Protegrity Data Security Gateway (DSG) Immutable Policy User Guide for AWS 3.1.0.5

Created on: Nov 19, 2024

Copyright

Table of Contents

Chapter 1

Protegrity Security Operations Cluster using Kubernetes

This section describes the Protegrity security operations cluster based on Kubernetes.

1.1 Business Problem

This section identifies the following problems that a company faces while protecting data:

- Protegrity customers are moving to the cloud. This includes data and workloads in support of transactional applications and analytical systems.
- It is impossible to keep up with the continual change in workloads by provisioning Protegrity products manually.
- Native Cloud capabilities can be used to solve this problem and deliver the agility and scalability required to keep up with the customers' business.
- Kubernetes can be configured with Protegrity data security components that can leverage the auto scaling capabilities of Kubernetes to scale.

1.2 Protegrity Solution

The Protegrity solution leverages cloud native capabilities to deliver a Protegrity data security operations cluster with the following characteristics:

- Cloud standard DSG form factor:
  - The delivery form factor for cloud deployments is a container. Protegrity has developed a DSG Container form factor based on the DSG Appliance form factor that we have been delivering for several years.
  - The DSG Container is a standard image that is familiar and expected in cloud deployments.
  - The DSG Container form factor makes the container a lightweight deployment of the DSG.
- Immutable Deployment:
  - Cloud deployments or containers do not change dynamically – they are immutable. In other words, when there is a change in Policy, Configuration over Programming (CoP), or the actual DSG Container software, the change is carried out by terminating the existing DSG Container with Policy and CoP, and instantiating a new one.
  - Changes to Policy, CoP, or the DSG Container happen through special deployment strategies available through Kubernetes. For Protegrity deployments, the recommended deployment strategy is a *Blue*/*Green* strategy.
- Auto Scalability:
  - Kubernetes can be setup to auto scale based on setting a configuration on CPU thresholds.

- Kubernetes will start with an initial set of pods. These will increase when the CPU threshold is passed, and will be shrinked when the CPU threshold is under the acceptable limits. This is automatic and hence gives the agility and scalability that is desired with cloud solutions. Similarly, the nodes on which the pods are run also auto scale when the node thresholds are reached.

- 3rd Party Integration:
  - The important implication is that the ESA is no longer required to be up and running when the Kubernetes runtime has all the required components and can auto scale, when needed.

1.3 Architecture and Components

This section describes the architecture, the individual components, and the workflow of the Protegrity DSG Containerization solution.

The DSG Container deployment consists of the following two stages:

- Stage 1: Creating immutable policy and CoP and uploading them to a cloud storage.
- Stage 2: Deploying the DSG Container Application on a Kubernetes cluster, which uses the policy and CoP stored in stage 1.

The following figures illustrates the process for creating a CoP and uploading it on AWS.

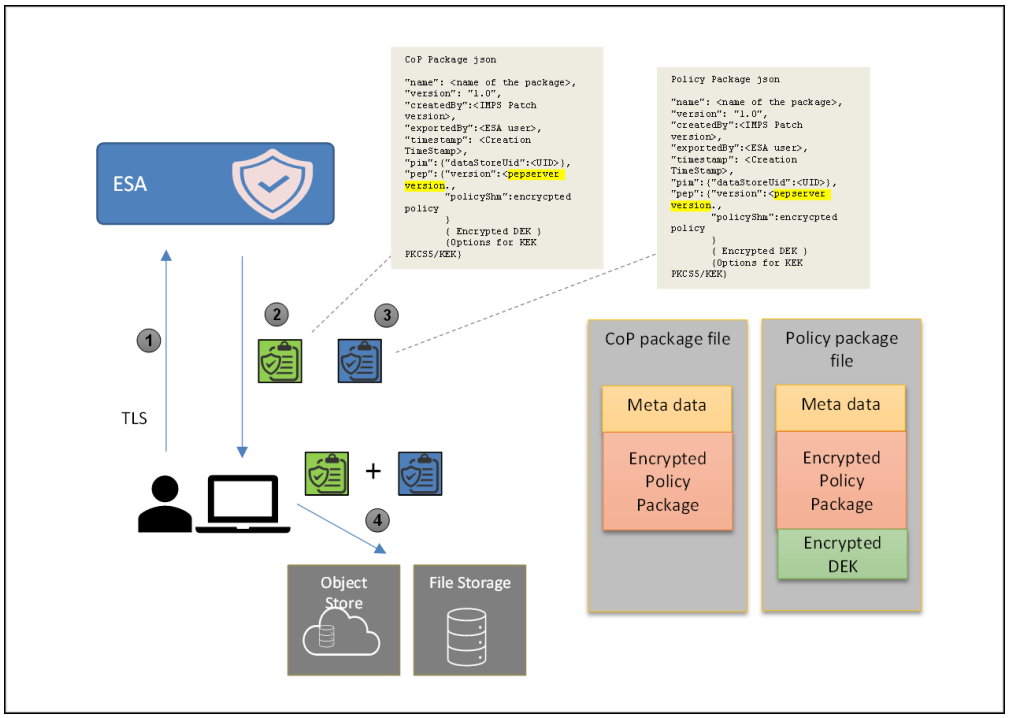


Figure 1-1: Creating CoP and uploading it on AWS

Stage 1

1. The user first creates the CoP and policy 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.

   > **Note:**
   > The CoP package cannot be encrypted using the Cloud Key Management System (KMS).

3. The user then exports the policy using the export API. This policy can be encrypted by one of the following methods:

- Using PBE where a passphrase and salt is used to encrypt the policy package.
- Using Cloud KMS to encrypt the policy package.

4. The user copies the encrypted CoP package and policy to an AWS S3 bucket or a persistent volume.

Stage 2

The following steps describe the workflow of deploying the DSG container on a Kubernetes cluster.

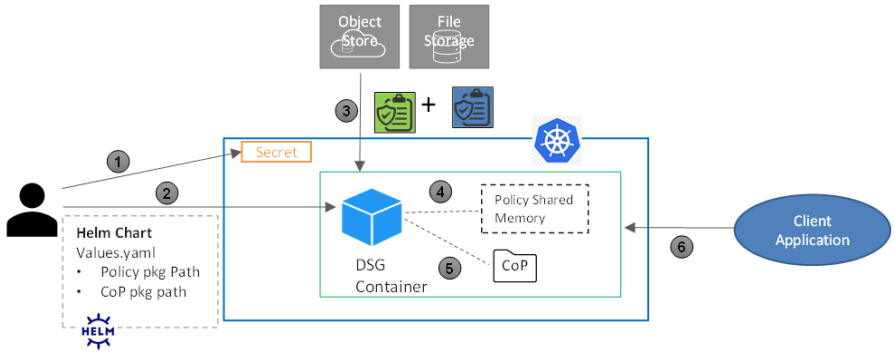


Figure 1-2: Deploying the DSG Container

1. The user creates a Kubernetes secret that is used to decrypt the CoP and the policy packages. This kubernetes secret contains the following entries:
   - Passphrase and salt for the CoP package
   - Passphrase and salt/KMS key ID for the policy package
2. The user runs the Helm chart to deploy the DSG container to the Kubernetes cluster.
3. The DSG container retrieves the CoP and the policy packages from the AWS S3 bucket or persistent volume, and decrypts the package using the secrets provided in the Kubernetes secrets.
4. The DSG container then loads the policy in the shared memory of the Pod.
5. 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 policy and CoP that are available in the Pod.
6. The client Application sends a request to the DSG container for protecting, unprotecting, and reprotecting data.

Chapter 2

Protegrity Reference Deployment Model

This section describes the Protegrity Reference Deployment model that customers can use to deploy the DSG application on a Kubernetes cluster.

2.1 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

2.1.1 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 *DSG_PAP-ALL-64_x86-64_3.1.0.5.6.HF-1* patch to extend the DSG Web UI on the ESA.

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

- *DSGI_RHUBI-ALL-64_x86-64_Generic.K8S_3.1.0.5.x*– Installation package for the Protegrity Reference Deployment. This package contains the files as described in the following table.

Table 2-1: Installation Package

Name	Description
DSG_K8S-ALL-64_x86-64_3.1.0.5.x.tar.gz	The DSG 3.1.0.5 image. This image is used to create the DSG Docker Container.
DSG-HELM_ALL-ALL-ALL_x86-64_AWS.K8S_3.1.0.5.x.tgz	Package containing the Helm charts, which are used to deploy the DSG application on the Kubernetes cluster.
DSG-Samples_Linux-ALL-ALL_x86-64_3.1.0.5.x.tgz	Package containing the sample application for testing the DSG container with sample data.
manifest.json	File specifying the name of the product and its components, the version number of the protector, and the build number of product.

2.1.2 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.

2.1.3 Verifying Prerequisites for AWS

The following additional prerequisites for AWS are required:

Table 2-2: 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

Prerequisite	Description
	• 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.
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

Prerequisite	Description
	• 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 • 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.

2.1.4 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.
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.

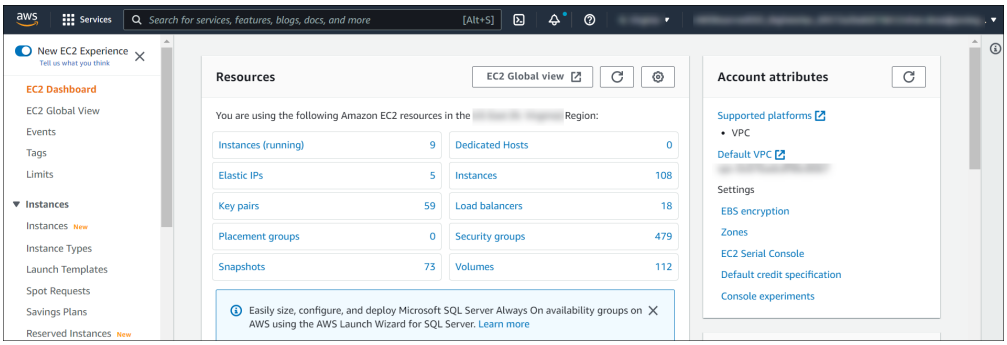


Figure 2-1: EC2 Dashboard Screen

4. On the left pane, click **NETWORK & SECURITY** > **Key Pairs**.

   The **Key pairs** screen appears.

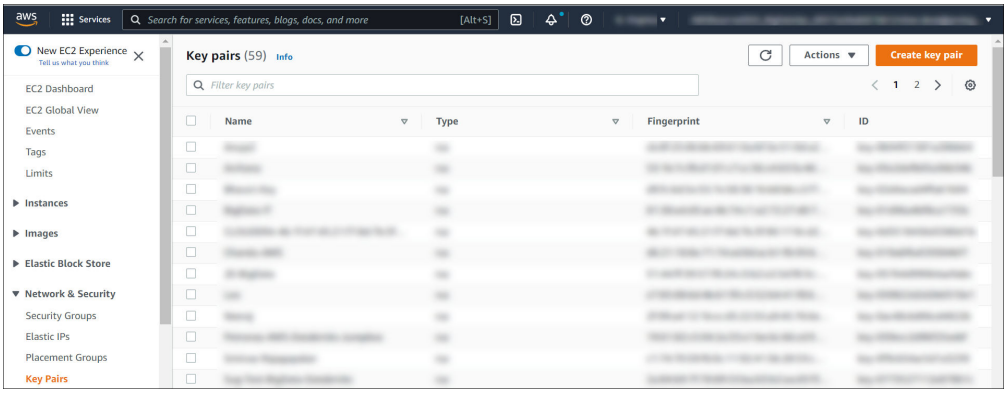


Figure 2-2: Key Pairs Screen

5. Click **Create key pair**.

   The **Create key pair** screen appears.

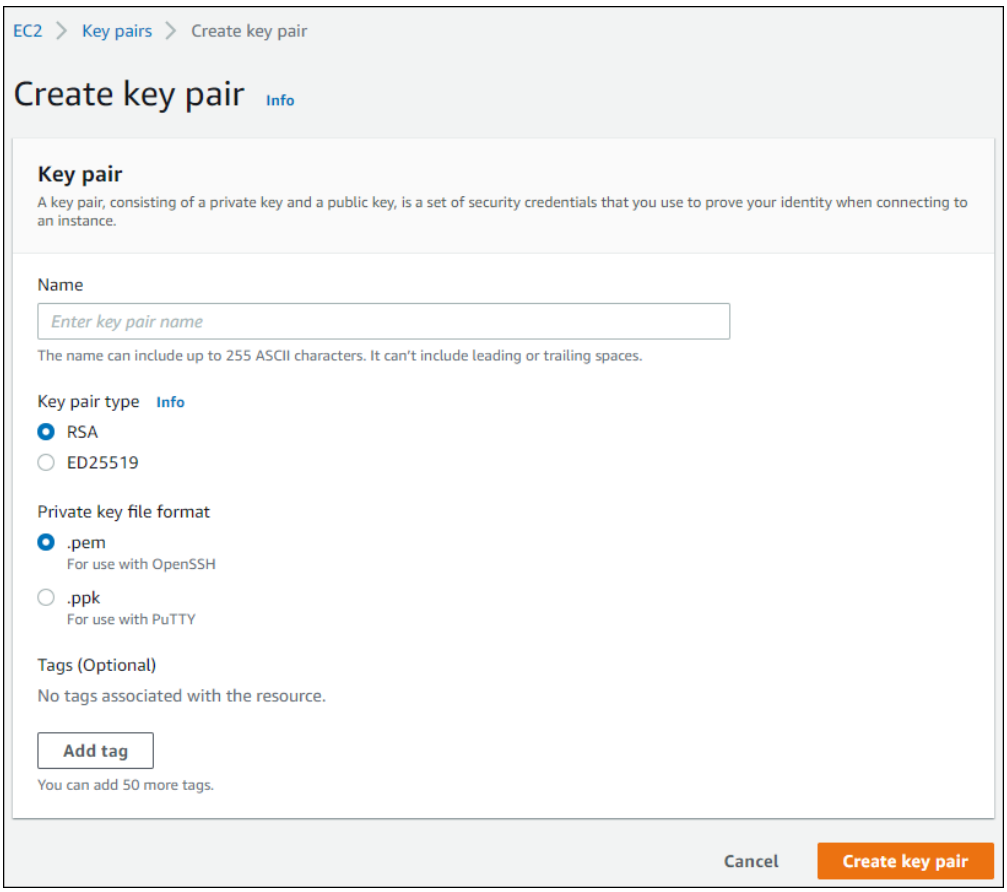


Figure 2-3: Create Key Pair Screen

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 *createCluster.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 **File format** field, specify the format as *pem* for use with OpenSSH.
8. Click **Create key pair**.

   The **Key pairs** screen appears, and the following message appears on top of the screen.

```
Successfully created key pair
```

2.1.5 Creating an AWS Customer Master Key

This section describes how to create the customer master AWS key, if you want to use the key to encrypt the policy package.

➤ To create an AWS customer master key:

1. Log in to the AWS environment.
2. Navigate to **Services.**

   A list of AWS services appears.

3. In **Security, Identity, & Compliance**, click **Key Management Service**.
   The AWS **Key Management Service (KMS)** console opens. By default, the **Customer managed keys** screen appears.

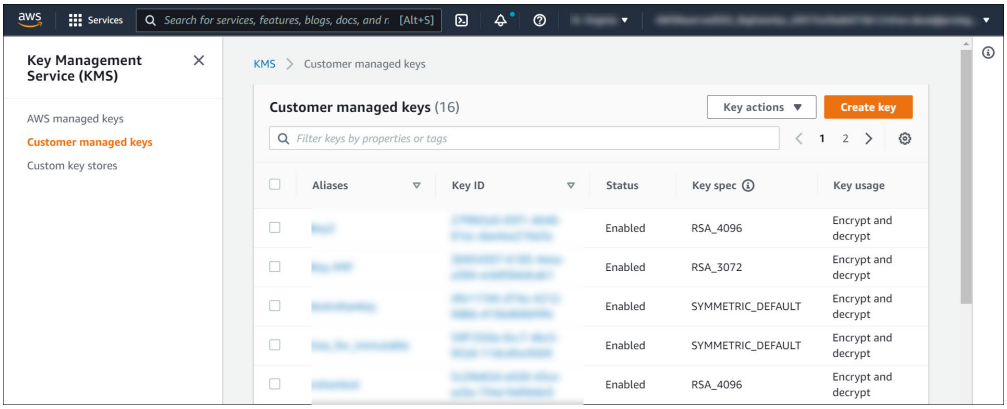


Figure 2-4: Customer Managed Keys Screen

4. Click **Create key**.
   The **Configure key** screen appears.

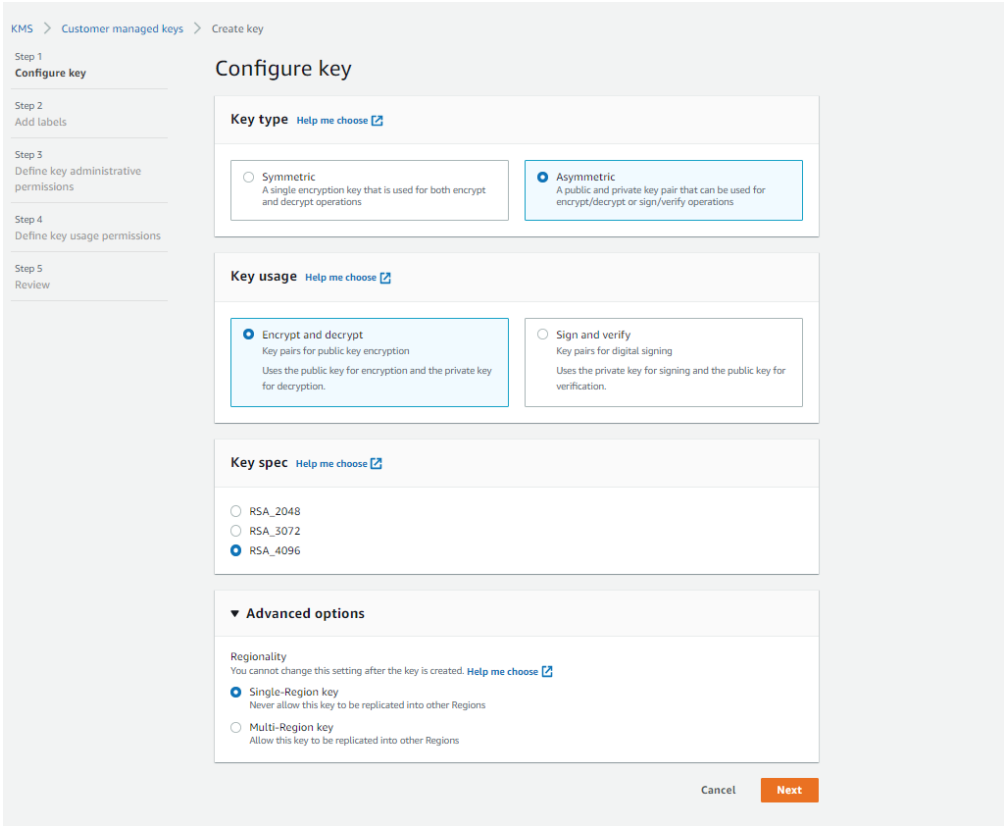
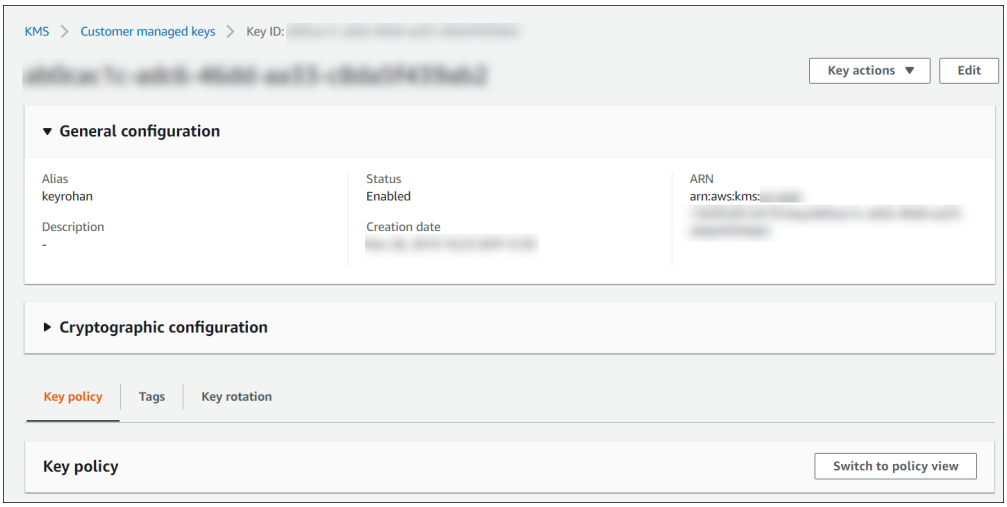


Figure 2-5: Configure Key Screen

5. In the **Key type** section, select the **Asymmetric** option to create a single customer master key that will be used to perform the encrypt and decrypt operations.
6. In the **Key usage** section, select the **Encrypt and decrypt** option.
7. In the **Key spec** section, select one option.
   For example, select **RSA_4096**.
8. In the **Advanced options** section, select the **Single-Region Key** option.
9. Click **Next**.
   The **Add labels** screen appears.

10. In the **Alias** field, specify the display name for the key, and then click **Next**.
    The **Review and edit key policy** screen appears.
11. Click **Finish**.
    The **Customer managed keys screen** appears, displaying the newly created customer master key.
12. Click the key alias.
    A screen specifying the configuration for the selected key appears.



13. 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 *KMSDecryptAccess* policy. You also need to specify this ARN value in the command for creating a Kubernetes secret for the key.

14. Navigate to **Services** > **IAM**.
15. Click **Policies**.
    The **Policies** screen appears.
16. Select the **KMSDecryptAccess** policy.
    The **Permissions** tab appears.
17. Click **Edit policy** to edit the policy in JSON format.
18. 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",
            "Resource": [
                "<ARN of the AWS Customer Master Key>"
            ]
        }
    ]
}
```

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

2.2 Downloading the Installation Package

After verifying the prerequisites, you must download the installation package for the Protegrity Reference Deployment model.

To download the installation package:

1. Download the *DSGI_RHUBI-ALL-64_x86-64_AWS.K8S_3.1.0.5.x.tgz* file on the Linux instance.
2. Run the following command to extract the files from the *DSGI_RHUBI-ALL-64_x86-64_AWS.K8S_3.1.0.5.x.tgz* file.

   ```
   tar -xvf DSGI_RHUBI-ALL-64_x86-64_AWS.K8S_3.1.0.5.x.tgz
   ```

   The following files are extracted:

   - *DSG_K8S-ALL-64_x86-64_3.1.0.5.x.tar.gz*
   - *DSG-HELM_ALL-ALL-ALL_x86-64_AWS.K8S_3.1.0.5.x.tgz*
   - *DSG-Samples_Linux-ALL-ALL_x86-64_3.1.0.5.x.tgz*
   - *DSG_K8S-ALL-64_x86-64_SRC_3.1.0.5.x.tgz*
   - *manifest.json*

3. Run the following command to extract the *.tgz* package that carries the DSG container Helm charts.

   ```
   tar -xvf DSG-HELM_ALL-ALL-ALL_x86-64_AWS.K8S_<version_number>.tgz
   ```

   The extracted package contains the *dsg* directory, which has the following contents:

   - *Chart.yaml* – A YAML file that provides information regarding the chart.
   - *pty-psp.yaml* – The Protegrity Pod Security Policies (PSP) file that is used to manage the security policies for a pod.
   - *pty-rbac.yaml* – The values for configuring the Protegrity Role Based Access Control (RBAC) for the pods.
   - *templates* – A directory that contains the templates required to generate the Kubernetes manifest files. Kubernetes uses the manifest files to deploy the application on the Kubernetes cluster.
   - *values.yaml* – The default values for configuring the Helm chart.

2.3 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 2-3:

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		For more information about installing *eksctl* on

Order	Tools Utilities	Description	Notes	References
		Kubernetes clusters on Amazon Elastic Kubernetes Service (Amazon EKS).		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` 2. `chmod 700 get_helm.sh` 3. `./get_helm.sh`	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.

2.4 Creating Certificates and Keys for TLS Authentication

This section describes the steps required to create certificates and keys for establishing secure communication between the client and the server.

> **Note:**
> If you have a server and client certificate that has been signed by a trusted third-party Certificate Authority (CA), then you do not need create a self-signed server and client certificate.

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'
```

> **Note:**
> If you have a CA certificate and a private key, then move to step 2.

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
```

Ensure that you specify a wildcard character as the subdomain name in the Common Name (CN) of the server certificate. This ensures that the same server certificate is valid for all the subdomains of the given domain name.

For example, consider that you have separate hostnames for the production and staging environments, *prod.example.com* and *staging.example.com*. By specifying a wildcard character in the Common Name of the server certificate, you can use the same server certificate to authenticate *prod.example.com* and *staging.example.com*.

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, *dsg-certs*, and copy all the certificates that have been created to this directory.

2.5 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 set up the Container Registry.

> **Note:**
> The steps listed in this section for uploading the container images to the Amazon 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 images to the Container Repository:

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 Amazon ECR.

   ```
   aws ecr get-login-password --region <Name of ECR region where you want to
   upload the container image> | docker login --username AWS --password-stdin
   <aws_account_id>.dkr.ecr.<Name of ECR region where you want to upload the container
   image>.amazonaws.com
   ```

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

3. Extract the installation package.

   The DSG container image is extracted.

   For more information about extracting the installation package, refer to the section *Downloading the Installation Package*.

4. Run the following command to load the DSG container image on Docker.

   ```
   docker load -i DSG_K8S-<Kubernetes_version>_x86-64_<version>.tgz
   ```

5. Run the following command to list the DSG container image.

   ```
   docker images
   ```

6. 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:

   ```
   docker tag dsg-<version number>:AWS <aws_account_id>.dkr.ecr.region.amazonaws.com/
   <account_name>/dsg:AWS
   ```

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

7. 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.region.amazonaws.com/<account_name>/dsg:AWS
   ```

8. Navigate to the directory where you have extracted the Helm charts package.
9. In the *values.yaml* file, update the required path for the *dsgImage* settings, with their tags.

2.6 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.

2.6.1 Creating and Exporting the Ruleset

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

To create a Ruleset:

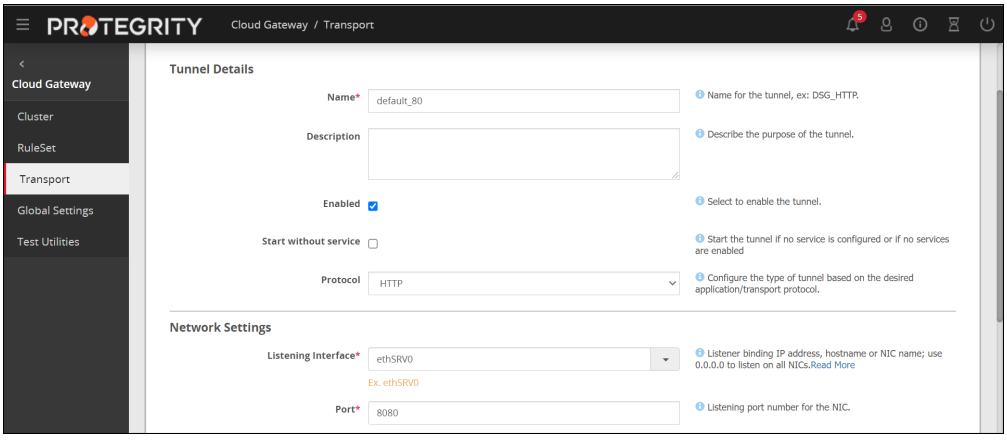
1. On the ESA Web UI, create a Configuration Over Programming (CoP) Ruleset.
   By default, the ESA provides you with default ruleset *REST API Examples* for testing the DSG functionality.

   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*.

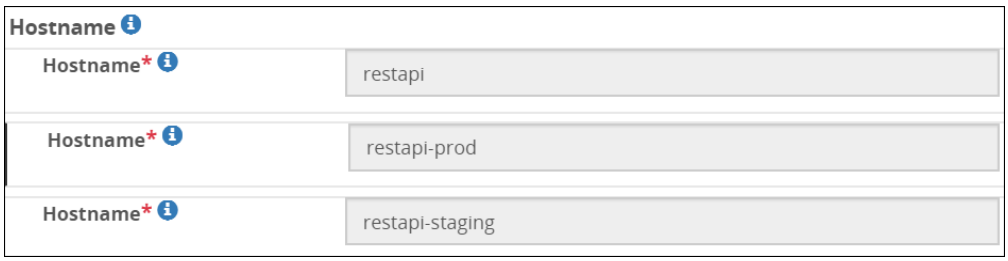
3. If you want to use the transaction metrics, then in the **Advanced Settings** section, specify the following value to support the X-Forwarded-For (XFF) header in the REST API requests sent to the DSG instances.

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

Figure 2-6: 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.

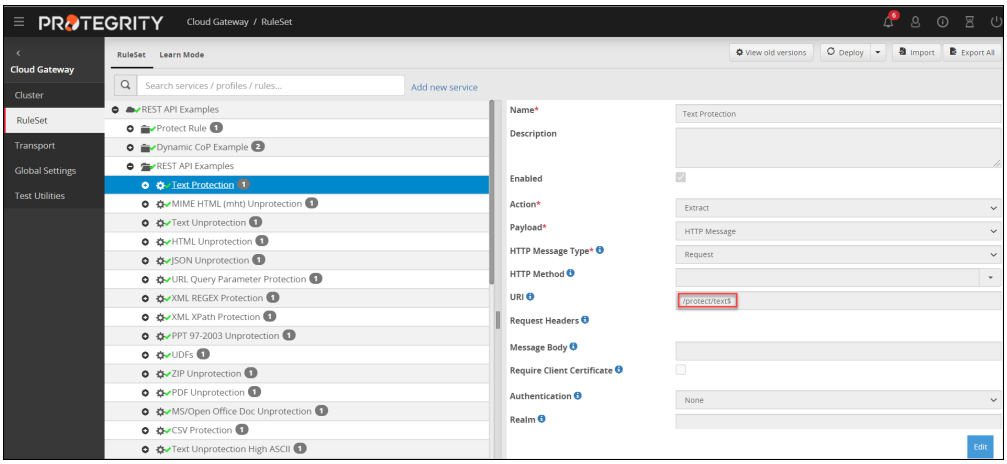
For more information about the *values.yaml* file parameters, refer to the section *Values.yaml*.

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/
```

2.7 Creating and Uploading the Policy to AWS Storage

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

2.7.1 Creating and Exporting the Policy

This section describes how you can create and export the policy on the ESA.

➤ To create a Ruleset:

1. On the ESA Web UI, create a policy.

For more information about creating a policy, refer to the section *Creating and Deploying Policies* in the *Policy Management Guide 9.1.0.0*.

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

   CURL request syntax for exporting policy using PKCS5

```
curl -v -k -u "<username>:<ESA password>"
https://<ESA_IP_address>/pty/v1/imp/export
-H "Content-Type: application/json"
--data '{"name": "<name>",
"pepVersion": "<pepVersion>",
"dataStoreName": "<dataStoreName>",
"emptyString":"NULL",
"caseSensitive":"YES","kek":{"pkcs5":
{"password":"<password>",
"salt":"<salt>"}}}' -o <Response file name in .json format>
```

   Note: Ensure that the **Export IMP** role is assigned to username. This role can be assigned from the **Roles** screen on the ESA Web UI.

   Sample CURL request for exporting policy using PKCS5

```
curl -v -k -u "export_user:admin1234"
https://xxx.xxx.xxx.xxx/pty/v1/imp/export
-H "Content-Type: application/json"
--data '{"name":"Policy1.json",
"pepVersion":"1.2.2+42",
"dataStoreName":"DS1",
"emptyString":"NULL",
"caseSensitive":"YES","kek":{"pkcs5":
{"password":"xxxxxxxxx",
"salt":"salt"}}}' -o Policy1.json
```

   Note: The response for the CURL request will be stored in the *Policy1.json* file.

   Note: For more information about the API to export the immutable policy, refer to the section *Appendix B: APIs for Immutable Protectors* in the *Protegrity APIs, UDFs, Commands Reference Guide 9.1.0.0*.

3. To export the policy package from the ESA and encrypt the policy package using a KMS, run the following command.

   CURL request syntax for exporting policy using KMS

```
curl -v -k -u "<username>:<password>"
https://<ESA IP>/pty/v1/imp/export
-H "Content-Type: application/json"
-o <Policy_package_name>.tgz
--data '{"name":"<name>",
"pepVersion":"<pepVersion>",
"dataStoreName":"<value>",
"emptyString":"<value>",
"caseSensitive":"<value>","kek":{"publicKey":
{"value":"Public Key of the
AWS Customer Key used to encrypt the policy"}}}'
-o Policy1.json
```

   Note:

> Ensure that the **Export IMP** role is assigned to username. This role can be assigned from the **Roles** screen on the ESA Web UI.

> **Note:**
> You can obtain the public key from the AWS Web UI. Navigate to **AWS** > **KMS** > **Customer Managed Keys**. Select the key created in *Creating an AWS Customer Master Key*. Enter the public key that appears in the **Public Key** box.

Sample CURL request for exporting policy using KMS

```
curl -v -k -u "IMPUser:<password>"
https://xxx.xxx.xxx.xxx/pty/v1/imp/export
-H "Content-Type: application/json"
-o policy_package_key.tgz
--data '{"name":"policypackage",
"pepVersion":"1.2.0+16",
"dataStoreName":"<value>",
"emptyString":"<value>","
caseSensitive":"<value>","kek":{"publicKey": {"value":"-----BEGIN PUBLIC KEY-----
\nMIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEAjwxqqrrNQ1kV84GEfYLR\nVK/SyOkGTs7kxE
ImEYMGtugSal2G9m7hFdVlWhHH/OvLDHwdOYe6GLXHVwycfbM/\nS6pzQPS5ujzyJEaLWYAfcRgAHi9g
8GvazDiv34Z+VO5FRbJzP9u9/23J4hExc+e1\nTezKsoBj6Ypx/zBkE4dGqdamfHJUF3ptB82nhJT/Pt
h15ejL/SVrD/q8sORU838O\negcIfQYmd5ziZJgbWLVzFaM9QZXH9hD/0JxAhqfHP3Fnhmc5MDVdg7D0
SQkufIjf\nu7djMy2I3ZRMgnkGxjq8PuBUFaH2s6DvAa4e6K0ppGjFoByVVe2tumrUVEvJg2EM\ndD/P
l15kzLelbDKjxrI8T1D8cWuGh43wf1xC+uRZsfoMSgwDWdpBmaOFP+q2k0d4\nrQWKIgZw71mxYADPKB
AUUwMgD/aREuvTDlpvpl0Vk2Y5i/4Xx/fK7GV5DCdy9TFM\nZgzAA/O9z5tuVzqcgodMu06+iqtXdTUo
OafA+BTvSIXLAgMBAAE=\n-----END PUBLIC KEY-----","algorithm":"RSA_OAEP_SHA256"}}}'
-o Policy1.json
```

> **Note:** For more information about the API to export the immutable policy, refer to the section *Appendix B: APIs for Immutable Protectors* in the *Protegrity APIs, UDFs, Commands Reference Guide 9.1.0.0*.

4. Upload the Policy package to an AWS S3 bucket by performing the following steps.
   a. Login to the AWS environment.
   b. Create an AWS S3 bucket.
   c. Upload the Policy package to an AWS S3 bucket.
5. Upload the Policy package to an AWS EFS by performing 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 Policy package from your local machine to the mount directory on the Linux instance.

```
cp <local_folder>/Policy1.json /test/xyz/
```

2.8 Creating an AWS

This section describes how to create an AWS EFS.

> **Important:**
> Instead of AWS S3, you can store the policy snapshot on the AWS EFS.

➤ To create an AWS EFS:

1. Login 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 **Create file system** screen appears.
5. In the **VPC** list, select the VPC where you will be creating the Kubernetes cluster.
6. Click **Create**.

The file system is created.

Note the value in the **File System ID** column.

2.9 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 ingress 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 edit the *Sample_App_EKSCreateCluster.yaml* file, which is provided as part of the Samples package, to specify the configurations for creating the Kubernetes cluster as per your requirements.

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

```
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: #In this section specify the subnet region and subnet id accordingly
    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 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
```

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

In the *ssh/publicKeyName* parameter, you must specify the name of the key pair that you have created in *step 6* of the section *Creating a Key Pair for the EC2 Instance*.

> **Important:**
> Ensure that you specify the name of the key pair without the extension.

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 *eksctl*documentation.

For more information about creating a configuration file to create a Kubernetes cluster, refer to the *eksctl* documentation.

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

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

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.

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 ensure that the Metrics 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>
```

Repeat this step for all the cluster subnets.

2.10 Setting up Splunk to Access Logs

This section describes how to setup a third-party tool, such as Splunk, to access the DSG and Container logs.

> **Important:**
> Ensure that you have a valid license for using Splunk.

2.10.1 Understanding the Logging Architecture

This section describes the sample architecture that is used to access the logs that are generated on the Kubernetes cluster.

The following figure represents a sample workflow that is used for accessing the DSG and Container logs. In this workflow, a third-party tool, such as Splunk, is used to view the generated logs.

For more information about Splunk, refer to the *Splunk documentation* website.

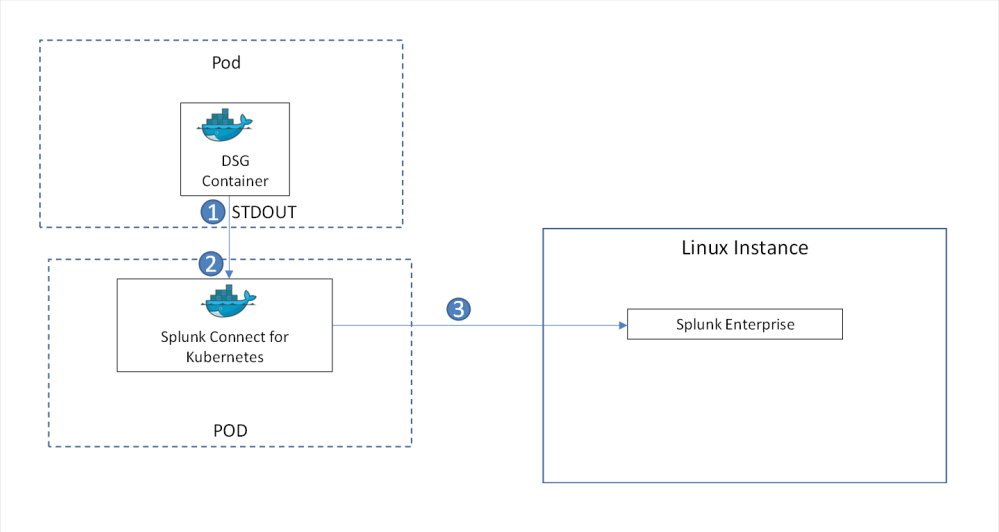


Figure 2-7: Sample Logging Workflow

The following steps describe the workflow of accessing the DSG and Container logs.

1. The DSG container write the logs to the Standard Output (STDOUT) stream.
2. The Splunk Connect for Kubernetes is used to collect the logs from the STDOUT stream.

   For more information about Splunk Connect for Kubernetes, refer to the *Splunk Connect for Kuberntes* page on Github.

3. The Splunk Connect for Kubernetes then forwards the logs to the Splunk Enterprise, which is used to view the logs.

2.10.2 Setting Up Splunk

This section describes how you can set up Splunk for accessing the DSG and Container logs.

To set up Splunk:

1. Create a Linux instance, either on-premise or on a Cloud environment.
2. Install the latest version of Splunk Enterprise on the Linux instance.

   By default, Splunk is installed in the */opt/splunk* directory.

   For more information about installing Splunk Enterprise on a Linux instance, refer to the section *Install Splunk Enterprise* in the Splunk documentation.

3. Navigate to the */opt/splunk/bin* directory and start Splunk by using the following command.

   ```
   ./splunk start --accept license
   ```

   You are prompted to create a username that will be used to login to the Splunk Enterprise.

   For more information about starting Splunk Enterprise on Linux, refer to the section *Start Splunk Enterprise on Linux* in the Splunk documentation.

4. Type the username for the administrator account for logging into Splunk Enterprise, and then press **Enter**.

You are prompted to create a password for the created user.

> **Note:** If you press **Enter** without specifying any username, then *admin* is used as the default username.

5. Type a password for the user that you have created in *step 4*, and then press **Enter**.
   The following message appears on the Console.

```
Waiting for web server at http://127.0.0.1:8000 to be available..... Done


If you get stuck, we're here to help.
Look for answers here: http://docs.splunk.com

The Splunk web interface is at http://[redacted]:8000
```

   By default, you can access the web interface of Splunk Enterprise from the port number *8000* of your Linux instance.

   For example, if the IP address of your Linux instance is *xxx.xxx.xxx.xxx*, then you can access Splunk Web at *http://xxx.xxx.xxx.xxx:8000*.

2.10.3 Configuring Splunk

This section describes how you can configure Splunk Enterprise for accessing the DSG and Container logs.

To configure Splunk Enterprise:

1. Login to the Splunk Web UI at *http://<IP of Linux instance>:8000*.
2. On the Splunk Web UI, type the username and password that you have created while installing Splunk Enterprise on the Linux instance.
   The Splunk Enterprise screen appears.

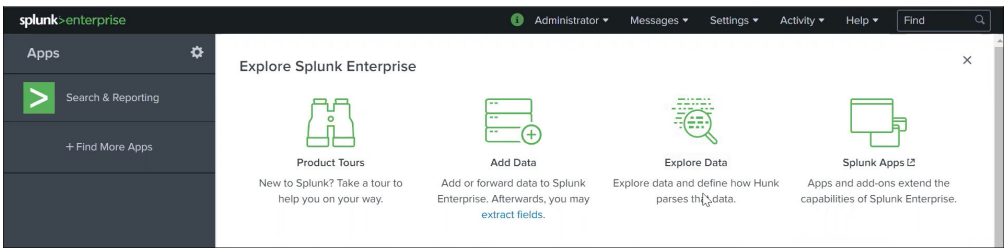


Figure 2-8: Splunk Enterprise Web UI

   For more information about the user credentials, refer to the section *Setting Up Splunk*.

3. Create an index, which is a repository for storing the Splunk Enterprise data, by performing the following steps.
   a. Navigate to **Settings** > **Index**.
      The **Indexes** screen appears.

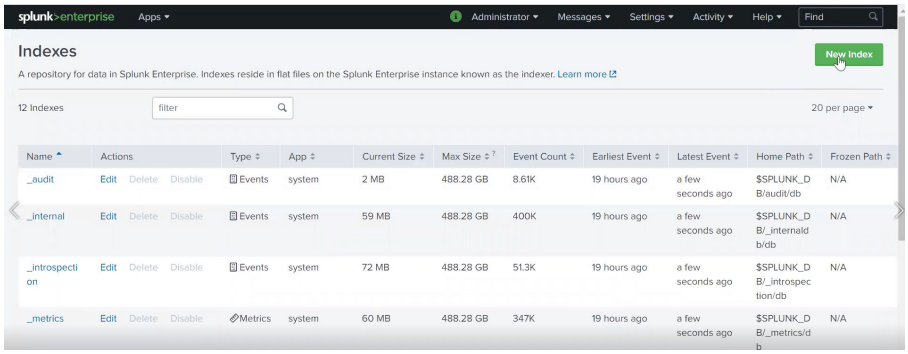


Figure 2-9: Indexes Screen

b. Click **New Index**.

The **New Index** dialog box appears.

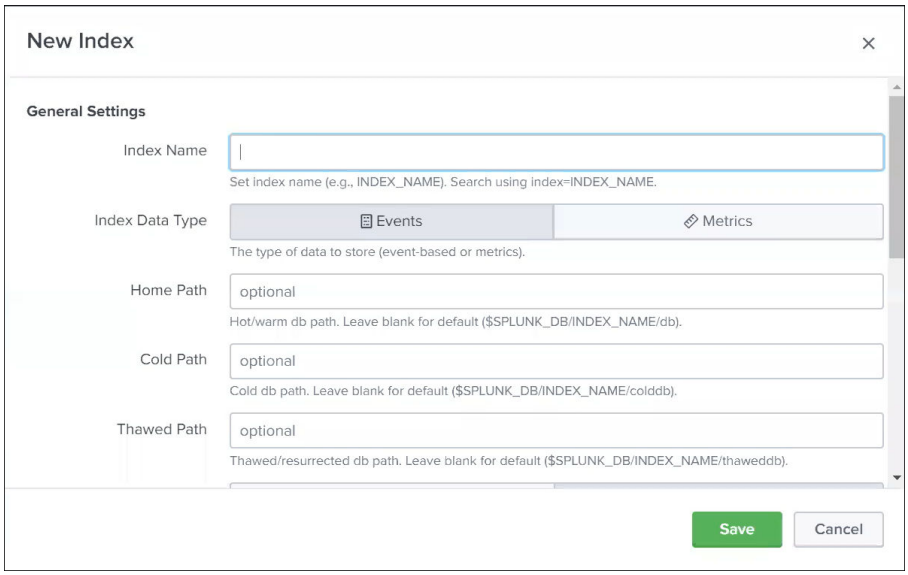


Figure 2-10: New Index Dialog Box

c. In the **General Settings** > **Index Name** field, type a name for the index that you want to create.

d. Click **Save**.

By default, an index with an index data type of *events* is created. This events index is used to store the DSG and Container logs.

For more information about indexes used in Splunk Enterprise, refer to the section *About managing indexes* in the Splunk documentation.

For more information about creating an index in Splunk Enterprise, refer to the section *Create custom indexes* in the Splunk documentation.

4. Perform the following steps to set up HTTP Event Collector (HEC) on the Splunk Web UI. The HEC enables you to receive the DSG and Container logs sent from Kubernetes.

For more information about HEC, refer to the section *Set up and use HTTP Event Collector in Splunk Web* in the Splunk documentation.

a. Navigate to **Settings** > **Data inputs**.

The **Data inputs** screen appears.

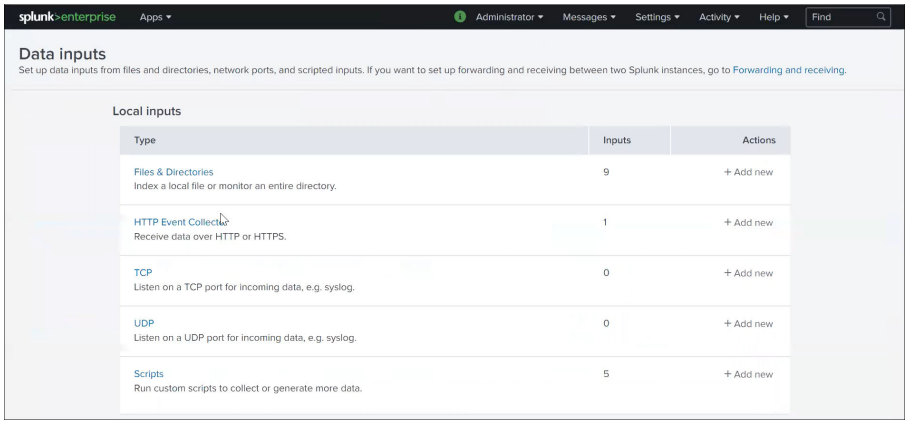


Figure 2-11: Data inputs Screen

b. Click **HTTP Event Collector**.

The **HTTP Event Collector** screen appears.

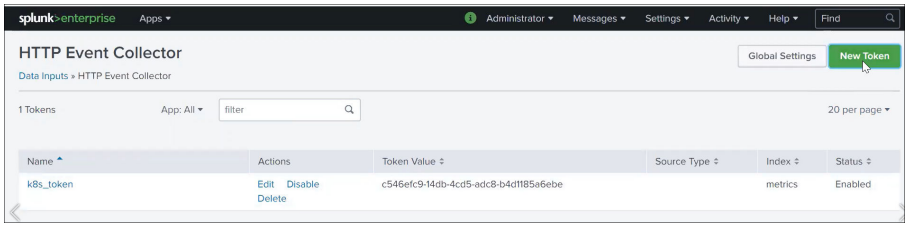


Figure 2-12: HTTP Event Collector Screen

c. Click **New Token**.

The **Add Data** > **Select Source** screen appears.

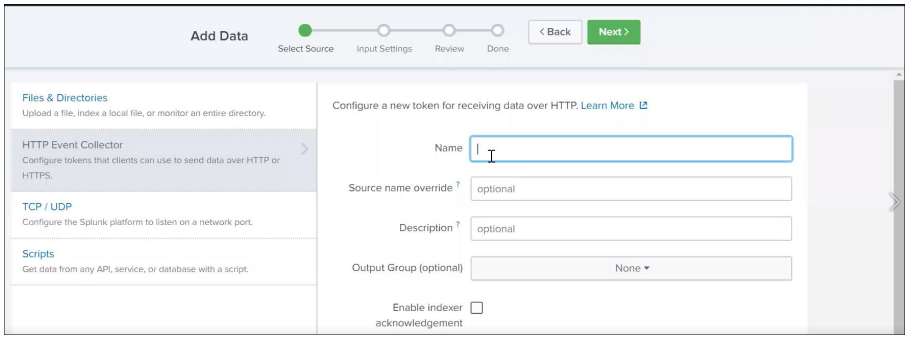


Figure 2-13: Select Source Screen

d. In the **Name** field, type a name for the token.

You need to create a token for receiving data over HTTPS.

For more information about HEC tokens, refer to the section *About Event Collector* tokens in the Splunk documentation.

e. Click **Next**.

The **Input Settings** screen appears.

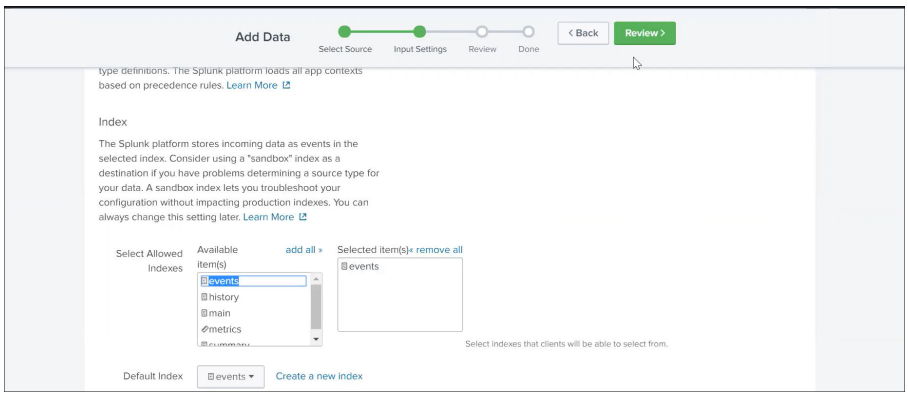


Figure 2-14: Input Settings Screen

f. In the **Available item(s)** list in the **Index** > **Select Allowed Indexes** section, select the index that you have created in *step 3*.

g. Double-click the index so that it appears in the **Selected item(s)** list.

h. Click **Review**.

The **Review** screen appears.

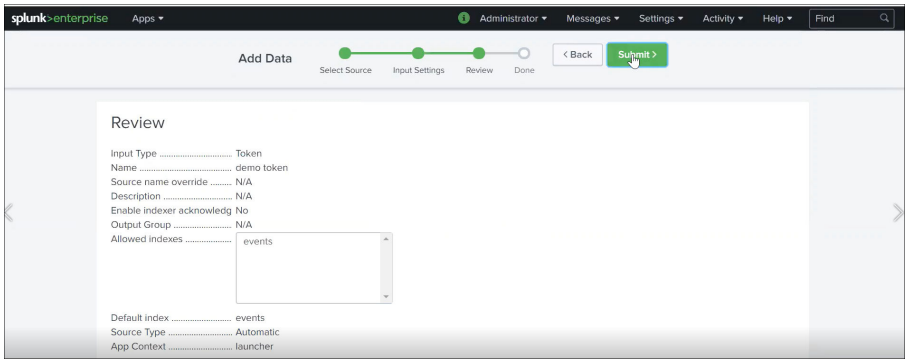


Figure 2-15: Review Screen

i. Click **Submit.**

The **Done** screen appears. The **Token Value** field displays the HEC token that you have created. You must specify this token value in the *values.yaml* file that you will use to deploy Splunk Connect for Kubernetes.

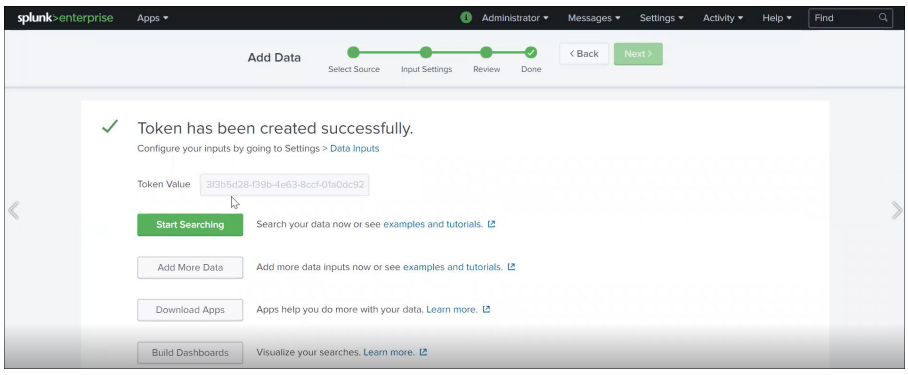


Figure 2-16: Done Screen

5. Configure the global settings for the HEC by performing the following steps.

a. Navigate to **Settings** > **Data inputs**.

The **Data inputs** screen appears.

b. Click **HTTP Event Collector**.

The **HTTP Event Collector** screen appears.

c. Click **Global Settings**.

The **Edit Global Settings** dialog box appears.

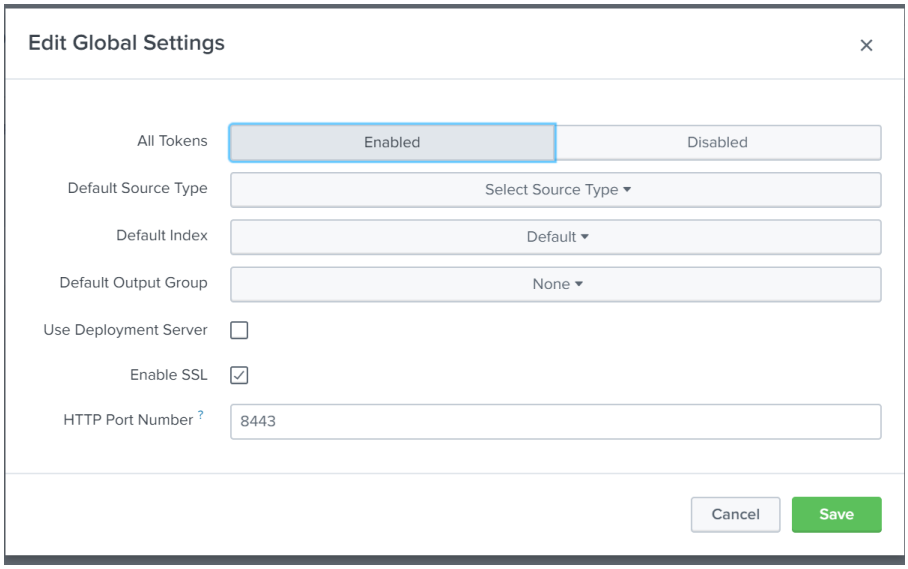


Figure 2-17: Edit Global Settings Dialog Box

d. Ensure the **Enable SSL** check box is selected for SSL communication between the Splunk Enterprise and the Splunk Connect for Kubernetes.

e. In the **HTTP Port Number** field, type a port number that will be used for the communication between the Splunk Enterprise and the Splunk Connect for Kubernetes.

> **Important:**
> Ensure that the pods in the Kubernetes cluster can communicate with the HEC on the configured port.

2.10.4 Deploying the Splunk Connect for Kubernetes Helm Chart on the Kubernetes Cluster

This section describes how you can deploy the Splunk Connect for Kubernetes Helm chart on the Kubernetes cluster where you have deployed the DSG container.

➤ To deploy the Splunk Connect for Kubernetes Helm chart:

1. Create a custom *custom-values.yaml* file for deploying the Splunk Connect for Kubernetes.

   The following snippet shows a sample *custom-values.yaml* file.

```
podSecurityPolicy:
  create: true
global:
  splunk:
    hec:
      protocol: https
      insecureSSL: true
      token: <Token value>
      host: 10.0.0.100
      port: 8443
      indexName: <Index name>
```

```
containers:
  logFormatType: cri
  logFormat: "%Y-%m-%dT%H:%M:%S.%N%:z"
```

Note:

If you use the *containerd* runtime, retain the **`containers:`** annotation. If you use the *dockerd* runtime, remove the **`containers:`** annotation.

Important:

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

2. Modify the default values in the *custom-values.yaml* file as required.

Parameter	Description
podSecurityPolicy/create	Specify whether you want to create the Pod Security Policy resources. You must set the value of this parameter to *true* if the PodSecurityPolicy has been enabled in the Kubernetes cluster. By default, the value of this parameter is set to *true*.
global/splunk/hec/protocol	Specify the protocol that is used to communicate between the Splunk Connect for Kubernetes and the Splunk Enterprise installed on the Linux instance. By default, the value of this parameter is set to *https*.
global/splunk/hec/insecureSSL	Specify whether an insecure SSL connection over HTTPS should be allowed between the Splunk Connect for Kubernetes and the Splunk Enterprise installed on the Linux instance. Specify the value of this parameter to *true*.
global/splunk/hec/token	Specify the value of the token that you have created in *step 4i* of the section *Configuring Splunk*.
global/splunk/hec/host	Specify the IP address of the Linux instance where you have installed Splunk Enterprise.
global/splunk/hec/port	Specify the port number that you have used to configure the HTTP Event Collector of the Splunk Enterprise in *step 5e* of the section *Configuring Splunk*.
global/splunk/hec/indexName	Specify the name of the index that you have created in *step 3* of the section *Configuring Splunk*.

For more information about the complete list of parameters that you can specify in the *custom-values.yaml* file, refer to the default *values.yaml* file in the Helm chart for Splunk Connect for Kubernetes.

3. If you want to forward only the Protegrity logs to the Splunk Enterprise, and do not want to forward the other logs, such as, the Kubernetes logs or the Container logs, then modify the *custom-values.yaml* file as follows.

```
global:
  splunk:
    hec:
      protocol: https
      insecureSSL: true
      token: <Token value>
      host: <Splunk server IP>
      port: 8443
```

```
      indexName: <Index name>
fluentd:
  path: /var/log/containers/*<Namespace where the DSG Container has been deployed>*.log
```

All the container logs are stored in the */var/log/containers* path. You can filter out the Protegrity namespace logs based on the namespace where the DSG Container has been deployed.

In the following example, the DSG Container has been deployed on the *protegrity-dsg* namespace. You can then filter out the Protegrity logs by specifying the value of the *path* parameter as shown in the following code snippet. This ensures that only the Protegrity logs are forwarded to the Splunk Enterprise.

```
global:
  splunk:
    hec:
      protocol: https
      insecureSSL: true
      token: <Token value>
      host: <Splunk server IP>
      port: 8443
      indexName: <Index name>
fluentd:
  path: /var/log/containers/*protegrity-dsg*.log
```

Note: If you want to use another log collector, such as, Filebeat, alongside Fluentd, and you want to forward all the logs, except the Protegrity logs, to an Elasticsearch server instead of Splunk Enterprise, then you need to create another *custom-values2.yaml* file for configuring Filebeat, as shown in the following snippet.

```
filebeatConfig:
  filebeat.yml: |
    filebeat.inputs:
    - type: container
      paths:
        - /var/log/containers/*.log
      exclude_files: ['.protegrity-dsg.']
```

The highlighted line in the snippet is used to exclude the Protegrity logs and forward the rest of the logs to the Elasticsearch server.

You must run the following command to deploy Filebeat Helm chart on the Kubernetes cluster.

```
helm repo add elastic https://helm.elastic.co
```

```
helm install filebeat -f custom-values2.yaml elastic/filebeat
```

4. Run the following command to deploy the Splunk Connect for Kubernetes Helm chart on the Kubernetes cluster.

```
helm install kube-splunk -f custom-values.yaml https://github.com/splunk/splunk-connect-
for-kubernetes/releases/download/1.5.0/splunk-kubernetes-logging-1.5.0.tgz --namespace
kube-system
```

5. Run the following command to list all the pods that are deployed.

```
kubetctl get pods -n kube-system
```

The *kube-splunk-splunk-kubernetes-logging-XXXXX* pod is listed on the console.

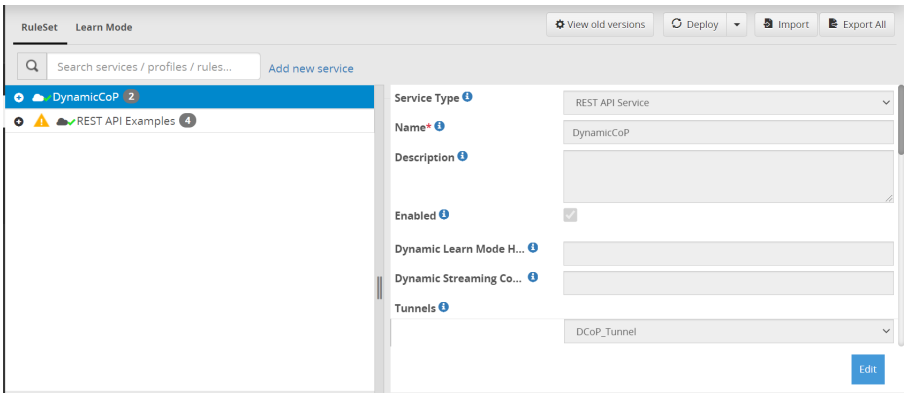
```
$ kubectl get pods
NAME                                              READY   STATUS    RESTARTS   AGE
kube-splunk-splunk-kubernetes-logging-            1/1     Running   0          15h
```

2.10.5 Enabling Transaction Metric Logs for Splunk

This section describes how you can enable the transaction metric logs so that the DSG can write these logs to the *STDOUT* stream. The Splunk for Kubernetes is then used to collect these logs and then send them to the Splunk Enterprise.

To enable the transaction metric logs:

1. Login to the ESA as *admin*.
2. Navigate to the **Cloud Gateway** > **RuleSet** tab.
   By default, the REST API Examples ruleset is selected.



3. Click **Edit** to edit the **DynamicCop**.
4. Under **Transaction Metrics Logging**, select the **Enabled** check box.
5. Click **Save** to save the changes.

2.11 Deploying the Containers

This section describes the different methods that you can use to deploy the DSG container. You can deploy the container as described in the following table.

Table 2-4: Deploying the DSG container

Methods	Steps
With Pod Security Policy (PSP) and Role-Based Access Control (RBAC)	1. *Apply PSP* 2. *Apply RBAC* 3. *Set up the Environment for Deploying the DSG Container* 4. *Deploy a Release on the Kubernetes Cluster with PSP and RBAC*
Without PSP and RBAC	*Deploy a Release on the Kubernetes Cluster without PSP and RBAC*

2.11.1 Applying a Pod Security Policy (PSP)

This section describes how to apply a PSP.

Important: The PSP determines the security policies for running a pod. You require the *cluster-admin* role to perform this task. The user with the cluster-admin role creates the RBAC and applies the PSP. This user also creates two Kubernetes service accounts that will install the Helm charts for deploying the DSG containers.

To apply a PSP:

1. On the Linux instance, run the following command to create the *namespace* required for Helm deployment.

   ```
   kubectl create namespace <Namespace name>
   ```

   For example:

   ```
   kubectl create namespace dsg
   ```

2. If you want to store the policy snapshot on AWS S3, then perform the following steps to create an IAM service account for the GDSG container.
   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. If you want to store the policy snapshot on a persistent volume, then perform the following steps to create a Kubernetes service account for the DSG containers.
   a. Run the following command to create a Kubernetes service account for the DSG container.

      ```
      kubectl create serviceaccount <Service_account_name> -n <Namespace>
      ```

      For example:

      ```
      kubectl create serviceaccount dsg-sa -n dsg
      ```

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

4. Navigate to the directory where you have extracted the Helm charts for deploying the DSG container, and run the following command to apply the Protegrity PSP to the Kubernetes cluster.

   ```
   kubectl apply -f pty-psp.yaml -n <Namespace>
   ```

   For example:

   ```
   kubectl apply -f pty-psp.yaml -n dsg
   ```

5. Create a file named *kube-system-roles.yaml* for applying privileged roles to the system-level pods created by the Kubernetes system.

   The following snippet displays the contents of the *kube-system-roles.yaml* file.

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: Role
   metadata:
     name: kube-system-role
     labels:
       addonmanager.kubernetes.io/mode: EnsureExists
   rules:
   - apiGroups: ['policy','extensions']
     resources: ['podsecuritypolicies']
     verbs:     ['use']
     resourceNames:
     - eks.privileged
   ---
   apiVersion: rbac.authorization.k8s.io/v1
   kind: RoleBinding
   metadata:
     name: kube-system-role-binding
     labels:
       addonmanager.kubernetes.io/mode: EnsureExists
   roleRef:
     apiGroup: rbac.authorization.k8s.io
     kind: Role
     name: kube-system-role
   subjects:
   - apiGroup: rbac.authorization.k8s.io
     kind: Group
     name: system:serviceaccounts:kube-system
   ```

   Important: If you want to copy the contents of the *kube-system-roles.yaml* file, then ensure that you indent the file as per YAML requirements.

6. Run the following command to create privileged roles for the *kube-system* pods, which are the system-level pods created by the Kubernetes system.

   ```
   kubectl apply -f kube-system-roles.yaml -n kube-system
   ```

   This ensures that the Amazon EKS privileged PSP is restrictively applied only to the system-level pods. This step is required because in *step 4*, the Protegrity PSP is applied only to the *dsg* namespace.

7. Run the following command to delete the cluster-level role binding for the privileged Amazon EKS PSP.

   ```
   kubectl delete clusterrolebindings eks:podsecuritypolicy:authenticated
   ```

   This step ensures that authenticated users cannot use the privileged Amazon EKS PSP.

2.11.2 Applying Role Based Access Control (RBAC)

This section describes how to apply RBAC.

> **Important:** You require the *cluster-admin* role to perform this task.

To apply an RBAC:

1. Perform the following steps to create service accounts that can be used to deploy the DSG container.

   > **Important:**
   > You require the *cluster-admin* role to perform these steps.

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

   a. Run the following command to create a service account that can be used to deploy the DSG container.

   ```
   kubectl create serviceaccount <Service account name> -n <Namespace>
   ```

   For example:

   ```
   kubectl create serviceaccount dsg-deployer --namespace dsg
   ```

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

   b. Run the following command to retrieve the token name for the service account.

   ```
   kubectl get serviceaccount <Service account name> --namespace <Namespace> -o
   jsonpath='{.secrets[0].name}'
   ```

   For example:

   ```
   kubectl get serviceaccount dsg-deployer --namespace dsg -o jsonpath='{.secrets[0].name}'
   ```

   c. Run the following command to obtain the service account token, which is stored as a Kubernetes secret, using the token name retrieved in *step 1b*.

   ```
   kubectl get secret <token name> --context <Valid admin context> --namespace <Namespace>
   -o jsonpath='{.data.token}'
   ```

   For example:

   ```
   kubectl get secret <token name> --context kubernetes-admin@kubernetes --namespace dsg
   -o jsonpath='{.data.token}'
   ```

   A Kubernetes context specifies the configuration for a specific Kubernetes cluster that is associated with a namespace and a corresponding service account.

You can obtain the context by running the following command:

```
kubectl config current-context
```

d. Run the following command to decode the service account token obtained in *step 1c*.

```
TOKEN=`echo -n <Token from step 1c> | base64 -d`
```

2. Navigate to the directory where you have extracted the Helm charts for deploying the DSG container and run the following command to apply the Protegrity RBAC to the Kubernetes cluster.

> **Important:**
> You require the *cluster-admin* role to perform these steps.

```
kubectl apply -f pty-rbac.yaml -n <Namespace>
```

For example:

```
kubectl apply -f pty-rbac.yaml -n dsg
```

For more information about the extracted Helm charts, refer to the of the section *Initializing the Linux Instance*.

2.11.3 Setting up the Environment for Deploying the DSG Container

This section describes how to set up the environment for deploying the DSG container on the Kubernetes cluster.

> **Note:**
> The following steps are applicable only if PSP and RBAC are applied.

> **Important:** You require the *cluster-admin* role to perform this task.

➤ To set up the environment for deploying the DSG container:

1. Perform the following steps if you want to use NGINX as the Ingress Controller. Skip these steps if you want to use the default Ingress controller provided by AWS.

> **Note:** Protegrity recommends using the NGINX Ingress Controller.

a. Run the following command to create a namespace where the NGINX Ingress Controller needs to be deployed.

```
kubectl create namespace <Namespace name>
```

For example:

```
kubectl create namespace nginx
```

b. Run the following command to add the repository from where the Helm charts for installing the NGINX Ingress Controller need to be fetched.

```
helm repo add stable https://charts.helm.sh/stable
```

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
```

c. Run the following command to install the NGINX Ingress Controller using Helm charts.

```
helm install nginx-ingress ingress-nginx/ingress-nginx --namespace
<Namespace name> --set controller.replicaCount=2 --set
controller.publishService.enabled=true --set controller.extraArgs.enable-ssl-
passthrough="" --set controller.ingressClassResource.name=<Namespace
name> --set controller.ingressClassResource.controllerValue="k8s.io/nginx-
dsg" --set controller.electionID="ingress-controller-leader-nginx-dsg"
--set podSecurityPolicy.enabled=true --set rbac.create=true --
set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-
internal"=\"true\"
```

For example:

```
helm install nginx-ingress ingress-nginx/ingress-nginx --
namespace nginx --set controller.replicaCount=2 --set
controller.publishService.enabled=true --set controller.extraArgs.enable-
ssl-passthrough="" --set controller.ingressClassResource.name=nginx
--set controller.ingressClassResource.controllerValue="k8s.io/nginx-dsg"
--set controller.electionID="ingress-controller-leader-nginx-dsg" --
set podSecurityPolicy.enabled=true --set rbac.create=true --
set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-
internal"=\"true\"
```

The NGINX ingress class name is specified in the *ingress/annotations/kubernetes.io/ingress.class* parameter in the *values.yaml* file.

For more information about the NGINX ingress class name, refer to the section *Ingress Configuration*.

For more information regarding the configuration parameters for installing the NGINX ingress Helm charts, refer to the *Readme file* of the Helm charts.

d. Check the status of the *nginx-ingress* release and verify that all the deployments are running accurately:

```
kubectl get pods -n <Namespace name>
```

For example:

```
kubectl get pods -n nginx
```

2. 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* parameter in the *values.yaml* file.

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

3. 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.

4. 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 on 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.

5. If you want to use AWS EFS for storing the CoP and policy snapshots and use PBE to decrypt the policy, then perform the following steps to create service accounts for the Sample Application container.
   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> --approve
   ```

   For example:

   ```
   eksctl create iamserviceaccount --name dsg-sa --namespace dsg --cluster clustername --
   approve
   ```

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

6. 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.

7. Run the following command to create the *policy-secret* secret for the policy package.
   a. If you are using PBE encryption, then run the following command to create the secret.

   ```
   kubectl create secret generic cop-secret --from-literal=passphrase='<passphrase for
   decrypting the policy>'--from-literal='salt=<salt for encrypting the policy> -n
   <SAME_AS_OF_CHART_DEPLOYMENT>
   ```

For example,

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

For more information about creating and exporting policy, refer to *Creating and Exporting the Policy*.

In the private key text box, enter the value that is copied in step 13 of the section *Creating an AWS Customer Master Key*.

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 2-5: 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.

Parameter	Description
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* of the section *Deploying a Release on the Kubernetes Cluster with PSP and RBAC*.

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 the 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 using the ESA, refer to the section *Working with Proxy Authentication* in the *Enterprise Security Administrator Guide 9.1.0.5*.

2.11.4 Deploying a Release on the Kubernetes Cluster with PSP and RBAC

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

Important:
The following steps are applicable only when PSP and RBAC are applied.

➤ To deploy a release on the Kubernetes cluster:

1. Configure the AWS CLI on your machine by running the following command.

```
aws configure
```

You are prompted to enter the *AWS Access Key ID*, *Secret Access Key*, *AWS Region*, and the default output format where these results are formatted.

For more information about configuring the AWS CLI, refer to the section *Configuring the AWS CLI* in the AWS documentation.

2. Enter the required details as shown in the following snippet.

```
AWS Access Key ID [None]: <AWS Access Key ID of IAM User 2>
AWS Secret Access Key [None]: <AWS Secret Access Key of IAM User 2>
Default region name [None]: <Region where want to deploy the Kubernetes Cluster>
Default output format [None]: json
```

You need to specify the credentials of IAM User 1 to deploy the DSG container.

For more information about the IAM User 1, refer to the section *Verifying Prerequisites on AWS*.

3. Run the following command to update the *kube-config* file, which is the configuration file that is generated when you create a Kubernetes cluster.

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

4. Perform the following steps to set up a Kubernetes context for the *dsg-deployer* service account.
   a. Run the following command to create user credentials in the *kube-config* file, using the token created in *step 1d* of the section *Applying Role Based Access Control (RBAC)*.

```
kubectl config set-credentials <username> --token <TOKEN from step 1d of the section
Applying Role-Based Access Control (RBAC)>
```

   b. Run the following command to create a context in the *kube-config* file for communicating with the Kubernetes cluster.

```
kubectl config set-context <Context Name> --cluster=<Name of the Kubernetes Cluster in
the kube-config file> --user=<username>
```

   c. Run the following command to set the Kubernetes context to the newly created context in *step 4b*.

```
kubectl config use-context <Context name from step 4b>
```

5. 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 the section *Initializing the Linux Instance*.

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

   In a *Blue-Green* deployment strategy, the Release 1 is considered as the *Blue* deployment.

6. Modify the default values in the *values.yaml* file as required.
   For more information about Helm charts and their default values, refer to the *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 AWS S3 for storing the policy snapshot, 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.
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 AWS EFS, 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 immutable policy package in the persistent volume.
privateKeySource	Specify one of the following methods used to encrypt the policy package: • *PKCS5* - Use Passphrase-Based Encryption for encrypting the policy package • *AWS_KMS* - Use AWS Customer Master Key for encrypting the policy package
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	If you want to use AWS S3 for storing the policy or AWS KMS for decrypting the policy snapshot, then specify the name of the service account that you have created in step 4b of the section *Setting up the Environment for Deploying the DSG Container*. If you want to use AWS EFS for storing and PBE for decrypting the policy snapshot, then specify the name of the service account that you have created in step 5b of the section *Setting up the Environment for Deploying the DSG Container*.
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.

Field	Description
	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 atleast 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 8* of the section *Setting up the Environment for Deploying the DSG Container*.
deploymentInUse	Specify the value as *blue*. This indicates that the *blue* deployment strategy is in use.
blueDeployment/enabled	Specify the value as *true*. While staging Release 1, the *blue* deployment strategy is enabled.
blueDeployment/policy/filepath	Specify the path in the persistent volume or the object store where you have stored the policy 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/pathInVolumeToPolicy*. If you have stored the policy package 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/policy.json*, where *test* is the bucket name, *LOB* is the directory inside the bucket, and *policy.json* is the name of the policy pakage. 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.
blueDeployment/cop/filepath	Specify the path in the persistent volume or the object store where you have stored the CoP package.

Field	Description
	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.
greenDeployment/enabled	Specify the value as *false*. While deploying Release 1, the *green* deployment strategy is always disabled.
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. For more information about sending a request to the Load Balancer, refer to the section *Using a load balancer* in the section *Running Security Operations with Static CoP*. • *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. For more information about sending a request to the cluster IP, refer to the section *Using a cluster IP* in the section *Running Security Operations with Static CoP*. This parameter is only applicable if you have set the value of the *ingress/enabled* parameter to *false*. 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*.
ingress/enabled	Specify this value as *true* to enable the ingress resource in Kubernetes. The ingress resource acts as a load balancer. It enables the simultaneous deployment of *blue and green* strategy by exposing both the deployments on the same external IP address, but to different URL paths. **Important:** If you want to configure the DSG container for SMTP and SFTP services, then you must specify this value as *false* to disable the ingress resource in Kubernetes.
ingress/annotations/kubernetes.io/ingress.class	Specify the value of the external ingress controller, if any. This is the same value that you have specified for the controller.ingressClass parameter in *step 2c* of the section *Setting up the Environment for Deploying the DSG Container*. For example, specify, *nginx-dsg* if you want to use the NGINX Ingress Controller. Leave this field as blank if you want to use the default ingress controller given by the Cloud provider.
ingress/annotations/ nginx.ingress.kubernetes.io/ssl-passthrough	Specify whether you want to use the NGINX Ingress Controller as an SSL passthrough, to ensure end-to-end encrypted communication between the REST API client and the DSG pods. Set the value of this field to *true*.
ingress/hosts/host	Specify the hostname of the ingress resource. If you are using the HTTPS channel for communication, then ensure that the hostname is the common name that you specified while creating the server certificate. Ensure that you specify separate subdomain names for the production and staging environment. For example, if you have specified the common name in the server certificates as **.example.com*, where example.com is the domain name, then ensure that you use the following hostnames for the production and staging environments: • For production environment: *prod.example.com* • For staging environment: *staging.example.com* In this example, *prod* and *staging* are the subdomains under the example.com domain.

Field	Description
	For more information about creating a server certificate, refer to the section *Creating Certificates and Keys for TLS Authentication*.
ingress/hosts/httpPaths/port	Specify the same value that you have provided in the *dsgService/tunnels/port* field.
ingress/hosts/httpPaths/prod	If you are using the NGINX Ingress Controller as an SSL passthrough, then you must specify the value of this field as /. This ensures that the data traffic from the REST API client is redirected to the appropriate production or staging URL based on the *hostname* value.
ingress/hosts/httpPaths/staging	If you are using the NGINX Ingress Controller as an SSL passthrough, then you must specify the value of this field as /. This ensures that the data traffic from the REST API client is redirected to the appropriate production or staging URL based on the hostname value.

7. Run the following command to deploy the DSG on the Kubernetes cluster.

```
helm install <Release Name> --namespace <Namespace where you want to deploy the DSG
application> <Location of the directory that contains the Helm charts>
```

For example:

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

8. 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
```

   b. Run the following command to get the status of the ingress.

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

For example:

```
kubectl get ingress -n dsg
```

This command is used to obtain the IP address of the ingress.

2.11.5 Deploying a Release on the Kubernetes Cluster without PSP and RBAC

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 NGINX as the Ingress Controller. Skip these steps if you want to use the default Ingress controller provided by AWS.

   > **Note:** Protegrity recommends using the NGINX Ingress Controller.

   a. Run the following command to create a namespace where the NGINX Ingress Controller needs to be deployed.

   ```
   kubectl create namespace <Namespace name>
   ```

   For example:

   ```
   kubectl create namespace nginx
   ```

   b. Run the following command to add the repository from where the Helm charts for installing the NGINX Ingress Controller need to be fetched.

   ```
   helm repo add stable https://charts.helm.sh/stable
   ```

   ```
   helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
   ```

   c. Run the following command to install the NGINX Ingress Controller using Helm charts.

   ```
   helm install nginx-ingress ingress-nginx/ingress-nginx --namespace
   <Namespace name> --set controller.replicaCount=2 --set
   controller.publishService.enabled=true --set controller.extraArgs.enable-ssl-
   passthrough="" --set controller.ingressClassResource.name=<Namespace
   name> --set controller.ingressClassResource.controllerValue="k8s.io/nginx-
   dsg" --set controller.electionID="ingress-controller-leader-nginx-dsg"
   --set podSecurityPolicy.enabled=true --set rbac.create=true --
   set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-
   internal"=\"true\"
   ```

   For example:

   ```
   helm install nginx-ingress ingress-nginx/ingress-nginx --
   namespace nginx --set controller.replicaCount=2 --set
   controller.publishService.enabled=true --set controller.extraArgs.enable-
   ssl-passthrough="" --set controller.ingressClassResource.name=nginx
   --set controller.ingressClassResource.controllerValue="k8s.io/nginx-dsg"
   --set controller.electionID="ingress-controller-leader-nginx-dsg" --
   set podSecurityPolicy.enabled=true --set rbac.create=true --
   set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-balancer-
   internal"=\"true\"
   ```

   The NGINX ingress class name is specified in the *ingress/annotations/kubernetes.io/ingress.class* parameter in the *values.yaml* file.

   For more information about the NGINX ingress class name, refer to the section *Ingress Configuration*.

   For more information regarding the configuration parameters for installing the NGINX ingress Helm charts, refer to the *Readme file* of the Helm charts.

   d. Check the status of the **`nginx-ingress`** release and verify that all the deployments are running accurately.

   ```
   kubectl get pods -n <Namespace name>
   ```

   For example:

   ```
   kubectl get pods -n nginx
   ```

2. 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.

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:

   - *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.

4. If you want to use AWS EFS for storing the CoP and policy snapshots and use PBE to decrypt the policy, then perform the following steps to create service accounts for the Sample Application container.
   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> --approve
```

For example:

```
eksctl create iamserviceaccount --name dsg-sa --namespace dsg --cluster clustername --
approve
```

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

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. Run the following command to create the *policy-secret* secret for the policy package.
   a. If you are using PBE encryption, then run the following command to create the secret.

```
kubectl create secret generic cop-secret --from-literal=passphrase='<Passphrase used
to encrypt the policy>'--from-literal='salt=<salt for encrypting the policy> -n
<SAME_AS_OF_CHART_DEPLOYMENT>
```

For example,

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

   b. If you are using KMS encryption, then run the following command to create the secret.

```
kubectl create secret generic cop-secret --from-literal='privatekey='<ARN of KMS key
used to encrypt the policy>' -n <SAME_AS_OF_CHART_DEPLOYMENT>
```

For example,

```
kubectl create secret generic key-secret --from-literal='privatekey=arn:aws:kms:us-
east-1:829528124735:key/2b954307-6185-4eea-a384-ecb85bbdcab1' -n dsg
```

For more information about creating and exporting policy, refer to *Creating and Exporting the Policy*

7. 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*.

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 2-6: 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.

Parameter	Description
	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 10* of the section *Initializing the Linux Instance*.

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

   In a *Blue-Green* deployment strategy, the Release 1 is considered as the *Blue* deployment.

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
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 AWS EFS, 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 immutable policy package in the persistent volume.
privateKeySource	Specify one of the following methods used to encrypt the policy package: • *PKCS5* - Use Passphrase-Based Encryption for encrypting the policy package • *AWS_KMS* - Use AWS Customer Master Key for encrypting the policy package
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.
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:**

Field	Description
	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 atleast 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*.
deploymentInUse	Specify the value as *blue*. This indicates that the *blue* deployment strategy is in use.
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. 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.
blueDeployment/policy/filepath	Specify the path in the persistent volume or the object store where you have stored the policy package.

Field	Description
	For example, if you are using AWS EFS as the persistent volume, then you can specify the file path as *file://var/data/ptypolicy/pathInVolumeToPolicy*. If you have stored the policy package 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/policy.json*, where *test* is the bucket name, *LOB* is the directory inside the bucket, and *policy.json* is the name of the policy pakage. 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.
blueDeployment/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.
greenDeployment/enabled	Specify the value as *false*. While deploying Release 1, the *Green* deployment strategy is always disabled.

Field	Description
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. For more information about sending a request to the Load Balancer, refer to the section *Using a load balancer* in the section *Running Security Operations with Static CoP*. • *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. For more information about sending a request to the cluster IP, refer to the section *Using a cluster IP* in the section *Running Security Operations with Static CoP*. This parameter is only applicable if you have set the value of the *ingress/enabled* parameter to *false*. If you specify the service type, then you also need to uncomment the annotations for the corresponding service type.
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*.
ingress/enabled	Specify this value as *true* to enable the ingress resource in Kubernetes. The ingress resource acts as a load balancer. It enables the simultaneous deployment of *blue* and *green* strategy by exposing both the deployments on the same external IP address, but on different URL paths.
ingress/annotations/kubernetes.io/ingress.class	Specify the value of the external ingress controller, if any. This is the same value that you have specified for the controller.ingressClass parameter in *step 2a*. For example, specify, *nginx-dsg* if you want to use the NGINX Ingress Controller.

Field	Description
	Leave this field as blank if you want to use the default ingress controller given by the Cloud provider.
ingress/annotations/nginx.ingress.kubernetes.io/ssl-passthrough	Specify whether you want to use the NGINX Ingress Controller as an SSL passthrough, to ensure end-to-end encrypted communication between the REST API client and the DSG pods. Set the value of this field to *true*.
ingress/hosts/host	Specify the hostname of the ingress resource. If you are using the external IP address of the ingress resource for performing the security operations, then leave this field blank.
ingress/hosts/httpPaths/port	Specify the same value that you have provided in the *dsgService/tunnels/port* field.
ingress/hosts/httpPaths/prod	Specify the URI used for pointing to current production deployment. This the URI on which the production data traffic is diverted. For example: */v1/restapi*.
ingress/hosts/httpPaths/staging	Specify the URI used for pointing to current deployment under test. For example: */v2/restapi*

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 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
```

    b. Run the following command to get the status of the ingress.

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

For example:

```
kubectl get ingress -n dsg
```

This command is used to obtain the IP address of the ingress.

2.12 Upgrading the Helm Charts

This section describes the steps for upgrading the Helm charts.

Protegrity recommends using the *Blue-Green* method of upgrading the Helm charts. In this method, you use two modes of deploying the application, production mode and staging mode. In the staging mode, you can test the changes to the application in a staging environment, without impacting the production environment. After you have tested the application in the staging environment, you can switch the environments by making the staging environment as the new production environment and the production environment as the new staging environment.

In this way, you can seamlessly divert the customer traffic to the modified application without any production downtime. After your modified application is running on the new production environment, you can shutdown the application that is running on the staging environment. In the *Blue-Green* method of deploying applications, only one Release is active at a given point of time.

➤ To upgrade the Helm charts:

1. Modify the policy or CoP as required.
   Similarly, create a new CoP or modify an existing CoP on the ESA.

   For more information about creating a Ruleset, refer to the section *Creating and Uploading the CoP to AWS Storage*.

2. On the Linux instance, navigate to the location where you have extracted the Helm charts for installing the DSG application.
   For more information about the extracted Helm charts, refer to the section *Initializing the Linux Instance*.

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

   In a *Blue-Green* deployment strategy, the Release 2 is considered as the *Green* deployment.

3. 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 AWS S3 for storing the policy snapshot, 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.
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 AWS EFS, 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 immutable policy package in the persistent volume.
privateKeySource	Specify one of the following methods used to encrypt the policy package: • *PKCS5* - Use Passphrase-Based Encryption for encrypting the policy package • *AWS_KMS* - Use AWS Customer Master Key for encrypting the policy package
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.
ldapXmlSecrets	Specify the value of the LDAP secret that you have created from the LDAP parameters using the `kubectl` command.
deploymentInUse	Specify the value as *blue*. This indicates that the *blue* deployment strategy is in use.
greenDeployment/enabled	Specify the value as *true*. While upgrading the release to Release 2, the *green* deployment strategy is enabled.
greenDeployment/policy/filepath	Specify the path in the persistent volume or the object store where you have stored the policy package.

Field	Description
	For example, if you are using AWS EFS as the persistent volume, then you can specify the file path as *file://var/data/ptypolicy/pathInVolumeToPolicy*. If you have stored the policy package 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/policy.json*, where *test* is the bucket name, *LOB* is the directory inside the bucket, and *policy.json* is the name of the policy pakage. 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.
greenDeployment/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:

Field	Description
	• *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. For more information about sending a request to the Load Balancer, refer to the section *Using a load balancer* in the section *Running Security Operations with Static CoP*. • *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. For more information about sending a request to the cluster IP, refer to the section *Using a cluster IP* in the section *Running Security Operations with Static CoP*. This parameter is only applicable if you have set the value of the *ingress/enabled* parameter to *false*. If you specify the service type, then you also need to uncomment the annotations for the corresponding service type.
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*.
ingress/enabled	Specify this value as *true* to enable the ingress resource in Kubernetes. The ingress resource acts as a load balancer. It enables the simultaneous deployment of *blue and green* strategy by exposing both the deployments on the same external IP address, but of different URL paths.
ingress/annotations/kubernetes.io/ingress.class	Specify the value of the external ingress controller, if any. For example, specify, *nginx-dsg* if you want to use the NGINIX Ingress Controller. Specify the value of the external ingress controller, if any. This is the same value that you have specified for the *controller.ingressClass* parameter in *step 1c* of the section *Setting up the Environment for Deploying the DSG Container*.

Field	Description
	For example, specify *nginx-dsg* if you want to use the NGINX Ingress Controller. Leave this field as blank if you want to use the default ingress controller given by the Cloud provider.
ingress/annotations/nginx.ingress.kubernetes.io/ssl-passthrough	Specify whether you want to use the NGINX Ingress Controller as an SSL passthrough, to ensure end-to-end encrypted communication between the REST API client and the DSG pods. Set the value of this field to *true*.
ingress/hosts/host	Specify the hostname of the ingress resource. If you are using the external IP address of the ingress resource for performing the security operations, then leave this field blank.
ingress/hosts/httpPaths/port	Specify the same value that you have provided in the *dsgService/tunnels/port* field.
ingress/hosts/httpPaths/prod	Specify the URI used for pointing to current production deployment. This is the URI to which the production data traffic is diverted. For example: */v1/restapi*.
ingress/hosts/httpPaths/staging	Specify the URI used for pointing to current deployment under test. For example: */v2/restapi*

4. Run the following command to upgrade the Helm charts:

```
helm upgrade <Release Name> <Location of the directory that contains the Helm charts>
--namespace <Namespace where you want to deploy the DSG application>
```

For example:

```
helm upgrade dsg-demo dsg --namespace dsg
```

Using this configuration ensures that the DSG application is deployed with the updated policy and CoP snapshot on the staging environment.

5. Verify whether the updated configuration is working accurately by running security operations on the staging environment.

For more information about running security operations, refer to the section *Running Security Operations*.

After you have verified that the updated configuration is working accurately, you can switch the production and staging environments so that all the production traffic is directed to the DSG containers with the updated configuration.

6. Modify the *values.yaml* file to change the value of the *deploymentInUse* field to *green*.

This ensures that the *green* deployment is now considered as the production environment, and the updated configuration handles all the production traffic from the customer application.

7. Run the following command to upgrade the Helm charts:

```
helm upgrade <Release Name> <Location of the directory that contains the Helm charts>
--namespace <Namespace where you want to deploy the DSG application>
```

For example:

```
helm upgrade dsg-demo dsg --namespace dsg
```

Using this configuration ensures that the DSG application is deployed with the updated policy and CoP snapshot on the staging environment.

After the update configuration is working correctly on the new production environment, you can shut down the pods that are running the Release 1 containers.

8. Modify the *values.yaml* file to change the value of the *blueDeployment/enabled* field to *false*.
   This will shutdown all the pods that were deployed as part of the *blue* deployment.

   > **Note:**
   > If you do not change the value of this parameter to *false* and upgrade the Helm charts, then the pods in the staging environment will keep consuming resources.

9. Run the following command to upgrade the Helm charts:

   ```
   helm upgrade <Release Name> <Location of the directory that contains the Helm charts>
   --namespace <Namespace where you want to deploy the DSG application>
   ```

   For example:

   ```
   helm upgrade dsg-demo dsg --namespace dsg
   ```

   Now, only the production environment is running with the updated configurations.

   If you want to modify any policy or CoP for subsequent releases, you need to repeat steps 1 to 9. The only difference is that you need to toggle the value of the *deploymentInUse* field from *green* to *blue* and vice versa depending on which deployment contains your modified configurations.

Chapter 3

Protecting Data using the DSG Container Deployment

This section describes how to protect data using the DSG Container that has been deployed on the Kubernetes cluster using the Postman client and cURL requests as an example. The Postman client is used to make REST calls to the DSG Container.

Protegrity provides you a sample application that you can use to test the data protection capabilities of the DSG Container deployment.

For more information about using the sample application, refer to the section *Appendix B: Using the Sample Application*.

3.1 Running Security Operations

This section describes how you can use the DSG instances running on the Kubernetes cluster to protect the data that is sent by a REST API client, by using static and dynamic CoP.

3.1.1 Running Security Operations with Static CoP

This section describes how you can use the DSG instances running on the Kubernetes cluster to protect the data that is sent by a REST API client, by using static CoP. In this example, the default *Text Protection* ruleset is used for protecting the data.

➤ To run security operations with static CoP:

Using Ingress:

1. Add an entry in the *hosts* file of your node from where you are making the REST API calls, to map the external IP address of the Ingress object to the hostname.

   > **Note:**
   > The external IP address is the IP address of the ingress object that you obtain after you run the *kubectl get ingress* command after deploying the application on the Kubernetes cluster.
   >
   > For more information about the external IP address associated with the ingress object, refer to *step 10* of the section *Deploying a Release on the Kubernetes Cluster*.

   For Windows:

a. Navigate to the `C:\Windows\System32\drivers\etc` directory.

b. Edit the *hosts* file in any text editor, such as Notepad, as an administrator, and add the following entry.

```
<External_IP> <Host name in your Ruleset>
```

For Linux:

a. Run the following command.

```
vi /etc/hosts
```

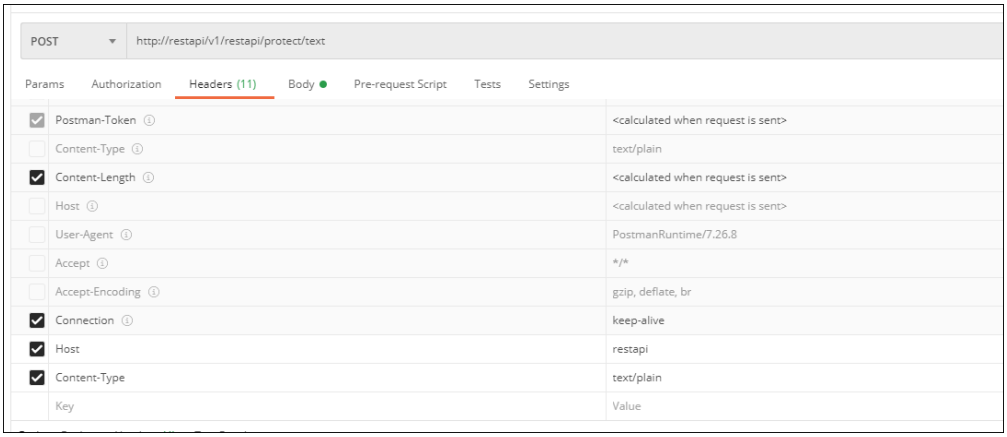
b. Add the following entry.

```
<External_IP> <Host name in your Ruleset>
```

2. In the URL field of the Postman client, enter the URL of the DSG Kubernetes cluster that you want to send the request for protecting data.

   For example, `http://restapi/prod/protect/text`

   This example uses the default *Text Protection* ruleset that is a part of the REST API Examples available on the ESA.



3. In the **Header** tab, specify the value for the Host key.

   For example, enter *restapi*.

4. 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 in the REST API client.

   This is used by the Transaction Metrics.

5. In the **Authorization** tab, select the authorization type as **Basic Auth** in the **Type** list, and then specify the values of the username and password in the **Username** and **Password** fields respectively.

> **Note:** Ensure that the Basic Authentication functionality has been enabled in the ESA.
>
> For more information about enabling the Basic Authentication functionality in the ESA, refer to *step 11* of the section *11*.

6. In the **Body** tab, specify the data that you want to protect.

7. Click **Send**.

   The DSG instance protects the data and returns the protected data.

Using a load balancer:

8. Run the following command to retrieve the load balancer IP address.

```
kubectl get svc -ndsg
```

9. Specify the details in the Postman client as shown in the following figure.

10. In the **Authorization** tab, select the authorization type as **Basic Auth** in the **Type** list, and then specify the values of the username and password in the **Username** and **Password** fields respectively.

> **Note:** Ensure that the Basic Authentication functionality has been enabled in the ESA.
>
> For more information about enabling the Basic Authentication functionality in the ESA, refer to *step 11* of the section *11*.

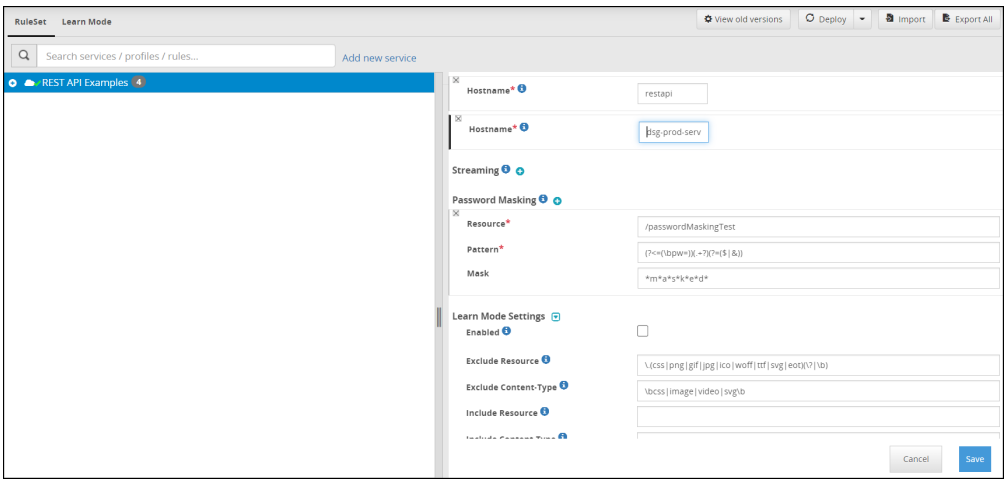
11. In the **Body** tab, specify the data that you want to protect.
12. Click **Send**.

    The DSG instance protects the data and returns the protected data.

Using a cluster IP:

13. Ensure that the clusterIP service is enabled and the Ingress is disabled in the *values.yaml* file.
14. Add the hostname in the CoP in the following format.

    <Kubernetes service name of DSG container>.<Namespace of DSG container>.svc.cluster.local



15. From a pod in the same cluster as the DSG container deployment, run the following cURL request.

```
curl --location --request POST 'http:// <Kubernetes service name of DSG
container>.<Namespace of DSG container>.svc.cluster.local:8080/v1/restapi/protect/text'
--header 'Host: <Kubernetes service name of DSG container>.<Namespace of
DSG container>.svc.cluster.local' --header 'Content-Type: text/plain' --data
'DataTobeProtected'
```

If you are using the Basic Authentication functionality, then you need you need to include the Base64-encoded value of the username and password as part of the authorization header, as shown in the following snippet.

```
curl --location --request POST 'http:// <Kubernetes service name of DSG
container>.<Namespace of DSG container>.svc.cluster.local:8080/v1/restapi/protect/text'
--header 'Host: <Kubernetes service name of DSG container>.<Namespace of
```

```
DSG container>.svc.cluster.local' --header 'Content-Type: text/plain' --data
'DataTobeProtected' -H 'Authorization: Basic <Base64 Encoding{Username:Password}>'
```

> **Note:** Ensure that the Basic Authentication functionality has been enabled in the ESA.
>
> For more information about enabling the Basic Authentication functionality in the ESA, refer to *step 11* of the section *11*.
>
> .

3.1.2 Running Security Operations with Dynamic CoP in HTTPS Mode

This section describes how you can use the DSG instances running on the Kubernetes cluster to protect the data that is sent by a REST API client, by using dynamic CoP in HTTPS mode.

To run security operations with dynamic CoP:

1. In the URL field of the Postman client, enter the URL of the DSG Kubernetes cluster that you want to send the request for protecting data.

   For example, ***`https://prod.example.com/protect`***

   In this example:

   - *protect* is the endpoint URI that you have specified while creating your ruleset.
   - *prod.example.com* is the hostname that you have configured in your ruleset. This value is also specified as the hostname of the Ingress service in the *values.yaml* file. In addition, ensure that you add this value in the *hosts* file of your node from where you are making the REST API calls, to map the external IP address of the Ingress object to the hostname.

     For more information regarding the external IP address associated with the ingress object, refer to *step 10* of the section *Deploying a Release on the Kubernetes Cluster*.

2. In the */etc/resolv.conf* file in Linux, resolve the IP address.
3. In the **Header** tab, specify the value for the **X-Protegrity-DCoP-Rules** key.

   For example, specify the following dynamic CoP as the header value:

```
[{"action":{"payload":{"encoding":{"type":"NO_ENCODING"},
"hasToMatch":"","pattern":"\\w+","type":"TEXT"},"type":"EXTRACT"},
"children":[{"action":{"method":{"dataElementName":"TE_Unicode_S23",
"encoding":{"type":"NO_ENCODING"},"method":"Protect","type":"PROTEGRITY_DATA_PROTECTION"},
"type":"TRANSFORM"},"enabled":true,"text":"transformrule"}],
"enabled":true,"text":"extracttext"}]
```

> **Note:**
> Ensure that the data element included in the header value is present in the ESA policy.
>
> For example, the data element *TE_Unicode_S23*, which is included in the header value, must be present in your ESA policy.

4. 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 in the REST API client.

   This is used by the transaction metrics.

5. In the **Authorization** tab, select the authorization type as **Basic Auth** in the **Type** list, and then specify the values of the username and password in the **Username** and **Password** fields respectively.

> **Note:** Ensure that the Basic Authentication functionality has been enabled in the ESA.
>
> For more information about enabling the Basic Authentication functionality in the ESA, refer to *step 11* of the section *Creating and Exporting the Ruleset*.

6. In the **Body** tab, specify the data that you want to protect.
7. Import the CA certificate, client certificate and the private key of the client certificate to the Postman client.

   For more information about how to create the CA certificate and client certificate and keys, refer to the section *Creating Certificates and Keys for TLS Authentication*.

   For more information about how to import the certificates to the Postman client, refer to the section *Importing Certificates to the Postman Client*.

8. Click **Send**.

   The DSG instance protects the data and returns the protected data.

3.2 Viewing Logs in Splunk

This section describes how you can view the DSG and Container logs in Splunk Enterprise.

➤ To view logs:

1. Login to Splunk Web at *http://<IP of Linux instance>:8000*.
2. On the Splunk Web UI, type the username and password that you have created while installing Splunk Enterprise on the Linux instance.

   The Splunk Enterprise screen appears.

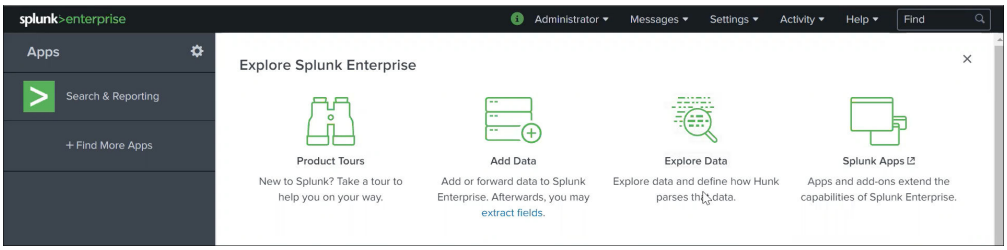


Figure 3-1: Splunk Enterprise Web UI

   For more information about the user credentials, refer to the section *Setting Up Splunk*.

3. On the left pane, click **Search & Reporting**.

   The **Search** screen appears.

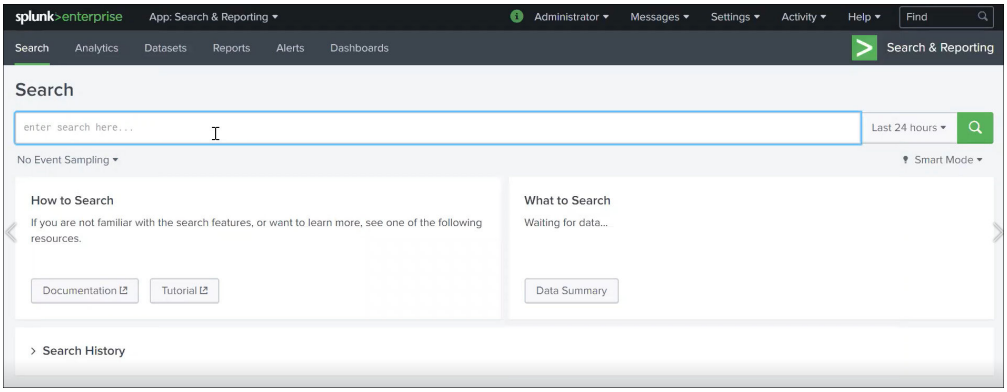


Figure 3-2: Search Screen

4. In the **Search** field, type the following text to filter the logs.

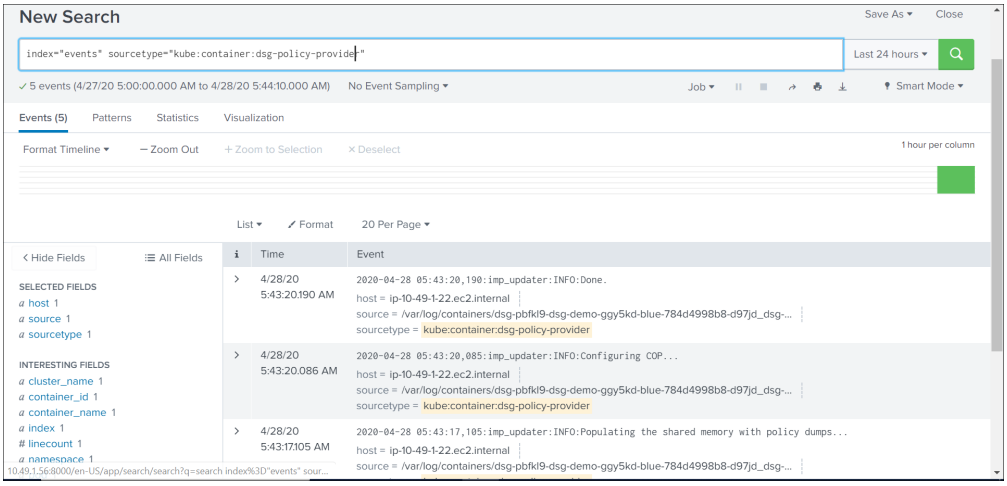
```
index="<Index_name>" sourcetype="kube:container:<Container_name>"
```

For example:

```
index="<Index_name>" sourcetype="kube:container:dsg"
```

5. Press **Enter**.

The search results appear in the **Events** tab. The search results display all the *STDOUT* logs from the specified container.

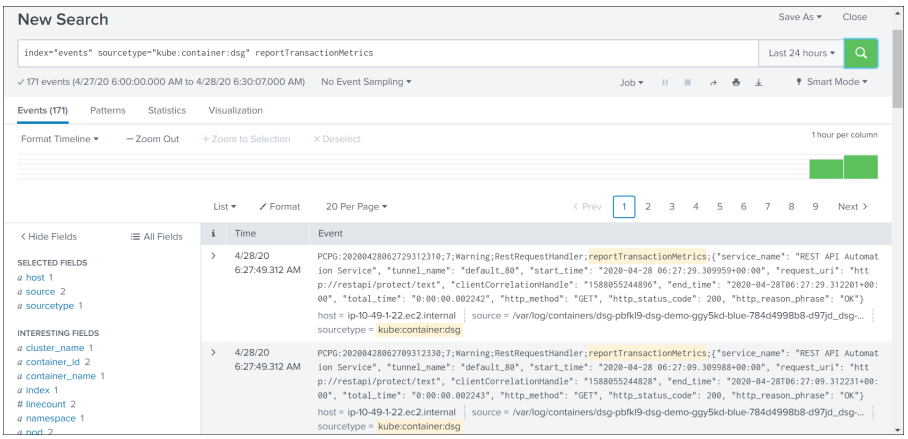


6. If you want to view only the transaction metric logs that are related to the DSG, then you can modify the text in the **Search** field to filter the logs, as shown in the following snippet.

```
index="<Index_name>" sourcetype="kube:container:dsg" reportTransactionMetrics
```

7. Press **Enter**.

The search results display only the logs that are related to the DSG.



3.3 Autoscaling the Protegrity Reference Deployment

You can use the autoscaling property of Kubernetes to autoscale the DSG instances based on a specific load. After the load crosses a pre-defined threshold, Kubernetes automatically scales up the DSG instances. Similarly, as the load dips below the pre-defined threshold, Kubernetes automatically scales down the DSG instances.

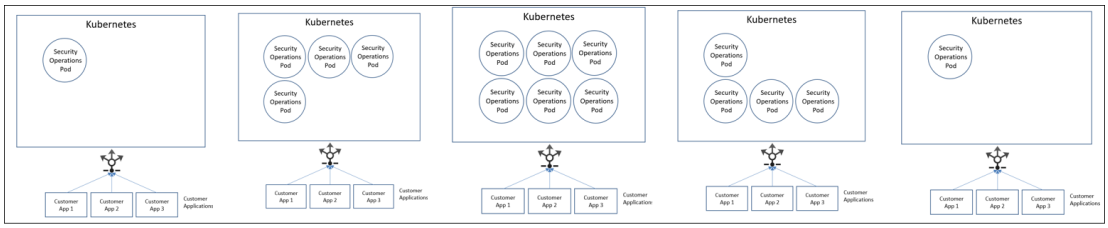


Figure 3-3: Autoscaling Kubernetes Cluster

As illustrated in the figure, the Customer Applications experience an increase and a decrease in workload required by the business. Kubernetes will automatically grow the Protegrity Security Operation pods to meet business needs. If the workloads reduce, then Kubernetes will shrink the cluster.

You do not have to provision additional DSG appliances to meet the business needs and then remove them if not required. Kubernetes performs the task of provisioning the DSG instances automatically.

The auto scaling capability ensures that the business needs are met dynamically while optimizing the costs.

Chapter 4

Troubleshooting Issues

This section includes the issues that you might encounter and the recovery actions to troubleshoot these issues.

Table 4-1: Troubleshooting Issues

Issue	Recovery Action
When you run the **`kubectl get pods`** command, the **`ImagePullBackOff`** status appears.	This issue occurs if the container images have not been pulled accurately from the container repository. Run the **`kubectl describe pod`** command to retrieve the pod details. This can help you in debugging the error.
When you run the **`kubectl get pods`** command, the **`Pending`** status appears.	This issue occurs if adequate memory or CPU is not available for creating a pod. As a workaround, you can try to increase the memory or CPU of the existing nodes where the pod is being deployed, or create a new node where you can deploy the pod.

The DSG container log displays the following error:
 	
`in bind_sockets sock.bind(sockaddr)`
`PermissionError: [Errno 13] Permission denied`
	This error can occur if you create a tunnel with port lesser than *1024*, while creating a ruleset on the DSG Web UI.
	
	To avoid this error, ensure that you create a tunnel with port greater than *1024* as the Docker containers do not allow any external communication on ports below *1024*.

If you send a request to the DSG container with the Basic Authentication functionality enabled, then the response takes more than 1 minute and returns the following error: `401 Unauthorized` In addition, the state of the serving DSG pod changes to *non-ready*.	Check the DSG pod logs. If the logs display the following error, then it indicates that the LDAP server is unavailable. `Error;LDAPConnection;CheckUserRole;LDAP Error: {u'info': 'Transport endpoint is not connected', 'errno': 107, 'desc': u"Can't contact LDAP server"}`
When you install the DSG container, the following message appears: `Warning FailedAttachVolume 2s (x2 over 6s) attachdetach-controller AttachVolume.Attach failed for volume "dsg-blue-pv" : rpc error: code = Internal desc = unknown Attach error: failed when waiting for zonal op: operation operation-1660032645200-5e5ca7821faac-50f601af-e3eb3e74 failed (RESOURCE_IN_USE_BY_ANOTHER_RESOURCE): The disk resource 'projects/dev-185720/zones/us-east1-b/disks/persistent-disk' is already being used by 'projects/dev-185720/zones/us-east1-b/instances/centos`	Detach the disk that is attached to the Linux instance on AWS.

Chapter 5

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.

5.1 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
```

5.2 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.

5.2.1 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>
```

5.2.2 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: ""
```

5.2.3 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.

5.2.4 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.

5.2.5 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.

5.2.6 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.

5.2.7 Container Security Context

This section specifies the privilege and access control settings for the container.

```
# Security Context object for DSG container
## leave the field empty if not applicable
  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.

5.2.8 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.

```
serviceAccount: <Name of the service account created for the pod>
```

This service account is mapped to the *AmazonS3ReadOnlyAccess* policy to ensure that only the pod, where the DSG Container application has been deployed. The DSG pod is able to download the COP and policy package stored in the S3 bucket/EFS.

5.2.9 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.

5.2.10 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
```

5.2.11 Private Secret Key

This section provides a brief overview for the *privateKeySecret* parameter. If you are using KMS to encrypt the policy package, then the DSG container uses the value specified in the *privateKeySecret* parameter to decrypt the policy package.

```
## k8s secret containing the private key id for decrypting policy
privateKeySecret: PKEY_SECRET_NAME
```

5.2.12 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
```

5.2.13 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
```

5.2.14 Deployment

These configurations refer to the DSG package-related information for the current release. The first release is considered to be *blueDeployment*.

When, *blueDeployment is enabled*, parameter is set to *true*, the configurations mentioned under the *blueDeployment* section are applied.

```
## start with blue deployment first
deploymentInUse: blue
```

```
blueDeployment:
  enabled: true
```

```
blueDeployment:
  enabled: true
  policy:
    # absolute path in ObjectStore from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/pathToPolicy"
    # <GS> "gs://bucketName/pathToPolicy"
    # <Azure> " https://<accountName>.blob.core.windows.net/<container name>/<pathToPolicy>"
    # absolute path in PV mount from where the policy dump file will be fetched.
    # <VOLUME> "file://var/data/ptypolicy/pathInVolumeToPolicy"
    # relative path in PV mount from where the policy dump file will be fetched.
    # it will prepend the '/var/data/ptypolicy' to below path
    # <VOLUME> "file:///pathInVolumeToPolicy"
    # <VOLUME> "pathInVolumeToPolicy" # without any leading '/'
    filePath: ABSOULTE_POLICY_PATH
  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 user is required to update the highlighted values as per the details where you have stored the CoP and policy package. Note that all entries are case sensitive.

5.2.15 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.

5.2.16 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"

  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*.This parameter is only applicable if you have set the value of the *ingress/enabled* parameter to *false*.

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 S3 bucket or the EFS storage.
- 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.

5.2.17 Ingress Configuration

This section explains the Ingress configuration for deployment.

```
## Ingress enables testing of Green (V2/Staging) Deployments alongside Blue ones
simultaneously.
## This is realized by exposing both deployments on the same External IP but on different URL
Paths.
## If you don't want expose both the deployments at the same time, keep 'ingress.enabled' as
false.
## This means that only Blue deployment gets exposed on the Load Balancer's External IP via
prod service.
ingress:
  enabled: true
  # specify ingressClass cluster resource. It associates which controller will implement the
resource.
  ingressClassName: nginx-dsg
  ## else keep the 'ingress.class' field empty
  ## to use cloud native Ingress Controller.
  annotations:
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
    ## specify the host names and paths for Ingress rules
  ## prod and staging http paths can be used as optional fields.
  hosts:
    - host: ""
      httpPaths:
      - port: 8080
        prod: "/v1/restapi"
        staging: "/v2/restapi"
```

In this case, we are referring to NGINX as the ingress. The following configurations can be edited by the user as per the CoP and deployment requirements.

```
ingress:
  enabled: true
```

When the *enabled* parameter is set to *true*, the ingress controller will be based on the value used in the annotation *ingressClassname*. If the parameter *enabled* is set to *false*, then the ingress controller will not be used. Instead, the default load balancer of the Kubernetes service will be used to expose the DSG Container services.

```
annotations:
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
```

If the *ingressClassName* value is set to *nginx-dsg*, then the NGINX Ingress Controller will be used. If the *ingress.ClassName* value is empty, then the native ingress controller will be used. For example, the AWS ingress controller will be used.

If the *nginx.ingress.kubernetes.io/ssl-passthrough* value parameter is set to *true*, then the NGINX Ingress Controller works as an SSL passthrough. This ensures an end-to-end encrypted communication channel between the REST API client and the DSG pods.

The following snippet refers to URLs and ports referred to for *blue green* deployments.

```
hosts:
    - host: ""
      httpPaths:
      - port: 8080
        prod: "/v1/restapi"
        staging: "/v2/restapi"
    ## Add host section with the hostname used as CN while creating server certificates.
    ## For accessing the staging end-points also, we need hostname as that of CN in server
certs.
    ## While creating the certificates you can use *.example.com as CN for the below example
    # - host: "prod.example.com"
    #   httpPaths:
    #   - port: 9443
```

```
    #     prod: "/"
    # - host: "staging.example.com"
    #   httpPaths:
    #   - port: 9443
    #     staging: "/"
    #
  ## If TLS is terminated on the Ingress Controller Load Balancer,
  ## K8s TLS Secret containing the certificate and key must also be provided
  # tls:
  #  - secretName: chart-example-tls
  #    hosts:
  #      - chart-example.local
```

The user must ensure the related COPs are made available on the storage before applying the changes. Ensure that you specify the values of the HTTP paths parameters as shown in the following snippet.

```
      httpPaths:
      - port: 8080
        prod: "/v1/restapi"
        staging: "/v2/restapi"
```

- *port* – Refers to the same value that you have provided in the *dsgService/tunnels/port* field.
- *prod* – Refers to the URI used for pointing to the current production deployment. This is the URI on which the production data traffic is diverted.
- *staging* – Refers to the URI used for pointing to the current deployment under test.

```
    # - host: "somehostname"
    #   httpPaths:
    #   - port: 8080
    #     staging: "/v3/restapi"
    # - host: "randomhostname"
    #   httpPaths:
    #   - port: 8080
    #     prod: "/v4/restapi"
```

If you specify hostname in the *host* field, then ensure that you have the same hostname value configured in your ruleset. In this case, the ingress maps the value of the provided hostname to the external IP address, and you also need to configure the hostname in your DNS.

> **Note:**
> In case of AWS, the hostname must be specified in the Fully Qualified Domain Name (FQDN) format, such as *abc.def.xyz*.

Using the hostname option, you have *prod* URLs running on one hostname and *staging* URLs running on another hostname. If you specify multiple hostnames in the *values.yaml* file, then you must ensure that you have added these hosts names in the DSG ruleset.

For more information about adding multiple hostnames to the DSG rule set, refer to the section *Creating a Ruleset*.

If you do not the specify hostname in the *host* "" field, then the ingress can be accessed on the external IP directly.

If you are using the NGINX Ingress Controller as an SSL passthrough, then you must specify the value of the *prod* and *staging* URL as */*. This ensures that the data traffic from the REST API client is redirected to the required production or staging URL based on the *hostname* value.

Chapter 6

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.

6.1 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.

6.1.1 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
Alphanumeric	deLastName
Alphanumeric	deStreet
Alphanumeric	deCity
Alpha	deState
Alphanumeric	deZipcode
Numeric	deSSN

Token Type	Data Element Name
Unicode	PTY_UNICODE

6.1.2 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.

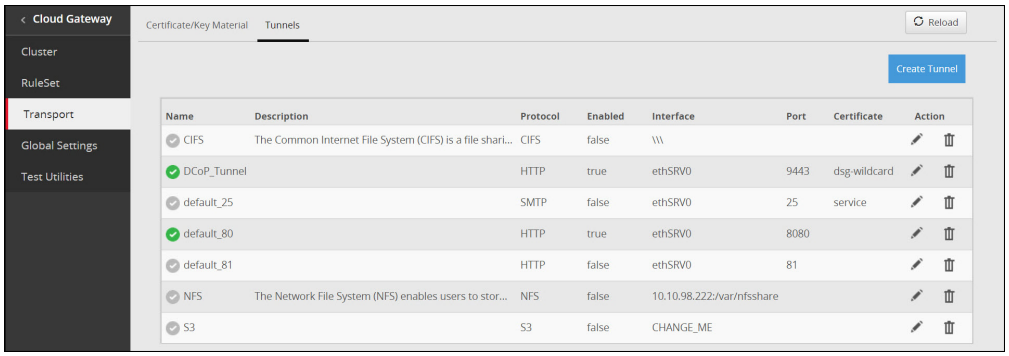
Ruleset 1

- 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.

6.1.3 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.

6.1.4 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
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

Field	Column Name	Token Type	Data Element Name
SSN	SSN	Numeric	deSSN
Customer ID	**Customer ID**	**Customer ID**	**Customer ID**

6.2 Running the Sample Application

1. The required Service accounts are created.

   For more information about the pre-requisites, refer to the section *Verifying Prerequisites for AWS*.

2. The AWS Bucket is created.
3. The AWS Customer Master Key is created.

   > **Note:**
   > This step is optional.

4. The cluster is created. For more information about creating a Kubernetes Cluster, refer to the section *Creating a Kubernetes Cluster*.

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

6.2.1 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.

6.2.2 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 AWS.
   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*.

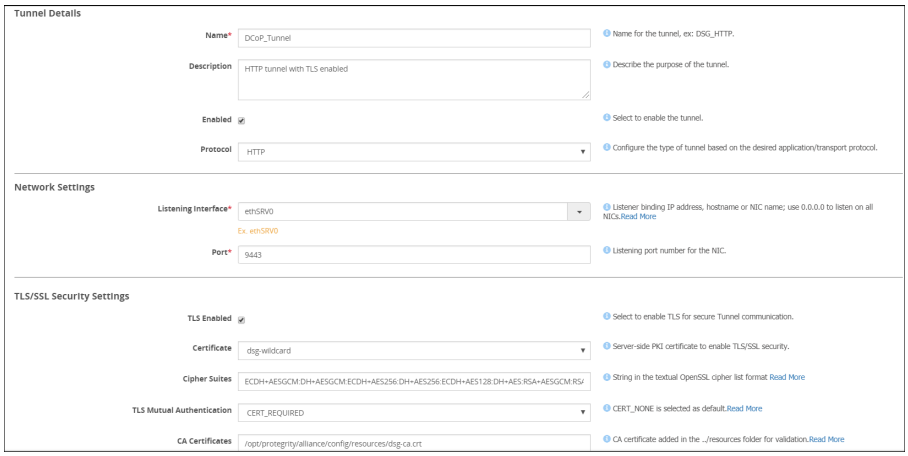


Figure 6-1: Create Tunnel Screen

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 *Creating Certificates and Keys for TLS Authentication*.

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.

6.2.3 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.

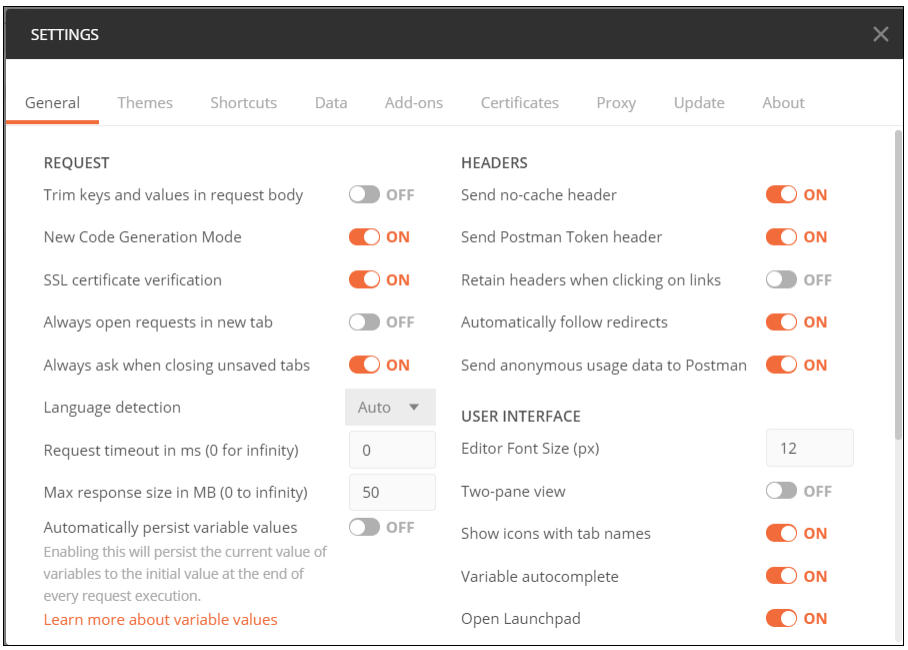


Figure 6-2: SETTING Dialog Box

3. Navigate to the **Certificates** tab.

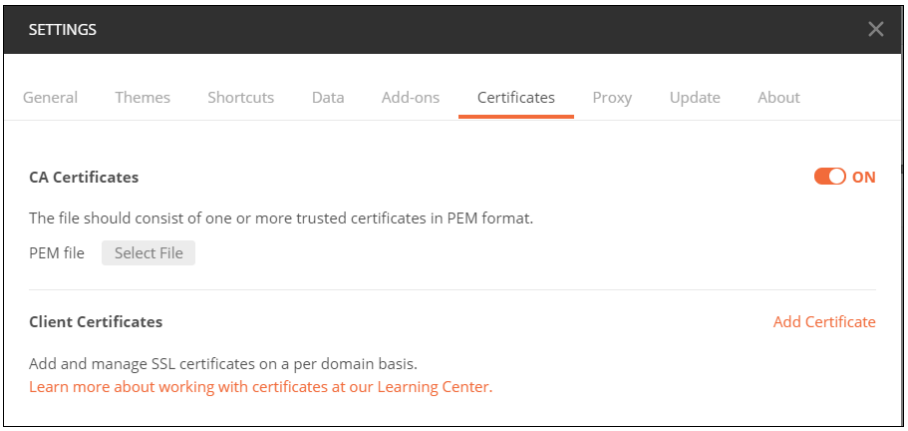


Figure 6-3: Certificates Tab

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 *Creating Certificates and Keys for TLS Authentication*.
5. In the **Client Certificates** section, click **Add Certificate**.

   The **Add Certificate** screen appears.

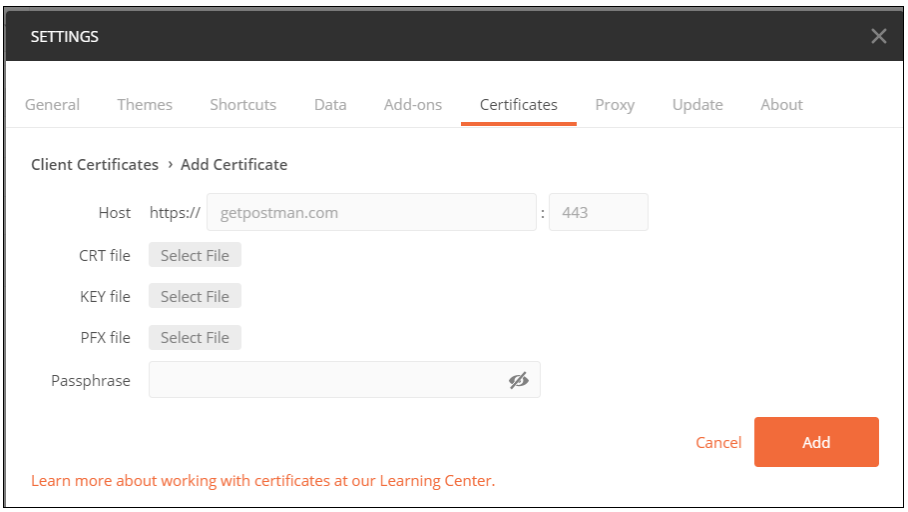


Figure 6-4: Add Certificate Screen

6. In the **Host** field, specify the value as *prod.example.com*.
   You need to specify the ingress 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*.

6.2.4 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 AWS Cognito 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 ID token.

For more information about AWS Cognito, refer to the *AWS Cognito* website.

➤ To enable the authentication functionality:

1. Perform the following steps to create and configure a user pool in AWS Cognito.
   a. On the Linux instance, run the following command to create a user pool in AWS Cognito.
      ```
      aws cognito-idp create-user-pool --pool-name <Container pool name>
      ```
      For example:

      ```
      aws cognito-idp create-user-pool --pool-name container_pool_test_1
      ```
      For more information about user pools, refer to the section *Amazon Cognito User Pools* in the AWS documentation.

      For more information about the parameters used in the *create-user-pool* command, refer to the section *create-user-pool API* in the AWS CLI Command Reference documentation.

   b. Login to the AWS environment.
   c. Navigate to **Services.**
      A list of AWS services appears.

d. In **Security, Identity & Compliance**, click **Cognito**.
The **Amazon Cognito** screen appears.

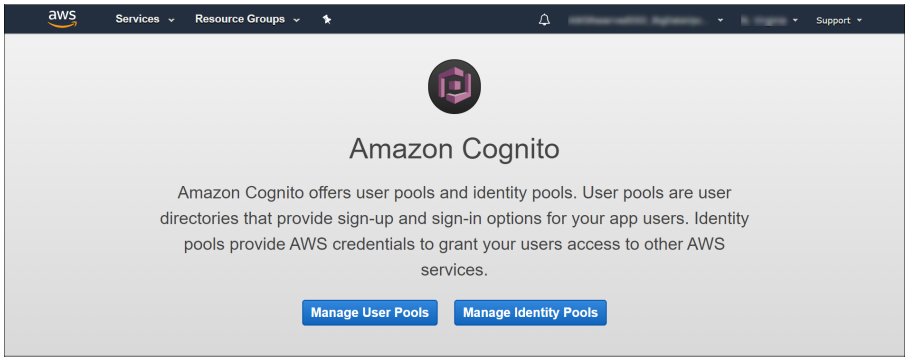


Figure 6-5: Amazon Congito Screen

e. Click **Manage User Pools**.
The **User Pools** screen appears, and it displays a list of user pools that you have created.

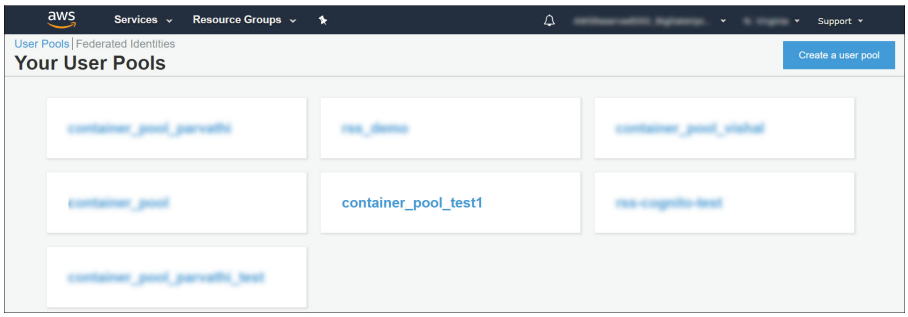


Figure 6-6: User Pools Screen

f. Click the user pool that you have created in *step 1a*.
The details for the selected user pool appear.

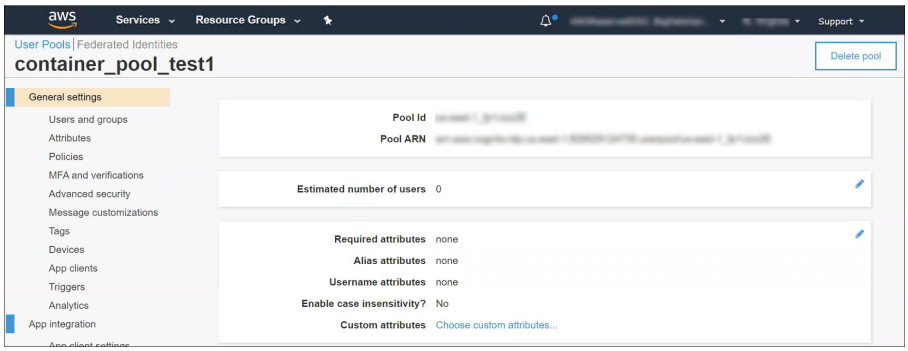


Figure 6-7: Select User Pool Details

Note the Pool Id of the user pool.

g. Navigate to **App integration** > **Domain name**.
The Domain name tab appears.

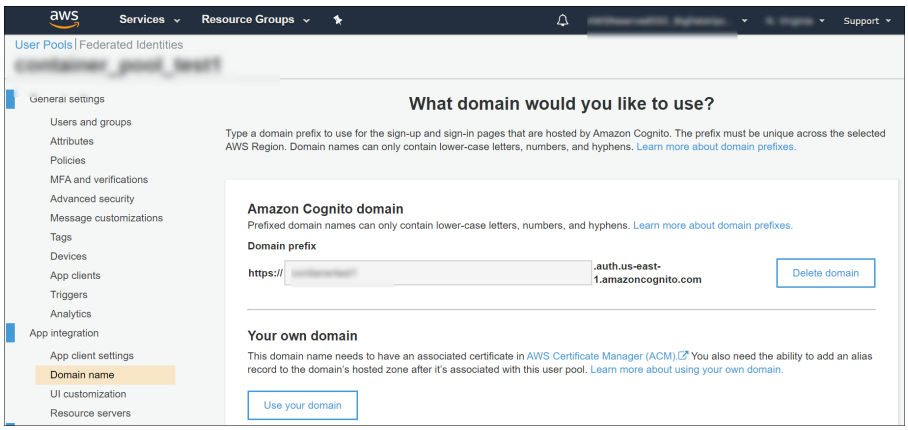


Figure 6-8: Domain Name Tab

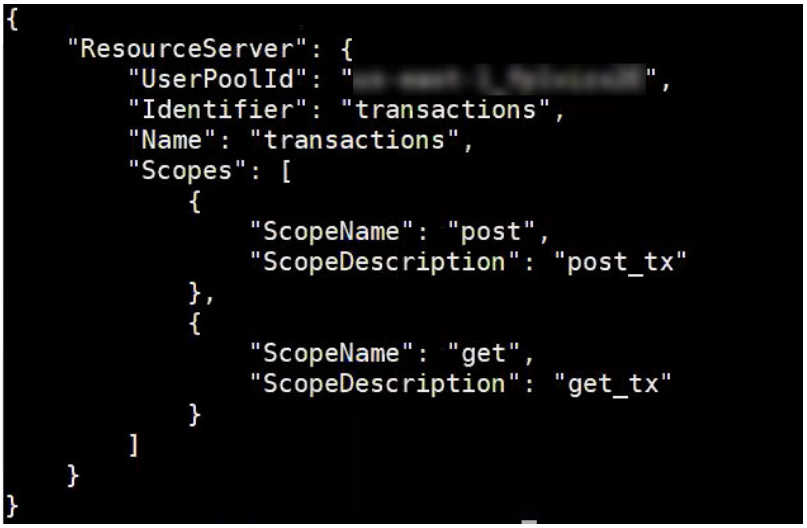
h. In the **Amazon Cognito Domain** > **Domain prefix** field, specify a prefix for the domain name that will be used to obtain the access token from AWS Cognito.

i. On the Linux instance, run the following command to create a resource server for your user pool.

```
aws cognito-idp create-resource-server --name transactions --identifier transactions
--user-pool-id <User Pool Id> --scopes ScopeName=get, ScopeDescription=get_tx
ScopeName=post, ScopeDescription=post_tx
```

Specify the value of the **Pool Id** field that you have noted in *step 1f*, as the value of the *user-pool-id* parameter.

The resource server is created for the selected user pool as shown in the following snippet.



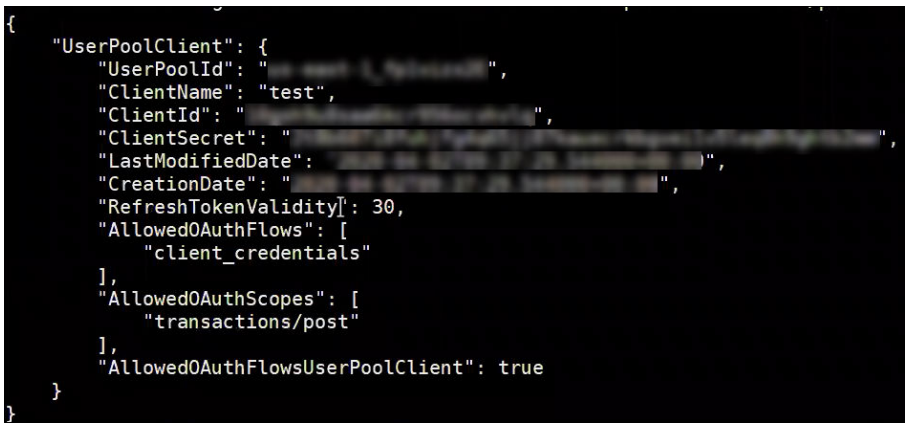
For more information about creating a resource server, refer to the section *Defining Resource Servers for Your User Pool* in the AWS documentation.

For more information about the parameters used in the *create-resource-server* command, refer to the section *create-resource-server* in the AWS CLI Command Reference documentation.

j. Run the following command to create a user pool client.

aws cognito-idp create-user-pool-client --user-pool-id <User Pool ID> --allowed-o-auth-flows client_credentials --client-name test --generate-secret --allowed-o-auth-scopes transactions/post --allowed-o-auth-flows-user-pool-client

The user pool client is created, as shown in the following snippet.



Note the values of the *ClientId* and *ClientSecret* fields. You need to combine these two values to create a Base64 encoded value.

For more information about user pools, refer to the section *Amazon Cognito User Pools* in the AWS documentation.

For more information about the parameters used in the *create-user-pool-client* command, refer to the section *create-user-pool-client API* in the AWS CLI Command Reference documentation.

k. Run the following command to create a Base64 encoded value from the client ID and client secret.

```
echo -n <Client ID>:<Client Secret>|base64
```

In this command, specify the values *ClientId* and *ClientSecret* fields generated in *step 1j* as the values for the *<Client ID>* and *<Client Secret>* parameters respectively.

The Base64 encoded value is generated. You need to specify this value when you send a CURL request for generating an access token.

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.

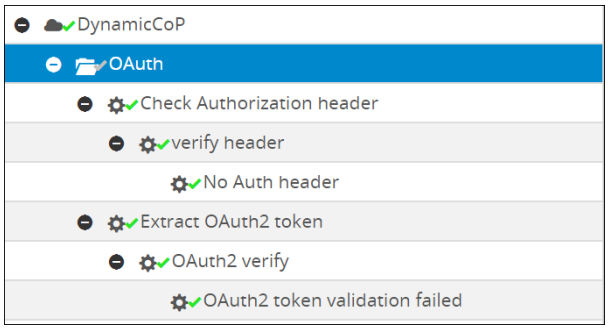


Figure 6-9: OAuth UDF

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.

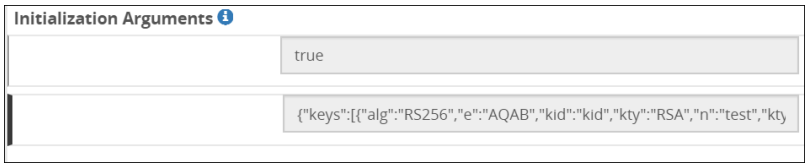


Figure 6-10: Initialization Arguments

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 access token. You can obtain the latest JWKS from AWS Cognito by accessing the following URL: *https://www.cognito-idp.us-east-1.amazonaws.com/<User Pool Id>/.well-known/jwks.json*		Required

i. Click **Save** to save the changes.

j. Click **Reload**.

3. Send the following CURL request to obtain the access token.

```
curl -X POST https://<Domain prefix>.auth.us-east-1.amazoncognito.com/oauth2/token
-H 'authorization: Basic <Base64 encoded value of the client ID
and client secret>' -H 'content-type: application/x-www-form-urlencoded' -d
'grant_type=client_credential&scope=transactions%2Fpost'
```

In the *<Domain prefix>* parameter, specify the value of the domain prefix that you entered in *step 1h*.

In the *<Base64 encoded value of the client ID and client secret>* parameter, specify the value that you have generated in *step 1l*.

The access token is generated.

The following snippet shows a sample access token that is generated.

{"access_token":"eyJraWQiOiJMZ2RZZnR4aWg1RzZWZUp2Y0tuK3RteEpUSmxYTW5mRHlVOEpYQllvdVJrPSIsImFsZyI6IlJTMjU2In0.eyJzdWIiOiIxMGdzaDl1OHNhYTZrY3I5NTZvY3Zodmxx
iwidG9rZW5fdXNlIjoiYWNjZXNzIiwic2NvcGUiOiJ0cmFuc2FjdGlvbnNcL3Bvc3QiLCJhdXRoX3RpbWUiOjE1ODU4MjA1NTEsImlzcyI6Imh0dHBzOlwvXC9jb2duaXRvLWlkcC51cy1lYXN0LTEuYWl
hem9uYXdzLmNvbVwvdXMtZWFzdC0xX2ZwMXZpengyRSIsImV4cCI6MTU4NTgyNDE1MSwiaWF0IjoxNTg1ODIwNTUxLCJ2ZXJzaW9uIjoyLCJqdGkiOiJmZjIzYjI3Mi0wZWE5LTQ2Y2ItOGJiMy1mODIyO
GUwMWVjMDkiLCJjbGllbnRfaWQiOiIxMGdzaDl1OHNhYTZrY3I5NTZvY3Zodmxx
In0.YwX9M1gzzuhcq7lIEPZbUaJTboeojpyRPaU4kg-cGNq1ZHEnNUiDt5blmJ0VhC-E3X-QXWrL0UKv9N87k0Gq_Pu
l0Ta9nAkKbMKDatzdJxqHezrQuUdblZKuvuUBsUBFazXua8gskm2BO4QiiW2rQEPYNu-xsZZXZdp6wR2dYpe3k4QPfTrMIvM55hD8MW0n45LH-W09XbhVnX3AxVfkaQEfpzNUieWmITO4pDWj1ZZ8jf-VU
njjoZTmRL75wR7_tchBfoaIuczMRHUC6cfljgs4JcXwXKu6puNvezQhNEmh_Q3n9q95jBCu4IA-0jmfarxIfCVSd_jZbhoWjlqZMA","expires_in":3600,"token_type":"Bearer"}

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:**
> The generated token has a default timeout of 60 minutes, and you need to regenerate the token after it times out.

6.2.5 Importing the Policy Sample on the ESA

The user needs to perform the following steps to import the *Sample_App_Policy_V5.tgz* file on the ESA.

➤ To import policy sample on the ESA:

1. Login to the ESA as *admin*.
2. Navigate to **Settings** > **Network** > **Web Settings**.

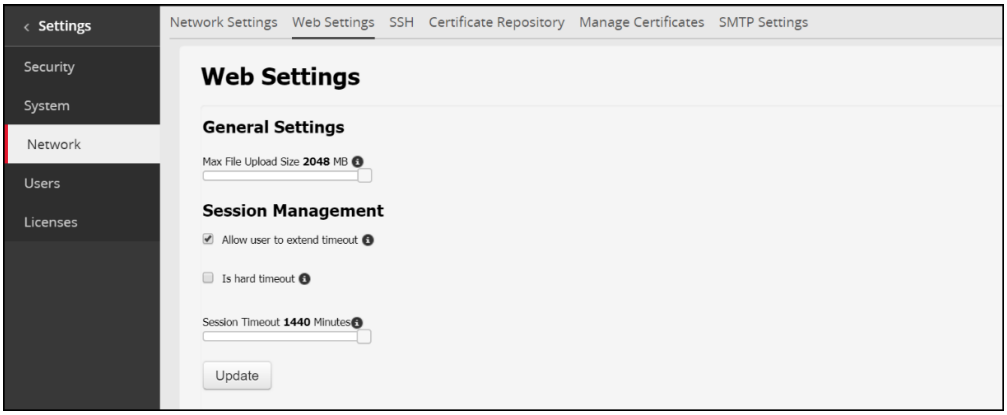


Figure 6-11: ESA Web Settings

3. In the **General Settings** section, change the **Max File Upload Size** value to the maximum value.
4. In the **Session Management** section, change the **Session Timeout** value to the maximum value.
5. Click **Update**.
6. Navigate to **Settings** > **System** > **File Upload**.
7. Click **Choose File** to select the *Sample_App_Policy_V5.tgz* file that you want to upload.
8. Navigate to **System** > **Backup and Restore** > **Import**, and select the *Sample_App_Policy_V5.tgz* file from the drop down and click **Import**.
9. Provide the password as **`admin1234`** and click **Import**.

   After successful import, the *Sample_policy* should be available in the section *Policies*.
10. Run the following command to export the Policy package.

```
curl -v -k -u "export_user:admin1234"
https://xxx.xxx.xxx.xxx/pty/v1/imp/export
-H "Content-Type: application/json"
--data '{"name":"Policy1.json",
"pepVersion":"1.2.2+42",
"dataStoreName":"DS1",
"emptyString":"NULL",
"caseSensitive":"YES","kek":{"pkcs5":
```

```
{"password":"Passphrase123@",
"salt":"salt"}}}' -o Policy1.json
```

For more information about exporting the Policy package, refer to the *step 2* or *step 3* in the section *Creating and Exporting the Policy*.

11. Upload the Policy package to an AWS S3 bucket or EFS.

    For more information about uploading the Policy package to an AWS S3 bucket, refer to the *step 4* in the section *Creating and Exporting the Ruleset*.

    For more information about uploading the Policy package to an AWS EFS, refer to the *step 5* in the section *Creating and Exporting the Ruleset*.

> **Important:**
> Ensure that the user has the required permissions to upload an object to the AWS S3 bucket or persistent volume.

6.2.6 Importing the CoP Sample on the ESA

The user needs to run the following steps to import the *Sample_App_COP_V5.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_V5.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 AWS S3 bucket or EFS.

For more information about uploading the CoP package to an AWS S3 bucket, refer to the *step 13* in the section *Creating and Exporting the Ruleset*.

For more information about uploading the CoP package to an AWS EFS, 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 AWS S3 bucket or persistent volume.

6.2.7 Creating Immutable Metadata Packages (IMP)

For Release 1

Set the following values for creating the IMP packages for release 1:

- policy/filepath: test/release1/policy
- cop/filepath: test/release1/cop

6.2.8 Deploy Release 1

1. Ensure that the *Values.yaml* file has the following configuration set on the Linux node.

```
blueDeployment:
  enabled: true
  securityConfigSource: ObjectStore
  policy:
    filepath: test/release1/policy
...
...

dsgService:
  tunnels:
     - name: "service1"
       port: 9443
       targetPort: 9443

ingress:
...
...
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
...
...
  hosts:
...
...
    - host: "prod.example.com"
       httpPaths:
       - port: 9443
         prod: "/"
     - host: "staging.example.com"
       httpPaths:
       - port: 9443
         staging: "/"
```

2. Deploy Release 1 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*.

6.2.9 Run 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.

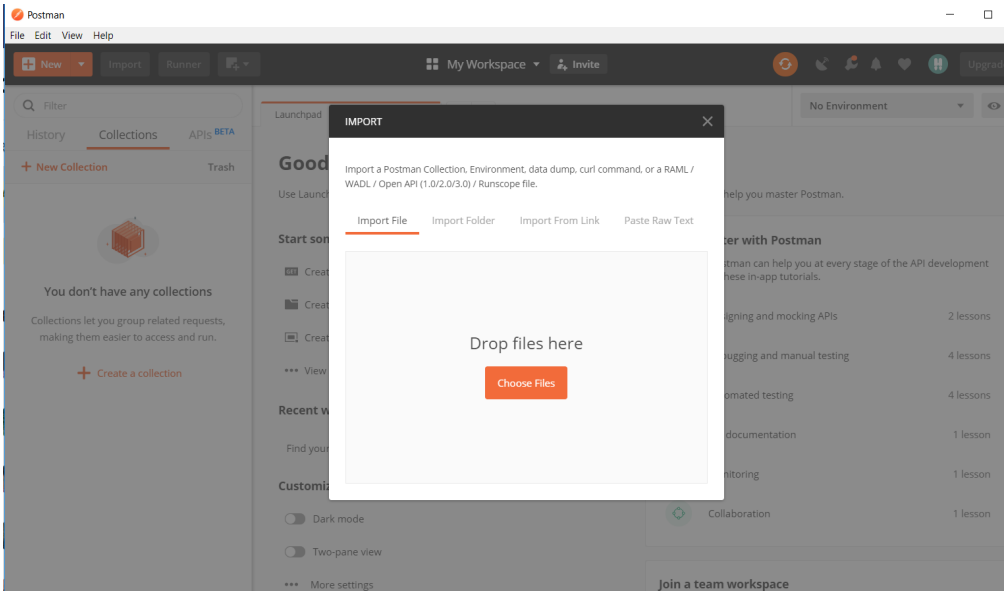
Header - DCOP request

The header is populated with the DCOP request, which protects data using the following Data Elements.

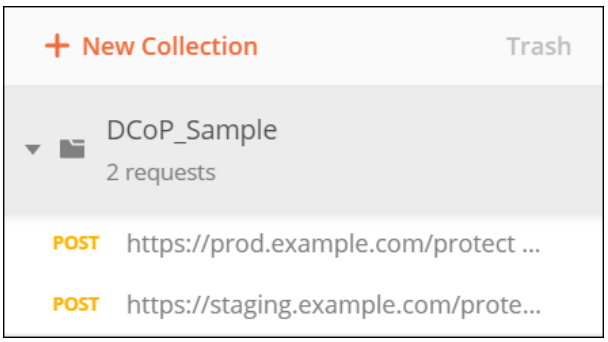
Field	Column Name	Token Type	Data Element Name
First Release			
First Name	first name	Alphanumeric	deFirstName
Last Name	last name	Alphanumeric	deLastName
Street	street address	Alphanumeric	deStreet
City	City	Alphanumeric	deCity
State	State Name	Alpha	deState
Zip Code	zip code	Alphanumeric	deZipcode
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.

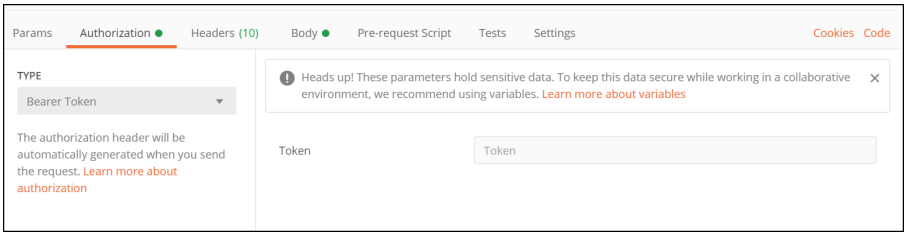


Figure 6-12: Authorization Tab

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 ID 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.

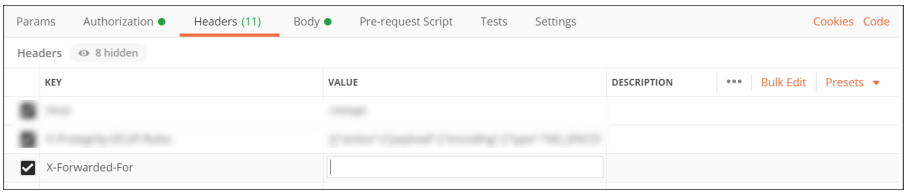


Figure 6-13: X-Forwarded-For Key

7. In the */etc/resolv.conf* file in Linux, specify the IP address of the Inbound Endpoint that you have created in *step 3* of the section *Appendix D: Creating a DNS Entry for the ELB Hostname in Route53*.

   For more information information about creating a host name in the Route53 service, refer to the section *Appendix D: Creating a DNS Entry for the ELB Hostname in Route53*.

8. Select **Release 1** in DCoP_Sample and click **Send**.

   User should get response as successful *200 OK* and receive protected Data.

For protecting data with Release 2

1. Add the data element for *Customer ID* on the ESA.
2. Create the IMP package R2.

For Release 2

Set the following values for creating the IMP package for release 2:

```
# policy and cop metadata configuration
job:
  ...
  ...
  securityConfigStorage: ObjectStore
  policyCOP:
    filepath: test/release2/policy
  cop:
    filepath: test/release2/cop
```

1. Run helm upgrade for switching to Release 2.

   For more information about upgrading the Helm Charts, refer to the section *Upgrading the Helm Charts*.

2. Ensure that the *values.yaml* file has the following configurations set on the Linux node.

```
greenDeployment:
    enabled: true
    securityConfigSource: ObjectStore
  policy:
   filepath: test/release2/policy
...
...

dsgService:
  tunnels:
     - name: "service1"
       port: 9443
       targetPort: 9443

ingress:
...
...
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
...
...
  hosts:
...
...
    - host: "prod.example.com"
       httpPaths:
       - port: 9443
         prod: "/"
     - host: "staging.example.com"
       httpPaths:
       - port: 9443
         staging: "/"
```

3. In the */etc/resolv.conf* file in Linux, specify the IP address of the Inbound Endpoint that you have created in *step 3* of the section *Appendix D: Creating a DNS Entry for the ELB Hostname in Route53*.

   For more information information about creating a host name in the Route53 service, refer to the section *Appendix D: Creating a DNS Entry for the ELB Hostname in Route53*.

4. Select **Release 2** in DCoP_Sample and click **Send**.

   The user should get the response as successful *200 OK* and receive protected data.

6.2.10 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 <Ingress 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.

Chapter 7

Appendix C: Creating a DNS Entry for the ELB Hostname in Route53

This section describes the steps to configure hostnames specified in the *values.yaml* file of the Sample Application Helm chart for resolving the hostname of the Elastic Load Balancer (ELB) that is created by the NGINX INgress Controller.

1. Configure Route53 for DNS resolution.
   - Create a private hosted zone in the Route53 service.
   - In our case, the domain name for the hosted zone is *protegrity.com*.
   - Select the VPC where the Kubernetes cluster is created.
2. Create a hostname for the ELB in the private hosted zone created in step 1.
   - Create a Record Set with type A - Ipv4 address.
   - Select Alias as *yes.*
   - Specify the Alias Target to the ELB created by the Nginx Ingress Controller.
3. Save the record Create Inbound endpoint for DNS queries from a network to the hosted VPC used in Kubernetes.
   - Select Configure endpoints in the Route53 Resolver service.
   - Select *Inbound Only* endpoint.
   - Give a name to the endpoint.
   - Select the VPC used in the Kubernetes cluster and Route53 private hosted zone.
   - Select the availability zone as per the subnet.
   - Review and create the endpoint.
   - Note the IP addresses from the Inbound endpoint page.

     In Linux, specify this IP address in the */etc/resolv.conf* file in *step 2* of the section *Running Security Operations*.

     In Windows, specify this IP address in the DNS Server Settings in *step 2* of the section *Running Security Operations*.

   - Send CURL request to the hostname created using the Route 53 service.

For more information about Amazon Route53, refer to the *Amazon Route53* documentation.