Version:          0.19.0
  GitCommit:
```

- To validate the Helm version installed on the base machine, execute the following command.

```
helm version
```

The following output appears.

```
version.BuildInfo{Version:"v3.16.2", GitCommit:"13654a52f7c70a143b1dd51416d633e1071faffb",
GitTreeState:"clean", GoVersion:"go1.22.7"}
```

4.8.3 Validating the Docker Images Uploaded in the Image Repository

This section describes the docker images to be validated after uploading it in the image repository.

- To validate the images tagged in the docker, execute the following command.

```
docker images
```

The following sample output appears on the CLI when you initially load the docker.

Table 4-7: Docker Images Output Format

REPOSITORY	TAG	IMAGE ID	SIZE
artifactory.protegrity.com/docker-registry/dsg/dsg-subscriber-sidecar	SUBSCRIBER-SIDECAR_RHUBI-8-64_x86-64_K8S_<Version>	<Image ID>	564MB

> **Note:** The example mentioned in the table considers only one image output.

The following sample output appears on the CLI after you create a tag.

Table 4-8: Docker Images Output Format After Tag Creation

REPOSITORY	TAG	IMAGE ID	SIZE
artifactory.protegrity.com/docker-registry/dsg/pty-subscriber-sidecar	CLIENT-SIDECAR_RHUBI-ALL-64_x86-64_K8S_<Version>	<Image ID>	221MB
<IMAGE_REPOSITORY_URL>/<PATH>	<TAG>	<Image ID>	221MB

4.8.4 Validating the Helm Chart

This section describes the Helm charts to be validated before deploying the various components of the DSG Container on the EKS or Azure platform.

- To validate the Helm chart, execute the following command.

```
helm template -f <values.yaml filepath> <name for this helm deployment> <chart name>
```

Example:

```
helm template pty-publisher/values.yaml publisher pty-publisher
```

> **Note:** A complete Helm chart appears with the values edited in the *values.yaml* file. If there is any error in the output, then it needs to be resolved before installing the Helm chart.

4.8.5 Validating the Status of the PV and PVC

This section describes the commands to validate the status of the PV and PVC.

- To get the list of Persistent Volume (PV) in the namespace, execute the following command.

```
kubectl -n <namespace> get pv
```

The following sample output appears on the CLI.

Table 4-9: PV Output

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	STORAGE CLASS
policy-store	1Gi	RWX	Retain	Bound	test-demo/esa-policy-store	

> **Note:** The status should be Bound post creation of the PVC.

- To get the list of Persistent Volume Claims (PVC) in the namespace, execute the following command.

```
kubectl -n <namespace> get pvc
```

The following sample output appears on the CLI.

Table 4-10: PVC Output

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGE CLASS
esa-policy-store	Bound	policy-store	1Gi	RWX	

> **Note:** The access mode should be ReadWriteMany (RWX). The status will change to Bound after the PVC is created and linked to the PV.

4.8.6 Validating the Publisher Containers

This section describes the commands to validate whether the policy and metadata are created successfully and the MetaServer is responding to the fetch metadata requests.

➤ To validate whether the policy and metadata are created successfully, execute the following steps.

1. To fetch the service name of the Publisher pod, execute the following command.

```
kubectl -n <namespace> get svc | grep publisher*
```

The following sample output appears on the CLI.

Table 4-11: Service Name of the Publisher Pod

NAME	TYPE	CLUSTER-IP	PORT(S)
publisher-pty-publisher	ClusterIP	172.30.15.221	11080/TCP

2. To enter into the interactive mode of the pod, execute the following command.

```
kubectl -n <namespace> exec -it <publisher pod name> -- /bin/bash
```

> **Note:** After executing the command, you will enter the pod in an interactive mode.

3. In the pod, execute the curl request using the following command.

```
curl http://<<name for this helm deployment>>.<<namespace>>:11080/status
```

4. Verify the sample output returned as *json* file with the policy information.

```
{
  "version": "1.0.0",
  "name": "policy_2023_09_06_07_32_12.tgz",
  "volume": "/var/data/policy",
  "path": "demo/policy",
  "timestamp": "1693985533",
  "hash_type": "SHA256",
  "hash": "cc04e6af39246ad60763a09a8df528e9bdec7861b50e67a98dfba04781072e15",
  "kek": "pkcs5"
}
```

4.8.7 Validating the Policy Download by the Publisher Containers

This section describes the commands to validate whether the PEP Server container has downloaded the policy from the ESA.

To get the *pty-publisher-pepserver* logs, execute the following command.

```
kubectl logs -n <<namespace>> <<PUBLISHER_POD_NAME>> -c pty-publisher-pepserver
```

For example, here is the sample output after executing the command. This output indicates that the connectivity to the ESA has been established and the PEP Server container has downloaded the policy from the ESA.

```
2023-08-30 07:47:47,952:pep_installer:INFO:Installing PepServer...

2023-08-30 07:47:47,952:pep_installer:DEBUG:Executing command: ['/bin/bash', '/opt/protegrity/
PepServerSetup_Linux_x64_1.1.0+99.g9521a.1.1.sh', '-dir', '/var/data/staging', '-esa',
'10.21.0.99', '-certuser', 'imp', '-certpw', '********']

2023-08-30 07:47:48,231:pep_installer:DEBUG:Unpacking...

Extracting files...

Downloading certificates from 10.21.0.99:8443...

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current

                                 Dload  Upload   Total   Spent    Left  Speed

100 30720  100 30720    0     0   188k      0 --:--:-- --:--:-- --:--:--  189k


Extracting certificates...

Certificates successfully downloaded and stored in /var/data/staging/defiance_dps/data


Protegrity PepServer installed in /var/data/staging/defiance_dps.
```

To get the *pty-publisher-metaserver* logs, execute the following command.

```
kubectl logs -n <<namespace>> <<PUBLISHER_POD_NAME>> -c pty-publisher-metaserver
```

For example, here is the sample output after executing the command.

```
[INFO ] 2023/08/30 07:47:48.203159 server.go:21: starting server at 0.0.0.0:11080

[DEBUG] 2023/08/30 07:48:09.301992 metainfoprovider.go:17: reading policy metadata from /var/
data/tmpfs/metadata.json
```

If you are storing the policy package in an NFS server, then you can run the following command to navigate to the following location to verify whether the policy package is available.

```
cd /<Path specified in the PVC>/<Path specified while installing the Publisher Pod>
```

If you are storing the policy package in AWS S3 bucket, then you can navigate to the bucket location to verify whether the policy package is available.

4.9 Uninstalling the DSG Container

This section describes the steps how to uninstall the DSG Container.

➤ To uninstall the DSG Container, perform the following steps.

1. Run the following command to uninstall the Helm charts.

   ```
   helm uninstall publisher dsg
   ```

2. Run the following command to delete the pods.

   ```
   kubectl delete pods <POD_Name1> <POD_Name2> .... <POD_NameN> --force
   ```

3. Run the following command to delete the Persistent Volume Claim.

   ```
   kubectl delete pvc <PVC_Name>
   ```

4. Run the following command to delete the Persistent Volume.

   ```
   kubectl delete pv <PV_Name>
   ```

5. Run the following command to delete the namespace.

   ```
   kubectl delete namespace <Namespace>
   ```

6. Run the following command to remove all the Docker images from the host node.

   ```
   docker rmi $(docker images -q) -f
   ```

Chapter 5

Installing the DSG Container on Azure Kubernetes Service (AKS)

This section describes the procedures to install the DSG Container on Azure.

The DSG Container will be used in combination with the Enterprise Security Administrator (ESA).

5.1 Verifying the Prerequisites

Ensure that the following prerequisites are met:

1. Verify the reference solution deployment
2. Verify the prerequisites for the reference application
3. Verify the prerequisites that are required on Azure

5.1.1 Verifying the Prerequisites for Reference Solution Deployment

Reference Solution Deployment

- *ESA* – Protegrity provides an AWS image *ESA_PAP-ALL-64_x86-64_AWS_9.1.0.5.xxxx*. Use this image to launch an ESA instance on AWS.
  - After launching the ESA instance, you must install the *ESA_PAP-ALL-64_x86-64_9.1.0.5.<Build_Number>.DSGUP.pty* patch to extend the DSG Web UI on the ESA.
  - Install the *DSG_PAP-ALL-64_x86-64_3.1.0.5.<Build_Number>.FE-1* patch to extend the DSG Web UI on the ESA.

  For more information about installing the DSG 3.1.0.5 FE-1 patch, refer to section *Installing the DSG* in the *Data Security Gateway Guide 3.1.0.5*.

- *DSG_RHUBI-ALL-64_x86-64_Generic.K8s_<Version>.tgz* – *Installation package* for the Metaserver-based Deployment.

5.1.2 Verifying the Prerequisites for Reference Application

Reference Application

This section describes the prerequisites for using the reference application:

- Policy – Ensure that you have defined the security policy on the ESA. In addition, ensure that you fulfill either of the following requirements:
  - You must link the policy to the default data store in the ESA.
  - You must add the Kubernetes node or pod IP address ranges as allowed servers on the data store to which the policy is linked.
- Configuration over Programming (CoP) – The Protegrity Reference Deployment uses the default **REST API Example** as the CoP.
- Client application – For example, the banking application, which contains the customer data that you want to protect using the DSG application.

5.1.3 Verifying Prerequisites for Azure

The following additional prerequisites for Azure are required:

Table 5-1: Azure Prerequisites

Prerequisite	Description
Linux Instance	This instance can be used to communicate with the Kubernetes cluster. This instance can be on-premise or on Azure. Ensure that Helm is installed on this Linux instance. You can also choose to install Docker on this Linux instance to communicate with the Container Registry, where you want to upload the Docker images.

Azure Subscription and a valid Azure Active Directory User	Access to an Azure subscription and a valid Azure Active Directory User is required for creating the following Azure services:
	• Resource Group - Used to organize resources. All the remaining Azure services are created within a Resource Group.
	• Azure Storage Account - Used for creating the Azure Storage Container or the Azure File Share, where the immutable policy and COP package will be uploaded.
	• Azure Storage Container - This prerequisite is optional, and is required only if you want to use the Azure Storage Container for storing the policy and COP package, instead of the Azure File Share.
	• Azure File Share - This prerequisite is optional, and is required only if you want to use the Azure File Share for uploading the immutable policy and COP package, instead of using an Azure Storage Container.
	• Two service principals:
	  • **Service principal 1** with *Read* permissions on the Azure Storage Account and *Decrypt* permission on Key Management of the Azure Key Vault.
	  > **Note:** The *Decrypt* permissions are required if you want to use the Azure Key Vault for encrypting the policy and COP package, instead of using the passphrase-based encryption.

Prerequisite	Description
	• **Service principal 2** is used to generate the access token for authenticating the requests that are sent from a REST API client to a DSG instance for performing security operations. **Note:** A service principal is created when you register an application on the Azure portal. **Note:** Ensure that the user has permissions to get the Azure Key Vault Public Key and to upload to an Azure Storage Container.
Permissions	Ensure that the following permissions are available: • Azure Kubernetes Service (AKS) to create a Kubernetes cluster • Azure Key Vault to create a key used in encryption and decryption operation • Azure Container Registry to upload the IMP Updater and DSG Container images
KMS Key	Create KMS key if you want to use it for Policy encryption.

5.2 Initializing the Linux Instance

After downloading the installation packages, you must initialize the Linux instance. This involves installing the tools and utilities in the order as shown in the following table.

Table 5-2:

Order	Tools Utilities	Description	Notes	References
1	Azure CLI	Provides a set of command line tools for the Azure Cloud Platform.	Configure Azure CLI on your machine by running the following command. `curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash`	For more information about installing Azure CLI on the Linux instance, refer to *https://learn.microsoft.com/en-us/cli/azure/install-azure-cli*.
2	*kubectl*	Command line interface for Kubernetes. Kubectl enables you to run commands from the Linux instance so that you can communicate with the Kubernetes cluster.	Install Kubectl by running the following command. `az aks install-cli`	For more information about installing *kubectl*, refer to the section *Installing kubectl*.
3	Helm Client		Run the following commands sequentially to install the Helm client on the Linux instance. 1. `curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/`	For more information about installing a Helm client, refer to *https://helm.sh/docs/intro/install/*. **Important:** Verify the version of the Helm client that must be used from the

Order	Tools Utilities	Description	Notes	References
			`helm/helm/main/scripts/get-helm-3` 2. `chmod 700 get_helm.sh` 3. `./get_helm.sh`	Readme provided with this build.

5.3 Uploading the Images to the Container Repository

This section describes how you can upload the DSG container image to the Container Repository.

Before you begin
Ensure that you setup the Container Registry.

> **Note:**
> The steps listed in this section for uploading the container images to the Azure Elastic Container Repository (ECR) are for reference use. You can choose to use a different Container Registry for uploading the container images.

➤ To upload the PEP server, Meta server, DSG, and Sidecar container images:

1. Install Docker on the Linux instance.
   For more information about installing Docker on a Linux machine, refer to the *Docker documentation*.
2. Run the following command to authenticate your Docker client with Azure.

```
az login --scope https://management.core.windows.net//.default
sudo docker login sampleacr.azurecr.io --username <acr username> --password <password>
```

> **Note:** For more information about authenticating your Docker client with Azure Container Registry, refer to the *https://learn.microsoft.com/en-us/azure/container-registry/container-registry-authentication* Azure Authentication documentation.

3. Run the following command to load the PEP Server container image on Docker.

```
docker load -i PUBLISHER-PEPSERVER_RHUBI-9-64_x86-64_K8S_<<Version>>.tar.gz
```

4. Run the following command to list the PEP Server container image.

```
docker images
```

5. Tag the image to the Azure Container Registry by running the following command.

```
docker tag <Container image>:<Tag> <Container registry path>/<Container image>:<Tag>
```

   For example:

```
docker tag <<LOADED_IMAGE>>:<<TAG>> <azure container registry name>.azurecr.io/images/
publisher-pepserver:latest
```

For more information about tagging an image, refer to the *Azure* documentation.

6. Push the tagged image to the Azure Container Registry by running the following command.

```
docker push <Container registry path>/<Container image>:<Tag>
```

For example:

```
docker push <azure container registry name>.azurecr.io/images/publisher-pepserver:latest
```

7. Repeat steps 4 to 7 for pushing the Meta server, DSG, and Sidecar container images to the Azure Container Registry.
8. Navigate to the directory where you have extracted the Helm charts package.
9. In the *values.yaml* file in the *pty-publisher* directory, update the required repository path and tag for the following entries:
   - PEP server image in the *pepServerImage* setting.
   - Meta server image in the *metaServerImage* setting.
10. Navigate to the directory where you have extracted the Helm charts packages for DSG container.
11. In the *values.yaml* file in the *dsg*, update the required repository path and tag for the following entries:
    - DSG image in the *dsgImage* setting.
    - Sidecar image in the *sidecarImage* setting.

5.4 Creating and Uploading the CoP to Azure Storage Container

This section describes how you can create a Ruleset on the ESA. In addition, this section describes how you can export a CoP from the ESA.

Important: Disable the Learn Mode on the DSG as it is not applicable to the containerized application.

5.4.1 Creating and Exporting the Ruleset

This section describes how you can create a Ruleset on the ESA.

To create a Ruleset:

1. On the ESA Web UI, create a Configuration Over Programming (CoP) Ruleset.
   By default, the ESA provides you with default ruleset *REST API Examples* for testing the DSG functionality.

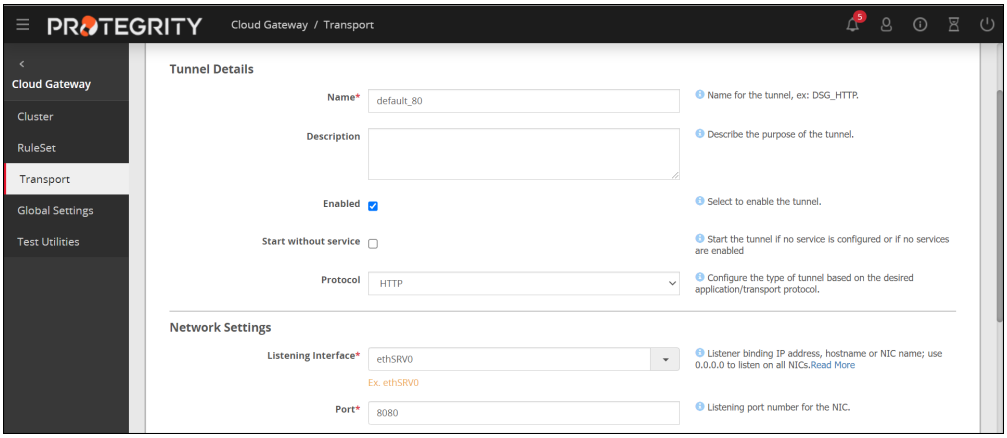
   For more information about creating a CoP Ruleset, refer to the section *Ruleset Menu* in the *Data Security Gateway User Guide 3.1.0.5.*

2. Navigate to **ESA Web UI** > **Cloud Gateway** > **Transport** > **Tunnels** > **Create Tunnel**.
   For more information about creating a tunnel, refer to the section *Tunnels tab* in the *Data Security Gateway User Guide 3.1.0.5*.

   Important:

Create a tunnel with port number higher than *1024* as the Docker containers do not allow any external communication on port numbers below *1024*.

The following figure shows a sample tunnel that uses the HTTP protocol on port *8080* using the service interface *ethSRV0*.



Note:

Ensure that any other tunnel that uses a port number lower than *1024* is disabled.

Note:

The NFS/CIFS tunnel and GPG transformations are not supported in the DSG containers.

Important:

Ensure that the port number that you specify in the **Port** field matches the value that you specify in the *targetPort* parameter in the *values.yaml* file under the **dsgService** > **tunnels** section.

For more information about the *values.yaml* file parameters, refer to the section *Values.yaml*.

3. If you want to use the transaction metrics, then in the **Advanced Settings** section, specify the following value to support the X-Forwarded-For (XFF) header in the REST API requests sent to the DSG instances.

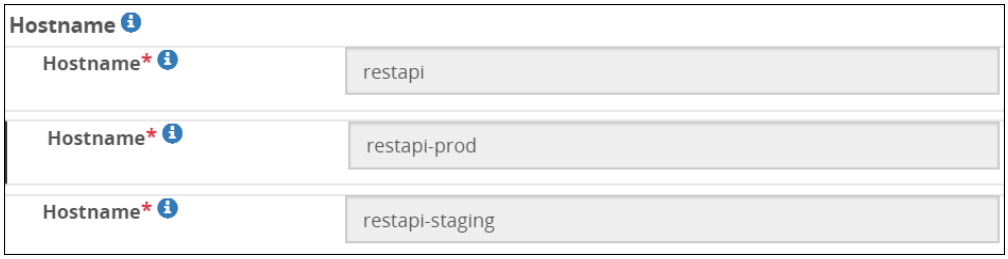
```
{
"xheaders": true
}
```

Figure 5-1: Advanced Settings

The XFF header is used to identify the client machine from which the REST API requests originate.

4. Click **Create** to create the tunnel.
5. On the ESA Web UI, navigate to **Cloud Gateway** > **RuleSet**.
   By default, the DSG provides you default rulesets for testing the DSG functionality.

6. If you want to specify multiple hostnames for the same ruleset, then perform the following steps.
   a. On the right-hand side screen, click **Edit**.
   b. Click the icon next to the **Hostname** field.
   c. Specify the required *hostname* in the **Hostname** fields.



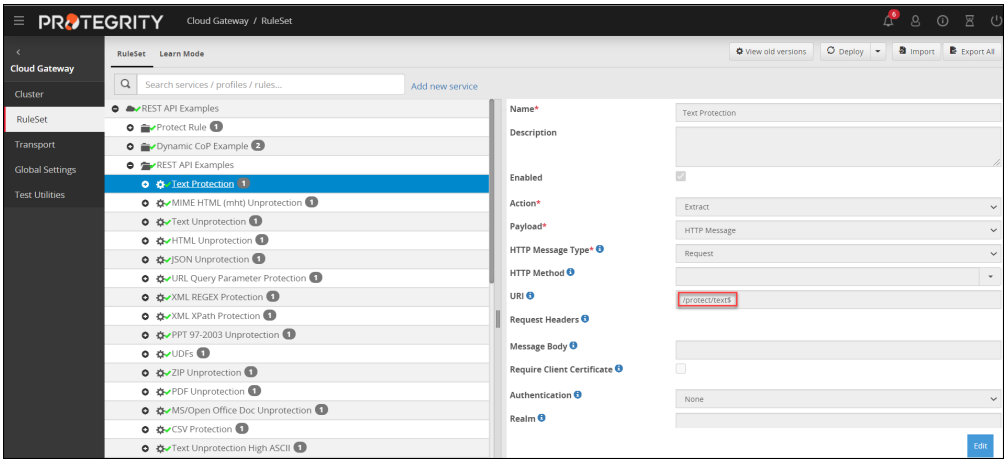
   You can use the multiple hostname option to run production URLs on one hostname and staging URLs on another hostname. If you want to use multiple hostnames, then you must add the same hostnames in the **hosts** > **host** field of the *values.yaml* file.

   For more information about the *values.yaml* file parameters, refer to the section *Values.yaml*.

7. Navigate to **REST API Examples** > **Protect Rule** > **Data Protection**.
8. On the right-hand side screen, click **Edit** and specify the data element name in the **DataElement Name** field.

> **Note:**
> Ensure that this data element is part of the security policy that you have created.

9. Click **Save**.
10. Expand **REST API Examples** and navigate to the Text Protection ruleset. Note the URI.
    The Postman client is used to send a request to the following location:
    http://<hostname>/protect/text



> **Note:** The Text Protection rule is used to check the health of the pods. Leave the **HTTP Method** text box blank or enter the **GET** method to perform this check.

11. Perform the following steps to enable the basic authentication functionality, which ensures that the user who is performing the security operations, is authenticated as part of an existing LDAP. The username and password of the user are sent as part of the REST API request to authenticate the user.

> **Important:** Ensure that the user that is utilized for authentication is added to the Active Directory, and assigned a role that includes the *Cloud Gateway Auth* permissions.
>
> You can also use the ESA LDAP for adding users.
>
> For more information about managing users in the ESA, refer to the section *Managing Users* in the *Appliances Overview Guide 9.1.0.5*.
>
> For more information about assigning permissions to user roles in the ESA, refer to the section *Managing Roles* in the *Appliances Overview Guide 9.1.0.5*.

For more information about the basic authentication functionality, refer to the section *Basic Authentication* in the *Data Security Gateway Guide 3.1.0.5*.

a. Navigate to the **Cloud Gateway** > **RuleSet** tab.
b. Select the required rule.
c. In the **Authorization** list, select **Basic**.

   For more information about enabling the basic authentication functionality, refer to the section *Enabling REST API Authentication* in the *Data Security Gateway Guide 3.1.0.5*.
d. Save the changes.
e. Click **Deploy** to push the configurations to all the DSG nodes.

> **Note:** The user can either click **Deploy** or **Deploy to Node Groups** on the Ruleset page. For more information about the deploy functionality, refer to the section *RuleSet Tab* in the *Data Security Gateway User Guide 3.1.0.5*.

12. To export the CoP package from the ESA and encrypt the policy package using a passphrase and a salt, run the following command.

CURL request syntax for exporting CoP

```
curl --location --request POST 'https://<ESA_IP_address>/exportGatewayConfigFiles' \
--header 'Authorization: Basic <Base64 encoded administrator credentials>' \
--header 'Content-Type: text/plain' \
--data-raw '{
"kek":{
"pkcs5":{
"passphrase": "<passphrase>",
"salt": "<salt>"
}
}
}' -k -o <Response file name in .json format>
```

Sample CURL request for exporting CoP

```
curl --location --request POST 'https://xxx.xxx.xxx.xxx/exportGatewayConfigFiles' \
--header 'Authorization: Basic <Base64 encoded administrator credentials>' \
--header 'Content-Type: text/plain' \
--data-raw '{
"kek":{
"pkcs5":{
"passphrase": "xxxxxxxxxx",
"salt": "salt"
}
```

```
}
}' -k -o cop_demo.json
```

Note: The response for the CURL request will be stored in *cop_demo.json* file.

Note: For more information about exporting a CoP, refer to the section *Appendix H: CoP Export API for deploying the CoP (Containers only)* in the *Data Security Gateway User Guide 3.1.0.5*.

13. If you want to upload the CoP package to an Azure Storage Container, then perform the following steps.
    a. Login to the Azure environment.
    b. Create Azure Storage Container.
    c. Upload the CoP package to an Azure Storage Container.

    Important: Ensure that the user has the required permissions to upload an object to the Azure Storage Container.

14. If you want upload the CoP package to an Azure File Share, then perform the following steps.
    a. Login to the Azure environment.
    b. Create an Azure File Share instance.

    For more information about creating an Azure File Share instance, refer to *Creating an Azure File Share*..

    After you have created a file system on the Azure File Share, you need to mount this file system to the Linux instance that has been created in the Azure environment.

    c. Navigate to the Azure File Share that you have created.
    d. Click **Upload**.
    e. In the **Files** field, browse to the location where you have downloaded the CoP package file, and select the file.
    f. Click **Upload** to upload the CoP package file to the Azure File Share.

5.5 Creating the Cloud Runtime Environment

This section describes how to create the Cloud runtime environment.

5.5.1 Creating the Azure Runtime Environment

This section describes how to create the Azure runtime environment.

Prerequisites

Before creating the runtime environment on Azure, ensure that you have a valid Azure account and the following information:

- Login URL for the Azure account
- Authentication credentials for the Azure account

Audience

It is recommended that you have working knowledge of Azure and knowledge of the following concepts:

- Introduction to Azure Storage
- Introduction to Azure Cybersecurity
- Introduction to Azure Kubernetes Service

You can follow the steps to setup the Azure cloud environment as described in the following table.

Table 5-3: Azure Cloud Environment Setup

Steps	References
Logging in to the Azure Environment	For more information about logging in to the Azure environment, refer to the section *Logging in to the Azure Environment*.
Registering an Application	For more information about registering an application in the Azure Active Directory, refer to the section *Registering an Application*.
Creating an Azure Storage Account	For more information about creating an Azure Storage Account, refer to *Creating an Azure Storage Account*.
Creating an Azure Storage Container	For more information about creating an Azure Storage Container, refer to *Creating an Azure Storage Container*.
Creating an Azure File Share	For more information about creating an Azure File Share, refer to *Creating an Azure File Share*.
Creating a Kubernetes Cluster	For more information about creating a Kubernetes Cluster, refer to the section *Creating Kubernetes Cluster*.

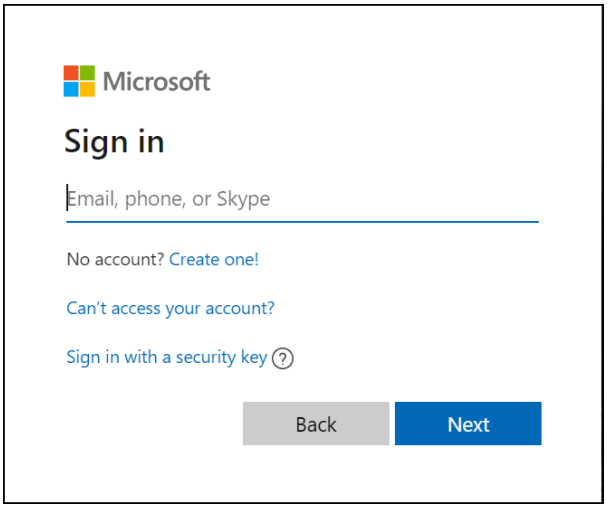
5.5.1.1 Logging in to the Azure Environment

This section describes how you can login to the Azure environment.

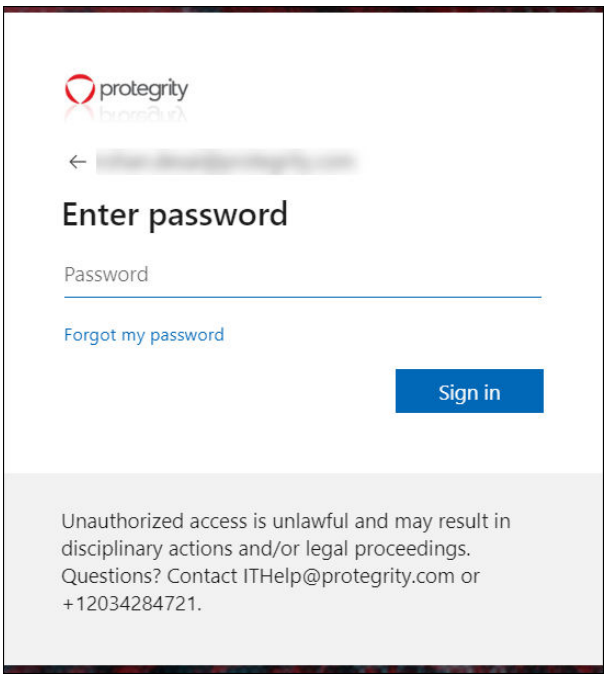
➤ To login to the Azure environment:

1. Access Azure using the following URL:
   https://azure.microsoft.com
   The **Microsoft Azure** home screen appears.

2. Click **Sign in**.
   The **Microsoft Sign in** screen appears.



3. On the **Microsoft Sign in** screen, enter the email address for logging in to Azure, and then click **Next**.
   The **Enter Password** screen appears.



4. Enter the password for authenticating your email address and click **Next**.
   After successful authentication, you are logged in to Microsoft Azure. The Azure home screen appears.
5. Click **Portal**, which is on the top-right side of the Azure home screen.
   The Azure Portal home screen appears. By default, the home screen displays a list of frequently used Azure services.

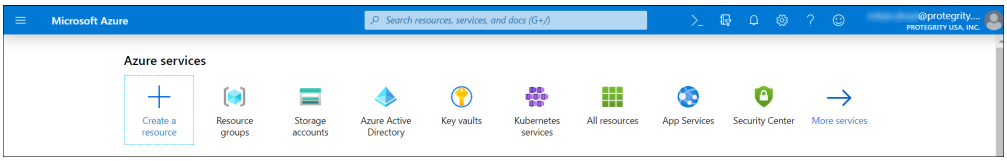
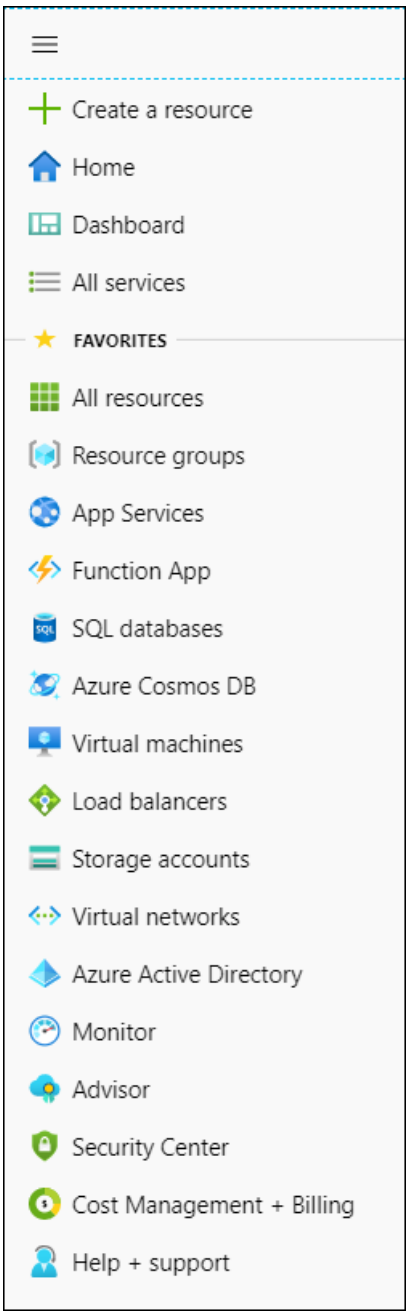
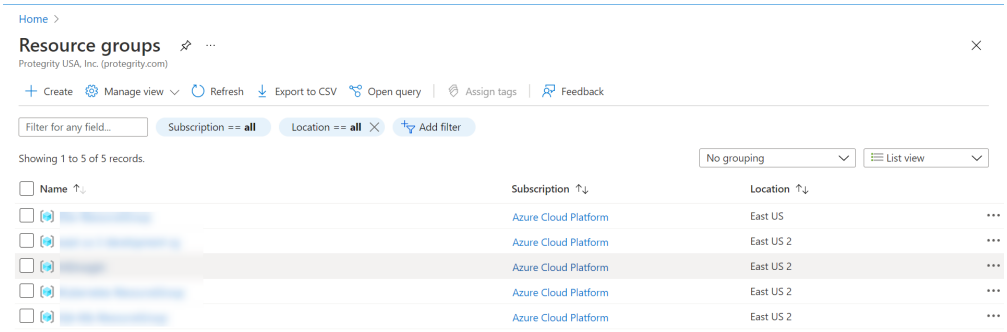


Figure 5-2: Azure Portal Home Screen

6. Click the **Portal** menu icon ( ).
   The **Portal** menu appears.



7. Click **Resource groups**.
   The **Resource groups** screen appears.



8. Select the required resource group from the available list.

> **Note:**

> If a resource group is not available, then contact your IT administrator for creating a resource group.

5.5.1.2 Registering an Application

This section describes how you can register an application in the Azure Active Directory. After you register an application, a corresponding service principal is automatically created. You then need to generate the secret for the service principal.

➤ To register an application:

1. Login to the Azure environment.
   For more information about logging in to the Azure environment, refer to the section *Logging in to the Azure Environment*.
2. Click the **Portal** menu icon ( ).
   The **Portal** menu appears.
3. Navigate to **Azure Active Directory**.
   The **Overview** screen of the Azure Active Directory appears.

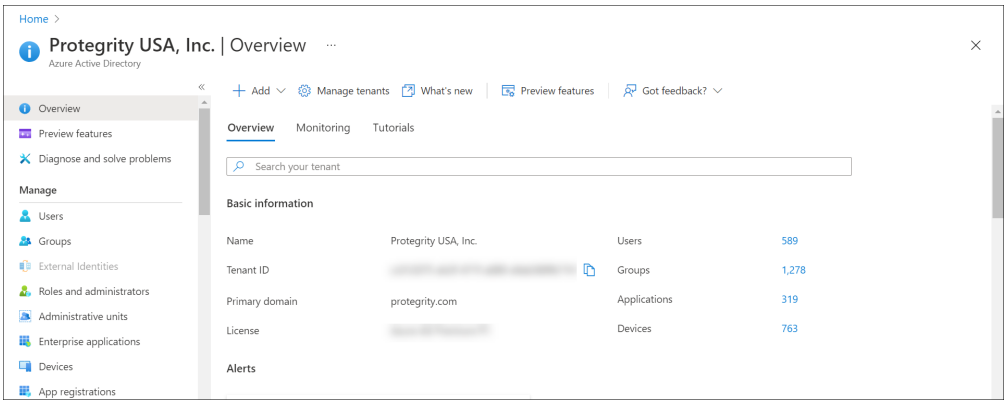


Figure 5-3: Azure Active Directory Overview Screen

4. Navigate to **App registrations**.
   The **App Registrations** screen appears.

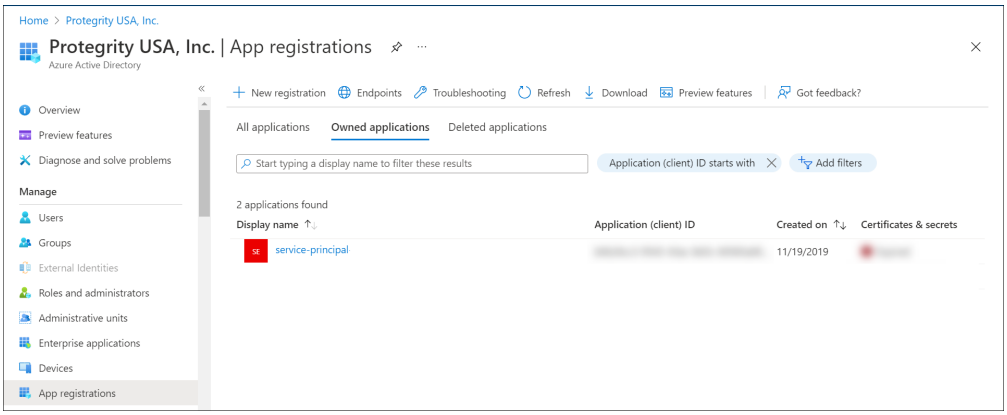


Figure 5-4: App Registrations Screen

5. Click **New registration**.

The **Register an application** screen appears.

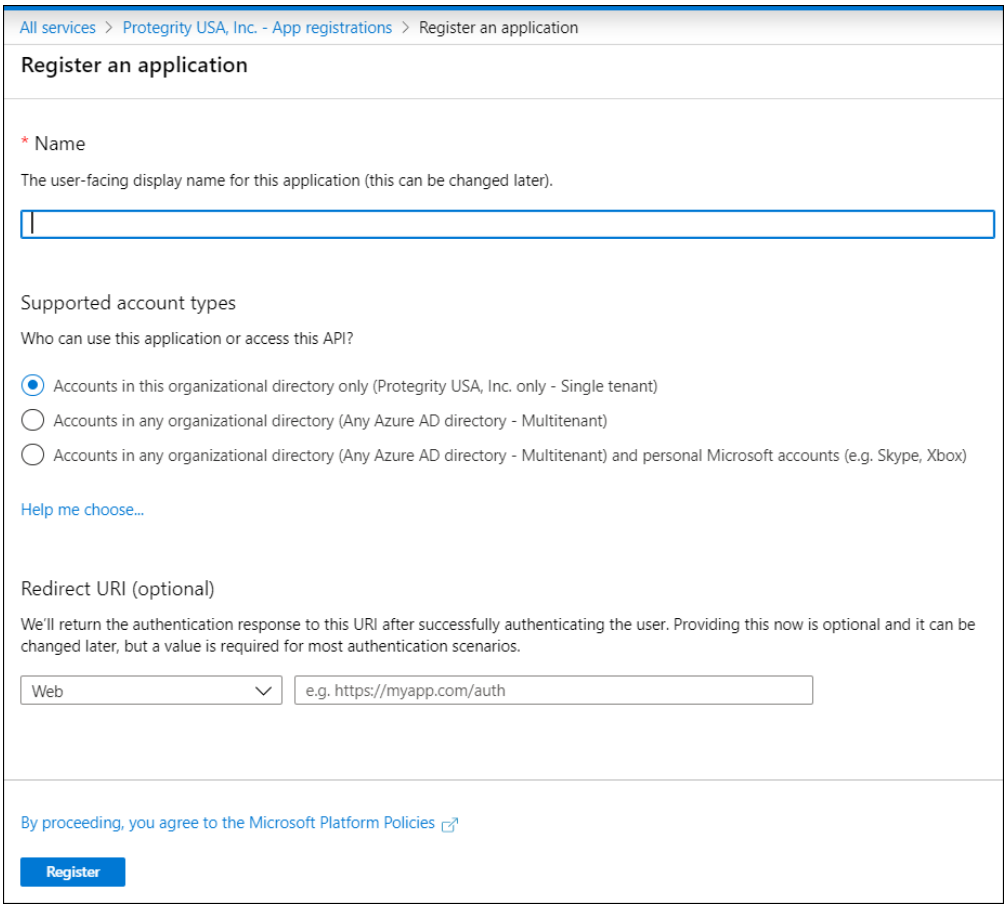
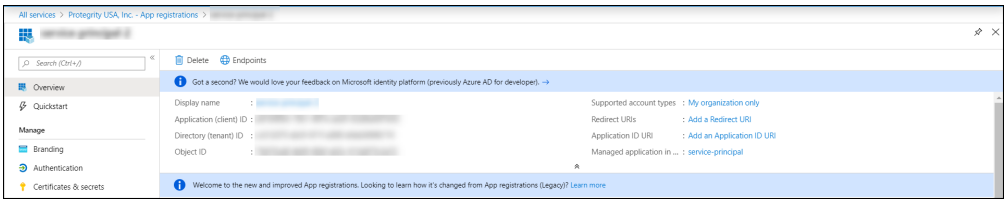


Figure 5-5: Register an Application Screen

6. In the **Name** field, specify a name for the service principal.
   For example, specify **service-principal-1**.
7. In the **Supported account types** field, retain the default option for single tenant.
8. Click **Register** to register your application.
   The application is registered and the **Overview** screen appears, displaying the details about the registered application.



> **Important:** Note the values of the **Application (client) ID** and **Directory (tenant) ID**. You need to specify these values in the credentials file for the corresponding service principal.

9. Perform the following steps to create a client secret for the registered application.
   a. On the left pane, click **Manage** > **Certificates & secrets**.
      The **Certificates & secrets** screen appears.

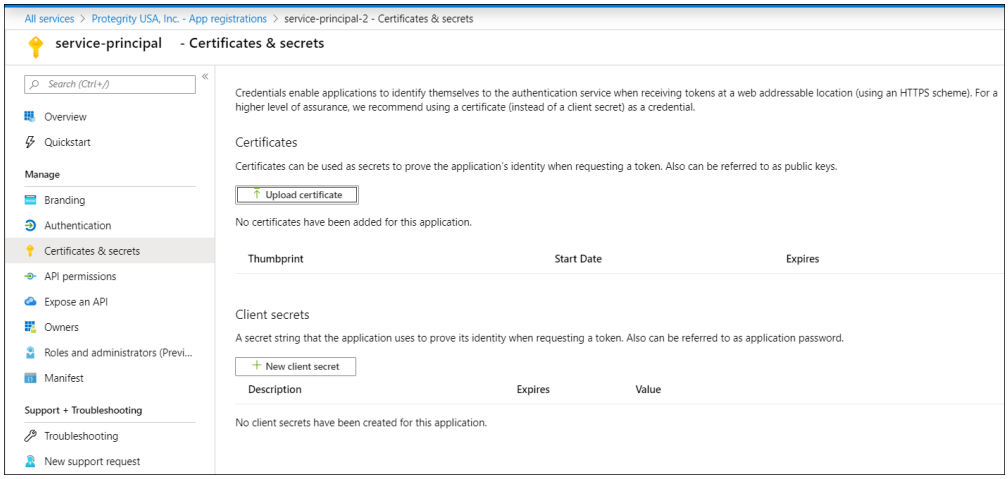


Figure 5-6: Certificates & Secrets Screen

b. Click **New client secret**.

The **Add a client secret** screen appears.

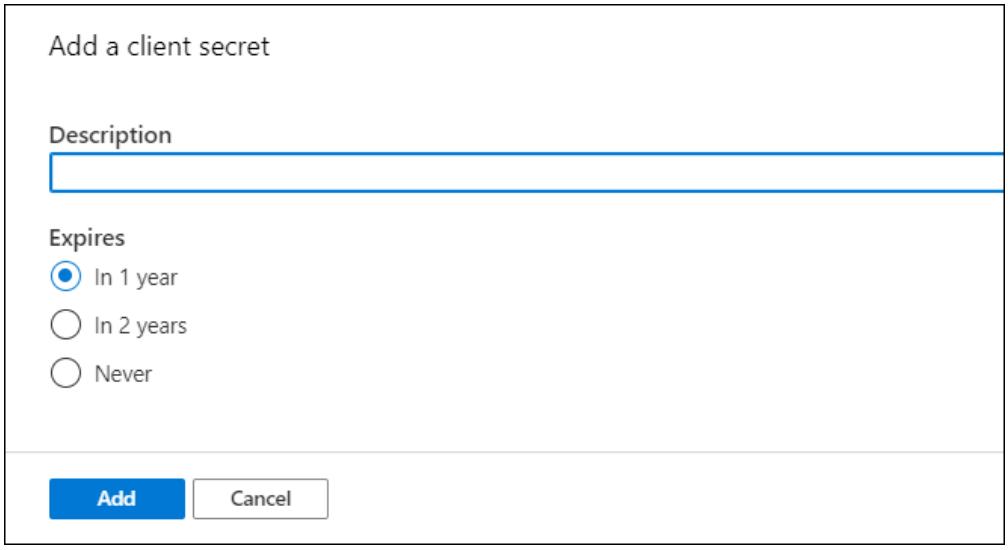


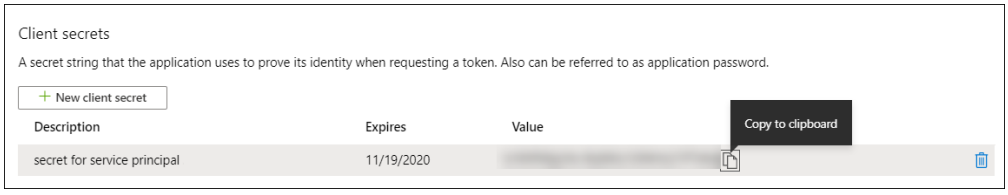
Figure 5-7: Add a Client Secret Scren

c. In the **Description** field, specify a description for the client secret.

d. In the **Expires** field, retain the default value.

e. Click **Add**.

The client secret for the registered application appears in the **Client secrets** section.



f. Click the Copy to clipboard icon next to the **Value** column to copy the value of the client secret.

> **Important:**
>
> Copy the client secret value to a secure location. You need to specify this value in the credentials file for the corresponding service principal.

10. Repeat *step 3* to *step 7* to register another application for creating service principal 2.

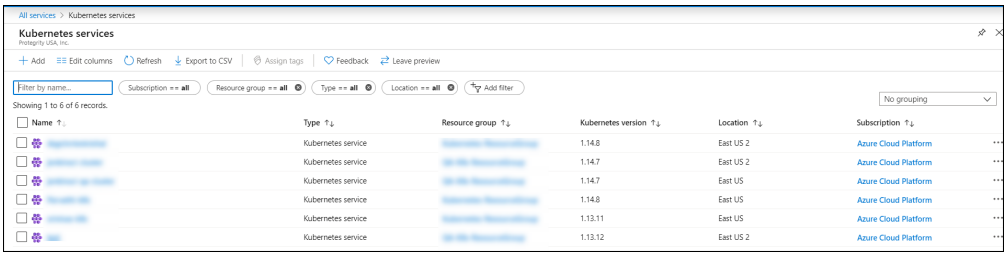
5.5.1.3 Creating a Kubernetes Cluster

This section describes how to create a Kubernetes Cluster on Azure.

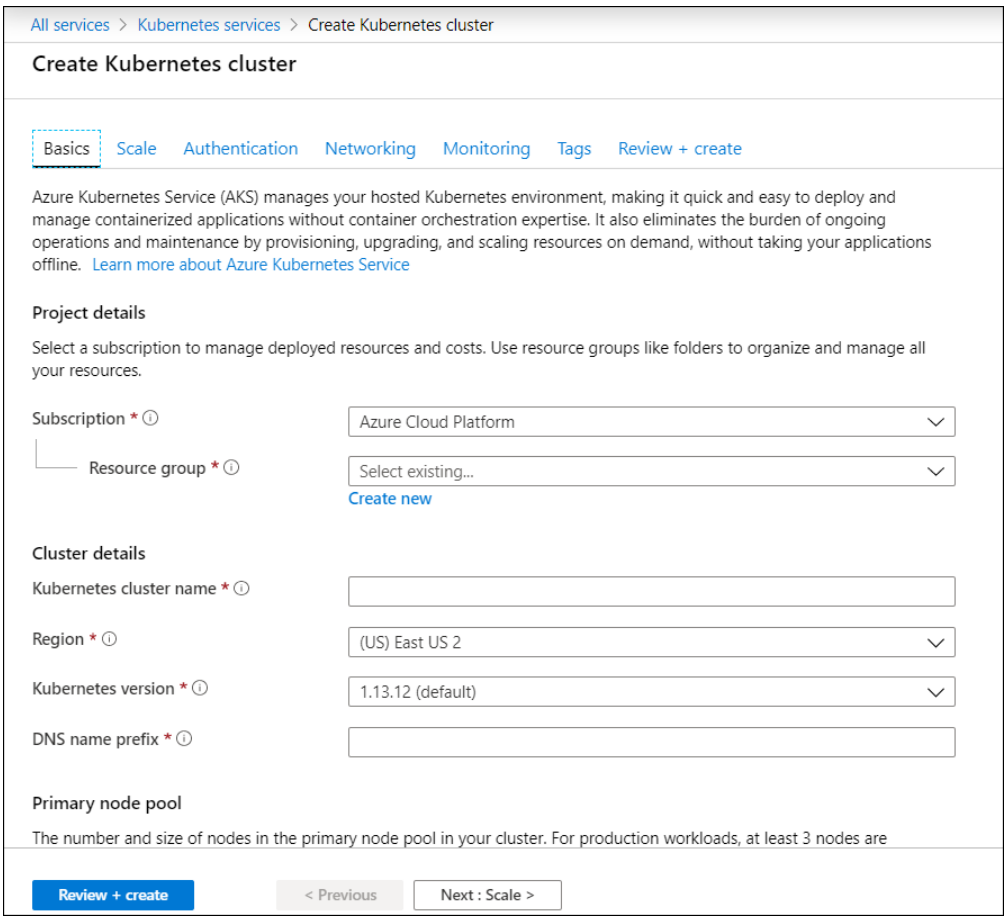
> **Note:**
> The steps listed in this procedure for creating a Kubernetes cluster are for reference use. If you have an existing Kubernetes cluster or want to create a Kubernetes cluster based on your requirements, then you can directly navigate to *step 9* to connect your Kubernetes cluster and the Linux instance. However, you must ensure that your DSG service port is enabled on the Network Security group of your VPC.

➤ To create a Kubernetes cluster:

1. Login to the Azure environment.
   For more information about logging in to the Azure environment, refer to the section *Logging in to the Azure Environment*.
2. Click the **Portal** menu icon ( ).
   The **Portal** menu appears.
3. Navigate to **All Services** > **Kubernetes services**.
   The **Kubernetes Services** screen appears.



4. Click **Add**.
   The **Create Kubernetes cluster** screen appears.



5. In the **Resource group** field, select the required resource group.
6. In the **Kubernetes cluster name** field, specify a name for your Kubernetes cluster.

   Retain the default values for the remaining settings.
7. Click **Review + create** to validate the configuration.
8. Click **Create** to create the Kubernetes cluster.

   The Kubernetes cluster is created.
9. Login to the Linux instance, and run the following command to connect your Linux instance to the Kubernetes cluster.

   az aks get-credentials --resource-group <Name of Resource Group> --name <Name of Kubernetes Cluster>

   The Linux instance is now connected with the Kubernetes cluster. You can now run commands using the Kubernetes command line interface (*kubectl*) to control the nodes on the Kubernetes cluster.
10. Validate whether the cluster is up by running the following command.

    kubectl get nodes

    The command lists the Kubernetes nodes available in your cluster.

5.6 Extracting the Package

This section provides information about extracting the installation package to verify that the individual components that are essential for the DSG Container installation are available.

1. Create a directory to download the package using the following command.

```
mkdir packaging
```

2. Navigate to the directory *packaging* using the following command.

```
cd packaging
```

3. Download the *DSG_RHUBI-ALL-64_x86-64_Generic.K8s_<Version>.tgz* package on the base machine.
4. Extract the files from the package using the following command.

```
tar -xvf DSG_RHUBI-ALL-64_x86-64_Generic.K8s_<Version>.tgz
```

The following is the list of the packages that are extracted:

- Helm charts and container images
- Additional packages and directories

Table 5-4: List of the Helm Charts and Container Images

List of Helm Charts and Docker Images	Description	Package Name
DSG container image	It is used to create the DSG container.	*DSG_K8S-ALL-64_x86-64_<Version>.tar.gz*
Sidecar image	It is used to create the Sidecar container.	*SUBSCRIBER-SIDECAR_RHUBI-8-64_x86-64_K8S_<Version>.tar.gz*
Helm Charts for the DSG Container	It is a package which contains the Helm Charts for deploying the Webhook container.	*DSG-HELM-DYNAMIC_ALL-ALL-ALL_x86-64_K8S_<Version>.tgz*
Meta server image	It is used to create the MetaServer container.	*PUBLISHER-METASERVER_RHUBI-8-64_x86-64_K8S_<Version>.tar.gz*
PEP server image	It is used to create the PEP Server container.	*PUBLISHER-PEPSERVER_RHUBI-8-64_x86-64_K8S_<Version>.tar.gz*
Helm Charts for the Meta server and PEP server	It is a package that contains the Helm Charts used to deploy the MetaServer and PEP Server containers.	*PUBLISHER-HELM_ALL-ALL-ALL_x86-64_K8S_<Version>.tgz*
Source package for the PEP server container	It is a package containing the Dockerfile that can be used to create a custom image for the PEP server container and the associated binary files.	*PUBLISHER-PEPSERVER_SRC_<Version>.tgz* For more information about building a custom image using the Dockerfile, refer to the section *Creating Custom Images Using Dockerfile*.
Source package for the Meta server container	It is a package containing the Dockerfile that can be used to create a custom image for the Meta server container and the associated binary files.	*PUBLISHER_METASERVER_SRC_<Version>.tgz* For more information about building a custom image using the Dockerfile, refer to the section *Creating Custom Images Using Dockerfile*.
Source package for the DSG containers	It is a package containing the Dockerfiles that can be used to create a custom image for the DSG containers and the associated binary files.	*DSG_K8S-ALL-64_x86-64_SRC_<Version>.tgz* For more information about building a custom image using the Dockerfile, refer to the section *Creating Custom Images Using Dockerfile*.
Source package for the Sidecar containers	It is a package containing the Dockerfiles that can be used to create a custom image for the Sidecar containers and the associated binary files.	*SIDECAR_SRC_<Version>.tgz* For more information about building a custom image using the Dockerfile, refer to

List of Helm Charts and Docker Images	Description	Package Name
		the section *Creating Custom Images Using Dockerfile*.
HOW-TO-BUILD-DOCKER-IMAGES	Text file specifying how to build the Docker images.	

Table 5-5: List of the Additional Packages and Directories

List of Additional Packages and Directories	Description
DSG-Samples_Linux-ALL-ALL_x86-64_<Version>.tgz	It includes the following files and directories: • *Sample_App_COP_V5.zip* - Sample COP package. • *Sample_App_Policy_V5.tgz* - Sample policy. • *Sample_App_EKSCreateCluster.yaml* - Sample YAML file used to create the Kubernetes cluster.
certs	Contains the *cnf* file to create the certificates required for secure communication.

5.7 Setting Up the Certificates

This section describes how you can set up the CA and TLS certificates for self-signed secure communication and communication between the Metserver and Sidecar.

➤ To set up certificates for secure communication:

1. Navigate to the *packaging/certs* directory using the following command.

   ```
   cd packaging/certs
   ```

   The certs directory contains the following files:

   - *example_ca.cnf* - File used to create the CA certificate
   - *example_svc.conf* - File used to create the TLS certificate
   - *README* - Readme file listing the commands to be executed for generating the CA and TLS certificates

2. Run the following command to create the CA certificate.

   ```
   openssl req -nodes -new -x509 -sha256 -newkey rsa:4096 -keyout ca_key.pem -out ca.pem \ -days 1825 -config example_ca.cnf
   ```

   The following files are created in the *certs* directory.

   - *ca.pem*
   - *ca_key.pem*

   The CA certificate is used to generate the TLS certificate in *step 3c*.

   The CA certificate is also used by the sidecar container to communicate with the PTY-Publisher pod.

3. Perform the following steps to generate the TLS certificate that can be used in both the Publisher deployments.

   a. Create a copy of the *example_svc.conf* file using the following command.

   ```
   cp example_svc.conf example_svc_1.conf
   ```

b. Edit the *example_svc_1.conf* file to update the hostname of the Publisher service that you want to deploy.

```
[alt_names]
DNS.1 = $PUBLISHER_METASERVER_SVC_HOSTNAME
```

> **Important:**
> The Publisher Service Hostname is not available until you deploy the Publisher Helm chart. It is formed from the following three components.
>
> - The Helm installation name.
> - Helm chart name.
> - The Kubernetes namespace, where you want to install the Helm chart.
>
> In this scenario, the TLS certificate is created before the Publisher Helm chart is deployed or the Publisher Service Hostname is available.

You can obtain the host name using the following method:

Hostname = ${HELM_INSTALLATION_NAME}-${HELM_CHART_NAME}.${NAMESPACE}.svc

Hostname = ${SERVICE_NAME}.${NAMESPACE}.svc

> **Note:**
> You need to identify the name that you are planning to use for the Publisher deployments, before the actual deployment.

For example:

```
export PTY_PUBLISHER_HELM_INSTALL_NAME=publisher
export PTY_PUBLISHER_HELM_CHART_NAME=pty-publisher
export WAREHOUSE_NAMESPACE=test_demo
export PUBLISHER_METASERVER_SVC_HOSTNAME = ${PTY_PUBLISHER_HELM_INSTALL_NAME}-$
{PTY_PUBLISHER_HELM_CHART_NAME}.${WAREHOUSE_NAMESPACE}.svc
```

```
echo $PUBLISHER_METASERVER_SVC_HOSTNAME

publisher-pty-publisher.test_demo.svc
```

c. Run the following command to create the TLS certificate.

```
openssl req -x509 -nodes -newkey rsa:4096 -days 1825 -CA ca.pem -CAkey ca_key.pem
\ -keyout key.pem -out certificate.pem -config example_svc_1.cnf
```

The *certificate.pem* and *key.pem* files are available in the *certs* directory.

5.8 Working with the Publisher Deployment

The *pty-publisher* pod hosts two containers - PEP server and Meta server.

The PEP server service fetches the policy from the ESA. After the post-deploy script gets triggered, the IMP Creator utility creates the policy package by encrypting the policy. It then stores package on the object store or to the persistent volume that is attached to the pod running the PEP server and Meta server services.

The policy package is encrypted in memory using Passphrase-Based Encryption or Key Management Service (KMS), and the encrypted package is stored in the persistent volume or object store. The latest copy of the policy package (n) and an earlier copy of the policy package (n-1) are stored on the persistent volume or the object store.

In addition, a metadata JSON file containing the policy related information is created in the *tmpfs* volume mounted on the pod. The volume is accessible to the Meta server service. The Meta server service reads the metadata information and serves information through an endpoint over an *https* request.

A backup of the metadata file is also stored on the persistent volume or the object store.

For more information about the PEP server and Meta server containers, refer to the section *Workflow of the PEP Server Container* and *Workflow of the Meta Server Container*.

5.8.1 Configuring the Publisher Helm Chart

Extract the *PUBLISHER-HELM_ALL-ALL-ALL_x86-64_K8S_DSG_<Version>.tgz* package to create a folder directory *pty-publisher* with the Helm Charts using the following command.

```
tar -xvf PUBLISHER-HELM_ALL-ALL-ALL_x86-64_K8S_DSG_<Version>.tgz
```

The following output appears in the CLI:

```
pty-publisher/Chart.yaml
pty-publisher/values.yaml
pty-publisher/credentials.json
pty-publisher/nfs-pv.yaml
pty-publisher/nfs-pvc.yaml
pty-publisher/templates/_helpers.tpl
pty-publisher/templates/deployment.yaml
pty-publisher/templates/configmap-encryption.yaml
pty-publisher/templates/deployment.yaml
pty-publisher/templates/service.yaml
```

Before you begin
Ensure that the following prerequisites are complete.

a. Create the TLS certificate secret.
   a. Navigate to the *packaging/certs/* directory.

      ```
      cd packaging/certs/
      ```

      The *certificate.pem* and *key.pem* files are available in the *certs* directory.

      For more information about creating a TLS certificate, refer to *step 3c* in the section *Setting Up the Certificates*.

   b. Create the TLS certificate secret (*pty-secret-tls-publisher*) for the Publisher Meta server using the following command.

      ```
      kubectl create -n $DSG_NAMESPACE secret tls $PUBLISHER_SERVER_CERT_SECRET --cert=./
      certs/certificate.pem --key=./certs/key.pem
      ```

      Example:

      ```
      export DSG_NAMESPACE=test-demo
      export PUBLISHER_SERVER_CERT_SECRET=pty-secret-tls-publisher
      ```

```
kubectl create -n $DSG_NAMESPACE secret tls $PUBLISHER_SERVER_CERT_SECRET --cert=./
certs/certificate.pem --key=./certs/key.pem
```

> **Note:** The secret is to be created from the *packaging/certs* directory.

b. Create a Kubernetes Secret in the namespace -

a. Create the ESA credentials secret (*pty-secret-esa-cred*) using the following command.

```
kubectl -n $DSG_NAMESPACE create secret generic $ESA_CREDENTIAL_SECRET --from-
literal=esa_user=$ESA_USER --from-literal=esa_pass=$ESA_PASS
```

Example:

```
export DSG_NAMESPACE=test-demo
export ESA_CREDENTIAL_SECRET=pty-secret-esa-cred
export ESA_User=imp
export ESA_PASS=Pwd$123#

kubectl -n $DSG_NAMESPACE create secret generic $ESA_CREDENTIAL_SECRET --from-
literal=esa_user=$ESA_USER --from-literal=esa_pass=$ESA_PASS
```

b. If you are using PBE to encrypt the policy package, then create the password-based encryption secret (*pty-secret-pbe*) using the following command.

```
kubectl -n $DSG_NAMESPACE create secret generic $PBE_SECRET --from-
literal=passphrase=$PASSPHRASE_PASSWORD --from-literal=salt=$SALT_PASSWORD
```

Example:

```
export DSG_NAMESPACE=test-demo
export PBE_SECRET=pty-secret-pbe
export PASSPHRASE_PASSWORD=<Passphrase>
export SALT_PASSWORD=<Salt_Password>

kubectl -n $DSG_NAMESPACE create secret generic $PBE_SECRET --from-
literal=passphrase=$PASSPHRASE_PASSWORD --from-literal=salt=$SALT_PASSWORD
```

> **Important:** Ensure that the passphrase has a minimum length of 12 characters, which includes at least the following:
>
> - 1 uppercase letter.
> - 2 digits.
> - 2 special characters.
> - 2 non-letters, such as, digits or special characters.
>
> The passphrase must also have a minimum value of entropybits as *30* and a minimum strength of *0.66*.
>
> Entropybits defines the randomness of the password, and the strength defines the complexity of the password.
>
> For more information about the password strength, refer to the section *Password Strength*.

c. Create a Kubernetes Secret to store the credentials needed to pull the Docker images from your image repository in the same namespace where you will deploy the PTY-Publisher pod.

```
kubectl -n $DSG_NAMESPACE create secret generic $IMAGE_REGISTRY_CREDS_SECRET --from-
file=.dockerconfigjson=$PATH_TO_DOCKER_CONFIG --type=kubernetes.io/dockerconfigjson
```

Example:

```
export DSG_NAMESPACE=test-demo
export IMAGE_REGISTRY_CREDS_SECRET=pty-secret-registry-cred
export PATH_TO_DOCKER_CONFIG=/etc/docker/config.json

kubectl -n $DSG_NAMESPACE create secret generic $IMAGE_REGISTRY_CREDS_SECRET --from-
file=.dockerconfigjson=$PATH_TO_DOCKER_CONFIG --type=kubernetes.io/dockerconfigjson
```

The *dockerconfigjson* file is generated when you set up the image container registry and perform the docker login command. This file stores the Docker credentials.

> **Note:**
> Create the Docker credentials only if required.

Edit the *imagePullSecrets* tag if using credentials to pull the Docker image.

d. If you are using Azure, then create the storage access secret (*storageAccessSecrets*) using the following command.

```
kubectl -n $DSG_NAMESPACE create secret generic $STORAGE_SECRET --from-
file=credentials=$HOME/pty-publisher/credentials.json --kubeconfig="${KUBECONFIG}"
```

You can also use service accounts to provide access to cloud services.

e. Run the following command to verify that the secrets have been created.

```
kubectl get secrets -n $DSG_NAMESPACE
```

For example:

```
export DSG_NAMESPACE=test-demo

kubectl get secrets -n $DSG_NAMESPACE
```

The following snippet shows the secrets that have been created.

```
NAME
TYPE                             DATA   AGE
pty-secret-esa-cred
Opaque                           2      34m
pty-secret-pbe
Opaque                           2      5d15h
pty-secret-tls-publisher
kubernetes.io/tls                2      5d15h
pty-secret-registry-cred                                              kubernetes.io/
dockerconfigjson   1      34m
```

> **Note:**
> Before editing the *values.yaml* file, ensure that the secrets are created in the same namespace where the Publisher and the DSG pods have been deployed.

To configure the Publisher Helm chart, perform the following steps.

1. Navigate to the *packaging/pty-publisher* directory using the following command.

```
cd packaging/pty-publisher
```

2. Open the *values.yaml* file using the following command.

```
vi values.yaml
```

3. If you want to use AWS S3 for storing the CoP and policy snapshots, instead of AWS EFS, then perform the following steps to create service accounts in the Kubernetes cluster. These steps ensure that only the pod on which the DSG container has been deployed has write access to the AWS S3 bucket, which enables the DSG container to upload the immutable policy snapshot to the AWS S3 bucket.

   a. Run the following command to create an OpenID Connect (OIDC) identity provider in the IAM console.

```
eksctl utils associate-iam-oidc-provider --name <Name of the Kubernetes cluster> --
approve
```

   OIDC is an authentication protocol. The OIDC identity provider enables you to map AWS IAM roles to Kubernetes service accounts.

   b. Run the following command to create an IAM service account for the Sample Application container.

```
eksctl create iamserviceaccount --name <Service_account_name>
--namespace <Namespace> --cluster <Kubernetes_cluster_name> --attach-
policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess --attach-policy-arn
arn:aws:iam::829528124735:policy/KMSDecryptAccess --approve
```

   This command contains the following policies:

   - *AmazonS3WriteOnlyAccess* is a default AWS policy, which is attached to the required service account. This policy allows the service account to upload the encrypted policy package to the S3 bucket.
   - *KMSEncryptAccess* is a custom AWS policy, which is attached to the required service account. This policy allows the service account to encrypt the policy package that has been encrypted using an AWS Customer key.

   For example:

```
eksctl create iamserviceaccount --name dsg-sa --namespace dsg --cluster cluster-name
--attach-policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess --attach-policy-arn
arn:aws:iam::829528124735:policy/KMSDecryptAccess --approve
```

   Important: Ensure that the name of the service account is *dsg-sa*, which has been defined in the *pty-psp.yaml* file.

4. Edit the *imagePullSecrets* tag.

```
# specify the image registry credential secret name
# in case if the registry is accessible without any credentials uncomment
# square brackets and comment name field
imagePullSecrets: # []
  - name: $IMAGE_REGISTRY_CREDS_SECRET
```

   Replace the *$IMAGE_REGISTRY_CREDS_SECRET* variable with the value of the *pty-secret-registry-cred* secret that you have created in the *prerequisites*.

5. Edit the *pepServerImage* tag.

```
pepServerImage:
  repository: "$PEPSERVER_IMAGE_REPOSITORY"
  tag: "$PEPSERVER_IMAGE_TAG"
  pullPolicy: IfNotPresent
  debug: false  # change debug to true to get debug logs on STDOUT.
```

Example:

```
pepServerImage:
  repository: <aws_account_id>.dkr.ecr.us-east-1.amazonaws.com/demo/publisher-pepserver
  tag: PUBLISHER_PEP_RHUBI-9-64_x86-64_K8S_9.1.0.0.xx
  pullPolicy: IfNotPresent
  debug: false  # change debug to true to get debug logs on STDOUT.
```

6. Edit the *metaServerImage* tag.

```
metaServerImage:
  repository: "$METASERVER_IMAGE_REPOSITORY"
  tag: "$METASERVER_IMAGE_TAG"
  pullPolicy: IfNotPresent
  debug: false  # change debug to true to get debug logs on STDOUT.
```

Example:

```
metaServerImage:
  repository: <aws_account_id>.dkr.ecr.us-east-1.amazonaws.com/demo/pty_metaserver
  tag: PUBLISHER_META_RHUBI-9-64_x86-64_K8S_9.1.0.0.xx
  pullPolicy: IfNotPresent
  debug: false  # change debug to true to get debug logs on STDOUT.
```

7. Edit the *publisherServerCertSecret* tag, which is used to specify the secret of the TLS certificate for the Publisher server.

```
## publisher server TLS certificate secret
## pty sidecar container will verify these certificates
# eg. kubectl command:
#     kubectl -n $DSG_NAMESPACE create secret tls pty-certs \
#     --cert=./certs/certificate.pem --key=./certs/key.pem
publisherServerCertSecret: $PUBLISHER_SERVER_CERT_SECRET
```

In the *publisherServerCertSecret* tag, specify the value of the *pty-secret-tls-publisher* that you have created in the *prerequisites*.

8. Edit the *esaCredSecrets* tag.

```
## k8s secret for storing ESA credentials
# eg. kubectl command:
#     kubectl -n $DSG_NAMESPACE create secret generic pty-esa \
#     --from-literal=esa_user=<esa_user> --from-literal=esa_pass='<esa_user_password>'
esaCredSecrets: $ESA_CREDENTIAL_SECRET
```

In the *esaCredSecrets* tag, specify the value of the *pty-secret-esa-cred* that you have created in the *prerequisites*.

9. If you are using PBE to encrypt the policy package, then edit the *pbeSecrets* tag.

```
## k8s secret containing passphrase for encrypting policy
# eg. kubectl command:
#     kubectl -n $DSG_NAMESPACE create secret generic pty-pbe \
#     --from-literal='passphrase=<complex chars>' --from-literal='salt=<complex chars>'
pbeSecrets: $PBE_SECRET
```

In the *pbeSecrets* tag, specify the value of the *pty-secret-pbe* that you have created in the *prerequisites*.

10. Specify one of the following options in the *securityDonfigDestination* tag for storing the policy package and COP:
    - VolumeMount
    - AWS
    - AZURE
11. If you are using Cloud KMS to encrypt the policy package, then edit the *privateKeySource* tag.

```
## If not using PBE, Choose your private key helper.
## Currently we support [AWS_KMS, AZ_VAULT]
privateKeySource: $KEY_SOURCE
```

```
privateKeyId: $KEY_ID
```

Specify one of the following values for the *privateKeySource* tag:

- *AWS_KMS* - Use AWS KMS
- *AZ_VAULT* - Use Azure Key Vault as the KMS

Also, specify the value of the *privateKeyID* tag.

Example 1 - AWS

```
## If not using PBE, Choose your private key helper.
## Currently we support [AWS_KMS, AZ_VAULT]
privateKeySource: "AWS_KMS"

privateKeyId: "a47844b7-ce4f-4650-8ce5-146875db5339"
```

Example 2 - Azure

```
## If not using PBE, Choose your private key helper.
## Currently we support [AWS_KMS, AZ_VAULT]
privateKeySource: "AZ_VAULT"

privateKeyId: "https://automation-cntrs.vault.azure.net/keys/dsg/
154e3a367e9240d5958cda0af44265f2"
```

12. Edit the *pvcName* tag only if you are using Persistent Volume for storage.

```
## specify the name of persistent volume claim to be used for this deployment.
pvcName: $PVC_NAME
```

Example:

```
## specify the name of persistent volume claim to be used for this deployment.
pvcName: esa-policy-store
```

13. Edit the *policy* tag.

```
policy:
  esaIP: $ESA_IP
  ## absolute path in PV where the policy dump file will be stored. <eg. /test/xyz/ >
  filePath: $ABSOLUTE_POLICY_PATH
```

Example:

```
policy:
  esaIP: 10.21.0.99
  filePath: demo/policy
```

> **Note:** Do not include the trailing '/' in the file path. For example; *test/xyz/policy*. The policy package file is placed in this path on the persistent volume.

5.8.1.1 Additional Configuration

The other values that you can edit from the *packaging/pty-publisher/values.yaml* file are mentioned in the following table.

Table 5-6: Additional Tags in values.yaml file

Tags	Default Configuration	Description
pepserverConfig[*1]	```## pepserver configurations that can be changed. ## emptystring: Set the output behavior for empty strings ## null = (default) null value is returned for empty input string ## empty = empty value is returned for empty input string ## encrypt = encrypted values is returned for empty input string ## level: Specifies the logging level for pepserver ## OFF (no logging) ## SEVERE ## WARNING ## INFO ## CONFIG ## ALL (default) ## caseSensitive: Specifies how policy users are checked against policy ## yes = (The default) policy users are treated in case sensitive manner ## no = policy users are treated in case insensitive manner. pepserverConfig: emptyString: "null" logLevel: "ALL" caseSensitive: "yes"```	It changes the default behavior of the PEP server settings. For more information about the PEP server configurations, refer to the section *PEP Server Configuration File* in the *Protegrity Installation Guide 9.1.0.5*. **Note:** If the input data is less than or equal to zero, then DSG processes the input as it is. Hence, in such cases, even if the *pepserverConfig*/emptyString parameter is set to **Null** or **Encrypt**, an Empty output is returned.
pepServerResources[*1]	```pepServerResources: # below allocation covers most of the use-cases which requires wide range of policy size. limits: cpu: 500m memory: 4000Mi requests: cpu: 200m memory: 256Mi```	It configures the resource requirements for the PEP server container.
metaServerResources[*2]	```metaServerResources: # below allocation covers most of the use-cases which requires wide range of policy size. limits: cpu: 500m memory: 512Mi requests: cpu: 200m memory: 256Mi```	It configures the resource requirements for the Meta server container.
serviceAccount[*2]	```## pod service account to be used ## leave the field empty if not applicable ## e.g. serviceAccount: user1 serviceAccount: name: ""```	It is a service account to run the pod. Currently, it set to default. Customers can configure the service account as per their requirement.

Tags	Default Configuration	Description

podSecurityContext[*2]	```## set the Pod's security context object
	podSecurityContext:
	  runAsUser: 1000
	  runAsGroup: 1000
	  fsGroup: 1000```
		It changes the default security context of the *pty-publisher* pod.
		Important: At least one of the three parameters must be set to *1000*.

containerSecurityContext[*2]	```## set the Container's security context object
	containerSecurityContext:
	  capabilities:
	  drop:
	  - ALL
	  allowPrivilegeEscalation: false
	  privileged : false
	  runAsNonRoot : true
	  readOnlyRootFilesystem: true```
		It changes the default security context of the containers.

livenessProbe[*2]	```## Configure the delays for Liveness Probe here
	livenessProbe:
	  initialDelaySeconds: 15
	  periodSeconds: 20```
		It changes the default delay interval of the liveness probe, which performs a diagnostic check to confirm if the containers are healthy.

readinessProbe[*2]	```## Configure the delays for Readiness Probe here
	readinessProbe:
	  initialDelaySeconds: 15
	  periodSeconds: 20```
		It changes the default delay interval of the readiness probe, which checks whether the containers are ready to accept the incoming traffic.
		If files with larger size are being processed, then the container might be busy in processing the file. This might lead to a probe failure message in the pod description. As a result, the pod changes its state from Ready to Not Ready.

Note:
[*1] - PEP server parameters

[*2] - Optional parameters

5.8.2 Deploying the Publisher Helm Chart

After performing the prerequisites and steps, you can deploy the Publisher Helm Chart.

1. Navigate to the *packaging* directory using the following command.

```
cd packaging
```

2. Deploy the Publisher Helm Chart using the following command.

```
helm install $PTY_PUBLISHER_HELM_INSTALL_NAME $PTY_PUBLISHER_HELM_CHART_LOCATION -n
$DSG_NAMESPACE
```

Example:

```
export DSG_NAMESPACE=test-demo
export PTY_PUBLISHER_HELM_INSTALL_NAME=publisher
export PTY_PUBLISHER_HELM_CHART_LOCATION=pty_publisher

helm install $PTY_PUBLISHER_HELM_INSTALL_NAME $PTY_PUBLISHER_HELM_CHART_LOCATION -n
$DSG_NAMESPACE
```

The following output appears if you have successfully installed the Publisher Helm Chart.

```
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /
home/ec2-user/kubeconfig
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /
home/ec2-user/kubeconfig
NAME: publisher
LAST DEPLOYED: Mon Apr  1 06:49:42 2024
NAMESPACE: test-demo
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

If you are an advanced user of Kubernetes, then you can also deploy the Helm charts by substituting the environment variables or using the *set* argument.

For more information about deploying a Helm chart by substituting the environment variables, refer to the Appendix *Deploying the Helm Charts by Substituting the Environment Variables*.

For more information about deploying a Helm chart by using the *set* argument, refer to the Appendix *Deploying the Helm Charts by Using the Set Argument*.

3. Run the following command to check the status of the pods.

```
kubectl get pods -n $DSG_NAMESPACE
```

For example:

```
export DSG_NAMESPACE=test-demo

kubectl get pods -n $DSG_NAMESPACE
```

4. Run the following command to check the logs.

```
kubectl get logs -f <Pod_name> -n $DSG_NAMESPACE
```

For example:

```
export DSG_NAMESPACE=test-demo

kubectl logs -f publisher-pty-publisher-7f596d88d5-jq4jq -n $DSG_NAMESPACE
```

For more information about the output of this command, refer to the Appendix *Publisher Pod Log Files*.

If you are unable to download the policy from the ESA even after specifying the correct Node IP, then verify whether the logs contain the following snippet.

```
2024-04-16 04:37:28 (WARNING) Node registration failed, please check IP/Range of data
store or set default data store.
2024-04-16 04:37:28 (INFO) This node has been registered as node ID 16
2024-04-16 04:37:28 (WARNING) Internal error - Failed to register node
```

This error indicates that the ESA is unable to register the node.

To troubleshoot this issue, perform the following steps:

a. Navigate to the ESA and specify the **Set as Default Data Store** option to **Yes**.

b. Click **Save** and then deploy the policy.

c. Validate whether the policy deployment is successful.

The following snippet of the Publisher pod log shows that the policy has been downloaded successfully.

```
....
....
....
2024-04-16 04:43:21,441:imp_creator:INFO:Done.
2024-04-16 04:43:21 (INFO) Post deploy application done!
2024-04-16 04:43:21 (INFO) Done with downloading policy files
```

d. In the **Nodes** tab of the ESA, verify the IP address of the registered node.

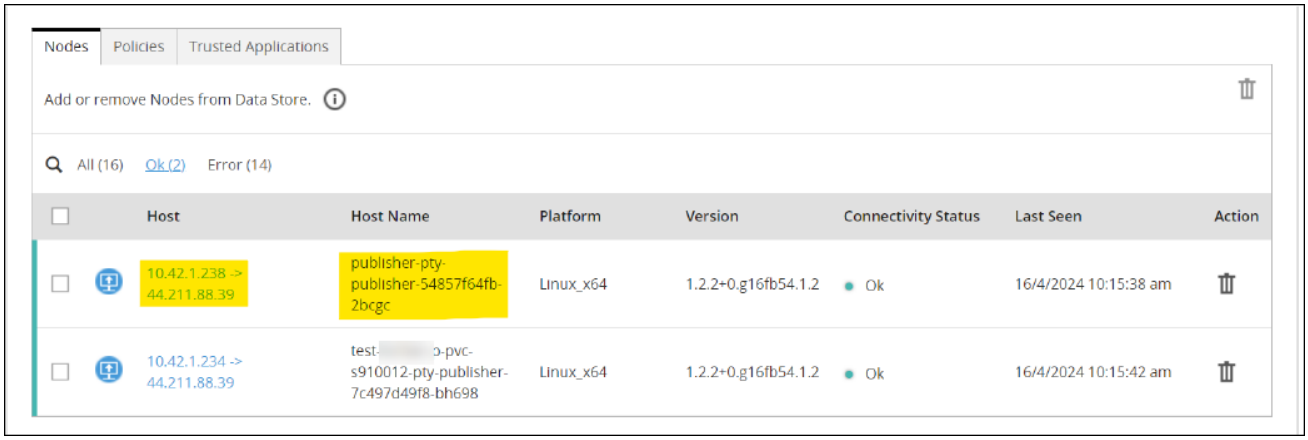


Figure 5-8: Nodes tab

In the **Host** column, the first IP address indicates the IP of the Publisher pod, while the second IP address indicates the IP of the Node where the Publisher pod has been deployed.

```
<IP address of Publisher Pod> --> <IP address of Node where Publisher Pod is deployed>
```

e. Configure the datastore in the ESA with the IP address of the Node that you have obtained in *step 4e*.

f. Specify the **Set as Default Data Store** option to **No**.

5.9 Deploying a Release on the Kubernetes Cluster

This section describes how to deploy a Release on the Kubernetes cluster by installing the Helm charts.

> **Note:** The procedures performed in this section require the user to have the *cluster-admin* role.

To deploy a release on the Kubernetes cluster:

1. Perform the following steps if you want to use an Azure File Share for storing the policy dump instead of the Azure Storage Container.
   a. Create a file named *pv.yaml* for creating a persistent volume resource.

      The following snippet shows the contents of the *pv.yaml* file.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: azurefile
  labels:
    target: policy
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  storageClassName: azurefile
  azureFile:
    secretName: azure-secret
    shareName: policy
    readOnly: false
  mountOptions:
  - dir_mode=0744
  - file_mode=0644
  - uid=1000
  - gid=1000
    #- mfsymlinks
    #- nobrl
```

> **Important:**
> If you want to copy the contents of the *pv.yaml* file, then ensure that you indent the file as per YAML requirements.

In the *storageClassName* parameter, ensure that you specify *azurefile*, for using the Azure File Share as the persistent volume.

In the *storageClassName* parameter, ensure that you specify *azurefile*, for using the Azure File Share as the persistent volume.

In the *azureFile/secretName* parameter, ensure that you specify name of the secret that you have created in *step 6*.

For example, specify the value of the *azureFile/secretName* parameter as *azure-secret*.

In the *azureFile/shareName* parameter, ensure that you specify name of the Azure File Share that you have created while creating the Azure File Share.

Ensure that the value specified in the *metadata/labels/target* is also specified as the value of the *selector/matchLabels/target* parameter in the *claim.yaml* file in *step 7c*.

You can specify the access control settings for the files and directories on the Azure File Share by using the *mountOptions* parameter.

For more information about the mount option parameters, refer to the section *Use Azure Files with Linux* in the Azure documentation.

b. Run the following command to create the persistent volume resource.

```
kubectl apply -f pv.yaml
```

A persistent volume resource is created.

c. Create a file named *claim.yaml* for creating a claim on the persistent volume that you have created in *step 6b*.

The following snippet shows the contents of the *claim.yaml* file.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: azurefile
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: azurefile
  resources:
    requests:
      storage: 1Gi
  selector:
    matchLabels:
      target: policy
```

> **Important:**
> If you want to copy the contents of the *claim.yaml* file, then ensure that you indent the file as per YAML requirements.

In the *storageClassName* parameter, ensure that you specify *azurefile*, for using the Azure File Share as the persistent volume.

In the *selector/matchLabels/target* parameter, ensure that you specify the value that was defined in the *metadata/labels/target* parameter in the *pv.yaml* file in *step 7a*.

d. Run the following command to create the persistent volume claim.

```
kubectl apply -f claim.yaml -n <Namespace>
```

For example:

```
kubectl apply -f claim.yaml -n dsg
```

A persistent volume claim is created.

2. If you want to use an Azure Storage Container for storing the policy package instead of the Azure File Share, then modify the *credentials.json* file in the *dsg* directory to update the credentials for the service principal 1.

The *credentials.json* file has the following format.

```
{
  "storage_account_name" : "AZURE_STORAGE_ACCOUNT_NAME",
  "tenant_id" : "AZURE_TENANT_ID",
  "client_id" : "AZURE_CLIENT_ID",
  "client_secret": "AZURE_CLIENT_SECRET"
}
```

3. If you want to use Azure Storage Container for storing the policy package, instead of the Azure File Share, then run the following command to create the *storageAccessSecret* secret, which is used by the DSG container to pull the Policy and CoP packages and use the KMS decryption.

```
kubectl create secret generic storageAccessSecret --from-file=dsg/credentials.json
--namespace <NAMESPACE>

kubectl create secret generic storageAccessSecret --from-file=dsg/credentials.json
--namespace dsg
```

For more information about the *credentials.json* file, refer to the section *Credentials.json*.

> **Note:**
> Ensure that you specify the name of the secret as the value of the *storageAccessSecretName* field in the *values.yaml* file.

For more information about the *storageAccessSecretName* parameter, refer to the section *Storage Access Secrets*.

4. If you want to use Azure File Share for storing the policy package, instead of the Azure Storage Container, then run the following command to create the *azure-secret* secret, which is used by the DSG container to pull the Policy and CoP packages.

```
kubectl create secret generic azure-secret --from-
literal=azurestorageaccountname='<Storage Account Name>' --from-
literal=azurestorageaccountkey='<Storage Account Key>' --namespace <Namespace>

kubectl create secret generic azure-
secret --from-literal=azurestorageaccountname='test' --from-
literal=azurestorageaccountkey='<Storage Account Key>' --namespace dsg
```

In the *azurestorageaccountname* parameter, specify the value of the Azure Storage Account that you have created while creating the Azure File Share.

In the *azurestorageaccountkey* parameter, specify the value of any one of the storage access keys for the Azure Storage Account.

> **Note:**
> Ensure that you specify the name of the secret as the value of the *azureFile/secretName* field in the *pv.yaml* file in *step 7a*.

5. Run the following command to create the *cop-secret* secret for the CoP package.

```
kubectl create secret generic cop-secret --from-literal='passphrase=<COP_PASS>' --from-
literal='salt=<COP_SALT>' -n <SAME_AS_OF_CHART_DEPLOYMENT>
```

For example,

```
kubectl create secret generic cop-secret --from-literal='passphrase=Passphrase123@' --
from-literal='salt=salt' -n dsg
```

> **Note:**
> As the value of the *<COP_PASS>* and *<COP_SALT>* parameters, you must specify the passwords that you created the section *Creating a Ruleset*, when you export the CoP package from the ESA.

6. If you want to authenticate the communication between the Kubernetes cluster and the Azure Container Registry by using your existing Docker credentials, then perform the following steps.
   a. Run the following command to login to Docker.

      ```
      docker login
      ```

      The login process creates a *config.json* file, which contains an authorization token.

   b. Run the following command to specify a secret for pulling the DSG container images from the Azure Container Registry to Kubernetes.:

      ```
      kubectl create secret generic <Secret_name> --from-file=.dockerconfigjson=/<Path to the
      config.json file> --type=kubernetes.io/dockerconfigjson --namespace dsg
      ```

      For example:

      ```
      kubectl create secret generic regcrd --from-file=.dockerconfigjson=/home/<user>/.docker/
      config.json --type=kubernetes.io/dockerconfigjson --namespace dsg
      ```

      For more information about creating secrets, refer to *https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/*.

      > **Note:**
      > Ensure that you specify the name of the secret as the value of the *imagePullSecrets/name* parameter in the *values.yaml* file.

      For more information about the *imagePullSecrets* field, refer to the section *Image Pull Secrets*.

7. Create a secret for the Certificate Authority (CA) certificate that you have generated for the PTY-Publisher pod.
   a. Navigate to the *certs* directory.

      The *ca.pem* files are available in the *certs* directory.

      For more information about creating a CA certificate, refer to step 2 in the section Setting Up the Certificates.

   b. Run the following command to create a Certificate Authority (CA) certificate for the PTY-Publisher pod. This CA certificate is used by the sidecar container to communicate with the PTY-Publisher pod.

      ```
      kubectl create -n $DSG_NAMESPACE create secret generic $PUBLISHER_CA_CERT_SECRET --
      fromfile=ca.pem=./certs/ca.pem
      ```

      Example:

      ```
      export DSG_NAMESPACE=test-demo
      export PUBLISHER_CA_CERT_SECRET=pty-secret-ca-cert

      kubectl create -n $WAREHOUSE_NAMESPACE create secret generic $CA_CERT_NAME --
      fromfile=ca.pem=./certs/ca.pem
      ```

8. If you want to authenticate the user using basic authentication, then run the following command to create an LDAP secret, which is used by the DSG container to connect to the LDAP.

   ```
   kubectl create secret generic <LDAP_secret_name> -n <Namespace> --from-
   literal=self.xml='<perldata> <hashref> <item key="LDAP_URI">ldap://{LDAP IP}:{port}/</
   item> <item key="LDAP_BASEDN">{BASE DN}</item> <item key="LDAP_USERSDN">{USER DN}</item>
   <item key="LDAP_BINDDN">{BIND DN}</item> <item key="LDAP_BINDPW">{BIND Password}</item>
   <item key="LDAP_STARTTLS">1</item> </hashref> </perldata>'
   ```

For example, the following snippet shows the LDAP parameters that are used to connect to the ESA LDAP:

```
kubectl create secret generic cmdldap -n dsg
--from-literal=self.xml='<perldata> <hashref> <item key="LDAP_URI">ldap://
10.120.0.58:389/</item> <item key="LDAP_BASEDN">dc=esa,dc=protegrity,dc=com</item>
<item key="LDAP_USERSDN">ou=people,dc=esa,dc=protegrity,dc=com</item> <item
key="LDAP_BINDDN">cn=ldap_bind_user,ou=people,dc=esa,dc=protegrity,dc=com</item> <item
key="LDAP_BINDPW">{BIND_Password}</item> <item key="LDAP_STARTTLS">1</item> </hashref> </
perldata>'
```

In this command, you need to specify the value for these LDAP parameters:

Table 5-7: LDAP Parameters

Parameter	Description
LDAP_URI	The IP address and port number used to connect to the LDAP server.
LDAP_BASEDN	Distinguished name, which uniquely identifies the base location that is used to search for a user in an LDAP directory. For example, if you are using the ESA LDAP and your domain name is *protegrity.com*, then your Base DN might be: `dc=esa,dc=protegrity,dc=com`
LDAP_USERSDN	Distinguished name, which uniquely identifies the user location that is used to search for a user in an LDAP directory. It is a subset of the Base DN. For example, if you are using the ESA LDAP, your domain name is *protegrity.com*, and your users are part of an organizational unit named *people*, then your Users DN might be: `ou=people, dc=esa,dc=protegrity,dc=com`
LDAP_BINDDN	Distinguished name of the user that is used to connect to the LDAP server. For example: `cn=ldap_bind_user,ou=people,dc=esa,dc=protegrity,dc=com`
LDAP_BINDPW	Password of the user that is used to connect to the LDAP server.
LDAP_STARTTLS	Set the value of this parameter to *1* to use the TLS protocol for encrypting the communication with the LDAP server.

You need to specify the LDAP secret name as the value of the *ldapXmlSecrets* parameter in the *values.yaml* file in *step 6*.

> **Note:** If you are using the ESA LDAP, then you can obtain the values for the LDAP parameters from the *self.xml* file.
>
> To access the *self.xml* file, perform the following steps.
>
> 1. Login to the CLI Manager using the administrator credentials.
> 2. Navigate to **Administration** > **OS Console**.
> 3. Enter the *root* password and press **OK**.

4. Navigate to the `/etc/ksa/conf/` directory to access the *self.xml* file.

Important: If you are using a third-party LDAP server, then ensure that you add the following attribute to the user with corresponding value in the LDAP server.

Attribute	Value
businessCategory	pty_role:cloud_gateway_auth

For more information about adding an attribute to the LDAP server, refer to the LDAP server documentation.

If you have connected the LDAP server to the ESA, and you add a user who has been assigned a role that includes the *Cloud Gateway Auth* permissions, then the *businessCategory* attribute is automatically added to the LDAP server.

For more information about connecting an LDAP server to the ESA using the CLI Manager, then refer to the section *Managing LDAP* in the *Appliances Overview Guide 9.1.0.5*.

For more information about authenticating and authorizing external LDAP users from the ESA, refer to the section *Working with Proxy Authentication* in the *Enterprise Security Administrator Guide 9.1.0.5*.

9. On the Linux instance, navigate to the location where you have extracted the Helm charts.

   For more information about the extracted Helm charts, refer to the *step 4* of the section *Extracting the Package*.

   The *values.yaml* file contains the default configuration values for deploying the DSG application on the Kubernetes cluster.

10. Modify the default values in the *values.yaml* file as required.

    For more information about Helm charts and their default values, refer to the section *Appendix A: DSG Container Helm Chart Details*.

Field	Description
podSecurityContext	Specify the privilege and access control settings for the pod. The default values are set as follows: • *runAsUser* - 1000 • *runAsGroup* - 1000 • *fsGroup* - 1000 • *supplementalGroups* - 1000 **Note:** If you want to use Azure Storage Container for storing the policy snapshot, then set the value of the *fsGroup* parameter to a non-root value. For example, you can set the value of the *fsGroup* parameter to any value between *1* and *65534*. **Note:** Ensure that at any given point of time, the value of at least one of the three parameters must be *1000*. This ensures that the pod has the required permissions to access the *pty* directories in the containers.

Field	Description
	For more information about the Pod Security Context, refer to the section *Configure a Security Context for a Pod or Container* in the Kubernetes documentation. For more information about the parameters, refer to the section *PodSecurityContext v1 Core* in the Kubernetes API documentation.
pvcName	Specify the name of the Persistent Volume Claim where you want to store the policy package. If you are storing the policy package in Azure File Share, then specify the name of the Persistent Volume Claim that you have specified in the *pv.yaml* file. Leave the value of this field blank if you want to use the Azure Storage Container for storing the policy package. **Important:** This field is required if you want to store the policy package in the persistent volume.
privateKeySource	Specify one of the following methods used to encrypt the policy package: • *AZ_VAULT* - Azure Vault for encrypting the policy package. • *AWS_KMS* - Use AWS Customer Master Key for encrypting the policy package.
privateKeyId	Specify the private key from Azure Key Vault
securityConfigDestination	Specify one of the following values for storing the policy package and COP: • VolumeMount • AWS • Azure
storageAccessSecrets	Specify the Kubernetes secret that contains the credentials for pulling the CoP and the Policy from the Azure Storage Container. The DSG Application container uses the value specified in the *storageAccessSecrets* parameter to decrypt the Policy package using the KMS.
copSecrets	Specify the Kubernetes secret that contains the passphrase and the salt that were used to generate key, which encrypted the CoP. The DSG Application container uses the value specified in the *CopSecrets* parameter to decrypt the CoP package.
pbeSecrets	Specify the Kubernetes secret that contains the passphrase and the salt that were used to generate key, which encrypted the policy. The DSG Application container uses the value specified in the *pbeSecrets* parameter to decrypt the policy. By default, the Passphrase-Based Encryption is used to encrypt the policy and CoP package. If you want to use the Azure KMS to decrypt the policy package, then you must comment out this entry or leave the value blank. **Note:** If you specify a value for *pbeSecrets* field, then Passphrase-Based Encryption is used instead of the Azure Key Vault.

Field	Description
sidecarObject/pingInterval	Specify the interval in seconds after which the Subscriber-Sidecar container checks for the policy update information.
sidecarObject/publisher/host	Specify the hostname of the service hosting the policy update information.
sidecarObject/publisher/port	Specify the port number of the service hosting the policy update information. BY default, this value is set to 443.
sidecarObject/publisher/tls/caCertSecret	Specify the CA secret that you have created in *step 6*.
ldapXmlSecrets	Specify the value of the LDAP secret that you have created in *step 8*.
cop/filepath	Specify the path in the persistent volume or the object store where you have stored the CoP package. For example, if you are using Azure File Share as the persistent volume, then you can specify the file path as *file://var/data/ptypolicy/pathInVolumeToCop*. If you have stored the IMP in an Object Store, such as an Azure Storage Container, then you can specify the storage container name as the first name in the *filepath* field. For example, you can specify the value of the *filepath* field, as *https://<accountName>.blob.core.windows.net/<container name>/<pathToCop>*. **Important:** Use only uppercase and lowercase letters, numbers, dot, and underscore in the file path. Do not use special characters in the file path. **Note:** This value is case-sensitive.
dsgService/type	Specify whether you want to use one of the following service types: • *LoadBalancer* - An external IP is created. You need to send a request to the IP address of the Azure Load Balancer for running the security operations. • *ClusterIP* - A cluster IP is created. You need to send a request to the cluster IP, which is a unique IP address assigned to the Kubernetes service, for running the security operations. If you specify the service type, then you also need to uncomment the annotations for the corresponding service type.
dsgService/healthCheckRestService/scheme	Specify the protocol used for sending a health check service request. By default, the value is set to *HTTP*.
dsgService/healthCheckRestService/host	Specify the host name that is used to send a health check service request. By default, this value is set to *restapi*. This default value corresponds to the value specified in the default REST API example ruleset available on the ESA.
dsgService/healthCheckRestService/port	Specify the port number for the health check service. By default, this value is set to *&healthCheckPort 8080*. This default value corresponds to the value specified in the default REST API example ruleset available on the ESA. This configuration is mandatory as the default port is used for running health checks on the container. Protegrity recommends you not to modify the default port number.
dsgService/healthCheckRestService/path	Specify the URI that is used to perform the health check service. By default, this value is set to *protect/text*.

Field	Description
	The health check is performed by sending a request to the *http://<Health_check_host>/protect/text* URI. This URI is defined as part of the *Text Protection* ruleset, which is the default ruleset available on the ESA. Ensure that you retain this default ruleset on the ESA so that the health check is performed successfully.
dsgService/tunnels/name	Specify a name for the tunnel to distinguish between ports.
dsgService/tunnels/port	Specify the port number on which you want to expose the Kubernetes service externally.
dsgService/tunnels/targetPort	Specify the port number configured while creating a tunnel on the ESA. By default, this value is set to **healthCheckPort*, which indicates the value specified in the *dsgService/healthCheckRestService/port* field. For more information about creating a tunnel and a CoP ruleset, refer to the section *Creating a Ruleset*.

11. Run the following command to deploy DSG on the Kubernetes cluster:

```
helm install <Release Name> <Location of the directory that contains the Helm charts>
--namespace <Namespace>
```

For example:

```
helm install dsg-demo packaging/DSG-HELM-DYNAMIC_ALL-ALL-ALL_x86-64.K8S_<Version>/dsg --
namespace dsg
```

12. Perform the following steps to check the status of the Kubernetes resources.
    a. Run the following command to check the status of the pods.

```
kubectl get pods -n <namespace>
```

For example:

```
kubectl get pods -n dsg
```

5.10 Verifying the Operations for the DSG Container Deployment

This section specifies the commands to verify the operations performed on the DSG Container deployment.

5.10.1 Generic Commands to Validate the Cluster

This section describes the generic commands to validate the cluster.

- To fetch the project list, execute the following command.

```
kubectl projects list
```

- To describe the pod specification, execute the following command.

```
kubectl -n <namespace> describe <pod name>
```

- To describe the pod specification in *yaml*, execute the following command.

```
kubectl -n <namespace> describe <pod name> -o yaml
```

- To check the pods list, execute the following command.

```
kubectl -n <namespace> get pods -n <namespace>
```

- To check the service name of the deployed containers, execute the following command.

```
kubectl -n <namespace> get svc -n <namespace>
```

- To check the logs of the containers, execute the following command.

```
kubectl -n <namespace> logs <pod name> -c <container name>
```

- To check the logs of the pod, execute the following command.

```
kubectl -n <namespace> logs <pod name>
```

5.10.2 Validating the Software Version Installed on the Base Machine

This section describes the versions to be validated for the software or client installed on the base machine.

- To validate the Kubectl version installed on the base machine, execute the following command.

```
kubectl version
```

The following output appears.

```
kubectl version
Client Version: v1.31.2
Kustomize Version: v5.4.2
Server Version: v1.30
```

- To validate the Docker version installed on the base machine, execute the following command.

```
docker version
```

The following output appears.

```
Client:
 Version:           24.0.7
 API version:       1.43
 Go version:        go1.22.2
 Git commit:        24.0.7-0ubuntu4.1
 Built:             Fri Aug  9 02:33:20 2024
 OS/Arch:           linux/amd64
 Context:           default

Server:
 Engine:
  Version:          24.0.7
  API version:      1.43 (minimum version 1.12)
  Go version:       go1.22.2
  Git commit:       24.0.7-0ubuntu4.1
  Built:            Fri Aug  9 02:33:20 2024
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.7.12
  GitCommit:
 runc:
  Version:          1.1.12-0ubuntu3.1
  GitCommit:
 docker-init:
```

```
  Version:          0.19.0
  GitCommit:
```

- To validate the Helm version installed on the base machine, execute the following command.

```
helm version
```

The following output appears.

```
version.BuildInfo{Version:"v3.16.2", GitCommit:"13654a52f7c70a143b1dd51416d633e1071faffb",
GitTreeState:"clean", GoVersion:"go1.22.7"}
```

5.10.3 Validating the Docker Images Uploaded in the Image Repository

This section describes the docker images to be validated after uploading it in the image repository.

- To validate the images tagged in the docker, execute the following command.

```
docker images
```

The following sample output appears on the CLI when you initially load the docker.

Table 5-8: Docker Images Output Format

REPOSITORY	TAG	IMAGE ID	SIZE
artifactory.protegrity.com/docker-registry/dsg/dsg-subscriber-sidecar	SUBSCRIBER-SIDECAR_RHUBI-8-64_x86-64_K8S_<Version>	<Image ID>	564MB

> **Note:** The example mentioned in the table considers only one image output.

The following sample output appears on the CLI after you create a tag.

Table 5-9: Docker Images Output Format After Tag Creation

REPOSITORY	TAG	IMAGE ID	SIZE
artifactory.protegrity.com/docker-registry/dsg/pty-subscriber-sidecar	CLIENT-SIDECAR_RHUBI-ALL-64_x86-64_K8S_<Version>	<Image ID>	221MB
<IMAGE_REPOSITORY_URL>/<PATH>	<TAG>	<Image ID>	221MB

5.10.4 Validating the Helm Chart

This section describes the Helm charts to be validated before deploying the various components of the DSG Container on the EKS or Azure platform.

- To validate the Helm chart, execute the following command.

```
helm template -f <values.yaml filepath> <name for this helm deployment> <chart name>
```

Example:

```
helm template pty-publisher/values.yaml publisher pty-publisher
```

> **Note:** A complete Helm chart appears with the values edited in the *values.yaml* file. If there is any error in the output, then it needs to be resolved before installing the Helm chart.

5.10.5 Validating the Status of the PV and PVC

This section describes the commands to validate the status of the PV and PVC.

- To get the list of Persistent Volume (PV) in the namespace, execute the following command.

```
kubectl -n <namespace> get pv
```

The following sample output appears on the CLI.

Table 5-10: PV Output

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	STORAGE CLASS
policy-store	1Gi	RWX	Retain	Bound	test-demo/esa-policy-store	

> **Note:** The status should be Bound post creation of the PVC.

- To get the list of Persistent Volume Claims (PVC) in the namespace, execute the following command.

```
kubectl -n <namespace> get pvc
```

The following sample output appears on the CLI.

Table 5-11: PVC Output

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGE CLASS
esa-policy-store	Bound	policy-store	1Gi	RWX	

> **Note:** The access mode should be ReadWriteMany (RWX). The status will change to Bound after the PVC is created and linked to the PV.

5.10.6 Validating the Publisher Containers

This section describes the commands to validate whether the policy and metadata are created successfully and the MetaServer is responding to the fetch metadata requests.

➤ To validate whether the policy and metadata are created successfully, execute the following steps.

1. To fetch the service name of the Publisher pod, execute the following command.

```
kubectl -n <namespace> get svc | grep publisher*
```

The following sample output appears on the CLI.

Table 5-12: Service Name of the Publisher Pod

NAME	TYPE	CLUSTER-IP	PORT(S)
publisher-pty-publisher	ClusterIP	172.30.15.221	11080/TCP

2. To enter into the interactive mode of the pod, execute the following command.

```
kubectl -n <namespace> exec -it <publisher pod name> -- /bin/bash
```

> **Note:** After executing the command, you will enter the pod in an interactive mode.

3. In the pod, execute the curl request using the following command.

```
curl http://<<name for this helm deployment>>.<<namespace>>:11080/status
```

4. Verify the sample output returned as *json* file with the policy information.

```
{
  "version": "1.0.0",
  "name": "policy_2023_09_06_07_32_12.tgz",
  "volume": "/var/data/policy",
  "path": "demo/policy",
  "timestamp": "1693985533",
  "hash_type": "SHA256",
  "hash": "cc04e6af39246ad60763a09a8df528e9bdec7861b50e67a98dfba04781072e15",
  "kek": "pkcs5"
}
```

5.10.7 Validating the Policy Download by the Publisher Containers

This section describes the commands to validate whether the PEP Server container has downloaded the policy from the ESA.

To get the *pty-publisher-pepserver* logs, execute the following command.

```
kubectl logs -n <<namespace>> <<PUBLISHER_POD_NAME>> -c pty-publisher-pepserver
```

For example, here is the sample output after executing the command. This output indicates that the connectivity to the ESA has been established and the PEP Server container has downloaded the policy from the ESA.

```
2023-08-30 07:47:47,952:pep_installer:INFO:Installing PepServer...

2023-08-30 07:47:47,952:pep_installer:DEBUG:Executing command: ['/bin/bash', '/opt/protegrity/
PepServerSetup_Linux_x64_1.1.0+99.g9521a.1.1.sh', '-dir', '/var/data/staging', '-esa',
'10.21.0.99', '-certuser', 'imp', '-certpw', '********']

2023-08-30 07:47:48,231:pep_installer:DEBUG:Unpacking...

Extracting files...

Downloading certificates from 10.21.0.99:8443...

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current

                                 Dload  Upload   Total   Spent    Left  Speed

100 30720  100 30720    0     0   188k      0 --:--:-- --:--:-- --:--:--  189k


Extracting certificates...

Certificates successfully downloaded and stored in /var/data/staging/defiance_dps/data


Protegrity PepServer installed in /var/data/staging/defiance_dps.
```

To get the *pty-publisher-metaserver* logs, execute the following command.

```
kubectl logs -n <<namespace>> <<PUBLISHER_POD_NAME>> -c pty-publisher-metaserver
```

For example, here is the sample output after executing the command.

```
[INFO ] 2023/08/30 07:47:48.203159 server.go:21: starting server at 0.0.0.0:11080

[DEBUG] 2023/08/30 07:48:09.301992 metainfoprovider.go:17: reading policy metadata from /var/
data/tmpfs/metadata.json
```

If you are storing the policy package in an NFS server, then you can run the following command to navigate to the following location to verify whether the policy package is available.

```
cd /<Path specified in the PVC>/<Path specified while installing the Publisher Pod>
```

If you are storing the policy package in AWS S3 bucket, then you can navigate to the bucket location to verify whether the policy package is available.

5.11 Uninstalling the DSG Container

This section describes the steps how to uninstall the DSG Container.

➤ To uninstall the DSG Container, perform the following steps.

1. Run the following command to uninstall the Helm charts.

   ```
   helm uninstall publisher dsg
   ```

2. Run the following command to delete the pods.

   ```
   kubectl delete pods <POD_Name1> <POD_Name2> .... <POD_NameN> --force
   ```

3. Run the following command to delete the Persistent Volume Claim.

   ```
   kubectl delete pvc <PVC_Name>
   ```

4. Run the following command to delete the Persistent Volume.

   ```
   kubectl delete pv <PV_Name>
   ```

5. Run the following command to delete the namespace.

   ```
   kubectl delete namespace <Namespace>
   ```

6. Run the following command to remove all the Docker images from the host node.

   ```
   docker rmi $(docker images -q) -f
   ```

Appendix A

Appendix A: DSG Container Helm Chart Details

The DSG containers are bundled with Helm charts, which are included in the package. This section explains the details of the Helm chart structure and the entities that the user needs to modify for adapting the charts for their environments.

6.1 Chart.yaml

The *Chart.yaml* file contains information about the Helm versions. These values can be left as default.

```
annotations:
  pepVersion: 1.2.2+42
apiVersion: v1
description: A DSG chart for Kubernetes
name: dsg
version: 1.0.0
```

6.2 Values.yaml

The *Values.yaml* file contains important fields that need to be edited by the user as per the requirements of each environment. The following descriptions explain the details of each field that needs to be replaced.

6.2.1 Image Pull Secrets

This section provides information for the credentials required to access the image repository.

```
## create image pull secrets and specify the name here.
## remove the [] after 'imagePullSecrets:' once you specify the secrets
imagePullSecrets: []
# - name: regcred
```

The user is required to create a Kubernetes secret for accessing the image registry. Kubernetes secrets can be created using existing Docker credentials.

For example:

```
kubectl create secret generic regcred --from-file=.dockerconfigjson=<PATH_TO_DOCKER_CONFIG>/
config.json --type=kubernetes.io/dockerconfigjson --namespace <NAMESPACE>
```

6.2.2 Name Override

These values indicate how naming for releases are managed for deployment. Protegrity recommends that these values should not be changed by the user.

If the ***fullnameoverride*** parameter value is specified, then the deployment names will use that value.

If the ***nameOverride*** parameter value is specified, then the deployment names will be combination of the ***nameOverride*** parameter value and the Helm chart release name.

If both the values are empty, then the Helm chart release name will be used.

```
nameOverride: ""
fullnameOverride: ""
```

6.2.3 Container Image Repository

This section provides the path for the images in the fields that the users are required to enter after loading the images into their registry.

```
dsgImage:
  repository: REPOSITORY_DSG_IMAGE
  tag: "DSG_IMAGE_TAG"
  pullPolicy: Always
  debug: false # change debug to true to get container debug logs on STDOUT.
```

The following list provides details for the *dsgImage* parameter, which refers to the details for the DSG container image:

- *repository* – Refers to the name of the registry.

  > **Note:**
  > Ensure that you only add the Container registry path as the repository value. Do not include the tag name in this value. You need to add tag name as the value in the *tag* field.

- *tag* – Refers to the tag mentioned at the time of image upload.
- *debug* - Set the value of this field to *true*, if you want debug logs for the DSG container. By default, the value of this field is set to *false*.
- *dsgImage* – Refers to details for the DSG container image.

6.2.4 Resources

This section specifies the resource limits that can be set for the containers in a pod.

```
dsgContainerResources:
  ## Below allocation covers largest policy that can be imported.
  ## Based on the policy size/application requirement update below limits.
  limits:
    cpu: 1000m
    memory: 3500Mi
  requests:
```

```
    cpu: 200m
    memory: 200Mi
```

You can specify the following parameters:

- *limits* - Specify the resource limits for the containers in a pod. These limits ensure that the running container does not use additional resources than the limit you set.
- *requests* - Specify the resource request for the containers in a pod. This information determines the nodes for deploying the pods.

For more about the resource parameters, refer to the section *Managing Resources for Containers* in the Kubernetes documentation.

6.2.5 Subscriber-Sidecar Container Image Repository

This section provides the path for the images in the fields that the users are required to enter after loading the images into their registry.

```
sidecarImage:
  repository: $SUBSCRIBER_SIDECAR_IMAGE_REPOSITORY
  tag: $SUBSCRIBER_SIDECAR_IMAGE_TAG
  pullPolicy: Always
  debug: false # change debug to true to get container debug logs on STDOUT.
```

The following list provides details for the *sidecarImage* parameter, which refers to the details for the Subscriber-Sidecar container image:

- *repository* – Refers to the name of the registry.

  > **Note:**
  > Ensure that you only add the Container registry path as the repository value. Do not include the tag name in this value. You need to add tag name as the value in the *tag* field.

- *tag* – Refers to the tag mentioned at the time of image upload.
- *debug* - Set the value of this field to *true* if you want debug logs for the Subscriber-Sidecar container. By default, the value of this field is set to *false*.
- *sidecarImage* – Refers to details for the Subscriber-Sidecar container image.

6.2.6 Subscriber-Sidecar Resources

This section specifies the resource limits that can be set for the containers in a pod.

```
sidecarResources:
  ## Below allocation covers largest policy that can be imported.
  ## Based on the policy size/application requirement update below limits.
    limits:
      cpu: 1000m
      memory: 3500Mi
    requests:
      cpu: 200m
      memory: 200Mi
```

You can specify the following parameters:

- *limits* - Specify the resource limits for the containers in a pod. These limits ensure that the running container does not use additional resources than the limit you set.
- *requests* - Specify the resource request for the containers in a pod. This information determines the nodes for deploying the pods.

For more about the resource parameters, refer to the section *Managing Resources for Containers* in the Kubernetes documentation.

6.2.7 Persistent Volume Claim

This section specifies the value of the persistent volume claim that will be used to store the policy and CoP package.

```
## persistent volume name e.g. nfs-pvc
pvcName: PVC_NAME
```

For more information about persistent volumes and persistent volume claims, refer to the section *Persistent Volumes* in the Kubernetes documentation.

6.2.8 Pod Security Context

This section specifies the privilege and access control settings for the pod.

```
## set the Pod's security context object
## leave the field empty if not applicable
podSecurityContext:
  # runAsUser: 1000
  # runAsGroup: 1000
  # fsGroup: 1000
```

For more information about the Pod Security Context, refer to the section *Configure a Security Context for a Pod or Container* in the Kubernetes documentation.

For more information about the parameters, refer to the section *PodSecurityContext v1 Core* in the Kubernetes API documentation.

6.2.9 Container Security Context

This section specifies the privilege and access control settings for the DSG and the Subscriber-Sidecar container.

```
# Security Context object for DSG container
## leave the field empty if not applicable
dsgContainerSecurityContext:
  capabilities:
    drop:
    - ALL
  allowPrivilegeEscalation: false
  privileged : false
  runAsNonRoot : true
  readOnlyRootFilesystem: true
  seccompProfile:
    type: RuntimeDefault

# Security Context object for Sidecar container
sidecarSecurityContext:
  capabilities:
    drop:
    - ALL
  allowPrivilegeEscalation: false
  privileged : false
  runAsNonRoot : true
  readOnlyRootFilesystem: true
  seccompProfile:
    type: RuntimeDefault
```

Leave the field blank if it is not applicable.

For more information about the Container Security Context, refer to the section *Configure a Security Context for a Pod or Container* in the Kubernetes documentation.

For more information about the parameter, refer to the section *SecurityContext v1 core* in the Kubernetes API documentation.

6.2.10 Pod Service Account

This section provides information for the name of the service account that must be created for the pod in the Kubernetes Cluster.

> **Note:** The *Pod Service Account* parameter is not applicable for the Azure release.

```
serviceAccount: <Name of the service account created for the pod>
```

6.2.11 Liveliness and Readiness Probe Settings

This section specifies the intervals required to run health checks for the DSG container images. The users must not change these settings.

```
## Configure the delays for Liveness Probe here
livenessProbe:
  initialDelaySeconds: 15
  periodSeconds: 20

## Configure the delays for Readiness Probe here
readinessProbe:
  initialDelaySeconds: 15
  periodSeconds: 20
```

The health check is performed by sending a request to the ***http://<hostname>/protect/text*** URI. This URI is defined as part of the *Text Protection* ruleset, which is the default ruleset available on the ESA. Ensure that you retain this default ruleset on the ESA so that the health check is performed successfully.

6.2.12 Sidecar Object

This section provides a brief overview for the *sidecarObject* parameter.

```
# The configuration for the subscriber sidecar container.
# Note: any secret required by the sidecar container must be created in the target namespace
# WARNING: ALL fields are MANDATORY
sidecarObject:
  # interval(in seconds) at which sidecar container check for policy update information
  pingInterval: 60

  # Address of the publisher service to fetch policy update information
  publisher:
    # hostname of service hosting the policy update information
    # hostname has following form
    # <name used for publisher helm chart install>-pty-publisher.<publisher namespace>.svc
    # OR
    # <publisher service name>.<publisher namespace>.svc
    host: $PUBLISHER_SVC_HOSTNAME
    port: 443
    tls:
      # publisher CA certificate secret name.
      # eg. kubectl command:
      #     kubectl -n $DSG_NAMESPACE create secret generic pty-pub-cert \
      #     --from-file=ca.pem=./certs/ca.pem
      caCertSecret: $PUBLISHER_CA_CERT_SECRET
```

6.2.13 Password-Based Encryption (PBE) Secrets

This section provides a brief overview for the *pbeSecrets* parameter. If you are using PBE encryption to encrypt the policy package, then the DSG container uses the value specified in the *pbeSecrets* parameter to decrypt the policy.

```
## k8s secret containing passphrase for decrypting policy
pbeSecrets: PBE_SECRET_NAME
```

6.2.14 Storage Access Secrets

This section provides a brief overview for the *storageAccessSecrets* parameter. This secret contains the credentials for pulling the CoP and the Policy from the Azure Storage Container. The DSG Application container uses the value specified in the *storageAccessSecrets* parameter to decrypt the Policy package using the KMS.

```
# k8s secret for storing credentials with read access to azure storage account
# keep empty for GCP and AWS
storageAccessSecrets:STORAGE_ACCESS_SECRET_NAME
```

6.2.15 Private Secret Key Source and ID

This section provides a brief overview for the *privateKeySource* and *privateKeyId* parameters. If you are using KMS to encrypt the policy package, the DSG container uses the value specified in the *privateKeyId* parameter to decrypt the policy package. It is based on the key source that you have specified in the *privateKeySource* parameter.

```
## If not using PBE, Choose your private key helper.
  ## Currently we support [AWS_KMS, AZ_VAULT]
  privateKeySource: $KEY_SOURCE

  privateKeyId: $KEY_ID
```

6.2.16 Security Config Destination

This section provides a brief overview for the *securityConfigDestination* parameter. It specifies the storage destination for the policy package and COP.

```
## choose your storage destination. Currently we support <VolumeMount|AWS|AZURE>
securityConfigDestination: VolumeMount
```

6.2.17 CoP Secret

This section provides a brief overview for the *copSecrets* parameter. The DSG container uses the value specified in the *copSecrets* parameter to decrypt the CoP package.

```
## k8s secret containing passphrase for decrypting cop
copSecrets: COP_SECRETE_NAME
```

6.2.18 LDAP Secrets

This section provides a brief overview for the *ldapXmlSecrets* parameter. The DSG container uses the value specified in the *ldapXmlSecrets* parameter to connect to the LDAP.

```
# LDAP configuration secret name
ldapXmlSecrets: LDAP_SECRET_NAME
```

6.2.19 COP

The *filePath* parameter specifies the location where the COP is stored.

```
cop:
    # absolute path in ObjectStore from where the cop dump file will be fetched.
    # <S3> "s3://bucketName/pathToCop"
    # <GS> "gs://bucketName/pathToCop"
    # <Azure> " https://<accountName>.blob.core.windows.net/<container name>/<pathToCop>"
    # absolute path in PV mount from where the cop dump file will be fetched.
    # <VOLUME> "file://var/data/ptypolicy/pathInVolumeToPolicy"
    # relative path in PV mount from where the cop dump file will be fetched.
    # it will prepend the '/var/data/ptypolicy' to below path
    # <VOLUME> "file:///pathInVolumeToCop"
```

```
    # <VOLUME> "pathInVolumeToCop" # without any leading '/'
    filePath: $ABSOULTE_COP_PATH
```

The path is case sensitive.

6.2.20 Autoscaling

This section specifies the settings for the autoscaling of pods on the cluster.

```
## specify the initial no. of DSG Pod replicas
replicaCount: 1

## HPA configuration
autoScaling:
  minReplicas: 1
  maxReplicas: 10
  targetCPU: 50
```

The following list provides a description for the parameters:

- *replicaCount* - Indicates the number of pods that get instantiated when the deployment starts.
- *minReplicas* - Indicates the minimum number of pods that will keep running in the deployment for a cluster.
- *maxReplicas* - Indicates the maximum number of pods that will keep running in the deployment for a cluster.
- *targetCPU* - Horizontal Pod Autoscaler (HPA) will increase and decrease the number of replicas (through the deployment) to maintain an average CPU utilization across all Pods based on the *%* value specified in the *targetCPU* parameter.

6.2.21 Service Configuration

This section has configurations for the REST service to be used on the cluster running the DSG containers.

```
## specify the ports exposed in your DSG configurations where,
## name - distinguishes between different ports.
## port - the port on which you wan't to expose the service externally.
## targetPort - the port no. configured while creating Tunnel.
dsgService:

  # only applicable if ingress is disabled
  # allows you to configure service type: LoadBalancer or ClusterIP
  type: LoadBalancer

  # only apply, if the ingress is disabled
  # Specify service related annotations here
  annotations:
    #service.beta.kubernetes.io/aws-load-balancer-internal: "true"
    #networking.gke.io/load-balancer-type: "Internal"
    #cloud.google.com/load-balancer-type: "Internal"
    #service.beta.kubernetes.io/azure-load-balancer-internal: "true"

  healthCheckRestService:
    scheme: HTTP
    host: restapi
    port: &healthCheckPort 8080
    path: /protect/text
  tunnels:
    - name: "restapi"
      port: 8080
      targetPort: *healthCheckPort
    # Add the below object for new tunnel,
    # specify the port used while creating TLS enabled HTTP Tunnel in "targetPort"
    # - name: "service1"
    #   port: 9443
    #   targetPort: 9443
```

You can specify the value of the *dsgService/type* parameter as *LoadBalancer* or *ClusterIP*.

The value of the *healthCheckRestService/port* parameter must be set to the same port value as that of the tunnel used in the *REST API Examples* service on the ESA. This port is used to perform the Readiness probe.

Update the health check-related section with the following details:

- *scheme* - The protocol used to send a request for checking the health of the DSG service.
- *host* - The host name of the heath check rule defined in the CoP.
- *port* - The port number configured in the DSG CoP ruleset for the health check rule.
- *path* - The URI of the health check rule configured in the DSG CoP ruleset.

If the user requires to add a new tunnel for the DSG Containers, then the following points must be ensured:

- The required CoP must be built on the ESA.
- The CoP and policy package should be made available on the Azure Storage Container or the Azure File Share.
- Add the tunnel-related section:
  - *name* - Distinguishes between different ports used in the DSG tunnels. The name must contain only lowercase alphanumeric characters, which is based on standard Kubernetes naming conventions.
  - *port* - Specifies the port on which you want to expose the service externally.
  - *targetPort* - Specifies the port number configured while creating the Tunnel.

6.3 credentials.json

The *credentials.json* file contains information about the credentials for the service principal 2 that are required to access the Azure storage account. The credentials are required to enable the service principal 2 to retrieve the IMP Metadata Package that has been uploaded to the Azure storage container. The following snippet is shown for Azure. Users are required to edit the file and replace the values for the service principal 2 credentials.

```
{
  "storage_account_name" : "AZURE_STORAGE_ACCOUNT_NAME",
  "tenant_id" : "AZURE_TENANT_ID",
  "client_id" : "AZURE_CLIENT_ID",
  "client_secret": "AZURE_CLIENT_SECRET"
}
```

Appendix B

Appendix B: Using the Sample Application

This section explains details and usage of the components included in included in the *DSG-Samples_Linux-ALL-ALL_x86-64_<DSG_Containerization_App_version>.tgz* archive.

7.1 Overview

Protegrity delivers a sample application as part of the DSG Container installation package. The sample application constitutes a canned policy (to get imported to the ESA), canned CoP, sample Postman collection (to test the DSG Container Pods serving deployed IMP) and an auto-scaling script (to push more load to the Kubernetes cluster to force auto-scaling of the DSG Container Pods).

On consuming the deliverables in the DSG Container installation package, the users must run this sample application end-to-end, as a sanity test, to confirm that the installation was completed accurately. In this section, we are providing details on the exact steps that the user needs to follow to run the sample application end-to-end. This section explains details and usage of the components included in the *DSG-Samples_Linux-ALL-ALL_x86-64_<DSG_Containerization_App_version>.tgz* archive.

The following components are included in the *DSG-Samples_Linux-ALL-ALL_x86-64_<DSG_Containerization_App_version>.tgz* archive.

7.1.1 Policy Sample

Package – *Sample_App_Policy_V5.tgz*

This component consists of the sample policy that can be imported on the ESA 9.1.0.5 for getting started with the DSG Containers use case. The following table contains the policy details.

Policy Name - Sample_policy

Token Type	Data Element Name
Alphanumeric	deFirstName

Token Type	Data Element Name
Alphanumeric	deLastName
Alphanumeric	deStreet
Alphanumeric	deCity
Alpha	deState
Alphanumeric	deZipcode
Numeric	deSSN
Unicode	PTY_UNICODE

7.1.2 CoP / RuleSet Sample

Package – *Sample_App_COP_V5.zip*

This component consists of sample ruleset for REST for getting started with the DSG Containers use case. The following are the details for the ruleset.

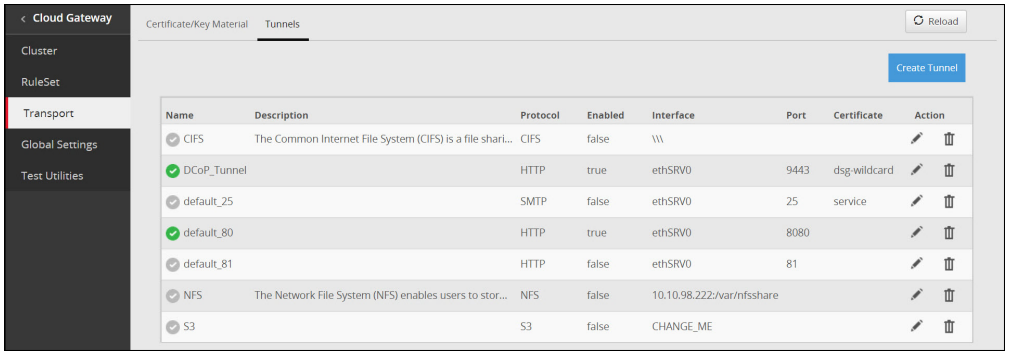
Ruleset 1

- RuleSet Name - REST API Example
- Port for REST – 8080

> **Caution:**
> This ruleset is only applicable for performing a health check. Ensure that you do not modify this ruleset.

Ruleset 2

- RuleSet Name - DynamicCoP
- Port for REST (with TLS) - 9443
- Format Supported for Dynamic CoP – ALL
- Path for Dynamic CoP - /protect



> **Note:**
> In the sample, if you are using the TLS for communication between the Postman client and the DSG instance on the Kubernetes pod, then ensure that the port number is set to *9443* in the Tunnel.

7.1.3 Autoscaling Script

Package – *Sample_App_autoscale.sh*

Script for making 10,000 REST calls to the DSG. This script can be triggered to test the autoscaling of the pods.

7.1.4 PostMan Collection

Package – *Sample_App_PostMan_Collection_V4.json*

This collection can be used to make REST calls to the DSG for protecting the data set provided in the *Sample App Test Data.csv* file. The collection has two entries (Release 1 and Release 2).

Release 1

Post Request Path - **`https://prod.example.com/protect`**

Body (Data to be Protected)

The request is built using the *Sample App Test Data.csv*. The contents of the *.csv* file are added to the *Body* of the request.

Header - DCOP request

The header is populated with the DCOP request, which protects data using the following Data Elements.

Field	Column Name	Token Type	Data Element Name
First Release			
First Name	first name	Alphanumeric	deFirstName
Last Name	last name	Alphanumeric	deLastName
Street	street address	Alphanumeric	deStreet
City	City	Alphanumeric	deCity
State	State Name	Alpha	deState
Zip Code	zip code	Alphanumeric	deZipcode
SSN	SSN	Numeric	deSSN

Release 2

Post Request Path - **`https://staging.example.com/protect`**

Body (Data to be Protected)

The request is built using the *Sample App Test Data.csv* file. The contents of the *.csv* file are added to the *Body* of the request.

Header - DCOP request

THe header is populated with the DCOP request, which protects data using the following Data Elements. Note that the *Customer ID* is added as a new data element to protect the *Customer ID* column.

Field	Column Name	Token Type	Data Element Name
First Release			
First Name	first name	Alphanumeric	deFirstName

Field	Column Name	Token Type	Data Element Name
Last Name	last name	Alphanumeric	deLastName
Street	street address	Alphanumeric	deStreet
City	City	Alphanumeric	deCity
State	State Name	Alpha	deState
Zip Code	zip code	Alphanumeric	deZipcode
SSN	SSN	Numeric	deSSN
Customer ID	**Customer ID**	**Customer ID**	**Customer ID**

7.2 Running the Sample Application

1. The Azure Storage Container is created.
2. The cluster is created. For more information about creating a Kubernetes Cluster, refer to the section *Creating a Kubernetes Cluster*.

Ensure that the *Prerequisites* are met before performing the following steps.

The user must perform the following steps.

1. Generate Certificates and Keys for TLS Authentication
2. Import Certificates on the ESA
3. Import certificates to the Postman Client
4. Enable the Authentication functionality
5. Import the Policy sample on ESA
6. Import the CoP sample on the ESA
7. Create Immutable Metadata Packages (IMP)
8. Deploy Release 1
9. Run Sample Application (PostMan collection)
10. Run Autoscaling Script

7.2.1 Generating Certificates and Keys for TLS Authentication

This section describes how you can create certificates and keys for establishing a secure communication between the Postman client and the server.

Before you begin
Ensure that OpenSSL is installed on the Linux instance to create the required certificates.

➤ To initialize the Linux instance:

1. On the Linux instance, run the following command to create a CA certificate and a private key for the certificate.
   ```
   openssl req -x509 -sha256 -newkey rsa:2048 -keyout dsg-ca.key -out dsg-ca.crt -days 356 -nodes -subj '/CN=DSG Certificate Authority'
   ```
2. On the Linux instance, create a server certificate and a private key that have been signed using the private key of the CA created in *step 1*.

```
openssl req -new -newkey rsa:2048 -keyout dsg-wildcard.key -out dsg-wildcard.csr
-nodes -subj '/CN=*.example.com'

openssl x509 -req -sha256 -days 365 -in dsg-wildcard.csr -CA dsg-ca.crt -CAkey dsg-
ca.key -set_serial 04 -out dsg-wildcard.crt
```

3. On the Linux instance, create a client certificate and key that have been signed using the private key of the CA created in *step 1*.

```
openssl req -new -newkey rsa:2048 -keyout dsg-client.key -out dsg-client.csr -nodes
-subj '/CN=My DSG Client'

openssl x509 -req -sha256 -days 365 -in dsg-client.csr -CA dsg-ca.crt -CAkey dsg-
ca.key -set_serial 02 -out dsg-client.crt
```

4. Copy all the certificates to a common directory.

   For example, create a directory named *dsg-certs* and copy all the certificates that have been created to this directory.

7.2.2 Importing Certificates on the ESA

The user needs to run the following steps to import CA and server certificates, and keys on the ESA, if you want to ensure secure communication between the Postman client and the DSG pods using TLS.

➤ To import CA and server certificates on the ESA:

1. Login to the ESA as *admin*.
2. Navigate to the **Cloud Gateway** > **Transport** > **Certificate/Key Material** tab.
3. Click **Choose File** to browse for the *dsg-ca.crt* certificate in the *dsg-certs* directory that you have created in the section *Creating Certificates and Keys for TLS Authentication*.
4. Select the certificate.
5. Click **Upload certificate**.
6. Repeat steps *3* to *5* to upload the following files:
   - *dsg-ca.key* - Private key for the CA certificate
   - *dsg-wildcard.crt* - Server certificate
   - *dsg-wildcard.key* - Private key for the server certificate

   > **Note:** If you are uploading a key, then you are prompted to specify a password in the **Password** field for encrypting the key. You also have to re-type the password in the **Confirm Password** field.
   >
   > However, you can choose to directly upload the key without specifying the password.

7. Perform the following steps to create a DSG tunnel for the TLS traffic between the REST API client and the DSG pods on the Azure.
   a. Navigate to the **Tunnels** tab.
   b. Click **Create Tunnel** to create a new tunnel for the TLS traffic.

      The **Create Tunnel** screen appears.
   c. Enter the required details to create a new tunnel.

For more information about creating a tunnel, refer to the section *Tunnels tab* in the *Data Security Gateway User Guide 3.1.0.5*.

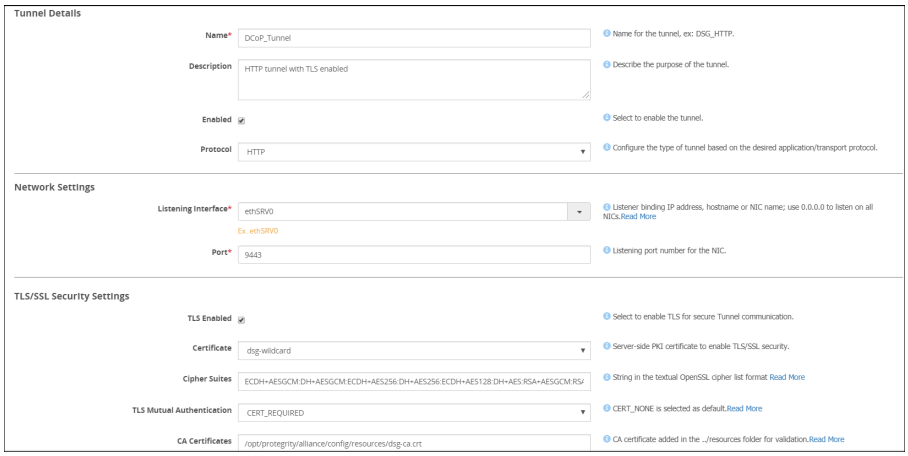


Figure B-1: Create Tunnel Screen

d. In the **TLS/SSL Security Settings** section, select the **TLS Enabled** check box to enable TLS for secure tunnel communication.

e. In the **Certificate** list, select the server certificate that you created in *step 2* of the section *Setting Up the Certificates*.

f. In the **TLS Mutual Authentication** list, select the value as *CERT_REQUIRED* to ensure mutual TLS authentication between the Postman client and the DSG instance.

g. In the **CA Certificates** field, specify the path where you have uploaded the CA certificate.

   For example, specify the path as */opt/protegrity/alliance/config/resources/dsg-ca.crt*.

h. Click **Reload** to refresh the list of tunnels on the **Tunnels** tab.
   Use this newly created tunnel while creating your ruleset.

7.2.3 Importing Certificates to the Postman Client

The user needs to perform the following steps to import the CA certificate, the client certificates, and keys to the Postman client, if you want to ensure secure communication between the Postman client and the DSG pods using TLS.

➤ To import certificates to the Postman client:

1. Open the Postman client.
2. In the header of the Postman client, click , and then select **Settings**.
   The **SETTINGS** dialog box appears.

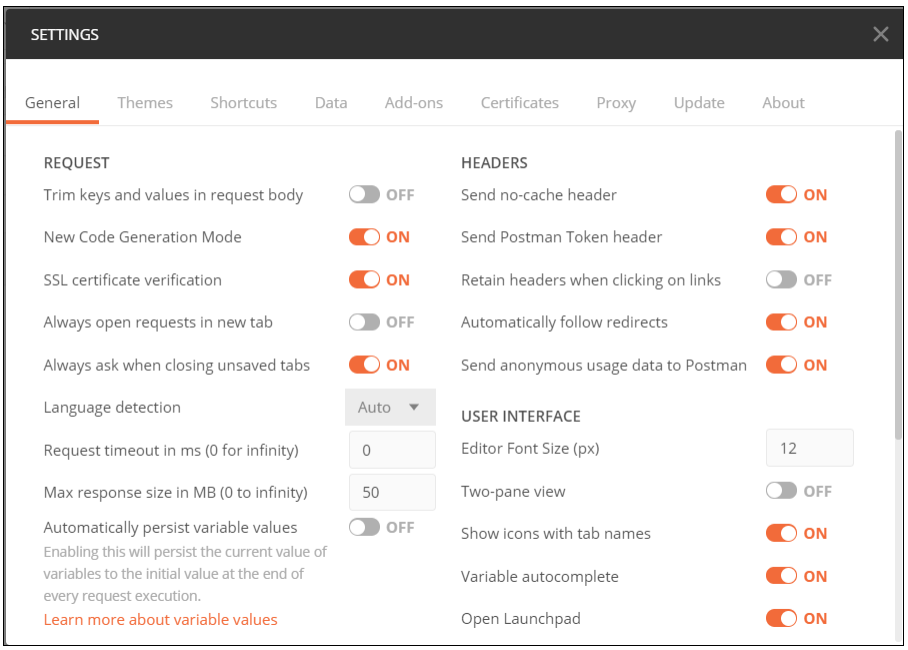


Figure B-2: SETTING Dialog Box

3. Navigate to the **Certificates** tab.

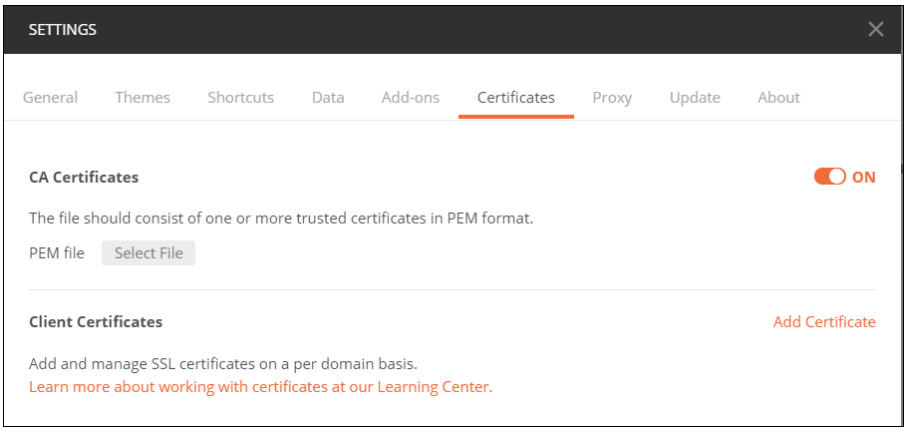


Figure B-3: Certificates Tab

4. In the **CA Certificates** section, click **Select File** and browse for the *dsg-ca.crt* file in the *dsg-certs* directory that you have created in the section *Setting Up the Certificates*.
5. In the **Client Certificates** section, click **Add Certificate**.

   The **Add Certificate** screen appears.

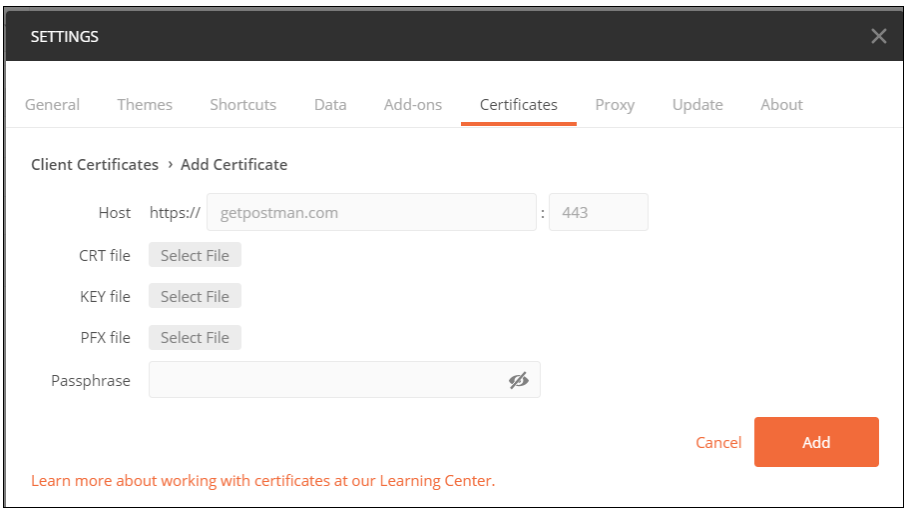


Figure B-4: Add Certificate Screen

6. In the **Host** field, specify the value as *prod.example.com*.
   You need to specify the DSG service port number, which is *443* by default.
7. In the **CRT file** field, click **Select File** to browse for the *dsg-client.crt* file in the *dsg-certs* directory.
8. In the **KEY file** field, click **Select File** to browse for the *dsg-client.key* file in the *dsg-certs* directory.
9. Click **Add** to add the client certificate and key to the Postman client.
10. Repeat step 5 to step 9 for adding client certificates for the host *staging.example.com*.

7.2.4 Enabling the Authentication Functionality

The sample application provides the OAuth UDF that is used to implement the authentication functionality for the DSG instance. This functionality uses an access token, which is a JSON Web Token (JWT), to authenticate the requests that are sent from a REST API client to a DSG instance for performing security operations. The access token is digitally signed by Microsoft and is then included in the authorization header of the request that is sent to the DSG instance. The OAuth UDF is used to authenticate this access token.

For more information about JWT, refer to the section *Using JSON Web Token (JWT)* in the *Appliances Overview Guide 9.1.0.5*.

➤ To enable the authentication functionality:

1. Generate a token for the application by performing the following steps.
   a. Navigate to the **App Registrations** page in Azure AD.
   b. Select the **New registration** option to create an application.
      The **Register an application** page appears.
   c. Specify the name of the application using the **Name** input box. Leave the other fields empty.
   d. Click **Register** to create a new application.
   e. Navigate to the new application using the **App registrations** page by selecting the new application by name.
      For example: app *://dsg-demo*
   f. Note the application (client) id from the **Overview** tab.
   g. Navigate to the **Certificates & secrets** page and create a new secret for the application. Note the newly created certificates.
   h. Navigate to the **API permissions** tab to remove any permission if present. It should have zero permissions.

i. Navigate to the **Expose an API** tab and create **Application ID URI**.

Note the application ID URI.

j. Create a client credentials *openid* flow based request using the following URL.

https://docs.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-client-creds-grant-flow

k. Send the following CURL request to obtain the access token.

```
curl -X POST -H "Content-Type: application/x-www-form-urlencoded" -d
'client_id=<client id from step 1f.>&scope=<url encoded application
id URI from step 1i>.default&client_secret=<client secret from
step 1g>&grant_type=client_credentials' 'https://login.microsoftonline.com/
<TENANT_ID>/oauth2/v2.0/token'
```

For example:

```
curl -X POST -H "Content-Type: application/x-www-form-
urlencoded" -d 'client_id=<client id from step
1f.>&scope=api%3A%2F%2Fdsg-demo%2F.default&client_secret=<client secret from
step 1g>&grant_type=client_credentials' 'https://login.microsoftonline.com/
<TENANT_ID>/oauth2/v2.0/token'
```

You need to copy this token and paste it in the *Token* field of the authorization header in the REST API request that you send to the DSG instance.

> **Important:**
> Ensure that the Application ID URI should end with a / (forward slash).
>
> For example: *api://test-auth/*

> **Important:**
> The generated token has a default timeout of 60 minutes, and you need to regenerate the token after it times out.

2. Perform the following steps to enable the OAuth UDF in the Dynamic CoP.

a. Login to the ESA as *admin*.

b. Navigate to the **Cloud Gateway** > **RuleSet** tab.

c. Expand **DynamicCoP** > **DynamicCoP** > **Extract HTTP request message body**.

d. Select the **OAuth** UDF.

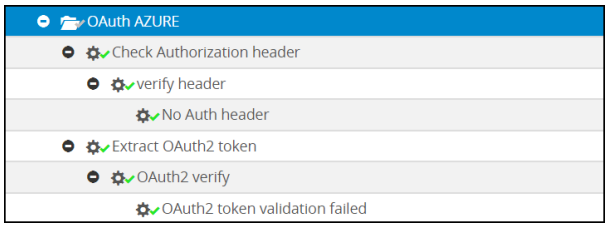


Figure B-5: OAuth UDF

The OAuth ruleset consists of the following rules:

- *Check Authorization header* - Used to verify whether you have included a token in the authorization header of the REST API request.
- *Extract Authorization OAuth Token* - Used to verify whether a valid token has been included in the authorization header.

By default, the OAuth UDF is not enabled.

e. Click **Edit** to edit the UDF.

f. Select the **Enabled** check box.

g. Expand the **Extract Authorization OAuth2 Token** > **OAuth2 verify** rule.

The **Initialization Arguments** section appears on the right side panel.

h. Specify the following initialization arguments.

Argument Name	Description	Value	Required / Optional
Debug logging	Specify whether you want to run the UDF in debugging mode	• true • false The default value is set to *true*.	Required
JSON Web Key Set (JWKS) certificates	Specify the public keys that are used to digitally sign the ID token. You can obtain the latest JWKS from Mircosoft by accessing the following URL: ***https://login.microsoftonline.com/TENANT_ID/discovery/v2.0/keys*** The *TENANT_ID* is the tenant ID of the application that you have created in *step 1*. Ensure that you specify valid keys. If you specify keys that are already expired, then the OAuth UDF automatically retrieves the latest keys by accessing the URL specified in the *JWKS URI* argument. However, the automatic retrieving of keys from the JWKS URL can impact the performance of the REST API request.		Required
JWKS URI	Specify the URL used to obtain the JWKS from Microsoft. For example, specify the following value as the URL: ***https://login.microsoftonline.com/TENANT_ID/discovery/v2.0/keys*** The *TENANT_ID* is the tenant ID of the application that you have created in *step 1*.		Required

i. Click **Save** to save the changes.

j. Click **Reload**.

7.2.5 Importing the CoP Sample on the ESA

The user needs to run the following steps to import the *Sample_App_COP_V4.zip* file on the ESA.

To import CoP sample on ESA:

1. Login to the ESA as *admin*.
2. Navigate to the **Cloud Gateway** > **Ruleset** tab.
3. Click **Import**.
4. Click **Choose File** and select *Sample_App_COP_V4.zip*.
5. Select the **Override existing configuration in case of conflicts** check box.
6. Click **Import Configuration**.
   After successful import, the *REST API Example* and *DynamicCop* rulesets should be available in the section RuleSet.
7. Run the following command to export the CoP package.

```
curl --location --request POST 'https://xxx.xxx.xxx.xxx/exportGatewayConfigFiles' \
--header 'Authorization: Basic YWRtaW46YWRtaW4xMjM0' \
--header 'Content-Type: text/plain' \
--data-raw '{
"nodeGroup": "LOB1",
"kek":{
"pkcs5":{
"passphrase": "Passphrase1#",
"salt": "salt"
}
}
}' -k -o cop_demo.json
```

   For more information about exporting the CoP package, refer to the *step 12* in the section *Creating and Exporting the Ruleset*.

8. Upload the CoP package to an Azure Storage Container or Azure File Share.
   For more information about uploading the CoP package to an Azure Storage Container, refer to the *step 13* in the section *Creating and Exporting the Ruleset*.

   For more information about uploading the CoP package to an Azure File Share, refer to the *step 14* in the section *Creating and Exporting the Ruleset*.

> **Important:**
> Ensure that the user has the required permissions to upload an object to the Azure Storage Container or persistent volume.

7.2.6 Deploying Release

1. Ensure that the *Values.yaml* file has the following configuration set on the Linux node.

```
...
...

# The configuration for the subscriber sidecar container.
```

```
# Note: any secret required by the sidecar container must be created in the target
namespace
# WARNING: ALL fields are MANDATORY
sidecarObject:
  # interval(in seconds) at which sidecar container check for policy update information
  pingInterval: 60

  # Address of the publisher service to fetch policy update information
  publisher:
    # hostname of service hosting the policy update information
    # hostname has following form
    # <name used for publisher helm chart install>-pty-publisher.<publisher namespace>.svc
    # OR
    # <publisher service name>.<publisher namespace>.svc
    host: $PUBLISHER_SVC_HOSTNAME
    port: 443
    tls:
      # publisher CA certificate secret name.
      # eg. kubectl command:
      #     kubectl -n $DSG_NAMESPACE create secret generic pty-pub-cert \
      #     --from-file=ca.pem=./certs/ca.pem
      caCertSecret: $PUBLISHER_CA_CERT_SECRET

  # the PBE passphrase and salt as Kubernetes secret required to decrypt the policy package
  # eg. kubectl command:
  #     kubectl -n $DSG_NAMESPACE create secret generic pty-pbe \
  #     --from-literal='passphrase=<complex chars>' --from-literal='salt=<complex chars>'
  pbeSecretName: $PBE_SECRET

  ## If not using PBE, Choose your private key helper.
  ## Currently we support [AWS_KMS, AZ_VAULT]
  privateKeySource: $KEY_SOURCE

  privateKeyId: $KEY_ID

## choose your storage destination. Currently we support <VolumeMount|AWS|AZURE>
securityConfigDestination: VolumeMount

## If securityConfigDestination is Volume Mount,
## specify the name of persistent volume claim to be used for this deployment.
pvcName: $PVC_NAME

## If securityConfigDestination is AZURE, then uncomment and
## specify k8s secret for storing credentials with access azure storage account.
# eg. kubectl command:
#     kubectl -n $DSG_NAMESPACE create secret generic storage-secret \
#     --from-file=credentials=./credentials.json'
storageAccessSecrets: $STORAGE_SECRET

## k8s secret containing passphrase for decrypting cop
copSecrets: $COP_SECRET_NAME

# LDAP configuration secret name
ldapXmlSecrets:

cop:
  # absolute path in ObjectStore from where the cop dump file will be fetched.
  # <S3> "s3://bucketName/pathToCop"
  # <GS> "gs://bucketName/pathToCop"
  # <Azure> " https://<accountName>.blob.core.windows.net/<container name>/<pathToCop>"
  # absolute path in PV mount from where the cop dump file will be fetched.
  # <VOLUME> "file://var/data/ptypolicy/pathInVolumeToPolicy"
  # relative path in PV mount from where the cop dump file will be fetched.
  # it will prepend the '/var/data/ptypolicy' to below path
  # <VOLUME> "file:///pathInVolumeToCop"
  # <VOLUME> "pathInVolumeToCop" # without any leading '/'
  filePath: $ABSOULTE_COP_PATH

...
...

## specify the ports exposed in your DSG configurations where,
## name - distinguishes between different ports.
```

```
## port - the port on which you wan't to expose the service externally.
## targetPort - the port no. configured while creating Tunnel.
dsgService:

  # only applicable if ingress is disabled
  # allows you to configure service type: LoadBalancer or ClusterIP
  type: LoadBalancer

  # only apply, if the ingress is disabled
  # Specify service related annotations here
  annotations:
    #service.beta.kubernetes.io/aws-load-balancer-internal: "true"
    #networking.gke.io/load-balancer-type: "Internal"
    #cloud.google.com/load-balancer-type: "Internal"
    #service.beta.kubernetes.io/azure-load-balancer-internal: "true"

  healthCheckRestService:
    scheme: HTTP
    host: restapi
    port: &healthCheckPort 8080
    path: /protect/text
  tunnels:
    - name: "restapi"
      port: 8080
      targetPort: *healthCheckPort
    # Add the below object for new tunnel,
    # specify the port used while creating TLS enabled HTTP Tunnel in "targetPort"
    # - name: "service1"
    #   port: 9443
    #   targetPort: 9443
```

2. Deploy the Release on the cluster.

   For more information about deploying a release on the Kubernetes Cluster, refer to the section *Deploying a Release on the Kubernetes Cluster*.

7.2.7 Running Sample Application (PostMan collection)

The component consists of the PostMan JSON file to generate the REST request for protecting data.

Post Request Path - **`https://prod.example.com/protect`**

> **Note:**
> If you want to test the sample application with the ESA, then you must specify the post request path as **`http://restapi:9090/protect`**.

Body (Data to be Protected)

The request is built using the *Sample App Test Data.csv* file. The contents of the *.csv* file are added to the *Body* of the request.

Header - DCOP request

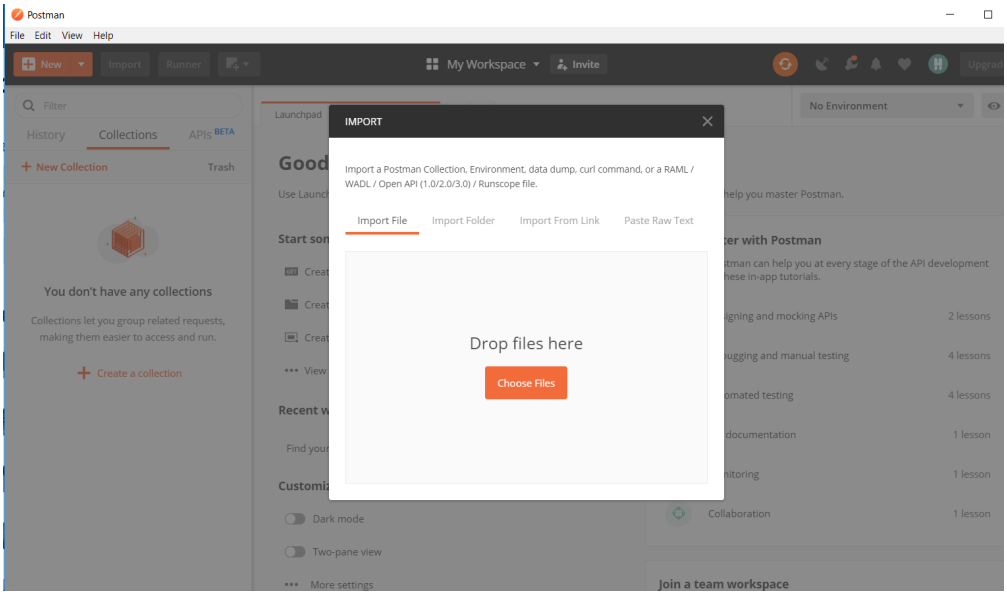
The header is populated with the DCOP request, which protects data using the following Data Elements.

Field	Column Name	Token Type	Data Element Name
First Release			
First Name	first name	Alphanumeric	deFirstName
Last Name	last name	Alphanumeric	deLastName
Street	street address	Alphanumeric	deStreet

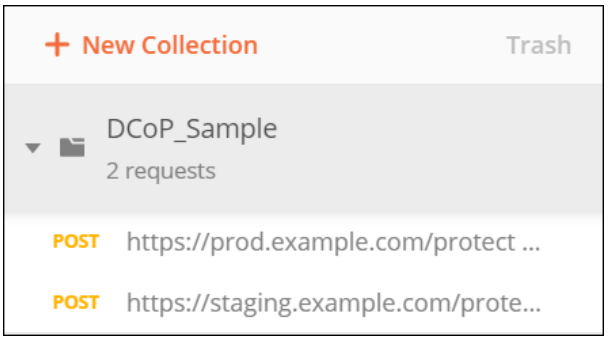
Field	Column Name	Token Type	Data Element Name
City	City	Alphanumeric	deCity
State	State Name	Alpha	deState
Zip Code	zip code	Alphanumeric	deZipcode
SSN	SSN	Numeric	deSSN

For protecting data with Release 1

1. Import the Postman collection.



2. After completing the import, the following two collections should be available.



3. Navigate to the **Authorization** tab.

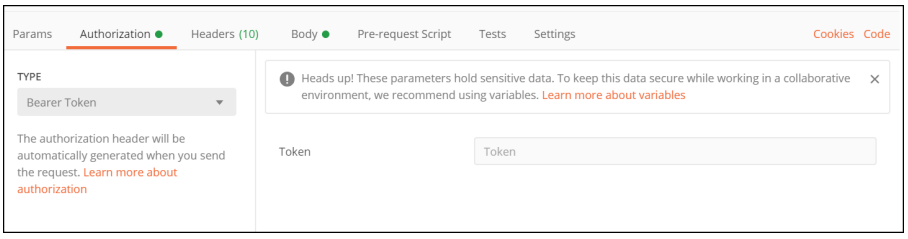


Figure B-6: Authorization Tab

4. Select *Bearer Token* as the type of authorization header from the **Type** drop-down list.

   The **Token** field appears.

5. In the **Token** field, specify the *access_token* that you have generated in *step 3* of the section *Enabling the Authentication Functionality*.
6. Navigate to the **Headers** tab and specify the IP address of the Linux instances, from where you are sending the REST API request, as the value for the **X-Forwarded-For** key.

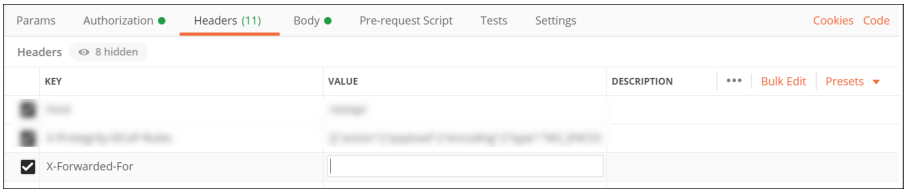


Figure B-7: X-Forwarded-For Key

7. Select **Release 1** in DCoP_Sample and click **Send**.

   User should get response as successful *200 OK* and receive protected Data.

7.2.8 Running Autoscaling Script

The Autoscaling script is used to issue continuous requests on the DSG containers. When the number of REST requests hitting the container are increased, with Horizontal Pod Autoscaling, Kubernetes automatically scales the number of pods based on the CPU utilization observed.

The user can run the autoscaling script from any Linux node, which can connect the external IP for the deployment.

Usage

```
./Sample_App_autoscale.sh <DSG service IP address> <CA certificate path> <Client certificate path> <Client certificate key path> [OAuth token] [IP address of the Linux instance]
```

After running this script, the user can observe that new pods are created to handle to the incoming traffic.

Appendix C

Using the Dockerfile to Build Custom Images

This section explains how to use the Dockerfiles, which are provided as part of the installation package, to build custom images for the Protegrity containers. This enables you to use a base image of your choice, instead of using the default RHEL Universal Base Image, which is used as the default base image for the Protegrity images.

8.1 Creating Custom Images Using Dockerfile

This section describes the steps for creating a custom images for the PEP server, Meta server, DSG, and Sidecar containers, using the Dockerfile provided with the installation package.

➤ To create custom image:

1. Download the installation package.
   For more information about downloading the installation package, refer to the section *Extracting the Package*.
2. Perform the following steps to build a Docker image for the PEP server container.
   a. Run the following command to extract the files from the *PUBLISHER-PEPSERVER_SRC_<version_number>.tgz* file to a directory.
   ```
   tar -xvf PUBLISHER-PEPSERVER_SRC_<version_number>.tgz -C <dir>
   ```
   The following files are extracted:
   - *PTY_PEPSERVER_RHUBI_DOCKERFILE_<version_number>*
   - *PepServerSetup_Linux_x64_<version_number>.sh*
   - *pep-startup.py*
   - *PtyShmCatSetup_Linux_x64_<version_number>.tgz*

   b. Switch to the directory where you have extracted the *PUBLISHER-PEPSERVER_SRC_<version_number>.tgz* package.

c. Edit the *PTY_PEPSERVER_RHUBI_DOCKERFILE_<version_number>* file and in the following code snippet, replace the placeholder *RHEL-MINIMAL-UBI* with the name of the repository from where you want to download the custom RHEL UBI.

d. Run the following command in the directory where you have extracted the contents of the *PUBLISHER-PEPSERVER_SRC_<version_number>.tgz* file.

```
docker build -t <image-name>:<image-tag> -f
PTY_PEPSERVER_RHUBI_DOCKERFILE_<version_number> .
```

For more information the Docker build command, refer to the *Docker* documentation.

For more information about tagging an image, refer to the *AWS* documentation.

e. Run the following command to list the PEP server container image.

```
docker images
```

f. Push the PEP server container image to your preferred Container Repository.

For more information about pushing an image to the repository, refer to the section *Uploading the Images to the Container Repository*.

3. Repeat step 2 for creating custom images for the Meta server, DSG, and Sidecar containers.

Appendix D

Deploying the Helm Charts by Substituting the Environment Variables

The Helm charts contain environment variables as tag values. You can deploy the Helm charts by substituting these environment variables instead of manually updating the Helm chart.

This section describes the steps for deploying the Helm charts by using the *envsubst* command, which enables you to substitute the environment variables.

Before you begin

Before configuring the environment variables, ensure that the prerequisites are met.

For more information about the prerequisites for deploying the Publisher Helm chart, refer to the section *Prerequisites*.

For more information about the prerequisites for deploying the DSG Helm chart, refer to the following sections:

- For AWS - *Verifying the Prerequisites*
- For Azure - *Verifying the Prerequisites*

To deploy Helm charts using the *envsubst* command:

1. Configure the environment variables that have been defined in the Helm charts.

   For example, run the following command to configure the environment variables.

```
export NAMESPACE='dsg'
export PUB_INSTALL_NAME=pty-publisher
export PEP_IMAGE_REPO_URI='<AWS_Account_ID>.dkr.ecr.us-east-1.amazon.com/test/pepserver'
export PEP_IMAGE_REPO_TAG='PUBLISHER-PEP_RHUBI-8-64_x86-64_K8S_<Version>'
export META_IMAGE_REPO_URI='<AWS_Account_ID>.dkr.ecr.us-east-1.amazon.com/test/metaserver'
export META_IMAGE_REPO_TAG='PUBLISHER-META_RHUBI-8-64_x86-64_K8S_<Version>'
export PUBLISHER_SERVER_CERT_SECRET=pty-secret-tls-publisher
export ESA_CREDENTIAL_SECRET=pty-secret-esa-cred
export PBE_SECRET=pty-secret-pbe
export ESA_IP='<ESA_IP_Address>'
export POLICY_FILE_PATH='test/path'
export IMAGE_REGISTRY_CREDS_SECRET=pty-secret-registry-cred
```

   You need to run the *export* command for all the environment variables.

2. Navigate to the directory where you have stored the *values.yaml* file for deploying the Publisher Helm chart.
3. Substitute the environment variables by running the following command.
   ```
   envsubst < values.yaml > values_new.yaml
   ```
4. Deploy the Helm chart using the following command.
   ```
   helm install <Deployment name> -f ./values_new.yaml <Path to Helm chart>
   ```
5. Repeat steps 2 and 3 for deploying the DSG Helm chart.

Appendix E

Deploying the Helm Charts by Using the Set Argument

The Helm charts contain environment variables as tag values. You can deploy the Helm charts by substituting these environment variables at runtime using the *set* argument instead of manually updating the Helm chart.

This section describes the steps for deploying the Helm charts by using the *set* argument, which enables you to substitute the environment variables.

Before you begin

Before configuring the environment variables, ensure that the prerequisites are met.

For more information about the prerequisites for deploying the Publisher Helm chart, refer to the section *Prerequisites*.

For more information about the prerequisites for deploying the DSG Helm chart, refer to the following sections:

- For AWS - *Verifying the Prerequisites*
- For Azure - *Verifying the Prerequisites*

To deploy Helm charts using the *envsubst* command:

1. Configure the environment variables that have been defined in the Helm charts.

   For example, run the following command to configure the environment variables.

   ```
   export NAMESPACE='dsg'
   export PUB_INSTALL_NAME=pty-publisher
   export PEP_IMAGE_REPO_URI='<AWS_Account_ID>.dkr.ecr.us-east-1.amazon.com/test/pepserver'
   export PEP_IMAGE_REPO_TAG='PUBLISHER-PEP_RHUBI-8-64_x86-64_K8S_<Version>'
   export META_IMAGE_REPO_URI='<AWS_Account_ID>.dkr.ecr.us-east-1.amazon.com/test/metaserver'
   export META_IMAGE_REPO_TAG='PUBLISHER-META_RHUBI-8-64_x86-64_K8S_<Version>'
   export PUBLISHER_SERVER_CERT_SECRET=pty-secret-tls-publisher
   export ESA_CREDENTIAL_SECRET=pty-secret-esa-cred
   export PBE_SECRET=pty-secret-pbe
   export ESA_IP='<ESA_IP_Address>'
   export POLICY_FILE_PATH='test/path'
   export IMAGE_REGISTRY_CREDS_SECRET=regcred
   ```

   You need to run the *export* command for all the environment variables.

2. Navigate to the directory where you have stored the *values.yaml* file for deploying the Publisher Helm chart.

3. Deploy the Publisher Helm Chart using the following command.

```
helm install <name for this helm deployment> <Location of the directory that contains the
Helm
chart> -n $WAREHOUSE_NAMESPACE
--set <tag 1>="${Env_Var_1}" \
--set <tag 2>="${Env_Var_2}" \
--set <tag 3>="${Env_Var_3}" \
--set <tag 4>="${Env_Var_4}"
```

Example:

```
helm install ${PUB_INSTALL_NAME} ./pty-publisher -n ${NAMESPACE}
--set pepServerImage.repository="${PEP_IMAGE_REPO_URI}" \
--set pepServerImage.tag="${PEP_IMAGE_REPO_TAG}" \
--set metaServerImage.repository="${META_IMAGE_REPO_URI}" \
--set metaServerImage.tag="${META_IMAGE_REPO_TAG}" \
--set publisherServerCertSecret="${PUBLISHER_SERVER_CERT_SECRET}" \
--set esaCredSecrets="${ESA_CREDENTIAL_SECRET}" \
--set pbeSecrets="${PBE_SECRET}" \
--set pvcName="${PVC_NAME}" \
--set policy.esaIP="${ESA_IP}" \
--set policy.filePath="${POLICY_FILE_PATH}" \
--set imagePullSecrets="${IMAGE_REGISTRY_CREDS_SECRET}" \
```

4. Repeat steps 1 and 2 for deploying the DSG Helm chart.

Appendix F

Publisher Pod Log Files

The following snippet shows the important logs that are generated after you run the *kubectl get logs* command for the Publisher pod.

```
Defaulted container "pty-publisher-pepserver" out of: pty-publisher-pepserver, pty-publisher-
metaserver
2024-10-16 09:36:30,577:pep_installer:INFO:Installing PepServer...
2024-10-16 09:36:31,242:pep_installer:INFO:PepServer installed successfully...
2024-10-16 09:36:31,242:pep_installer:INFO:Metadata file not present on tmpfs
2024-10-16 09:36:31,242:pep_installer:INFO:Check and copy metadata from volume mount
2024-10-16 09:36:31,243:pep_installer:INFO:Starting pepserver...
Protegrity PEP Server
Version 1.2.2+42.g01eb3.1.2
Copyright (c) 2004-2024 Protegrity Corporation. All Rights Reserved.
2024-10-16 09:36:31 (INFO) Application starting, pid=57
2024-10-16 09:36:31 (INFO) Starting Protegrity PEP Server
2024-10-16 09:36:31 (INFO) Version: 1.2.2+42.g01eb3.1.2
2024-10-16 09:36:31 (INFO) Platform: Linux_x64
2024-10-16 09:36:31 (INFO) Hostname: test-eks-p-dsg-310510-v1-pty-publisher-85bcfcbcb-dn6l8
2024-10-16 09:36:31 (INFO) IP Address: 10.49.7.134
2024-10-16 09:36:31 (CONFIG) Policy will be downloaded from https://<ESA_IP_Address>
2024-10-16 09:36:31 (CONFIG) Policy refresh interval: 89 seconds
2024-10-16 09:36:31 (INFO) Key handler loaded: internal
2024-10-16 09:36:32 (CONFIG) Communication id: 0
2024-10-16 09:36:32 (CONFIG) Semaphore resources: 4000
2024-10-16 09:36:32 (INFO) Shared memory GID: -1
2024-10-16 09:36:32 (CONFIG) Is shared memory worldreadable: yes
2024-10-16 09:36:32 (CONFIG) Creating semaphore, id: 781728 with initial count: 4000
2024-10-16 09:36:32 (INFO) Semaphore id : 781728 created with gid 1000 and Permission 0666
2024-10-16 09:36:32 (CONFIG) Creating shared memory, id: 781728, size(bytes): 20000000
2024-10-16 09:36:32 (INFO) Shared mem id: 781728 created/opened with GID 1000 and Permission 0666
2024-10-16 09:36:32 (CONFIG) Creating semaphore, id: 703396 with initial count: 4000
2024-10-16 09:36:32 (INFO) Semaphore id : 703396 created with gid 1000 and Permission 0666
2024-10-16 09:36:32 (CONFIG) Creating shared memory, id: 703396, size(bytes): 2000000
2024-10-16 09:36:32 (INFO) Shared mem id: 703396 created/opened with GID 1000 and Permission 0666
2024-10-16 09:36:32 (CONFIG) Creating semaphore, id: 1028090 with initial count: 4000
2024-10-16 09:36:32 (INFO) Semaphore id : 1028090 created with gid 1000 and Permission 0666
2024-10-16 09:36:32 (CONFIG) Creating shared memory, id: 1028090, size(bytes): 544
2024-10-16 09:36:32 (INFO) Shared mem id: 1028090 created/opened with GID 1000 and Permission
0666
2024-10-16 09:36:32 (CONFIG) Creating semaphore, id: 896416 with initial count: 4000
2024-10-16 09:36:32 (INFO) Semaphore id : 896416 created with gid 1000 and Permission 0666
2024-10-16 09:36:32 (CONFIG) Creating shared memory, id: 896416, size(bytes): 20000000
2024-10-16 09:36:32 (INFO) Shared mem id: 896416 created/opened with GID 1000 and Permission 0666
2024-10-16 09:36:32 (CONFIG) Creating semaphore, id: 789988 with initial count: 4000
2024-10-16 09:36:32 (INFO) Semaphore id : 789988 created with gid 1000 and Permission 0666
2024-10-16 09:36:32 (CONFIG) Creating shared memory, id: 789988, size(bytes): 402653184
2024-10-16 09:36:32 (INFO) Shared mem id: 789988 created/opened with GID 1000 and Permission 0666
2024-10-16 09:36:32 (CONFIG) Publishing policy to shared memory: enabled
2024-10-16 09:36:32 (INFO) Application protector listener is disabled
```

```
2024-10-16 09:36:32 (CONFIG) Administration interface created on port 16700
2024-10-16 09:36:32 (CONFIG) Administration interface is using ESA for authentication
2024-10-16 09:36:32 (INFO) Application started, pid=57
2024-10-16 09:36:32 (INFO) Registering node test-lochan-eks-p-dsg-310510-v1-pty-
publisher-85bcfcbcb-dn6l8(10.49.7.134)
2024-10-16 09:36:32 (INFO) This node has been assigned to data store Test
2024-10-16 09:36:32 (INFO) This node has been registered as node ID 34
2024-10-16 09:36:32 (INFO) Downloading signingkey from https://<ESA_IP_Address>/dps/v1/
deployment/keys/f9b38c89-9df8-4203-b0b4-ce394b694b26
2024-10-16 09:36:32 (ALL) Downloaded signingkey has UID: f9b38c89-9df8-4203-b0b4-ce394b694b26
2024-10-16 09:36:32 (INFO) PepServer Started
2024-10-16 09:36:32 (INFO) Downloading policy files
2024-10-16 09:36:32 (INFO) There is no license.
2024-10-16 09:36:33 (ALL) Successfully downloaded pepserver.lic
2024-10-16 09:36:33 (ALL) Successfully downloaded policy.json
2024-10-16 09:36:33 (ALL) Successfully downloaded new_capabilities.json
2024-10-16 09:36:33 (ALL) Successfully downloaded tokenelements/2228FADFE57845D2BA3C25CF98B19BEB
(TE_ANUM_SLT13_PL data element)
2024-10-16 09:36:33 (ALL) Successfully downloaded tokenelements/C3B0A3754DD558AEA52822E75D84BCBF
(TE_A_S13_L0R0_Y_ASY data element)
2024-10-16 09:36:33 (INFO) Policy download was successful.
2024-10-16 09:36:33 (ALL) Will publish policy
2024-10-16 09:36:33 (WARNING) License will expire in 14 days.
2024-10-16 09:36:33 (INFO) License state: OK
2024-10-16 09:36:33 (CONFIG) case-sensitive = yes (Policy users are treated in case-sensitive
manner)
2024-10-16 09:36:33 (INFO) Policy successfully published
2024-10-16 09:36:33 (INFO) Policy INFO successfully published
2024-10-16 09:36:33 (INFO) Policy Alphabets successfully published
2024-10-16 09:36:33 (ALL) Request packed
2024-10-16 09:36:33 (INFO) Received service request id: 4306
2024-10-16 09:36:33 (INFO) Loaded token element from cache: C3B0A3754DD558AEA52822E75D84BCBF
2024-10-16 09:36:33 (INFO) Loaded token element from cache: 2228FADFE57845D2BA3C25CF98B19BEB
2024-10-16 09:36:33 (INFO) Size of token elements in memory: 3094900 bytes
2024-10-16 09:36:33 (ALL) Response unpacked
2024-10-16 09:36:33 (INFO) Sending status to: dps/v1/deployment/nodes/34/status
2024-10-16 09:36:33 (INFO) Executing post deploy application.
Executing imp_creator script to create policy dumps on Volume Mount\n
2024-10-16 09:36:34,249:imp_creator:INFO:Storage Provider: VOLUME
2024-10-16 09:36:34,249:commons.utils.shell:DEBUG:Executing command: ['rm', '-rf', '/var/data/
staging/.staging']
2024-10-16 09:36:34,252:commons.utils.shell:DEBUG:
2024-10-16 09:36:34,252:commons.utils.shell:DEBUG:Executing command: ['mkdir', '-p', '/var/data/
staging/.staging']
2024-10-16 09:36:34,254:commons.utils.shell:DEBUG:
2024-10-16 09:36:34,255:imp_creator:INFO:Creating policy dumps...
2024-10-16 09:36:34,255:commons.utils.shell:DEBUG:Executing command: ['pgrep', 'pepserver']
2024-10-16 09:36:34,259:commons.utils.shell:DEBUG:57

2024-10-16 09:36:34,259:imp_creator:DEBUG:Checking policy health...
2024-10-16 09:36:34,259:commons.utils.shell:DEBUG:Executing command: ['/opt/protegrity/
healthcheck/bin/shmmon']
2024-10-16 09:36:34,261:commons.utils.shell:DEBUG:Initiating Policy Health Check

Opening Shared Memory for Policy Health Check
Shared Memory Opened for Policy Health Check

Number Of DataElements : 2
Number Of Users : 3

Policy Health Check Done


2024-10-16 09:36:34,261:commons.utils.shell:DEBUG:Executing command: ['/var/data/staging/
defiance_dps/bin/dpsadmin', '-s', 'print(gettokenbookheader())']
2024-10-16 09:36:36 (ALL) Sending 165 bytes
2024-10-16 09:36:36,670:commons.utils.shell:DEBUG:--- Shared memory size (bytes) : 402653184 ---
---   Used (bytes)             : 3094900 ---
---   Available (bytes)        : 399558284 ---


2024-10-16 09:36:36,671:commons.utils.shell:DEBUG:Executing command: ['/opt/protegrity/
shmutilities/bin/ptyshmcat', '3094900']
```

```
2024-10-16 09:36:36,776:commons.utils.shell:DEBUG:
2024-10-16 09:36:36,777:commons.utils.shell:DEBUG:Executing command: ['tar', '--mode=0400',
'-zcf', '/var/data/staging/.staging/policy_2024_10_16_09_36_34.tgz', 'CODEBOOK_ID_KEY',
'ALPHABET_ID_KEY', 'SIGNINGKEY_ID_KEY', 'POLICY_INFO_ID_KEY', 'POLICY_ID_KEY']
2024-10-16 09:36:39,040:commons.utils.shell:DEBUG:
2024-10-16 09:36:39,041:commons.utils.shell:DEBUG:Executing command: ['rm', '-f',
'CODEBOOK_ID_KEY', 'ALPHABET_ID_KEY', 'SIGNINGKEY_ID_KEY', 'POLICY_INFO_ID_KEY', 'POLICY_ID_KEY']
2024-10-16 09:36:39,056:commons.utils.shell:DEBUG:
2024-10-16 09:36:39,057:commons.utils.shell:DEBUG:Executing command: ['tar', '-zcf', '/var/data/
staging/.staging/policy_dump.tgz', 'policy_2024_10_16_09_36_34.tgz']
2024-10-16 09:36:39,341:commons.utils.shell:DEBUG:
2024-10-16 09:36:39,341:commons.utils.shell:DEBUG:Executing command: ['rm', '-f',
'policy_2024_10_16_09_36_34.tgz']
2024-10-16 09:36:39,346:commons.utils.shell:DEBUG:
2024-10-16 09:36:39,346:imp_creator:INFO:Encrypting the dumps...
2024-10-16 09:36:40,063:commons.utils.shell:DEBUG:Executing command: ['rm', '-f', '/var/data/
staging/.staging/policy_dump.tgz']
2024-10-16 09:36:40,067:commons.utils.shell:DEBUG:
2024-10-16 09:36:40,067:imp_creator:INFO:Uploading to storage media...
2024-10-16 09:36:40,067:commons.utils.shell:DEBUG:Executing command: ['cp', '-f', '/var/data/
staging/.staging/final_policy.tgz', '/var/data/staging/.staging/policy_2024_10_16_09_36_40.tgz']
2024-10-16 09:36:40,073:commons.utils.shell:DEBUG:
2024-10-16 09:36:40,076:commons.cloud.volume:DEBUG:Overwrite flag value is: False
2024-10-16 09:36:40,076:commons.utils.shell:DEBUG:Executing command: ['mkdir', '-p', '/var/data/
policy/dsg-310510/policy-v1']
2024-10-16 09:36:40,084:commons.utils.shell:DEBUG:
2024-10-16 09:36:40,084:commons.utils.shell:DEBUG:Executing command: ['/bin/cp', '-f',
'/var/data/staging/.staging/policy_2024_10_16_09_36_40.tgz', '/var/data/policy/dsg-310510/policy-
v1/policy_2024_10_16_09_36_40.tgz']
2024-10-16 09:36:40,157:commons.utils.shell:DEBUG:
2024-10-16 09:36:40,161:commons.cloud.volume:DEBUG:Overwrite flag value is: True
2024-10-16 09:36:40,161:commons.utils.shell:DEBUG:Executing command: ['mkdir', '-p', '/var/data/
policy/dsg-310510/policy-v1']
2024-10-16 09:36:40,165:commons.utils.shell:DEBUG:
2024-10-16 09:36:40,166:commons.utils.shell:DEBUG:Executing command: ['/bin/cp', '-f', '/var/
data/tmpfs/metadata.json', '/var/data/policy/dsg-310510/policy-v1/metadata.json']
2024-10-16 09:36:40,174:commons.utils.shell:DEBUG:
2024-10-16 09:36:40,175:commons.utils.shell:DEBUG:Executing command: ['rm', '-rf', '/var/data/
staging/.staging']
2024-10-16 09:36:40,179:commons.utils.shell:DEBUG:
2024-10-16 09:36:40,179:imp_creator:INFO:Done.
2024-10-16 09:36:40 (INFO) Post deploy application done!
2024-10-16 09:36:40 (INFO) Done with downloading policy files
2024-10-16 09:38:01 (INFO) Downloading policy files
2024-10-16 09:38:01 (INFO) Done with downloading policy files
```

For more information about the kubectl get logs command for the Publisher pod, refer to the step *4* in the section *Deploying the Publisher Helm Chart*.

Appendix G

DSG Container Log Files

The following snippet shows the important logs that are generated after you run the *kubectl get logs* command for the DSG Container.

```
INFO  2024/10/16 09:50:32.629716 copRunner.go:111: fetching cop from storage media, VOLUME
DEBUG 2024/10/16 09:50:32.629840 volume.go:78: using default mount location /var/data/ptypolicy
to read cop package
DEBUG 2024/10/16 09:50:32.642983 volume.go:128: downloaded cop package file size 145176
INFO  2024/10/16 09:50:32.644280 copRunner.go:121: fetched cop package
DEBUG 2024/10/16 09:50:32.671217 extras.go:33: detected 3 user(s) in policy
DEBUG 2024/10/16 09:50:32.671361 extras.go:34: detected 2 dataelement(s) in policy
INFO  2024/10/16 09:50:32.671371 extras.go:35: health check status: healthy
Info : Decryption is done for CoP Export API
INFO 2024/10/16 09:50:32.000951 cop_deploy_util:84: COP Extracted
PCPG:20241016095033263139;1;Warning;SourcelessFileLoader;exec_module;Loading feature
configuration
PCPG:20241016095033263425;1;Information;SourcelessFileLoader;exec_module;first trying /opt/
protegrity/alliance/bin/config
PCPG:20241016095033263607;1;Information;SourcelessFileLoader;exec_module;trying /opt/protegrity/
alliance/config
PCPG:20241016095033263823;1;Warning;SourcelessFileLoader;exec_module;enhanced-http-transaction-
metrics:Detailed information about outbound http connections in the HTTP-GW service are
included in transaction metrics. BY DEFAULT ENABLED. Use disable-enhanced-http-transaction-
metrics to disable.
PCPG:20241016095033263905;1;Warning;SourcelessFileLoader;exec_module;disable-sftp-client-key-
check:This feature will disable the sftp client key check.
PCPG:20241016095033265883;1;Information;Manager;get_logger;Tunnel log level set to Information
PCPG:20241016095033265978;1;Warning;Logger;increase_severity_level;Log level set to Information
PCPG:20241016095033266071;1;Information;Manager;get_logger;Service log level set to Information
PCPG:20241016095033266126;1;Warning;Logger;increase_severity_level;Log level set to Information
PCPG:20241016095033299546;1;Information;Manager;get_logger;RuleSet log level set to Information
PCPG:20241016095033299656;1;Warning;Logger;increase_severity_level;Log level set to Information
PCPG:20241016095033301379;1;Information;Manager;get_logger;Admin log level set to Information
PCPG:20241016095033301461;1;Warning;Logger;increase_severity_level;Log level set to Information
PCPG:20241016095033310462;1;Information;Manager;get_logger;DiskBuffer log level set to
Information
PCPG:20241016095033310578;1;Warning;Logger;increase_severity_level;Log level set to Information
PCPG:20241016095033894986;1;Warning;<NA>;_run_module_as_main;Running in containerized mode
PCPG:20241016095033896214;1;Warning;Gateway;run;Protegrity Cloud Gateway 3.1.0.5.7 starting up...
PCPG:20241016095033896301;1;Warning;Gateway;run;Openssl Version OpenSSL 1.1.1k  FIPS 25 Mar 2021
PCPG:20241016095033896355;1;Warning;Gateway;run;PyCurl Version PycURL/7.45.3 libcurl/7.61.1
OpenSSL/1.1.1k zlib/1.2.11 brotli/1.0.6 libidn2/2.2.0 libpsl/0.20.2 (+libidn2/2.2.0)
libssh/0.9.6/openssl/zlib nghttp2/1.33.0
PCPG:20241016095033896418;1;Warning;Gateway;run;Python Version 3.10.0 (default, Oct  3
2024, 13:40:44) [GCC 8.5.0 20210514 (Red Hat 8.5.0-22)]
PCPG:20241016095033896463;1;Information;Gateway;run;tornado Version 6.4.1
PCPG:20241016095033896526;1;Information;Gateway;run;boto3 Version 1.35.32
PCPG:20241016095033896662;1;Warning;Gateway;load_settings;Loading server configuration /opt/
protegrity/alliance/config/gateway.json
PCPG:20241016095033902826;1;Warning;Settings;load_stats_config;stats {'enabled': False,
```

```
'logDirectory': '/opt/protegrity/alliance/log', 'enableStandardOutput': True}
PCPG:20241016095033904266;1;Information;Settings;__init__;Getting user account info for
"alliance"
PCPG:20241016095033906694;1;Warning;BaseLogger;increase_severity_level;base log level set
to Warning
PCPG:20241016095033906814;1;Warning;BaseLogger;increase_severity_level;Log level set to Warning
PCPG:20241016095033906873;1;Warning;Logger;increase_severity_level;Tunnel log level set to
Warning
PCPG:20241016095033906951;1;Warning;Logger;increase_severity_level;Log level set to Warning
PCPG:20241016095033906999;1;Warning;Logger;increase_severity_level;Service log level set
to Warning
PCPG:20241016095033907061;1;Warning;Logger;increase_severity_level;Log level set to Warning
PCPG:20241016095033907109;1;Warning;Logger;increase_severity_level;RuleSet log level set
to Warning
PCPG:20241016095033907171;1;Warning;Logger;increase_severity_level;Log level set to Warning
PCPG:20241016095033907220;1;Warning;Logger;increase_severity_level;Admin log level set to Warning
PCPG:20241016095033907292;1;Warning;Logger;increase_severity_level;Log level set to Warning
PCPG:20241016095033907353;1;Warning;Logger;increase_severity_level;DiskBuffer log level set
to Warning
PCPG:20241016095033907403;1;Warning;Logger;increase_severity_level;Log level set to Warning
PCPGS:20241016095033907495;1;STATS;Gateway;load_settings;stats enabled: False
PCPG:20241016095033907535;1;Warning;Gateway;set_max_process_memory;setting maximum address
space to 6178907136.0
PCPG:20241016095033926474;1;Warning;HttpTunnel;__init__;HTTP Tunnel <default_80> will start
only if there is an enabled service assigned to this tunnel.
PCPG:20241016095033926633;1;Warning;HttpTunnel;__init__;Threshold delta configured 0.1
PCPG:20241016095033926678;1;Warning;HttpTunnel;__init__;Maximum requests configured -1.000000
PCPG:20241016095033926736;1;Warning;HttpTunnel;__init__;Bytes threshold configured -1.000000
PCPG:20241016095033928396;1;Warning;Gateway;load_tunnels;'Tunnel default_81 disabled.'
PCPG:20241016095033928493;1;Warning;HttpTunnel;__init__;HTTP Tunnel <default_443> will start
only if there is an enabled service assigned to this tunnel.
PCPG:20241016095033928572;1;Warning;HttpTunnel;__init__;Threshold delta configured 0.1
PCPG:20241016095033928618;1;Warning;HttpTunnel;__init__;Maximum requests configured -1.000000
PCPG:20241016095033928658;1;Warning;HttpTunnel;__init__;Bytes threshold configured -1.000000
PCPG:20241016095033952591;1;Warning;Gateway;load_tunnels;'Tunnel default_25 disabled.'
PCPG:20241016095033952723;1;Warning;Gateway;load_tunnels;'Tunnel S3 disabled.'
PCPG:20241016095033952814;1;Warning;Gateway;load_tunnels;'Tunnel NFS disabled.'
PCPG:20241016095033952913;1;Warning;Gateway;load_tunnels;'Tunnel CIFS disabled.'
PCPG:20241016095033993066;1;Warning;HttpService;__init__;Service REST DSG Service maximum
outbound 599 error retries configured to 1
PCPG:20241016095034010126;1;Warning;HttpService;__init__;Service DSG_Service maximum outbound
599 error retries configured to 1
PCPG:20241016095034257120;1;Warning;UserDefinedFunction;__init__;User Defined Extract rule 'User
Defined Extraction Impl' defined in profile 'REST API Examples', service name 'REST API Examples'
PCPG:20241016095034257896;1;Warning;UserDefinedFunction;__init__;User Defined Transform rule
'User Defined Transformation Impl' defined in profile 'REST API Examples',
service name 'REST API Examples'
PCPG:20241016095034265894;1;Warning;UserDefinedFunction;__init__;User Defined Extract rule
'Extract Custom Error (UDF)' defined in profile 'REST API Examples', service
name 'REST API Examples'
PCPG:20241016095034270036;1;Warning;HttpService;__init__;Service REST API Examples maximum
outbound 599 error retries configured to 1
PCPG:20241016095034272584;1;Warning;Gateway;run;Unable to get group id dsggroup
PCPG:20241016095034282831;1;Warning;Gateway;lockdown_child_account;Unable to drop privileges
to pwd.struct_passwd(pw_name='alliance', pw_passwd='x', pw_uid=1000, pw_gid=1000, pw_gecos='',
pw_dir='/home/alliance', pw_shell='/bin/bash')
PCPG:20241016095034282941;1;Warning;Gateway;run;Protegrity Cloud Gateway 3.1.0.5.7 server
is running
```

Appendix H

Subscriber-Sidecar Container Log Files

The following snippet shows the important logs that are generated after you run the *kubectl get logs* command for the Subscriber-Sidecar Container.

```
2024-10-16 09:49:32,607:pty_client:INFO:Starting client...
2024-10-16 09:49:32,608:pty_client:INFO:Sending request to fetch metadata.
2024-10-16 09:49:32,687:pty_client:INFO:Found metadata update. Triggering Policy update.
2024-10-16 09:49:32,695:imp_updater:INFO:Storage Provider: VOLUME
2024-10-16 09:49:32,695:imp_updater:INFO:Downloading policy from Storage media...
2024-10-16 09:49:32,724:imp_updater:INFO:Validating checksum of downloaded file.
2024-10-16 09:49:32,729:imp_updater:INFO:Decrypting the dumps...
/opt/protegrity/shmutilities/scripts/commons/crypto/decryptor.py:39: RuntimeWarning: The default
behavior of tarfile extraction has been changed to disallow common exploits (including
CVE-2007-4559). By default, absolute/parent paths are disallowed and some mode bits are cleared.
See https://access.redhat.com/articles/7004769 for more details.
  tar.extractall()
2024-10-16 09:49:33,169:imp_updater:INFO:Populating the shared memory with policy...
2024-10-16 09:49:34,271:imp_updater:INFO:Done.
2024-10-16 09:50:34,271:pty_client:INFO:Sending request to fetch metadata.
2024-10-16 09:51:34,332:pty_client:INFO:Sending request to fetch metadata.
```