Protegrity Application Protector Java Immutable Policy User Guide for AWS Gen2 9.1.0.0

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 problems that a company faces while protecting data:

- Protegrity customers are moving to the cloud. This includes data and workloads in support of transactional application 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 autoscaling 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 Application Protector Java form factor:
  - The delivery form factor for cloud deployments is an SDK and a supporting Dockerfile. Customers can use this Dockerfile to build an AP Java Container, which is based on the Application Protector form factor that Protegrity been delivering for several years.
  - The AP Java Container is a standard Docker Container that is familiar and expected in cloud deployments. Customers can create their own container image by integrating their applications with the AP Java libraries.
  - The AP Java Container is a lightweight deployment of the AP Java.
- Immutable Deployment:
  - Cloud deployments or containers do not change dynamically – they are immutable. In other words, when there is a change in Policy, the change is carried out by terminating the existing AP Java Container with the Policy and instantiating a new one.
  - Changes to Policy or the AP Java 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 set up to autoscale based on setting a configuration on CPU thresholds.

  - Kubernetes will start with an initial set of Protegrity Pods. These will increase when the CPU threshold is passed, and they will be shrinked when the CPU threshold is under the acceptable limits. This is automatic and hence gives the agility and scalability that is so desired with cloud solutions. Similarly, the nodes on which the pods are run also autoscale 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 autoscale, when needed.

1.3 Architecture and Components

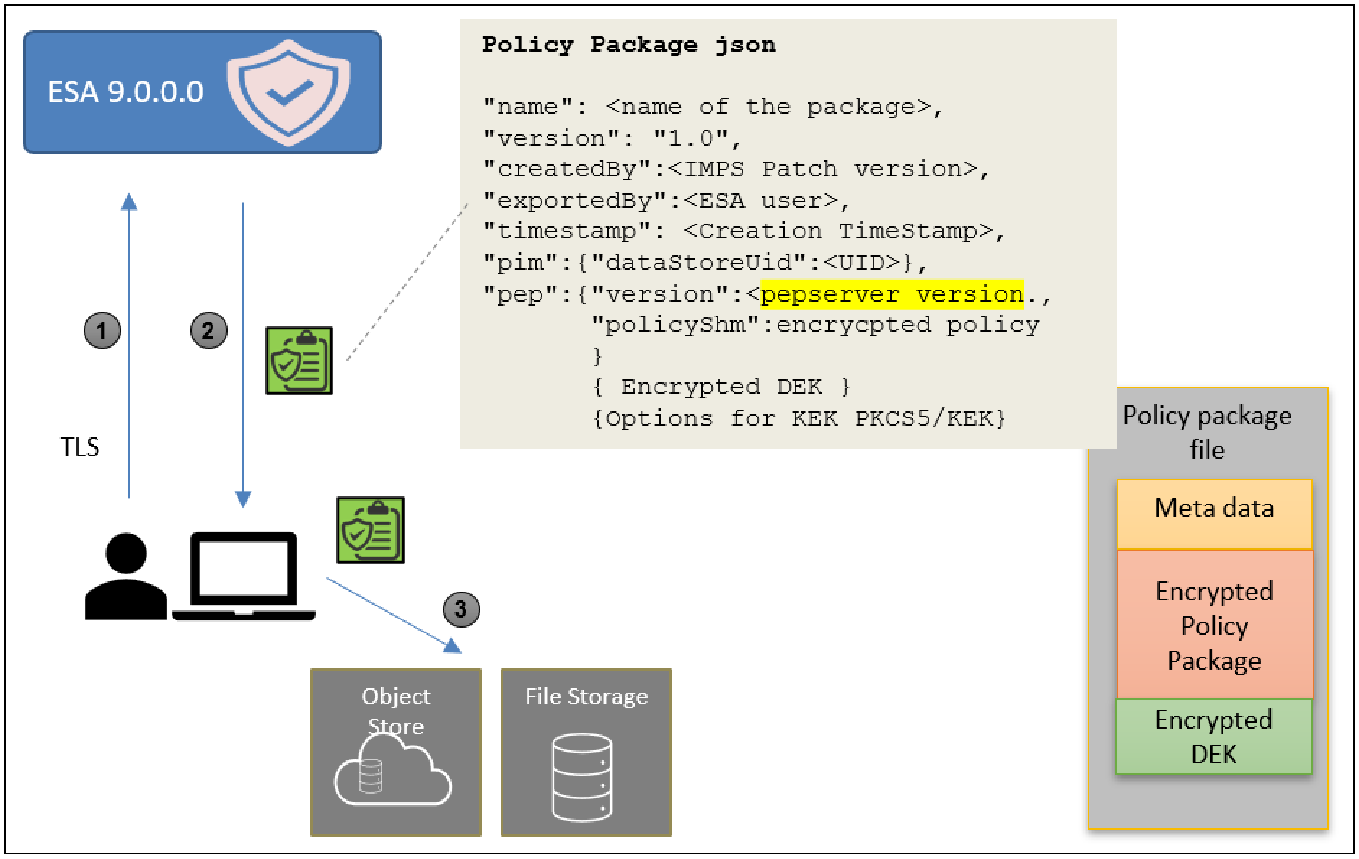
This section will describe the architecture, the individual components, and the workflow of the Protegrity Immutable Application Protector solution.

The IAP Gen2 deployment consists of the following two stages:

- Stage 1: Running a cURL command to fetch the ESA policy and create the policy snapshot.
- Stage 2: Deploying the Sample Application on a Kubernetes cluster, which uses the policy snapshot created in stage 1.

Note: In the production environment, you will deploy the Customer Application instead of the Sample Application.

The following describes the architecture diagram and the steps for Stage 1 - Retrieving the policy from the ESA.



```
Policy Package json

"name": <name of the package>,
"version": "1.0",
"createdBy":<IMPS Patch version>,
"exportedBy":<ESA user>,
"timestamp": <Creation TimeStamp>,
"pim":{"dataStoreUid":<UID>},
"pep":{"version":<pepserver version.,
       "policyShm":encrycpted policy
       }
       { Encrypted DEK }
       {Options for KEK PKCS5/KEK}
```

Figure 1-1: Stage 1: Fetching the ESA Policy and Creating the Policy Package

The following steps describe the workflow of retrieving the policy from the ESA.

1. The user sends the IMPS API request to the ESA to create a policy package. You can choose to encrypt the policy package using a Passphrase-Based Encryption or a Key Management System (KMS). You need to specify the encryption parameters as part of the IMPS API request body.
2. The ESA generates a JSON file for the policy package, which contains metadata and the encrypted policy package.
3. The user needs to upload the policy package to an Object Store or a File Store.

The following describes the architecture diagram and the steps for Stage 2 - Deploying the Sample Application on a Kubernetes cluster.

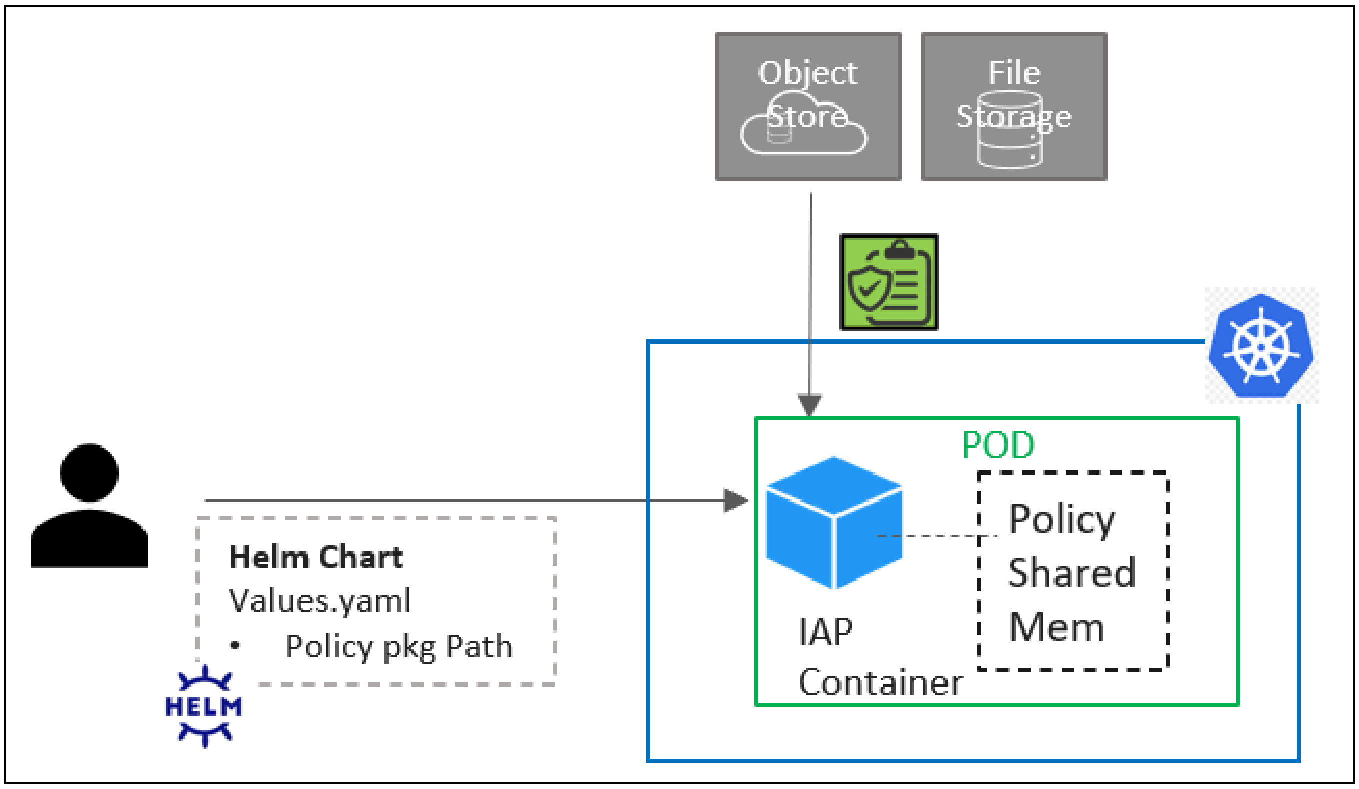


Figure 1-2: Stage 2: Deploying the Sample Application Container

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

1. The user runs the Helm chart to deploy the Sample Application to the Kubernetes cluster. The Sample Application is integrated with the Application Protector Java libraries.

   > **Note:** The Sample Application has been used for demonstration purposes. You can choose to create a custom image by integrating your custom application with the Application Protector Java libraries.

2. The Sample Container reads the immutable policy snapshot from the AWS S3 bucket or a persistent volume, decrypts it using the Kubernetes secret configured for Passphrase-Based Encryption or KMS-related secrets, and stores the policy in the shared memory of the Pod.
3. The Client Application sends a request to the Sample application over REST for protecting, unprotecting, and reprotecting data.
4. The Sample Application internally sends the request to the Application Protector Java, and then returns the protected, unprotected, or reprotected value to the client application.

Chapter 2

Immutable Application Protector (IAP) Deployment Model

This section describes the IAP Reference Deployment model that customers can use to deploy the Sample Application containers on a Kubernetes cluster.

2.1 Verifying the Prerequisites

Ensure that the following prerequisites are met:

Reference Solution Deployment

- *IAP_RHUBI-ALL-64_x86-64_ALL.K8S.JRE-1.7_<IAP_version>.tgz* – Installation package for the IAP Gen2 Deployment. This package contains the following packages:
  - *APJavaIMSetup_<OS>_x64_<App_Protector_version>.tgz* - Immutable AP Java installation package.
  - *IAP-SAMPLE-APP-HELM_ALL-ALL-ALL_x86-64_K8_<version>.tgz* – Package containing the Helm charts, which are used to deploy the sample application on the Kubernetes cluster.
  - *IAP-SAMPLE-APP_SRC_<version_number>.tgz* - Package containing the Dockerfile that can be used to create a custom image for the Sample Application container and the associated binary files.

    For more information about building a custom image using the Dockerfile, refer to the section *Creating Custom Images Using Dockerfile*.
  - *PolicyUtil_Linux_x64_<version>.tgz* - Binary file containing the scripts used to retrieve the policy package from the ESA.
  - *HOW-TO-BUILD* - Text file specifying how to build the Sample Application and Docker images.
  - *manifest.json* - File specifying the name of the product and its components, the version number of the protector, and the build number of product.

Reference Application

This section describes the prerequisites for using the reference application:

- Policy – Ensure that you have defined the security policy in the ESA 9.1.0.5.

For more information about defining a security policy, refer to the section *Creating and Deploying Policies* in the *Policy Management Guide 9.1.0.5*.

- Trusted Application - Create a Trusted Application with the name as *com.protegrity.sample.apjavarest.APJavaSpringApp* and user name as *ptyituser*.

  Note: If you are deploying a Customer Application container, then you must specify the application name and user name, which is defined in the Customer Application container image, that is used to run the Application Protector Java.

  Important: If you are deploying a Customer Application container, then you must apply the NSS wrapper, which is a *nss-wrapper* library for the Name Service Switch (NSS) API, to the Customer Application container image.

  For more information about applying the NSS wrapper to the Customer Application container image, refer to the section *Appendix C: Applying the NSS Wrapper to the Customer Application Container Image*.

  For more information about the *nss_wrapper* library, refer to the section *The nss_wrapper library*.

  For more information about setting up a Trusted Application, refer to the section *Working with Trusted Applications* in the *Policy Management Guide 9.1.0.5*.

- ESA user - Create an ESA user that will be used to invoke the IMP REST API for retrieving the security policy and the certificates from the ESA 9.1.0.5. Ensure that the user is assigned the *IMP Export* role.

  For more information about assigning roles, refer to the section *Managing Roles* in the *Appliances Overview Guide 9.1.0.5*.

Additional Prerequisites

The following additional prerequisites are required:

- *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.
- *Linux instance* - This additional Linux instance is used to install Splunk Enterprise, in case you want to view the Application Protector and Container logs on Splunk. This instance can be on-premise or on AWS.
- Access to an AWS account.
- AWS Elastic File System (EFS), if you want to use AWS EFS for uploading the immutable policy snapshot, instead of using AWS S3.
- Install the latest version of the *EFS-CSI* driver, which is required if you are using AWS EFS as the persistent volume.

  For more information about installing the EFS-CSI driver, refer to the *Amazon EFS CSI driver* documentation.

- AWS S3 buckets for uploading the immutable policy snapshot. This prerequisite is optional, and is required only if you want to use AWS S3 for storing the policy snapshot, instead of AWS EFS.
- *KMSDecryptAccess* - This is a custom policy that allows the user to decrypt the policy packages that have been encrypted using AWS Customer Master Key. The following action must be permitted on the IAM service:

  ```
  kms:Decrypt
  ```

- *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.

- IAM User 1 - Required to create the Kubernetes cluster. This user requires the following permissions:
  - *AmazonEC2FullAccess* - This is a managed policy by AWS
  - *AmazonEKSClusterPolicy* - This is a managed policy by AWS
  - *AmazonEKSServicePolicy* - This is a managed policy by AWS
  - *AWSCloudFormationFullAccess* - This is a managed policy by AWS
  - Custom policy that allows the user to create a new role and an instance profile, retrieve information about 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
  - Custom policy that allows the user to delete a role and an instance profile, detach a policy from a specified role, delete a policy from the specified role, remove an IAM role from the specified EC2 instance profile, and so on. The following actions must be permitted on the IAM service:
    - GetOpenIDConnectProvider
    - CreateOpenIDConnectProvider
    - DeleteInstanceProfile
    - DeleteRole
    - RemoveRoleFromInstanceProfile
    - DeleteRolePolicy
    - DetachRolePolicy
    - PutRolePolicy
  - Custom policy that allows the user to manage EKS clusters. The following actions must be permitted on the EKS service:
    - ListClusters
    - ListNodegroups
    - ListTagsForResource
    - ListUpdates
    - DescribeCluster
    - DescribeNodegroup
    - DescribeUpdate
    - CreateCluster
    - CreateNodegroup
    - DeleteCluster
    - DeleteNodegroup
    - UpdateClusterConfig
    - UpdateClusterVersion
    - UpdateNodegroupConfig
    - UpdateNodegroupVersion
  - Permission to create a bucket on AWS S3. This prerequisite is optional, and is required only if you want to use AWS S3 for storing the policy snapshot, instead of AWS EFS.
  - Permission to create a Kubernetes cluster.

- Permission to create a persistent volume in the Kubernetes cluster.

  For more information about persistent volumes and persistent volume claims, refer to the section *Persistent Volumes* in the Kubernetes documentation.

For more information about creating an IAM user, refer to the section *Creating an IAM User in Your AWS Account* in the AWS documentation. Contact your system administrator for creating the IAM users.

For more information about the AWS-specific permissions, refer to the API Reference document for *AWS S3* and *Amazon EKS*.

- IAM User 2 - This user requires the following permissions:
  - Access to AWS Elastic Container Registry (ECR) to upload the Sample Application container image.
  - Access to Route53 for mapping the hostname of the Elastic Load Balancer to a DNS entry in the Amazon Route53 service. This is required if you are terminating the TLS connection from the client application on the Load Balancer.

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

  - *AmazonEC2FullAccess* permission to create and manage Linux host instances and also to mount persistent volume and check the policy snapshot
  - AWS S3 viewing permissions to check the policy snapshot that has been created. This permission is required if you want to the store the policy snapshot on AWS S3.
  - AWS EFS viewing permissions to check the policy snapshot that has been created. This permission is required if you want to the store the policy snapshot on AWS EFS.
  - Permission to create a persistent volume claim in the Kubernetes cluster.

    For more information about persistent volumes and persistent volume claims, refer to the section *Persistent Volumes* in the Kubernetes documentation.

2.2 Downloading the Installation Package

This section describes the steps to download the installation package for the IAP Reference Deployment model.

To download the installation package:

1. Download the *IAP_RHUBI-ALL-64_x86-64_ALL.K8S.JRE-1.7_<IAP_version>.tgz* file on the Linux instance.
2. Run the following command to extract the files from the *IAP_RHUBI-ALL-64_x86-64_ALL.K8S.JRE-1.7_<IAP_version>.tgz* file.

   ```
   tar -xvf IAP_RHUBI-ALL-64_x86-64_ALL.K8S.JRE-1.7_<IAP_version>.tgz
   ```

   The following files are extracted:
   - *APJavaIMSetup_<OS>_x64_<App_Protector_version>.tgz*
   - *IAP-SAMPLE-APP-HELM_ALL-ALL-ALL_x86-64_K8_<version>.tgz*
   - *IAP-SAMPLE-APP_SRC_<version_number>.tgz*
   - *PolicyUtil_Linux_x64_<version>.tgz*
   - *HOW-TO-BUILD*
   - *manifest.json*

2.3 Initializing the Linux Instance

This section describes how you can initialize the Linux instance.

To initialize the Linux instance:

1. Install the AWS CLI, which provides a set of command line tools for the AWS Cloud Platform.

   For more information about installing the AWS CLI on the Linux instance, refer to the section *Installing the AWS CLI* in the AWS documentation.
2. 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.
3. Enter the required details as shown in the following snippet.

   ```
   AWS Access Key ID [None]: <AWS Access Key ID of IAM User 1>
   AWS Secret Access Key [None]: <AWS Secret Access Key of IAM User 1>
   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 allow the user to create the Kubernetes cluster.

   For more information about the IAM User 1, refer to the section *Additional Prerequisites*.
4. Install *eksctl*, which is a command line utility to create and manage Kubernetes clusters on Amazon Elastic Kubernetes Service (Amazon EKS).

   For more information about installing *eksctl* on the Linux instance, refer to the section *Getting started with eksctl* in the AWS documentation.
5. Install Kubectl, which is the 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* in the AWS documentation.
6. Install *aws-iam-authenticator*, which is a tool that uses your IAM credentials to authenticate to your Kubernetes cluster.

   For more information about installing *aws-iam-authenticator*, refer to the section *Installing aws-iam-authenticator* in the AWS documentation.
7. Run the following commands sequentially to install the Helm client on the Linux instance.

   ```
   curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/
   scripts/get-helm-3

   chmod 700 get_helm.sh

   ./get_helm.sh
   ```

For more information about installing a Helm client, refer to the section *Installing Helm* in the Helm documentation.

Important: Verify the version of the Helm client that must be used from the Readme document provided with this build.

You can verify the version of the Helm client installed by running the following command.

```
helm version --client
```

8. Run the following command to extract the *.tgz* package containing the Helm charts for deploying the Sample Application container.

```
tar -xvf <Helm chart package>
```

For example:

```
tar -xvf IAP-SAMPLE-APP-HELM_ALL-ALL-ALL_x86-64_K8S_<version>.tgz
```

The extracted package contains the *spring-apjava* directory, which has the following contents:

- *Chart.yaml* – A YAML file that provides information about the chart
- *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.
- *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.

2.4 Creating Certificates and Keys for TLS Authentication

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

Note:

If you already have a server that has been signed by a trusted third-party Certificate Authority (CA), then you do not need create a self-signed server 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 iap-ca.key -out iap-ca.crt -days
356 -nodes -subj '/CN=IAP Certificate Authority'
```

Note:

If you already have a CA certificate and a private key, then you can skip this step.

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 iap-wildcard.key -out iap-wildcard.csr
   -nodes -subj '/CN=*.example.com'

   openssl x509 -req -sha256 -days 365 -in iap-wildcard.csr -CA iap-ca.crt -CAkey iap-
   ca.key -set_serial 04 -out iap-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 iap-client.key -out iap-client.csr -nodes
   -subj '/CN=My IAP Client'

   openssl x509 -req -sha256 -days 365 -in iap-client.csr -CA iap-ca.crt -CAkey iap-
   ca.key -set_serial 02 -out iap-client.crt
   ```

4. Copy all the certificates to a common directory.

   For example, create a directory named *iap-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 Sample Application images to the Container Repository.

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

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

Note: Before deploying the Customer Application container, ensure that you have applied the NSS wrapper to the Customer Application container image.

For more information about applying the NSS wrapper to the Customer Application container image, refer to the section *Appendix C: Applying the NSS Wrapper to the Customer Application Container Image*.

➤ 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 to 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 to Amazon ECR, refer to the *AWS CLI Command Reference* documentation.

3. Create a custom image for the Sample Application container.

   For more information about creating a custom image, refer to the section *Creating Custom Images Using Dockerfile*.
4. Upload the Sample Application container image to Amazon ECR by performing the following steps.
   a. Run the following command to list the Sample Application container image.
      ```
      docker images
      ```
   b. Tag the image to the Amazon ECR by running the following command.
      ```
      docker tag <Container image>:<Tag> <Container registry path>/<Container
      image>:<Tag>
      ```
      For example:
      ```
      docker tag spring-
      app:v9.1 <aws_account_id>.dkr.ecr.us-east-1.amazonaws.com/iap:IAP-Sample-
      Application-POLICY_AWS.K8S-<Kubernetes_version>-64_x86-64_<version>
      ```
      For more information about tagging an image, refer to the *AWS* documentation.
   c. Push the tagged image to the Amazon ECR by running the following command.
      ```
      docker push <Container registry path>/<Container image>:<Tag>
      ```
      For example:
      ```
      docker push <aws_account_id>.dkr.ecr.us-east-1.amazonaws.com/iap:IAP-Sample-
      Application-POLICY_AWS.K8S-<Kubernetes_version>-64_x86-64_<version>
      ```
5. Navigate to the directory where you have extracted the Helm charts packages for the Sample Application container.
6. In the *values.yaml* file, update the required path for the *springappImage* settings, with the tag.

2.6 Creating the Cloud Runtime Environment

This section describes how to create the Cloud runtime environment.

2.6.1 Creating the AWS Runtime Environment

This section describes how to create the AWS runtime environment.

Prerequisites

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

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

Audience

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

- Introduction to AWS S3
- Introduction to AWS Cloud Security

- Introduction to AWS EKS

2.6.1.1 Logging in to the AWS Environment

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

To login to the AWS environment:

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

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

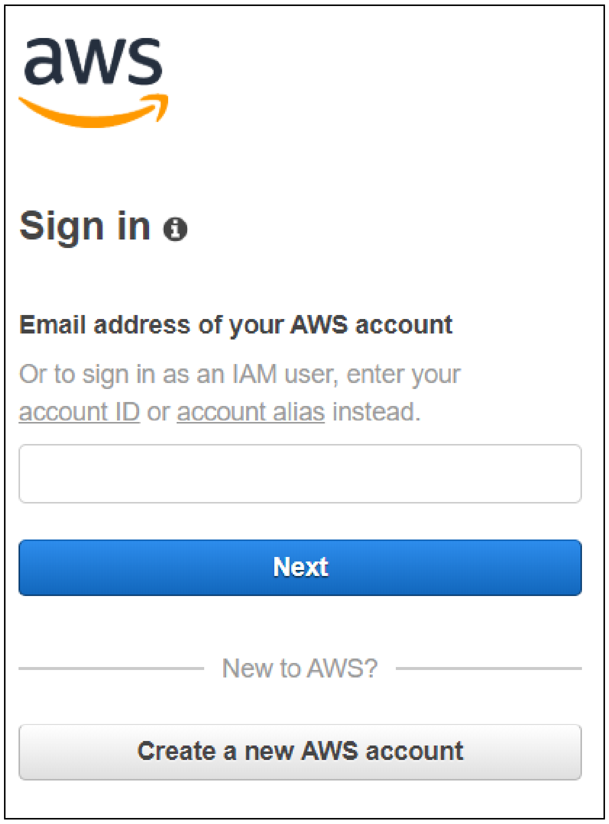
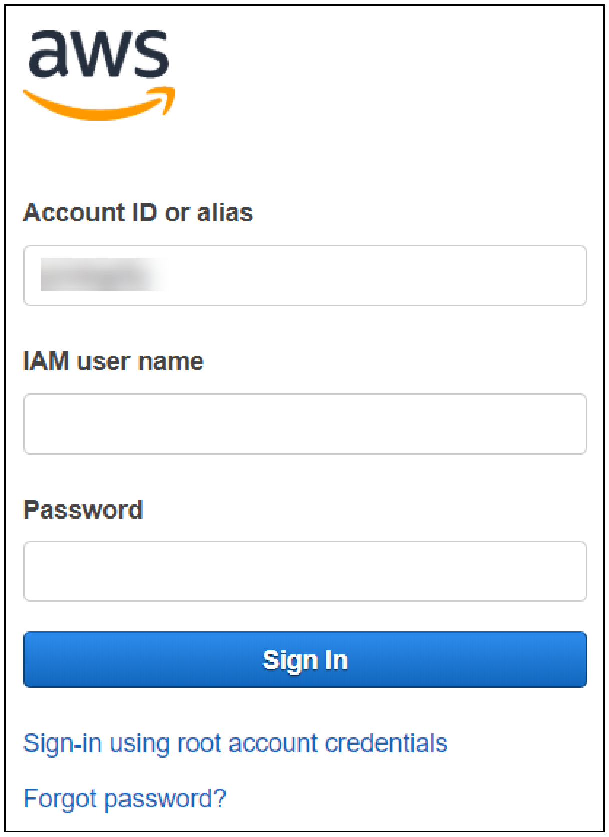
   
   aws
   Sign in
   Email address of your AWS account
   Or to sign in as an IAM user, enter your account ID or account alias instead.
   Next
   New to AWS?
   Create a new AWS account

   Figure 2-1: AWS Sign in screen
3. Enter the account ID or alias that has been provided to you by your administrator, and then click **Next**.
   A screen appears that prompts you to enter the IAM user credentials for signing into the AWS Management Console.
4. Enter the following details:
   - Identity and Access Management (IAM) user name
   - Password



Account ID or alias

[redacted]

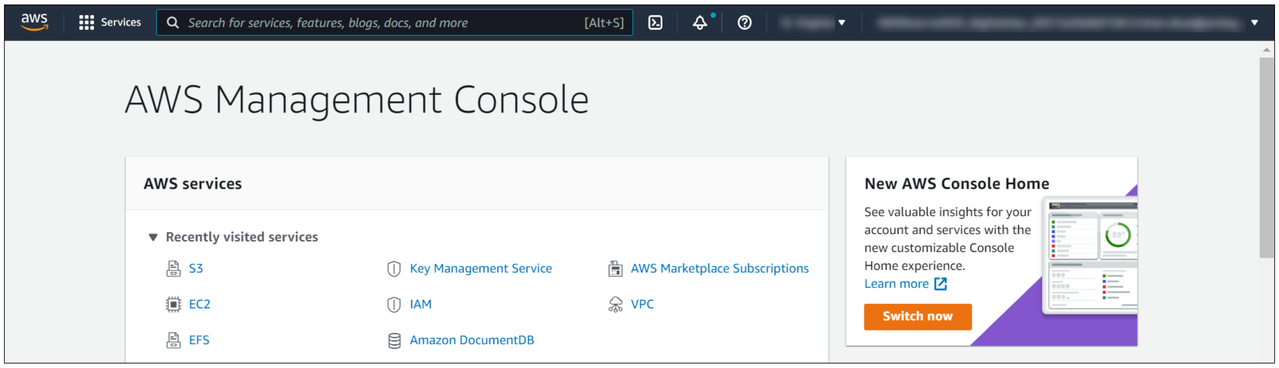
IAM user name

Password

Figure 2-2: AWS login screen

5. Click **Sign in**.

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



Search for services, features, blogs, docs, and more [Alt+S]

[redacted]

AWS Management Console

AWS services

▼ Recently visited services

S3	Key Management Service	AWS Marketplace Subscriptions
EC2	IAM	VPC
EFS	Amazon DocumentDB	

New AWS Console Home

See valuable insights for your account and services with the new customizable Console Home experience.

Learn more

Figure 2-3: AWS Management Console Screen

2.6.1.2 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.

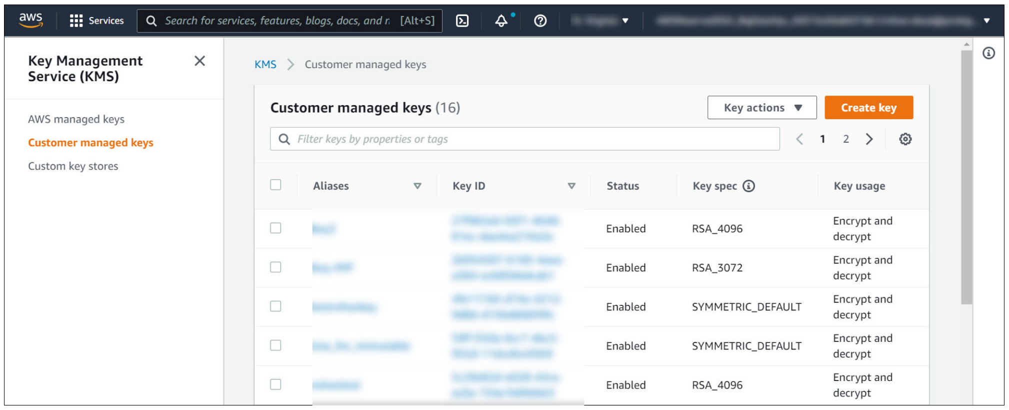
   For more information about logging in to the AWS environment, refer to the section *Logging in to the AWS Environment*.

2. Navigate to **Services.**

   A list of AWS services appears.

3. In **Security, Identity, & Compliance**, click **Key Management Service**.

   The AWS **Key Management Service (KMS)** console opens. By default, the **Customer managed keys** screen appears.



aws Services Search for services, features, blogs, docs, and r [Alt+S] [redacted]

Key Management Service (KMS)

AWS managed keys
Customer managed keys
Custom key stores

KMS > Customer managed keys

Customer managed keys (16) Key actions ▼ Create key

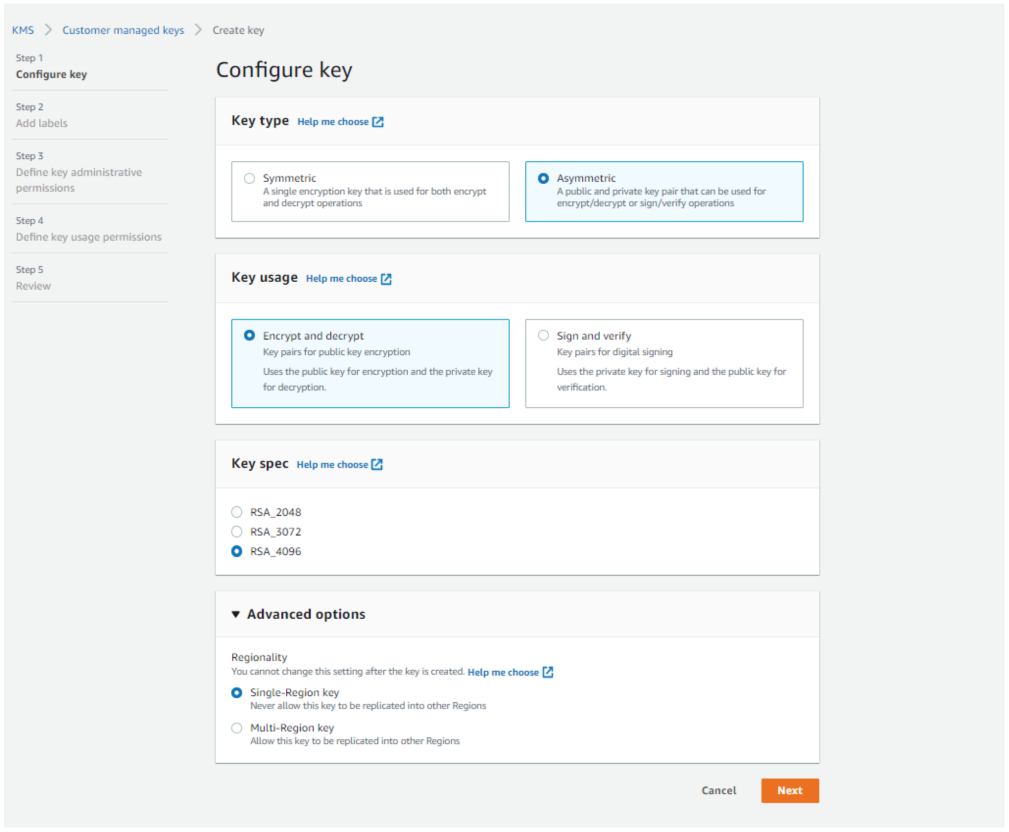
Filter keys by properties or tags < 1 2 >

	Aliases	Key ID	Status	Key spec	Key usage
☐	[redacted]	[redacted]	Enabled	RSA_4096	Encrypt and decrypt
☐	[redacted]	[redacted]	Enabled	RSA_3072	Encrypt and decrypt
☐	[redacted]	[redacted]	Enabled	SYMMETRIC_DEFAULT	Encrypt and decrypt
☐	[redacted]	[redacted]	Enabled	SYMMETRIC_DEFAULT	Encrypt and decrypt
☐	[redacted]	[redacted]	Enabled	RSA_4096	Encrypt and decrypt

Figure 2-4: Customer Managed Keys Screen

4. Click **Create key**.

   The **Configure key** screen appears.



KMS > Customer managed keys > Create key

Step 1
Configure key

Step 2
Add labels

Step 3
Define key administrative permissions

Step 4
Define key usage permissions

Step 5
Review

Configure key

Key type Help me choose

○ Symmetric
A single encryption key that is used for both encrypt and decrypt operations

● Asymmetric
A public and private key pair that can be used for encrypt/decrypt or sign/verify operations

Key usage Help me choose

● Encrypt and decrypt
Key pairs for public key encryption
Uses the public key for encryption and the private key for decryption.

○ Sign and verify
Key pairs for digital signing
Uses the private key for signing and the public key for verification.

Key spec Help me choose

○ RSA_2048
○ RSA_3072
● RSA_4096

▼ Advanced options

Regionality
You cannot change this setting after the key is created. Help me choose

● Single-Region key
Never allow this key to be replicated into other Regions

○ Multi-Region key
Allow this key to be replicated into other Regions

Cancel Next

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.

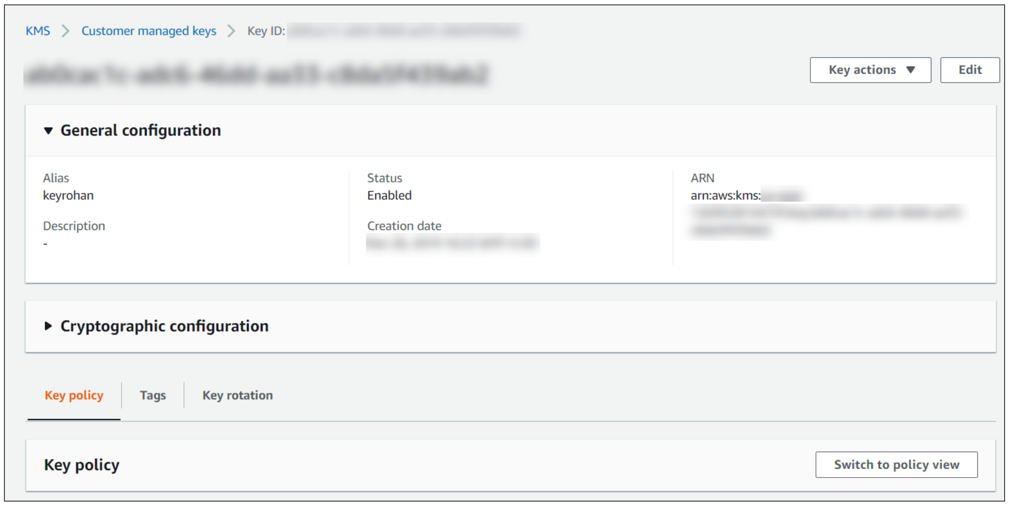
9. Click **Next**.

   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.



Alias
keyrohan

Description
-

Status
Enabled

Creation date
[redacted]

ARN
arn:aws:kms:[redacted]

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>"
            ]
        }
    ]
}
```

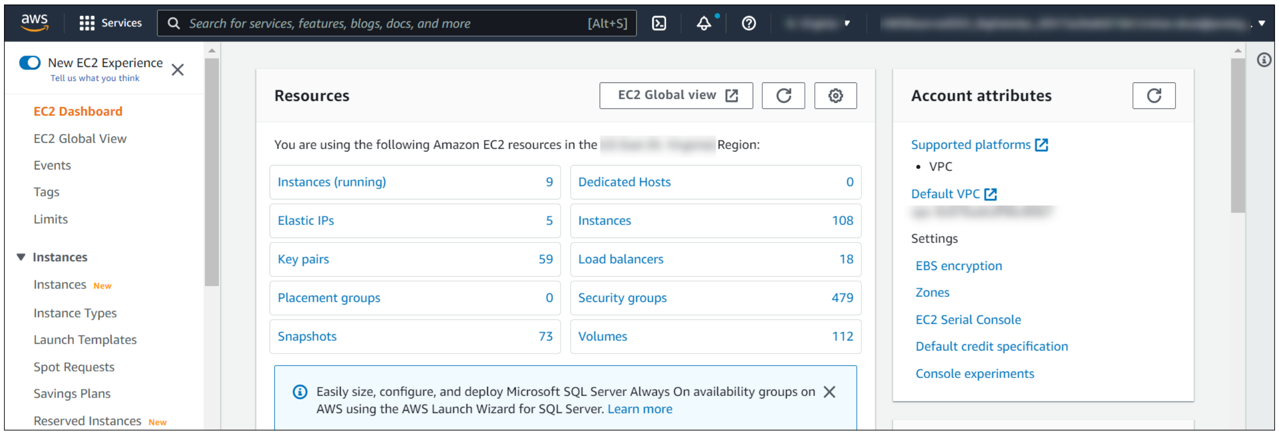
19. Click **Review policy**, and then click **Save changes** to save the changes to the policy.

2.6.1.3 Creating a Key Pair for the EC2 Instance

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

➤ To create a key pair for the EC2 instance:

1. Login to the AWS environment.
   For more information about logging in to the AWS environment, refer to the section *Logging in to the AWS Environment*.
2. Navigate to **Services.**
   A list of AWS services appears.
3. In **Compute**, click **EC2**.
   The **EC2** console opens. By default, the **EC2 Dashboard** screen appears.



Resources

You are using the following Amazon EC2 resources in the [redacted] Region:

Instances (running)	9	Dedicated Hosts	0
Elastic IPs	5	Instances	108
Key pairs	59	Load balancers	18
Placement groups	0	Security groups	479
Snapshots	73	Volumes	112

Easily size, configure, and deploy Microsoft SQL Server Always On availability groups on AWS using the AWS Launch Wizard for SQL Server. Learn more

Account attributes

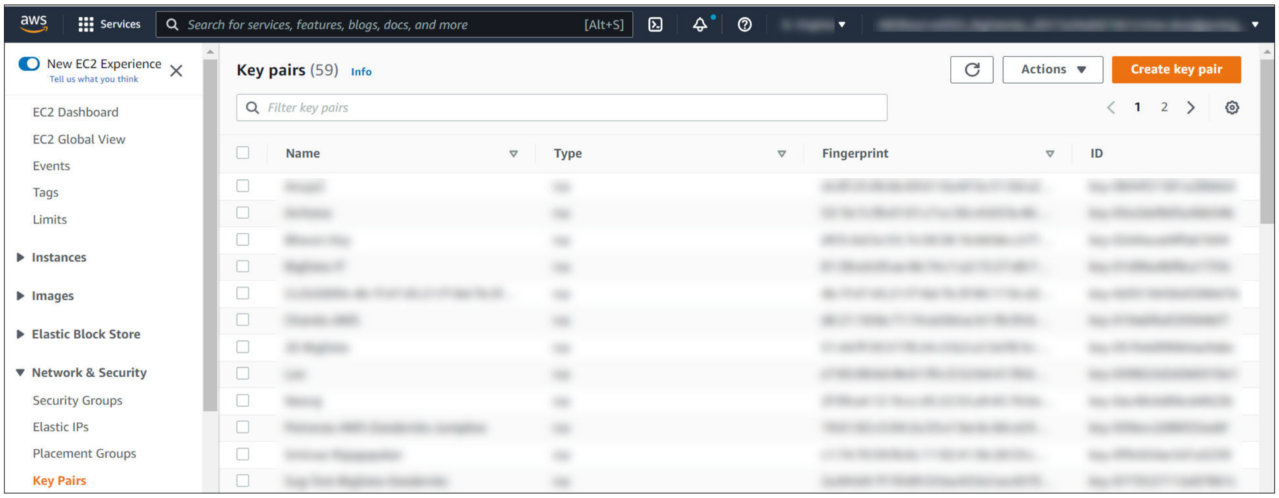
Supported platforms
- VPC

Default VPC
[redacted]

Settings
EBS encryption
Zones
EC2 Serial Console
Default credit specification
Console experiments

Figure 2-6: EC2 Dashboard Screen

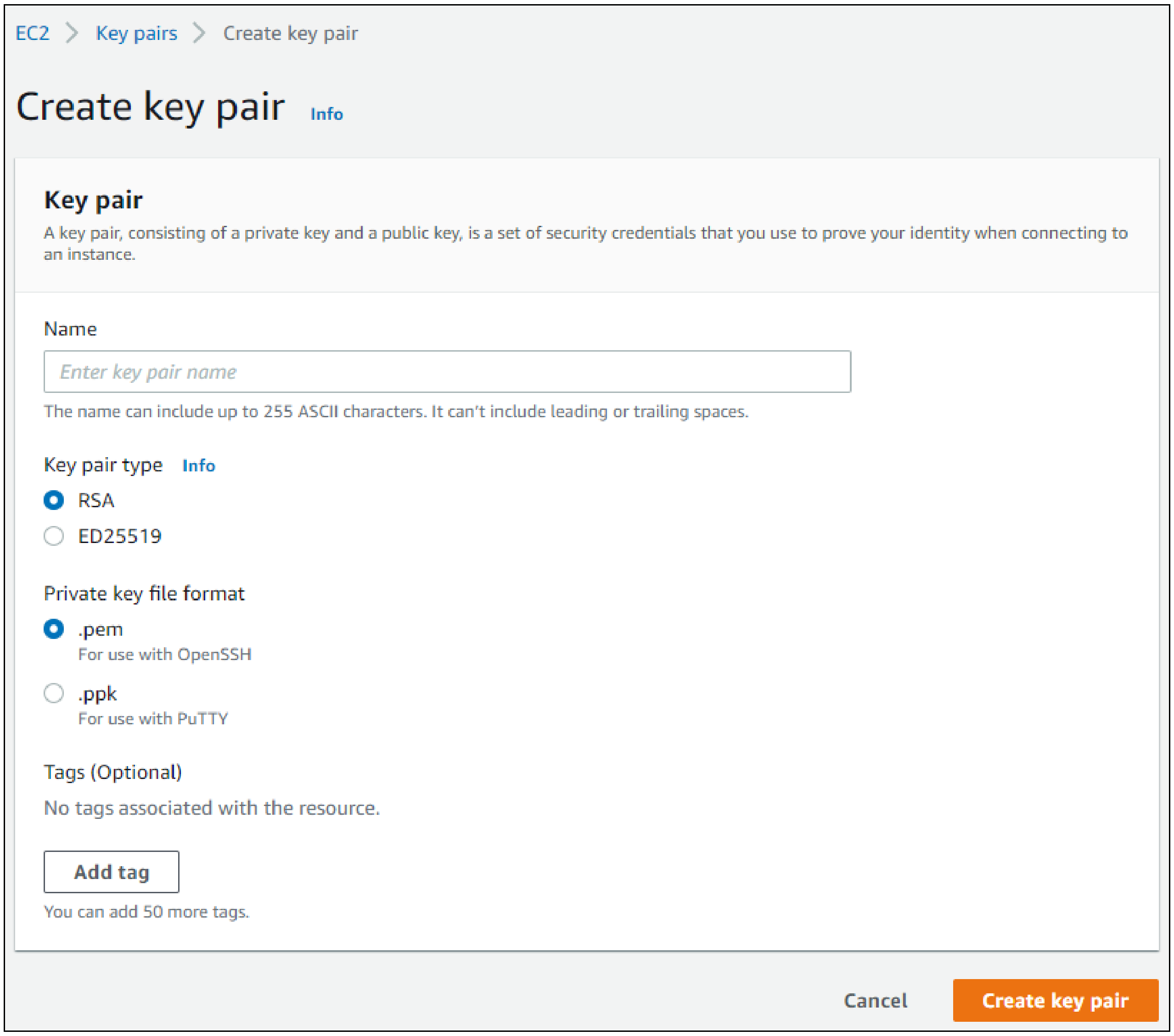
4. On the left pane, click **NETWORK & SECURITY** > **Key Pairs**.
   The **Key pairs** screen appears.



Name	Type	Fingerprint	ID
[redacted]	[redacted]	[redacted]	[redacted]

Figure 2-7: Key Pairs Screen

5. Click **Create key pair**.
   The **Create key pair** screen appears.


EC2 > Key pairs > Create key pair

Create key pair Info

Key pair

A key pair, consisting of a private key and a public key, is a set of security credentials that you use to prove your identity when connecting to an instance.

Name

Enter key pair name

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type Info

[✓ RSA]

ED25519

Private key file format

[✓ .pem]
For use with OpenSSH

.ppk
For use with PuTTY

Tags (Optional)

No tags associated with the resource.

Add tag

You can add 50 more tags.

Cancel　Create key pair

Figure 2-8: 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 **Key pair type** field, select *RSA*.
8. In the **Private key file format** field, specify the format as *pem* for use with OpenSSH.
9. Click **Create key pair**.

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

```
Successfully created key pair
```

2.6.1.4 Creating an AWS S3 Bucket

This section describes how to create an AWS S3 bucket.

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

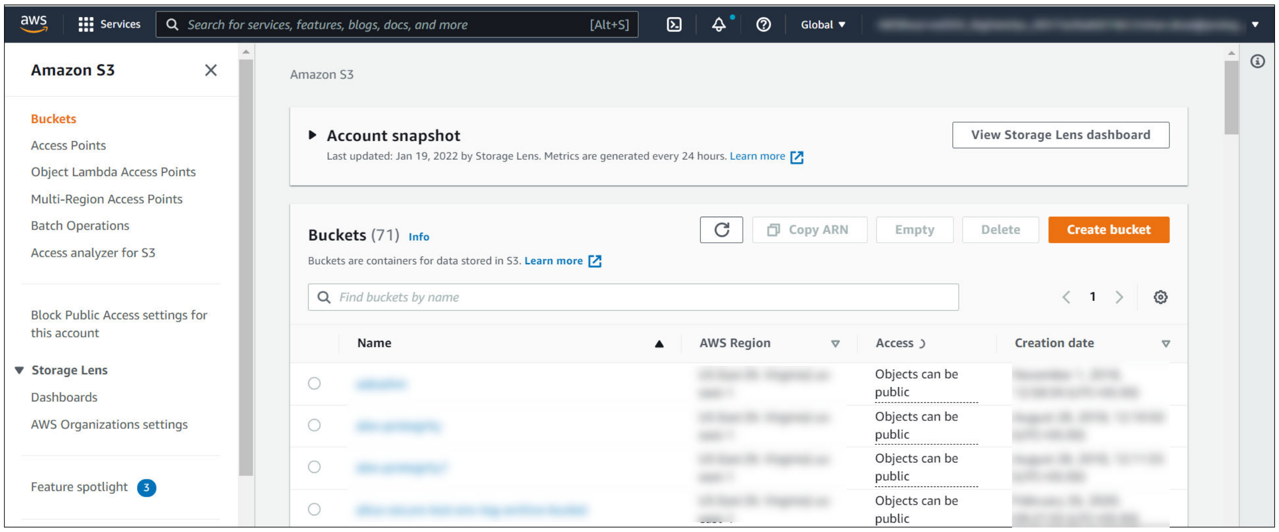
To create an AWS S3 bucket:

1. Login to the AWS environment.

   For more information about logging in to the AWS environment, refer to the section *Logging in to the AWS Environment*.
2. Navigate to **Services.**

   A list of AWS services appears.
3. In **Storage**, click **S3**.

   The **S3 buckets** screen appears.



Last updated: Jan 19, 2022 by Storage Lens. Metrics are generated every 24 hours. Learn more

Buckets are containers for data stored in S3. Learn more

Figure 2-9: S3 Buckets Screen

4. Click **Create bucket**.

   The **Create bucket** screen appears.

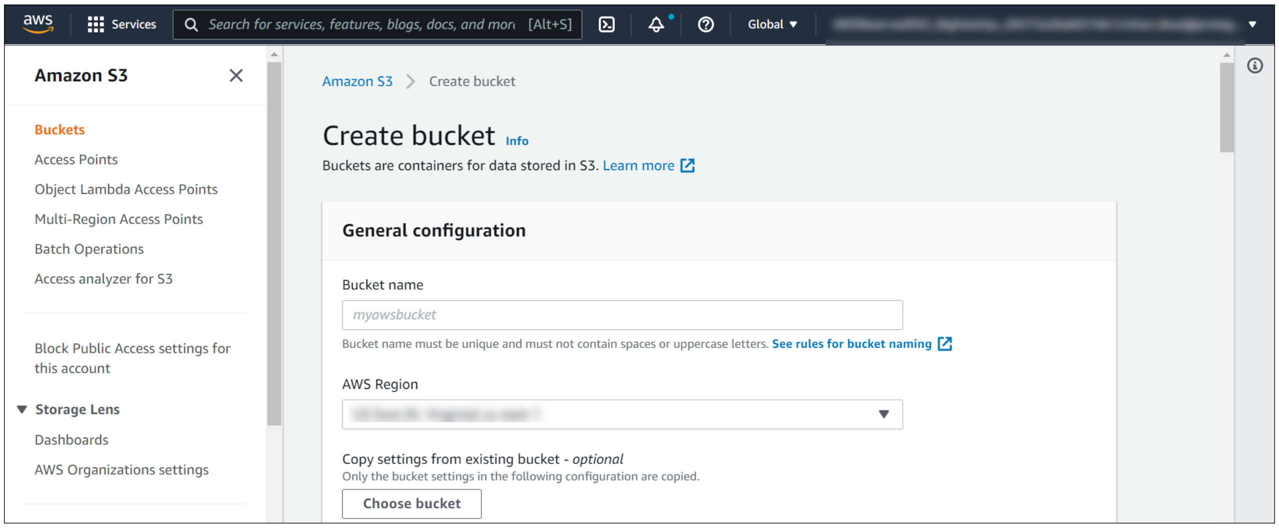

Services
Search for services, features, blogs, docs, and more [Alt+S]
Global
Amazon S3
Buckets
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
Access analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
AWS Organizations settings
Amazon S3 > Create bucket
Create bucket Info
Buckets are containers for data stored in S3. Learn more
General configuration
Bucket name
myawsbucket
Bucket name must be unique and must not contain spaces or uppercase letters. See rules for bucket naming
AWS Region
Copy settings from existing bucket - optional
Only the bucket settings in the following configuration are copied.
Choose bucket

Figure 2-10: Create Bucket Dialog Box

5. In the **General configuration** screen, specify the following details.
   a. In the **Bucket name** field, enter a unique name for the bucket.
   b. In the **AWS Region** field, choose the same region in which you want to create your EC2 instance.

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

   The bucket is created.

2.6.1.5 Creating an AWS EFS

This section describes how to create an AWS EFS.

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

To create an AWS EFS:

1. Login to the AWS environment.

   For more information about logging in to the AWS environment, refer to the section *Logging in to the AWS Environment*.
2. Navigate to **Services.**

   A list of AWS services appears.
3. In **Storage**, click **EFS**.

   The **File Systems** screen appears.
4. Click **Create file system**.

   The **Configure network access** screen appears.
5. In the **VPC** list, select the VPC where you will be creating the Kubernetes cluster.
6. Click **Next Step**.

   The **Configure file system settings** screen appears.
7. Click **Next Step**.

The **Configure client access** screen appears.

8. Click **Next Step**.

   The **Review and create** screen appears.

9. Click **Create File System**.

   The file system is created.

   Note the value in the **File System ID** column. You need to specify this value as the value of the *volumeHandle* parameter in the *pv.yaml* file in *step 10c*.

10. Perform the following steps if you want to use a persistent volume for storing the policy 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: fs-618248e2:/
```

    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 4b*.

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 10a*.

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 10d*.

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 10b*. The value of the *storage* parameter in the *pvc.yaml* 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 10a*.

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 iap
```

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 10g*.

```
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 10c*.

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

2.6.1.6 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 create a file named *createCluster.yaml* to specify the configurations for creating the Kubernetes cluster.

   The following snippet displays the contents of the *createCluster.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 your Kubernetes cluster>: "owned"
    privateNetworking: true
    securityGroups:
      withShared: true
      withLocal: true
      attachIDs: ['<Security group linked to your VPC>']
    ssh:
     publicKeyName: '<EC2 keypair>'
    iam:
      attachPolicyARNs:
        - "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
      withAddonPolicies:
        autoScaler: true
```

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

For more information about the sample configuration file used to create a Kubernetes cluster, refer to the section *Example with custom IAM and VPC config* in the eksctl documentation.

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

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

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

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

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

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

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

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

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

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

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

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

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

   ```
   kubectl get nodes
   ```

   The command lists the Kubernetes nodes available in your cluster.

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
```

After you have created the Kubernetes cluster, you can deploy the Sample Application container with PSP and RBAC, or without PSP and RBAC.

For more information about deploying the containers with PSP and RBAC, refer to the section *Deploying the Containers with a Pod Security Policy (PSP) and Role-Based Access Control (RBAC)*.

For more information about deploying the containers without PSP and RBAC, refer to the section *Deploying the Containers without a PSP and RBAC*.

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.

2.7 Deploying the Containers with a Pod Security Policy (PSP) and Role-Based Access Control (RBAC)

This section describes the steps that you need to perform for deploying the Sample Application container with a Protegrity Pod Security Policy (PSP) and Role Based Access Control (RBAC). A PSP determines the security policies for running a pod.

The user with the *cluster-admin* role creates the RBAC and applies the PSP. This user also creates a Kubernetes service account that will install the Helm charts for deploying the Sample Application container.

Important:
The PSP has been deprecated in Kubernetes v1.21, and has been removed from Kubernetes v1.25.

For more information about the deprecation and removal of the PSP, refer to the *Kubernetes documentation*.

For more information about deploying the sample container without a PSP, refer to the section *Deploying the Containers without a PSP and RBAC*.

2.7.1 Applying a PSP

This section describes how to apply a PSP.

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

➤ 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 iap
   ```

2. If you want to store the policy snapshot on AWS S3 or use AWS KMS to decrypt the policy, then perform the following steps to create an IAM service account 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> --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 iap-sa --namespace iap --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 *iap-sa*, which has been defined in the *pty-psp.yaml* file.

3. If you want to store the policy snapshot on AWS EFS and use PBE to decrypt the policy, then perform the following steps to create an IAM service account 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 iap-sa --namespace iap --cluster cluster-name --approve
```

Important: Ensure that the name of the service account is *iap-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 Sample Policy application, 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 iap
```

Important: Skip *step 5* and *step 7* if you have already deployed a custom privileged PSP, instead of using the default Amazon EKS PSP that is automatically applied when a Kubernetes cluster is created.

If you are using a custom PSP, then ensure that it has privileged access to the Kubernetes cluster.

In addition, ensure that the custom PSP is not applied to the *iap* namespace that you have created for deploying the Sample Application containers on the Kubernetes cluster.

For more information about the extracted Helm charts, refer to the *step 8* of the section *Initializing the Linux Instance*.

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 *iap* 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.

2.7.2 Applying RBAC

This section describes how to apply RBAC.

Important: You require a *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 Sample Application containers.

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

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

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

      For example:

      ```
      kubectl create serviceaccount iap-deployer --namespace iap
      ```

      Important: Ensure that the name of the service account is *iap-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 iap-deployer --namespace iap -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
iap -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 step3> | base64 -d`
```

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

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

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

For example:

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

For more information about the extracted Helm charts, refer to the *step 8* of the section *Initializing the Linux Instance*.

2.7.3 Retrieving the Policy from the ESA

This section describes how to invoke the Immutable Service APIs to retrieve the policy package using the ESA.

For more information about the Immutable Service APIs, refer to the section *Appendix B: APIs for Immutable Protectors* in the *Protegrity APIs, UDFs, Commands Reference Guide 9.1.0.5*.

To retrieve the policy package from the ESA:

1. Run the following command to verify the PEP server versions that are supported by the Immutable Service on the ESA.

```
curl -v -k -u "<ESA user>:<ESA password>" https://<ESA IP>/pty/v1/imp/version
```

The command returns the following output.

```
{
    "version": "1.0.0",
    "buildnumber": "1.0.0",
    "pepVersion": ["1.1.0+85"]
}
```

Note: Ensure that the PEP server version returned by the IMP version API matches the PEP server version specified in the *manifest.json* file in the installation package.

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

2. If you want to download the policy package from the ESA and encrypt the policy package using a passphrase and a salt, then run the following command.

```
curl -v -k -u "<ESA user>:<ESA password>" https://<ESA IP>/pty/v1/imp/export -H
"Content-Type: application/json" -o policy_package_key.tgz --data
'{"name":"policypackage","pepVersion":"1.1.0+85","dataStoreName":"<Value>","emptySt
ring":"<Value>","caseSensitive":"<Value>","kek":{"pkcs5":
{"password":"<Password>","salt":"<Salt>"}}}'
```

For example:

```
curl -v -k -u "IMPUser:<Password>" https://10.49.1.218/pty/v1/imp/export -H
"Content-Type: application/json" -o policy_package_key.json --data
'{"name":"policypackage","pepVersion":"1.1.0+85","dataStoreName":"<Value>","emptySt
ring":"<Value>","caseSensitive":"<Value>","kek":{"pkcs5":
{"password":"<Password>","salt":"<Salt>"}}}'
```

The policy is encrypted using the key generated by using the passphrase and salt. The salt is used as an initialization vector for generating the key.

The encrypted policy package is downloaded to your machine.

> **Important:** Ensure that the user is assigned the **Export IMP** permission through roles.
>
> For more information about assigning permissions to user roles, refer to the section *Managing Roles* in the *Appliances Overview 9.1.0.5* guide.
>
> For more information about managing users, refer to the section *Managing Users* in the *Appliances Overview 9.1.0.5* guide.

For more information about the Export API, refer to the section *Sample Immutable Service API - Exporting Policy from a PEP Server Version* in the *Protegrity APIs, UDFs, Commands Reference Guide 9.1.0.5*.

> **Important:** Ensure that the password complexity meets the following requirements:
>
> - The password must contain a minimum of 10 characters and a maximum of 128 characters
> - The user should be able to specify Unicode characters, such as Emoji, in the password
> - The user should avoid commonly used passwords, such as, *123456789*, *11111111*, *password*, and *qwertyuiop*

3. If you want to download the policy package from the ESA and encrypt the policy package using a KMS, then run the following command.

```
curl -v -k -u "<ESA user>:<ESA password>" https://<ESA IP>/pty/v1/imp/export -H
"Content-Type: application/json" -o <Policy_package_name>.tgz --data
'{"name":"policypackage","pepVersion":"1.1.0+85","dataStoreName":"<value>","emptySt
ring":"<value>","caseSensitive":"<value>","kek":{"publicKey": {"value":"Public Key
of the AWS Customer Key used to encrypt the policy"}}}'
```

For example:

```
curl -v -k -u "IMPUser:<password>" https://10.49.1.218/pty/v1/imp/export -H
"Content-Type: application/json" -o policy_package_key.tgz --data
'{"name":"policypackage","pepVersion":"1.1.0+85","dataStoreName":"<value>","emptySt
ring":"<value>","caseSensitive":"<value>","kek":{"publicKey": {"value":"-----BEGIN
PUBLIC KEY-----
\nMIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEAjwxqqrrNQ1kV84GEfYLR\nVK/
SyOkGTs7kxEImEYMGtugSal2G9m7hFdVlWhHH/OvLDHwdOYe6GLXHVwycfbM/
\nS6pzQPS5ujzyJEaLWYAfcRgAHi9g8GvazDiv34Z+VO5FRbJzP9u9/23J4hExc+e1\nTezKsoBj6Ypx/
zBkE4dGqdamfHJUF3ptB82nhJT/Pth15ejL/SVrD/
q8sORU838O\negcIfQYmd5ziZJgbWLVzFaM9QZXH9hD/
0JxAhqfHP3Fnhmc5MDVdg7D0SQkufIjf\nu7djMy2I3ZRMgnkGxjq8PuBUFaH2s6DvAa4e6K0ppGjFoByVV
e2tumrUVEvJg2EM\ndD/
Pl15kzLelbDKjxrI8T1D8cWuGh43wf1xC+uRZsfoMSgwDWdpBmaOFP+q2k0d4\nrQWKIgZw71mxYADPKBAU
UwMgD/aREuvTDlpvpl0Vk2Y5i/4Xx/fK7GV5DCdy9TFM\nZgzAA/
O9z5tuVzqcgodMu06+iqtXdTUoOafA+BTvSIXLAgMBAAE=\n-----END PUBLIC
KEY-----","algorithm":"RSA_OAEP_SHA256"}}}'
```

Note: Ensure that the new line in the public key is replaced with the *\n* character.

The policy package is downloaded to your machine.

4. Copy the policy package file to an AWS S3 bucket or AWS EFS, as required.

2.7.4 Setting up the Environment for Deploying the Sample Application Container

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

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

To set up the environment for deploying the Sample Application container:

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 iap
```

2. 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 --namespace <Namespace name>
--set controller.replicaCount=2 ingress-nginx/ingress-nginx --set
controller.publishService.enabled=true --set podSecurityPolicy.enabled=true --
version 3.36.0 --set rbac.create=true --set controller.ingressClass=<Ingress
Class Name> --set controller.service.annotations."service\.beta\.kubernetes\.io/
aws-load-balancer-internal"="true" --set controller.extraArgs.enable-ssl-
passthrough=""
```

For example:

```
helm install nginx-ingress --namespace nginx --set controller.replicaCount=2
ingress-nginx/ingress-nginx --set controller.publishService.enabled=true
--set podSecurityPolicy.enabled=true --version 3.36.0 --
set rbac.create=true --set controller.ingressClass=nginx-iap
--set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-
balancer-internal"="true" --set controller.extraArgs.enable-ssl-passthrough=""
```

For more information about 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
```

2.7.5 Deploying the Sample Application Container on the Kubernetes Cluster

This section describes how to deploy the Sample Application container on the Kubernetes cluster by installing the Helm charts.

Important: Use the *iap-deployer* service account, which you have created in *step 1* of the section *Applying RBAC*, to perform the steps in this section.

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 2 to deploy the Sample Application container.

   For more information about the IAM User 2, refer to the section *Additional Prerequisites*.

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 *iap-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 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> --user=<Service account name>
      ```

   c. Run the following command to set the Kubernetes context to the newly created context in *step 2b*.

      ```
      kubectl config use-context <Context name from step 2b>
      ```

5. If you have used Passphrase-Based Encryption to encrypt the policy package, then run the following command to create a passphrase secret, which is used by the Sample Application container to decrypt the policy.

   ```
   kubectl create secret generic pbe-secret --from-literal='passphrase=<passphrase>'
   --from-literal='salt=<salt>' -n iap
   ```

   Ensure that you specify the same passphrase and salt that you have used in *step 2* of the section *Retrieving the Policy from the ESA* for encrypting the policy package.

   You need to specify the PBE secret name as the value of the *pbeSecrets* parameter in the *values.yaml* file in *step 8*.

6. If you have used AWS KMS to encrypt the policy package, then run the following command to create a private key secret, which is used by the Sample Application container to decrypt the policy.

   ```
   kubectl create secret generic key-secret --from-literal='privatekey=<ARN of the AWS
   Customer Master Key used to encrypt the policy package> ' -n iap
   ```

   For more information about the ARN of the AWS Customer Master Key, refer to *step 15* of the section *Creating an AWS Customer Master Key*.

   You need to specify the private key secret name as the value of the *privateKeySecret* parameter in the *values.yaml* file in *step 8*.

7. On the Linux instance, navigate to the location where you have extracted the Helm charts to deploy the Sample application.

   For more information about the extracted Helm charts, refer to the *step 9* of the section *Initializing the Linux Instance*.

   The *spring-apjava > values.yaml* file contains the default configuration values for deploying the Sample application on the Kubernetes cluster.

   ```
   ...
   ..
   .
   ## Configure the delays for Liveness Probe here
   livenessProbe:
     initialDelaySeconds: 15
     periodSeconds: 20

   ## Configure the delays for Readiness Probe here
   readinessProbe:
     initialDelaySeconds: 15
     periodSeconds: 20

   ## pod service account to be used
   ## leave the field empty if not applicable
   serviceAccount:

   ## set the Pod's security context object
   ## leave the field empty if not applicable
   podSecurityContext:
   ```

```
  runAsUser: 1000
  runAsGroup: 1000
  fsGroup: 1000

## set the Spring App Container's security context object
## leave the field empty if not applicable
springContainerSecurityContext:
  capabilities:
    drop:
    - ALL
  allowPrivilegeEscalation: false
  privileged : false
  runAsNonRoot : true
  readOnlyRootFilesystem: true

## persistent volume name e.g nfs-pvc
pvcName: PVC_NAME

## Choose your private key helper. Currently we support [PKCS5, AWS_KMS]
privateKeySource: PKCS5

## k8s secret containing passphrase for decrypting policy
pbeSecrets: PBE_SECRET_NAME

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

## start with blue deployment first
deploymentInUse: blue

## Policy metadata for Blue Deployments
blueDeployment:
  enabled: true
  policy:
    # absolute path in ObjectStore from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/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: "POLICY_PATH"

## Policy metadata for Green Deployments
greenDeployment:
  enabled: false
  policy:
    # absolute path in ObjectStore from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/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: "POLICY_PATH"

## specify the initial no. of springapp Pod replicas
replicaCount: 1

...
..
.


## specify the ports exposed in your springapp 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.
springappService:
  name: "restapi"
  port: 8080
  targetPort: 8080

## 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 external Ingress Controller if any,
  ## else keep the 'ingress.class' field empty
  ## to use cloud native Ingress Controller.
  annotations:
    kubernetes.io/ingress.class: nginx
    ## Enable client certificate authentication
    # nginx.ingress.kubernetes.io/auth-tls-verify-client: "on"
    ## Name of the Secret along with the Namespace, where its created,
    ## that contains the full CA chain ca.crt,
    ## that is enabled to authenticate against this Ingress
    # nginx.ingress.kubernetes.io/auth-tls-secret: "<NAMESPACE>/ca-secret"

  ## specify the host names and paths for Ingress rules
  ## prod and staging http paths can be used as optional fields.
  hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8080
        prod: "/"
    ## 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: 8080
    #     prod: "/"
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8080
    #     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: example-tls
  #    hosts:
  #      - prod.example.com
```

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

8. Modify the default values in the *values.yaml* file as required.

   For more information about Helm charts and their default values, refer to the section *Appendix A: Sample Application Helm Chart Details*.

Field	Description
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 2b* of the section *Applying a PSP*. 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 3b* of the section *Applying a PSP*.

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 **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. 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 snapshot. Leave the value of this field blank if you want to use AWS S3 buckets for storing the policy snapshot.
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
pbeSecrets	Specify the Kubernetes secret created in *step 5* that contains the passphrase and the salt that were used to generate key, which encrypted the policy. The Sample Application container uses the value specified in the *pbeSecrets* parameter to decrypt the policy. This parameter is applicable if you have specified *PKCS5* as the value of the *privateKeySource* parameter. **Important:** Ensure that the password complexity meets the following requirements: • The password must contain a minimum of 10 characters and a maximum of 128 characters • The user should be able to specify Unicode characters, such as Emoji, in the password

Field	Description
	• The user should avoid commonly used passwords, such as, *123456789*, *11111111*, *password*, and *qwertyuiop*
privateKeySecret	Specify the Kubernetes secret created in *step 6* that contains the private key of the AWS Customer Master Key, which encrypted the policy. The Sample Application container uses the value specified in the *privateKeySecret* parameter to decrypt the policy. This parameter is applicable if you have specified *AWS_KMS* as the value of the *privateKeySource* parameter.
deploymentInUse	Specify the value as *blue*. This indicates that the *blue* deployment strategy is in use.
blueDeployment/policy/filePath	Specify the path in the persistent volume or the object store where you have stored the policy. For example, if you are using AWS EFS as the persistent volume, then you can specify the file path as *file://<Mount directory>/xyz/imp*. In this case, a policy with the name as *imp* will be stored in the */<Mount directory>/xyz* directory in the required persistent volume. **Note:** The *<Mount directory>* is a directory inside the Mount point. If you have stored the policy in an Object Store, such as a 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/imp*, where *test* is the bucket name, *LOB* is the directory inside the bucket, and *imp* is the name of the policy. 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, hyphens, 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 disabled by default.
springappService/name	Specify a name for the tunnel to distinguish between ports.
springappService/port	Specify the port number on which you want to expose the Kubernetes service externally.
springappService/targetPort	Specify the port on which the Sample application is running inside the Docker container.
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

Field	Description
	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. For example, specify, *nginx* if you want to use the NGINX Ingress Controller.
nginx.ingress.kubernetes.io/auth-tls-verify-client	Set the value of this field to *on*, if you want to enable TLS mutual authentication between the REST API client and the AWS load balancer. By default, this field is *disabled*.
nginx.ingress.kubernetes.io/auth-tls-secret	Specify a value for the CA secret, if you have enabled the TLS mutual authentication between the REST API client and the AWS load balancer. By default, this field is *disabled*. For more information about the value of this field, refer to the section *Ingress Configuration*.
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 port on which the Sample application is running inside the Docker container.
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 required 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 required production or staging URL based on the hostname value.

9. Run the following command to deploy the Sample Application container on the Kubernetes cluster.

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

   For example:

   ```
   helm install spring-app --namespace iap spring-apjava
   ```

10. Perform the following steps to check the status of the Kubernetes resources.

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

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

    For example:

    ```
    kubectl get pods -n iap
    ```

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

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

For example:

```
kubectl get ingress -n iap
```

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

2.7.6 Upgrading the Helm Charts

This section describes the steps for upgrading the Helm charts.

Note: 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. On the Linux instance, navigate to the location where you have extracted the Helm charts for installing the Sample Application.

   For more information about the extracted Helm charts, refer to the *step 9* of the section *Initializing the Linux Instance*.

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

```
...
..
.
## Configure the delays for Liveness Probe here
livenessProbe:
  initialDelaySeconds: 15
  periodSeconds: 20

## Configure the delays for Readiness Probe here
readinessProbe:
  initialDelaySeconds: 15
  periodSeconds: 20

## pod service account to be used
## leave the field empty if not applicable
serviceAccount:

## set the Pod's security context object
## leave the field empty if not applicable
podSecurityContext:
  runAsUser: 1000
  runAsGroup: 1000
  fsGroup: 1000

## set the Spring App Container's security context object
## leave the field empty if not applicable
springContainerSecurityContext:
  capabilities:
    drop:
```

```
    - ALL
  allowPrivilegeEscalation: false
  privileged : false
  runAsNonRoot : true
  readOnlyRootFilesystem: true

## persistent volume name e.g nfs-pvc
pvcName: PVC_NAME

## Choose your private key helper. Currently we support [PKCS5, AWS_KMS]
privateKeySource: PKCS5

## k8s secret containing passphrase for decrypting policy
pbeSecrets: PBE_SECRET_NAME

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

## start with blue deployment first
deploymentInUse: blue

## Policy metadata for Blue Deployments
blueDeployment:
  enabled: true
  policy:
    # absolute path in ObjectStore from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/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: "POLICY_PATH"

## Policy metadata for Green Deployments
greenDeployment:
  enabled: false
  policy:
    # absolute path in ObjectStore from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/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: "POLICY_PATH"

## specify the initial no. of springapp Pod replicas
replicaCount: 1

...
..
.


## specify the ports exposed in your springapp 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.
springappService:
  name: "restapi"
  port: 8080
  targetPort: 8080

## 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 external Ingress Controller if any,
  ## else keep the 'ingress.class' field empty
  ## to use cloud native Ingress Controller.
  annotations:
    kubernetes.io/ingress.class: nginx
    ## Enable client certificate authentication
    # nginx.ingress.kubernetes.io/auth-tls-verify-client: "on"
    ## Name of the Secret along with the Namespace, where its created,
    ## that contains the full CA chain ca.crt,
    ## that is enabled to authenticate against this Ingress
    # nginx.ingress.kubernetes.io/auth-tls-secret: "<NAMESPACE>/ca-secret"

  ## specify the host names and paths for Ingress rules
  ## prod and staging http paths can be used as optional fields.
  hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8080
        prod: "/"
    ## 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: 8080
    #     prod: "/"
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8080
    #     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: example-tls
  #    hosts:
  #      - prod.example.com
```

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

2. 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 B: Sample Application 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 bucket for storing the policy snapshot.
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
pbeSecrets	Specify the Kubernetes secret that contains the passphrase and the salt that were used to generate key, which encrypted the policy.

Field	Description
	The Sample Application container uses the value specified in the *pbeSecrets* parameter to decrypt the policy. **Important:** Ensure that the password complexity meets the following requirements: • The password must contain a minimum of 10 characters and a maximum of 128 characters • The user should be able to specify Unicode characters, such as Emoji, in the password • The user should avoid commonly used passwords, such as, *123456789*, *11111111*, *password*, and *qwertyuiop*
privateKeySecret	Specify the Kubernetes secret created that contains the private key of the AWS Customer Master Key, which encrypted the policy. The Sample Application container uses the value specified in the *privateKeySecret* parameter to decrypt the policy. This parameter is applicable if you have specified *AWS_KMS* as the value of the *privateKeySource* parameter.
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/securityConfigSource	Specify one of the following options: • *VolumeMount* - Specify this value if you have stored the policy on a volume that is mounted on the pod. Use this option if you are using AWS EFS for storing the policy. • *ObjectStore* - Specify this value if you have stored the policy in an AWS S3 bucket. By default, the value is set to *VolumeMount*.
greenDeployment/policy/filePath	Specify the path in the persistent volume or the object store where you have stored the policy. For example, if you are using AWS EFS as the persistent volume, then you can specify the file path as *file:///<Mount directory>/xyz/imp*. In this case, a policy with the name as *imp* will be stored in the */<Mount directory>/xyz* directory in the required persistent volume. **Note:** The *<Mount directory>* is a directory inside the Mount point. If you have stored the policy in an Object Store, such as a 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/imp*, where *test* is the bucket name, *LOB* is the directory inside the bucket, and *imp* is the name of the policy. In

Field	Description
	this example, the bucket name is specified as the first name of the file path. **Important:** Use only uppercase and lowercase letters, numbers, dot, hyphens, and underscore in the file path. Do not use special characters in the file path. **Note:** This value is case-sensitive.
springappService/name	Specify a name for the tunnel to distinguish between ports.
springappService/port	Specify the port number on which you want to expose the Kubernetes service externally.
springappService/targetPort	Specify the port on which the Sample application is running inside the Docker container.
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. For example, specify, *nginx* if you want to use the NGINX Ingress Controller.
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 port on which the Sample application is running inside the Docker container.
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 required 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 required production or staging URL based on the hostname value.

3. Run the following command to upgrade the Helm charts.

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

For example:

```
helm upgrade spring-app --namespace iap spring-apjava
```

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

4. 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 Sample application containers with the updated configuration.

5. 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.

6. Run the following command to upgrade the Helm charts.

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

   For example:

   ```
   helm upgrade spring-app --namespace iap spring-apjava
   ```

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

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

7. Modify the *values.yaml* file to change the value of the *blueDeployment/enabled* field to *false*.

   This will be shut down the 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.

8. Run the following command to upgrade the Helm charts.

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

   For example:

   ```
   helm upgrade spring-app --namespace iap spring-apjava
   ```

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

   If you want to modify any policy or subsequent releases, then you need to repeat step *1* to step *8*. 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.

2.7.7 Upgrading the IAP Java from v7.1 MR4 to v9.1.0.5

This section describes how to seamlessly upgrade the IAP Java from v7.1 MR4 to v9.1.0.5 by upgrading the Helm charts.

To upgrade IAP Java from v7.1 MR4 to v9.1.0.5:

Before you begin

Ensure that the following prerequisites are met before upgrading the IAP from v7.1 MR4 to v9.1.0.5.

- The policy is created in the ESA 9.1.0.5.
- The policy package is generated by invoking the Immutable Service APIs.

  For more information about invoking the Immutable Service APIs, refer to the section *Retrieving the Policy from the ESA*.
- If you have used AWS KMS to encrypt the policy package, then ensure that you have created the private key secret, which is used by the Sample Application container to decrypt the policy.
- If you have used Passphrase-Based Encryption to encrypt the policy package, then ensure that you have create the passphrase secret, which is used by the Sample Application container to decrypt the policy.
- Ensure that you have created a custom image for the Sample Application container or the Custom container using the Dockerfile provided in the v9.1.0.5 installation package.

  For more information about building a custom image, refer to the section *Creating Custom Images Using Dockerfile*.

1. On the Linux instance, navigate to the location where you have extracted the Helm charts for installing the Sample Application.

   For more information about the extracted Helm charts, refer to the *step 9* of the section *Initializing the Linux Instance*.

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

```
...
..
.
## Configure the delays for Liveness Probe here
livenessProbe:
  initialDelaySeconds: 15
  periodSeconds: 20

## Configure the delays for Readiness Probe here
readinessProbe:
  initialDelaySeconds: 15
  periodSeconds: 20

## pod service account to be used
## leave the field empty if not applicable
serviceAccount:

## set the Pod's security context object
## leave the field empty if not applicable
podSecurityContext:
  runAsUser: 1000
  runAsGroup: 1000
  fsGroup: 1000

## set the Spring App Container's security context object
## leave the field empty if not applicable
springContainerSecurityContext:
  capabilities:
    drop:
    - ALL
  allowPrivilegeEscalation: false
  privileged : false
  runAsNonRoot : true
  readOnlyRootFilesystem: true

## persistent volume name e.g nfs-pvc
pvcName: PVC_NAME
```

```
## Choose your private key helper. Currently we support [PKCS5, AWS_KMS]
privateKeySource: PKCS5

## k8s secret containing passphrase for decrypting policy
pbeSecrets: PBE_SECRET_NAME

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

## start with blue deployment first
deploymentInUse: blue

## Policy metadata for Blue Deployments
blueDeployment:
  enabled: true
  policy:
    # absolute path in ObjectStore from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/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: "POLICY_PATH"

## Policy metadata for Green Deployments
greenDeployment:
  enabled: false
  policy:
    # absolute path in ObjectStore from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/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: "POLICY_PATH"

## specify the initial no. of springapp Pod replicas
replicaCount: 1

...
..
.


## specify the ports exposed in your springapp 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.
springappService:
  name: "restapi"
  port: 8080
  targetPort: 8080

## 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 external Ingress Controller if any,
  ## else keep the 'ingress.class' field empty
  ## to use cloud native Ingress Controller.
  annotations:
    kubernetes.io/ingress.class: nginx
    ## Enable client certificate authentication
    # nginx.ingress.kubernetes.io/auth-tls-verify-client: "on"
    ## Name of the Secret along with the Namespace, where its created,
    ## that contains the full CA chain ca.crt,
    ## that is enabled to authenticate against this Ingress
    # nginx.ingress.kubernetes.io/auth-tls-secret: "<NAMESPACE>/ca-secret"

  ## specify the host names and paths for Ingress rules
  ## prod and staging http paths can be used as optional fields.
  hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8080
        prod: "/"
    ## 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: 8080
    #     prod: "/"
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8080
    #     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: example-tls
  #    hosts:
  #      - prod.example.com
```

2. 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: Sample Application 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 bucket for storing the policy snapshot.
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
pbeSecrets	Specify the Kubernetes secret that contains the passphrase and the salt that were used to generate key, which encrypted the policy. The Sample Application container uses the value specified in the *pbeSecrets* parameter to decrypt the policy. **Important:** Ensure that the password complexity meets the following requirements: • The password must contain a minimum of 10 characters and a maximum of 128 characters • The user should be able to specify Unicode characters, such as Emoji, in the password • The user should avoid commonly used passwords, such as, *123456789*, *11111111*, *password*, and *qwertyuiop*

Field	Description
privateKeySecret	Specify the Kubernetes secret created that contains the private key of the AWS Customer Master Key, which encrypted the policy. The Sample Application container uses the value specified in the *privateKeySecret* parameter to decrypt the policy. This parameter is applicable if you have specified *AWS_KMS* as the value of the *privateKeySource* parameter.
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/securityConfigSource	Specify one of the following options: • *VolumeMount* - Specify this value if you have stored the policy on a volume that is mounted on the pod. Use this option if you are using AWS EFS for storing the policy. • *ObjectStore* - Specify this value if you have stored the policy in an AWS S3 bucket. By default, the value is set to *VolumeMount*.
greenDeployment/policy/filePath	Specify the path in the persistent volume or the object store where you have stored the policy. For example, if you are using AWS EFS as the persistent volume, then you can specify the file path as *file:///<Mount directory>/xyz/imp*. In this case, a policy with the name as *imp* will be stored in the */<Mount directory>/xyz* directory in the required persistent volume. **Note:** The *<Mount directory>* is a directory inside the Mount point. If you have stored the policy in an Object Store, such as a 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/imp*, where *test* is the bucket name, *LOB* is the directory inside the bucket, and *imp* is the name of the policy. 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, hyphens, and underscore in the file path. Do not use special characters in the file path. **Note:** This value is case-sensitive.
springappService/name	Specify a name for the tunnel to distinguish between ports.
springappService/port	Specify the port number on which you want to expose the Kubernetes service externally.

Field	Description
springappService/targetPort	Specify the port on which the Sample application is running inside the Docker container.
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. For example, specify, *nginx* if you want to use the NGINX Ingress Controller. **Important:** In v7.1MR4, if you have deployed the NGINX Ingress Controller without specifying this parameter, then ensure that you comment out this parameter in the *values.yaml* file. Otherwise, the requests that are sent from the client application to the NGINX Ingress Controller will not be forwarded to the pods in the Sample Application container.
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 port on which the Sample application is running inside the Docker container.
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 required 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 required production or staging URL based on the hostname value.

3. Run the following command to upgrade the Helm charts.

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

For example:

```
helm upgrade spring-app --namespace iap spring-apjava
```

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

Important: Ensure that you use the same release name that you had used while deploying the Sample Application container in v7.1 MR4.

2.8 Deploying the Containers without a PSP and RBAC

This section describes the steps that you need to perform for deploying the Sample Application containers without a PSP and RBAC.

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

2.8.1 Retrieving the Policy from the ESA

This section describes how to invoke the Immutable Service APIs to retrieve the policy package using the ESA.

For more information about the Immutable Service APIs, refer to the section *Appendix B: APIs for Immutable Protectors* in the *Protegrity APIs, UDFs, Commands Reference Guide 9.1.0.5*.

To retrieve the policy package from the ESA:

1. Run the following command to verify the PEP server versions that are supported by the Immutable Service on the ESA.

   ```
   curl -v -k -u "<ESA user>:<ESA password>" https://<ESA IP>/pty/v1/imp/version
   ```

   The command returns the following output.

   ```
   {
       "version": "1.0.0",
       "buildnumber": "1.0.0",
       "pepVersion": ["1.1.0+85"]
   }
   ```

   Note: Ensure that the PEP server version returned by the IMP version API matches the PEP server version specified in the *manifest.json* file in the installation package.

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

2. If you want to download the policy package from the ESA and encrypt the policy package using a passphrase and a salt, then run the following command.

   ```
   curl -v -k -u "<ESA user>:<ESA password>" https://<ESA IP>/pty/v1/imp/export -H
   "Content-Type: application/json" -o policy_package_key.tgz --data
   '{"name":"policypackage","pepVersion":"1.1.0+85","dataStoreName":"<Value>","emptySt
   ring":"<Value>","caseSensitive":"<Value>","kek":{"pkcs5":
   {"password":"<Password>","salt":"<Salt>"}}}'
   ```

   For example:

   ```
   curl -v -k -u "IMPUser:<Password>" https://10.49.1.218/pty/v1/imp/export -H
   "Content-Type: application/json" -o policy_package_key.json --data
   '{"name":"policypackage","pepVersion":"1.1.0+85","dataStoreName":"<Value>","emptySt
   ring":"<Value>","caseSensitive":"<Value>","kek":{"pkcs5":
   {"password":"<Password>","salt":"<Salt>"}}}'
   ```

   The policy is encrypted using the key generated by using the passphrase and salt. The salt is used as an initialization vector for generating the key.

The encrypted policy package is downloaded to your machine.

> **Important:** Ensure that the user is assigned the **Export IMP** permission through roles.
>
> For more information about assigning permissions to user roles, refer to the section *Managing Roles* in the *Appliances Overview 9.1.0.5* guide.
>
> For more information about managing users, refer to the section *Managing Users* in the *Appliances Overview 9.1.0.5* guide.

For more information about the Export API, refer to the section *Sample Immutable Service API - Exporting Policy from a PEP Server Version* in the *Protegrity APIs, UDFs, Commands Reference Guide 9.1.0.5*.

> **Important:** Ensure that the password complexity meets the following requirements:
> - The password must contain a minimum of 10 characters and a maximum of 128 characters
> - The user should be able to specify Unicode characters, such as Emoji, in the password
> - The user should avoid commonly used passwords, such as, *123456789*, *11111111*, *password*, and *qwertyuiop*

3. If you want to download the policy package from the ESA and encrypt the policy package using a KMS, then run the following command.

```
curl -v -k -u "<ESA user>:<ESA password>" https://<ESA IP>/pty/v1/imp/export -H
"Content-Type: application/json" -o <Policy_package_name>.tgz --data
'{"name":"policypackage","pepVersion":"1.1.0+85","dataStoreName":"<value>","emptySt
ring":"<value>","caseSensitive":"<value>","kek":{"publicKey": {"value":"Public Key
of the AWS Customer Key used to encrypt the policy"}}}'
```

For example:

```
curl -v -k -u "IMPUser:<password>" https://10.49.1.218/pty/v1/imp/export -H
"Content-Type: application/json" -o policy_package_key.tgz --data
'{"name":"policypackage","pepVersion":"1.1.0+85","dataStoreName":"<value>","emptySt
ring":"<value>","caseSensitive":"<value>","kek":{"publicKey": {"value":"-----BEGIN
PUBLIC KEY-----
\nMIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEAjwxqqrrNQ1kV84GEfYLR\nVK/
SyOkGTs7kxEImEYMGtugSal2G9m7hFdVlWhHH/OvLDHwdOYe6GLXHVwycfbM/
\nS6pzQPS5ujzyJEaLWYAfcRgAHi9g8GvazDiv34Z+VO5FRbJzP9u9/23J4hExc+e1\nTezKsoBj6Ypx/
zBkE4dGqdamfHJUF3ptB82nhJT/Pth15ejL/SVrD/
q8sORU838O\negcIfQYmd5ziZJgbWLVzFaM9QZXH9hD/
0JxAhqfHP3Fnhmc5MDVdg7D0SQkufIjf\nu7djMy2I3ZRMgnkGxjq8PuBUFaH2s6DvAa4e6K0ppGjFoByVV
e2tumrUVEvJg2EM\ndD/
Pl15kzLelbDKjxrI8T1D8cWuGh43wf1xC+uRZsfoMSgwDWdpBmaOFP+q2k0d4\nrQWKIgZw71mxYADPKBAU
UwMgD/aREuvTDlpvpl0Vk2Y5i/4Xx/fK7GV5DCdy9TFM\nZgzAA/
O9z5tuVzqcgodMu06+iqtXdTUoOafA+BTvSIXLAgMBAAE=\n-----END PUBLIC
KEY-----","algorithm":"RSA_OAEP_SHA256"}}}'
```

> **Note:** Ensure that the new line in the public key is replaced with the *\n* character.

The policy package is downloaded to your machine.

4. Copy the policy package file to an AWS S3 bucket or AWS EFS, as required.

2.8.2 Deploying the Sample Application Container on the Kubernetes Cluster

This section describes how to deploy the Sample Application container on the Kubernetes cluster by installing the Helm charts.

To deploy a release on the Kubernetes cluster:

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 iap
   ```

2. 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 --namespace <Namespace name>
   --set controller.replicaCount=2 ingress-nginx/ingress-nginx --set
   controller.publishService.enabled=true --set podSecurityPolicy.enabled=true --
   version 3.36.0 --set rbac.create=true --set controller.ingressClass=<Ingress
   Class Name> --set controller.service.annotations."service\.beta\.kubernetes\.io/
   aws-load-balancer-internal"="true" --set controller.extraArgs.enable-ssl-
   passthrough=""
   ```

   For example:

   ```
   helm install nginx-ingress --namespace nginx --set controller.replicaCount=2
   ingress-nginx/ingress-nginx --set controller.publishService.enabled=true
   --set podSecurityPolicy.enabled=true --version 3.36.0 --
   set rbac.create=true --set controller.ingressClass=nginx-iap
   --set controller.service.annotations."service\.beta\.kubernetes\.io/aws-load-
   balancer-internal"="true" --set controller.extraArgs.enable-ssl-passthrough=""
   ```

   For more information about 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 iap
   ```

3. If you want to use AWS S3 for storing the policy snapshot, instead of AWS EFS, or use AWS KMS to decrypt the policy, then perform the following steps to create a service account in the Kubernetes cluster. These steps ensure that only the pod on which the Sample application has been deployed will be able to access the S3 bucket where the immutable policy snapshot has been uploaded.

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 the required service account.

```
eksctl create iamserviceaccount --name <Name of the service account> --
namespace <Namespace where the Kubernetes cluster has been created> --cluster
<Name of the Kubernetes cluster> --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 pod-s3-eks-serviceaccount --
namespace iap --cluster spring-app --attach-policy-arn arn:aws:iam::aws:policy/
AmazonS3ReadOnlyAccess --attach-policy-arn arn:aws:iam::829528124735:policy/
KMSDecryptAccess --approve
```

4. If you have used Passphrase-Based Encryption to encrypt the policy package, then run the following command to create a passphrase secret, which is used by the Sample Application container to decrypt the policy.

```
kubectl create secret generic pbe-secret --from-literal='passphrase=<passphrase>'
--from-literal='salt=<salt>' -n iap
```

Ensure that you specify the same passphrase and salt that you have used in *step 2* of the section *Retrieving the Policy from the ESA* for encrypting the policy package.

You need to specify the PBE secret name as the value of the *pbeSecrets* parameter in the *values.yaml* file in *step 8*.

5. If you have used AWS KMS to encrypt the policy package, then run the following command to create a private key secret, which is used by the Sample Application container to decrypt the policy.

```
kubectl create secret generic key-secret --from-literal='privatekey=<ARN of the AWS
Customer Master Key used to encrypt the policy package> ' -n iap
```

For more information about the ARN of the AWS Customer Master Key, refer to *step 15* of the section *Creating an AWS Customer Master Key*.

You need to specify the private key secret name as the value of the *privateKeySecret* parameter in the *values.yaml* file in *step 8*.

6. On the Linux instance, navigate to the location where you have extracted the Helm charts to deploy the Sample application.

For more information about the extracted Helm charts, refer to the *step 9* of the section *Initializing the Linux Instance*.

The *spring-apjava > values.yaml* file contains the default configuration values for deploying the Sample application on the Kubernetes cluster.

If you have used the AWS S3 bucket for storing the policy snapshot, then use the following snippet.

```
...
..
.
```

```
## Configure the delays for Liveness Probe here
livenessProbe:
  initialDelaySeconds: 15
  periodSeconds: 20

## Configure the delays for Readiness Probe here
readinessProbe:
  initialDelaySeconds: 15
  periodSeconds: 20

## pod service account to be used
## leave the field empty if not applicable
serviceAccount:

## set the Pod's security context object
## leave the field empty if not applicable
podSecurityContext:
  runAsUser: 1000
  runAsGroup: 1000
  fsGroup: 1000

## set the Spring App Container's security context object
## leave the field empty if not applicable
springContainerSecurityContext:
  capabilities:
    drop:
    - ALL
  allowPrivilegeEscalation: false
  privileged : false
  runAsNonRoot : true
  readOnlyRootFilesystem: true

## persistent volume name e.g nfs-pvc
pvcName: PVC_NAME

## Choose your private key helper. Currently we support [PKCS5, AWS_KMS]
privateKeySource: PKCS5

## k8s secret containing passphrase for decrypting policy
pbeSecrets: PBE_SECRET_NAME

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

## start with blue deployment first
deploymentInUse: blue

## Policy metadata for Blue Deployments
blueDeployment:
  enabled: true
  policy:
    # absolute path in ObjectStore from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/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: "POLICY_PATH"

## Policy metadata for Green Deployments
greenDeployment:
  enabled: false
  policy:
    # absolute path in ObjectStore from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/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: "POLICY_PATH"

## specify the initial no. of springapp Pod replicas
replicaCount: 1

...
..
.


## specify the ports exposed in your springapp 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.
springappService:
  name: "restapi"
  port: 8080
  targetPort: 8080

## 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 external Ingress Controller if any,
  ## else keep the 'ingress.class' field empty
  ## to use cloud native Ingress Controller.
  annotations:
    kubernetes.io/ingress.class: nginx
    ## Enable client certificate authentication
    # nginx.ingress.kubernetes.io/auth-tls-verify-client: "on"
    ## Name of the Secret along with the Namespace, where its created,
    ## that contains the full CA chain ca.crt,
    ## that is enabled to authenticate against this Ingress
    # nginx.ingress.kubernetes.io/auth-tls-secret: "<NAMESPACE>/ca-secret"

  ## specify the host names and paths for Ingress rules
  ## prod and staging http paths can be used as optional fields.
  hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8080
        prod: "/"
    ## 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: 8080
    #     prod: "/"
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8080
    #     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: example-tls
  #    hosts:
  #      - prod.example.com
```

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

7. 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 B: Sample Application Helm Chart Details*.

Field	**Description**
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 3*, or else leave it blank.
podSecurityContext	Specify the privilege and access control settings for the pod. The default values are set as follows: • *runAsUser* - 1000 • *runAsGroup* - 1000 • *fsGroup* - 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. 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 snapshot. Leave the value of this field blank if you want to use AWS S3 buckets for storing the policy snapshot.
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
pbeSecrets	Specify the Kubernetes secret created in *step 4* that contains the passphrase and the salt that were used to generate key, which encrypted the policy. The Sample Application container uses the value specified in the *pbeSecrets* parameter to decrypt the policy. **Important:** Ensure that the password complexity meets the following requirements: • The password must contain a minimum of 10 characters and a maximum of 128 characters

Field	Description
	• The user should be able to specify Unicode characters, such as Emoji, in the password • The user should avoid commonly used passwords, such as, *123456789*, *11111111*, *password*, and *qwertyuiop*
privateKeySecret	Specify the Kubernetes secret created in *step 6* that contains the private key of the AWS Customer Master Key, which encrypted the policy. The Sample Application container uses the value specified in the *privateKeySecret* parameter to decrypt the policy. This parameter is applicable if you have specified *AWS_KMS* as the value of the *privateKeySource* parameter.
deploymentInUse	Specify the value as *blue*. This indicates that the *blue* deployment strategy is in use.
blueDeployment/policy/filePath	Specify the path in the persistent volume or the object store where you have stored the policy. For example, if you are using AWS EFS as the persistent volume, then you can specify the file path as *file://<Mount directory>/xyz/imp*. In this case, a policy with the name as *imp* will be stored in the */<Mount directory>/xyz* directory in the required persistent volume. **Note:** The *<Mount directory>* is a directory inside the Mount point. If you have stored the policy in an Object Store, such as a 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/imp*, where *test* is the bucket name, *LOB* is the directory inside the bucket, and *imp* is the name of the policy. 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, hyphens, 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 disabled by default.
springappService/name	Specify a name for the tunnel to distinguish between ports.
springappService/port	Specify the port number on which you want to expose the Kubernetes service externally.
springappService/targetPort	Specify the port on which the Sample application is running inside the Docker container.

Field	Description
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. For example, specify, *nginx* if you want to use the NGINX Ingress Controller.
nginx.ingress.kubernetes.io/auth-tls-verify-client	Set the value of this field to *on*, if you want to enable TLS mutual authentication between the REST API client and the AWS load balancer. By default, this field is *disabled*.
nginx.ingress.kubernetes.io/auth-tls-secret	Specify a value for the CA secret, if you have enabled the TLS mutual authentication between the REST API client and the AWS load balancer. By default, this field is *disabled*. For more information about the value of this field, refer to the section *Ingress Configuration*.
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 port on which the Sample application is running inside the Docker container.
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 required 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 required production or staging URL based on the hostname value.

8. Run the following command to deploy the Sample Application container on the Kubernetes cluster.

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

   For example:

   ```
   helm install spring-app --namespace iap spring-apjava
   ```

9. 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 iap
```

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

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

For example:

```
kubectl get ingress -n nginx
```

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

2.8.3 Upgrading the Helm Charts

This section describes the steps for upgrading the Helm charts.

Note: 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. On the Linux instance, navigate to the location where you have extracted the Helm charts for installing the Sample Application.

   For more information about the extracted Helm charts, refer to the *step 9* of the section *Initializing the Linux Instance*.

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

```
...
..
.
## Configure the delays for Liveness Probe here
livenessProbe:
  initialDelaySeconds: 15
  periodSeconds: 20

## Configure the delays for Readiness Probe here
readinessProbe:
  initialDelaySeconds: 15
  periodSeconds: 20

## pod service account to be used
## leave the field empty if not applicable
serviceAccount:

## set the Pod's security context object
## leave the field empty if not applicable
podSecurityContext:
  runAsUser: 1000
  runAsGroup: 1000
  fsGroup: 1000

## set the Spring App Container's security context object
## leave the field empty if not applicable
springContainerSecurityContext:
  capabilities:
    drop:
    - ALL
  allowPrivilegeEscalation: false
  privileged : false
  runAsNonRoot : true
  readOnlyRootFilesystem: true

## persistent volume name e.g nfs-pvc
pvcName: PVC_NAME

## Choose your private key helper. Currently we support [PKCS5, AWS_KMS]
privateKeySource: PKCS5

## k8s secret containing passphrase for decrypting policy
pbeSecrets: PBE_SECRET_NAME

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

## start with blue deployment first
deploymentInUse: blue

## Policy metadata for Blue Deployments
blueDeployment:
  enabled: true
  policy:
    # absolute path in ObjectStore from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/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: "POLICY_PATH"

## Policy metadata for Green Deployments
greenDeployment:
  enabled: false
  policy:
    # absolute path in ObjectStore from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/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: "POLICY_PATH"

## specify the initial no. of springapp Pod replicas
replicaCount: 1

...
..
.


## specify the ports exposed in your springapp 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.
springappService:
  name: "restapi"
  port: 8080
  targetPort: 8080

## 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 external Ingress Controller if any,
  ## else keep the 'ingress.class' field empty
  ## to use cloud native Ingress Controller.
  annotations:
    kubernetes.io/ingress.class: nginx
    ## Enable client certificate authentication
    # nginx.ingress.kubernetes.io/auth-tls-verify-client: "on"
    ## Name of the Secret along with the Namespace, where its created,
    ## that contains the full CA chain ca.crt,
    ## that is enabled to authenticate against this Ingress
    # nginx.ingress.kubernetes.io/auth-tls-secret: "<NAMESPACE>/ca-secret"

  ## specify the host names and paths for Ingress rules
  ## prod and staging http paths can be used as optional fields.
  hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8080
        prod: "/"
    ## 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: 8080
    #     prod: "/"
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8080
    #     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: example-tls
  #    hosts:
  #      - prod.example.com
```

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

2. 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 B: Sample Application 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 bucket for storing the policy snapshot.
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
pbeSecrets	Specify the Kubernetes secret that contains the passphrase and the salt that were used to generate key, which encrypted the policy.

Field	**Description**
	The Sample Application container uses the value specified in the *pbeSecrets* parameter to decrypt the policy. **Important:** Ensure that the password complexity meets the following requirements: • The password must contain a minimum of 10 characters and a maximum of 128 characters • The user should be able to specify Unicode characters, such as Emoji, in the password • The user should avoid commonly used passwords, such as, *123456789*, *11111111*, *password*, and *qwertyuiop*
privateKeySecret	Specify the Kubernetes secret created that contains the private key of the AWS Customer Master Key, which encrypted the policy. The Sample Application container uses the value specified in the *privateKeySecret* parameter to decrypt the policy. This parameter is applicable if you have specified *AWS_KMS* as the value of the *privateKeySource* parameter.
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/securityConfigSource	Specify one of the following options: • *VolumeMount* - Specify this value if you have stored the policy on a volume that is mounted on the pod. Use this option if you are using AWS EFS for storing the policy. • *ObjectStore* - Specify this value if you have stored the policy in an AWS S3 bucket. By default, the value is set to *VolumeMount*.
greenDeployment/policy/filePath	Specify the path in the persistent volume or the object store where you have stored the policy. For example, if you are using AWS EFS as the persistent volume, then you can specify the file path as *file:///<Mount directory>/xyz/imp*. In this case, a policy with the name as *imp* will be stored in the */<Mount directory>/xyz* directory in the required persistent volume. **Note:** The *<Mount directory>* is a directory inside the Mount point. If you have stored the policy in an Object Store, such as a 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/imp*, where *test* is the bucket name, *LOB* is the directory inside the bucket, and *imp* is the name of the policy. In

Field	Description
	this example, the bucket name is specified as the first name of the file path. **Important:** Use only uppercase and lowercase letters, numbers, dot, hyphens, and underscore in the file path. Do not use special characters in the file path. **Note:** This value is case-sensitive.
springappService/name	Specify a name for the tunnel to distinguish between ports.
springappService/port	Specify the port number on which you want to expose the Kubernetes service externally.
springappService/targetPort	Specify the port on which the Sample application is running inside the Docker container.
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. For example, specify, *nginx* if you want to use the NGINX Ingress Controller.
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 port on which the Sample application is running inside the Docker container.
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 required 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 required production or staging URL based on the hostname value.

3. Run the following command to upgrade the Helm charts.

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

For example:

```
helm upgrade spring-app --namespace iap spring-apjava
```

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

4. 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 Sample application containers with the updated configuration.

5. 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.
6. Run the following command to upgrade the Helm charts.

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

   For example:

   ```
   helm upgrade spring-app --namespace iap spring-apjava
   ```

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

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

7. Modify the *values.yaml* file to change the value of the *blueDeployment/enabled* field to *false*.

   This will be shut down the 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.

8. Run the following command to upgrade the Helm charts.

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

   For example:

   ```
   helm upgrade spring-app --namespace iap spring-apjava
   ```

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

   If you want to modify any policy or subsequent releases, then you need to repeat step *1* to step *8*. 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.

2.8.4 Upgrading the IAP Java from v7.1 MR4 to v9.1.0.5

This section describes how to seamlessly upgrade the IAP Java from v7.1 MR4 to v9.1.0.5 by upgrading the Helm charts.

To upgrade IAP Java from v7.1 MR4 to v9.1.0.5:

Before you begin
Ensure that the following prerequisites are met before upgrading the IAP from v7.1 MR4 to v9.1.0.5.

- The policy is created in the ESA 9.1.0.5.
- The policy package is generated by invoking the Immutable Service APIs.

  For more information about invoking the Immutable Service APIs, refer to the section *Retrieving the Policy from the ESA*.
- If you have used AWS KMS to encrypt the policy package, then ensure that you have created the private key secret, which is used by the Sample Application container to decrypt the policy.
- If you have used Passphrase-Based Encryption to encrypt the policy package, then ensure that you have create the passphrase secret, which is used by the Sample Application container to decrypt the policy.
- Ensure that you have created a custom image for the Sample Application container or the Custom container using the Dockerfile provided in the v9.1.0.5 installation package.

  For more information about building a custom image, refer to the section *Creating Custom Images Using Dockerfile*.

1. On the Linux instance, navigate to the location where you have extracted the Helm charts for installing the Sample Application.

   For more information about the extracted Helm charts, refer to the *step 9* of the section *Initializing the Linux Instance*.

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

```
...
..
.
## Configure the delays for Liveness Probe here
livenessProbe:
  initialDelaySeconds: 15
  periodSeconds: 20

## Configure the delays for Readiness Probe here
readinessProbe:
  initialDelaySeconds: 15
  periodSeconds: 20

## pod service account to be used
## leave the field empty if not applicable
serviceAccount:

## set the Pod's security context object
## leave the field empty if not applicable
podSecurityContext:
  runAsUser: 1000
  runAsGroup: 1000
  fsGroup: 1000

## set the Spring App Container's security context object
## leave the field empty if not applicable
springContainerSecurityContext:
  capabilities:
    drop:
    - ALL
  allowPrivilegeEscalation: false
  privileged : false
  runAsNonRoot : true
  readOnlyRootFilesystem: true

## persistent volume name e.g nfs-pvc
pvcName: PVC_NAME
```

```
## Choose your private key helper. Currently we support [PKCS5, AWS_KMS]
privateKeySource: PKCS5

## k8s secret containing passphrase for decrypting policy
pbeSecrets: PBE_SECRET_NAME

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

## start with blue deployment first
deploymentInUse: blue

## Policy metadata for Blue Deployments
blueDeployment:
  enabled: true
  policy:
    # absolute path in ObjectStore from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/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: "POLICY_PATH"

## Policy metadata for Green Deployments
greenDeployment:
  enabled: false
  policy:
    # absolute path in ObjectStore from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/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: "POLICY_PATH"

## specify the initial no. of springapp Pod replicas
replicaCount: 1

...
..
.


## specify the ports exposed in your springapp 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.
springappService:
  name: "restapi"
  port: 8080
  targetPort: 8080

## 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 external Ingress Controller if any,
  ## else keep the 'ingress.class' field empty
  ## to use cloud native Ingress Controller.
  annotations:
    kubernetes.io/ingress.class: nginx
    ## Enable client certificate authentication
    # nginx.ingress.kubernetes.io/auth-tls-verify-client: "on"
    ## Name of the Secret along with the Namespace, where its created,
    ## that contains the full CA chain ca.crt,
    ## that is enabled to authenticate against this Ingress
    # nginx.ingress.kubernetes.io/auth-tls-secret: "<NAMESPACE>/ca-secret"

  ## specify the host names and paths for Ingress rules
  ## prod and staging http paths can be used as optional fields.
  hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8080
        prod: "/"
    ## 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: 8080
    #     prod: "/"
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8080
    #     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: example-tls
  #     hosts:
  #       - prod.example.com
```

2. 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: Sample Application 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 bucket for storing the policy snapshot.
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
pbeSecrets	Specify the Kubernetes secret that contains the passphrase and the salt that were used to generate key, which encrypted the policy. The Sample Application container uses the value specified in the *pbeSecrets* parameter to decrypt the policy. **Important:** Ensure that the password complexity meets the following requirements: • The password must contain a minimum of 10 characters and a maximum of 128 characters • The user should be able to specify Unicode characters, such as Emoji, in the password • The user should avoid commonly used passwords, such as, *123456789*, *11111111*, *password*, and *qwertyuiop*

Field	Description
privateKeySecret	Specify the Kubernetes secret created that contains the private key of the AWS Customer Master Key, which encrypted the policy. The Sample Application container uses the value specified in the *privateKeySecret* parameter to decrypt the policy. This parameter is applicable if you have specified *AWS_KMS* as the value of the *privateKeySource* parameter.
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/securityConfigSource	Specify one of the following options: • *VolumeMount* - Specify this value if you have stored the policy on a volume that is mounted on the pod. Use this option if you are using AWS EFS for storing the policy. • *ObjectStore* - Specify this value if you have stored the policy in an AWS S3 bucket. By default, the value is set to *VolumeMount*.
greenDeployment/policy/filePath	Specify the path in the persistent volume or the object store where you have stored the policy. For example, if you are using AWS EFS as the persistent volume, then you can specify the file path as *file:///<Mount directory>/xyz/imp*. In this case, a policy with the name as *imp* will be stored in the */<Mount directory>/xyz* directory in the required persistent volume. **Note:** The *<Mount directory>* is a directory inside the Mount point. If you have stored the policy in an Object Store, such as a 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/imp*, where *test* is the bucket name, *LOB* is the directory inside the bucket, and *imp* is the name of the policy. 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, hyphens, and underscore in the file path. Do not use special characters in the file path. **Note:** This value is case-sensitive.
springappService/name	Specify a name for the tunnel to distinguish between ports.
springappService/port	Specify the port number on which you want to expose the Kubernetes service externally.

Field	Description
springappService/targetPort	Specify the port on which the Sample application is running inside the Docker container.
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. For example, specify, *nginx* if you want to use the NGINX Ingress Controller. **Important:** In v7.1MR4, if you have deployed the NGINX Ingress Controller without specifying this parameter, then ensure that you comment out this parameter in the *values.yaml* file. Otherwise, the requests that are sent from the client application to the NGINX Ingress Controller will not be forwarded to the pods in the Sample Application container.
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 port on which the Sample application is running inside the Docker container.
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 required 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 required production or staging URL based on the hostname value.

3. Run the following command to upgrade the Helm charts.

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

For example:

```
helm upgrade spring-app --namespace iap spring-apjava
```

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

Important: Ensure that you use the same release name that you had used while deploying the Sample Application container in v7.1 MR4.

Chapter 3

Accessing Logs Using Splunk

This section describes how to access the Application Protector and Container logs using a third-party tool, such as Splunk, which is used as an example to process the logs from the IAP deployment.

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

3.1 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 Application Protector 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.

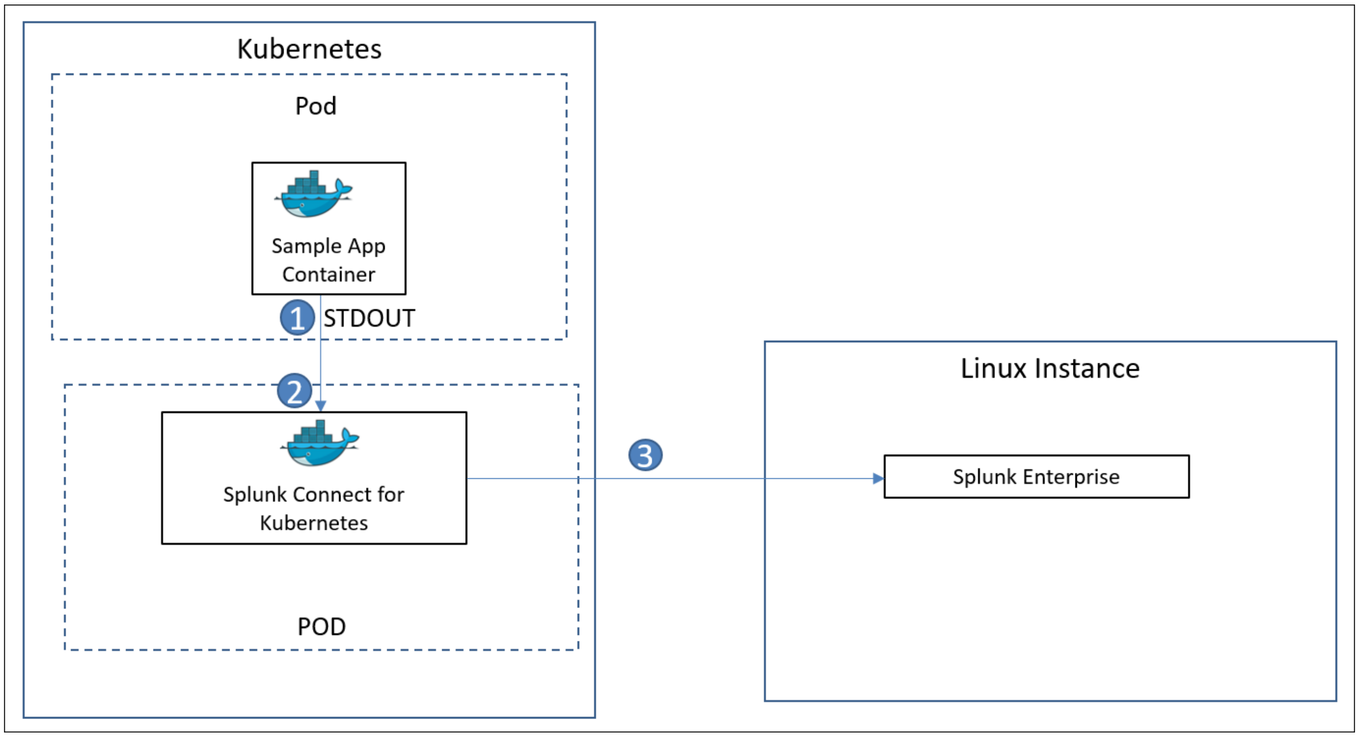


Figure 3-1: Sample Logging Workflow

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

1. The Sample Application container writes 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 Kubernetes* page on Github.

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

3.2 Setting Up Splunk

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

To set up Splunk:

1. Create a Linux instance, either in an 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 using the following command.

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

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

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

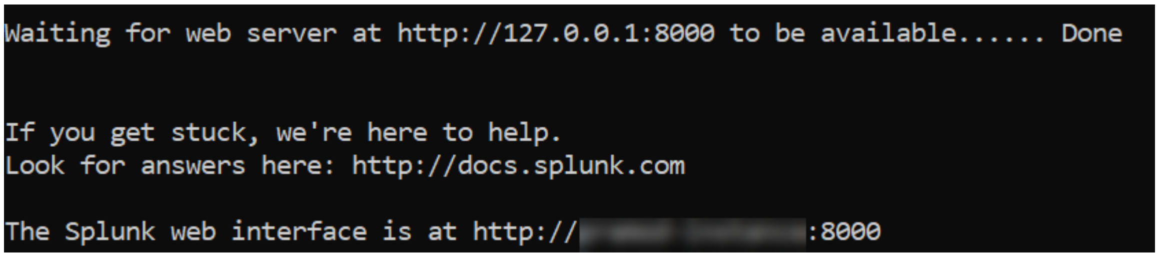
4. Type the username for the administrator account for logging into Splunk Enterprise, and then press **Enter**.

   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 *10.xx.xx.xx*, then you can access Splunk Web at *http://10.xx.xx.xx:8000*.

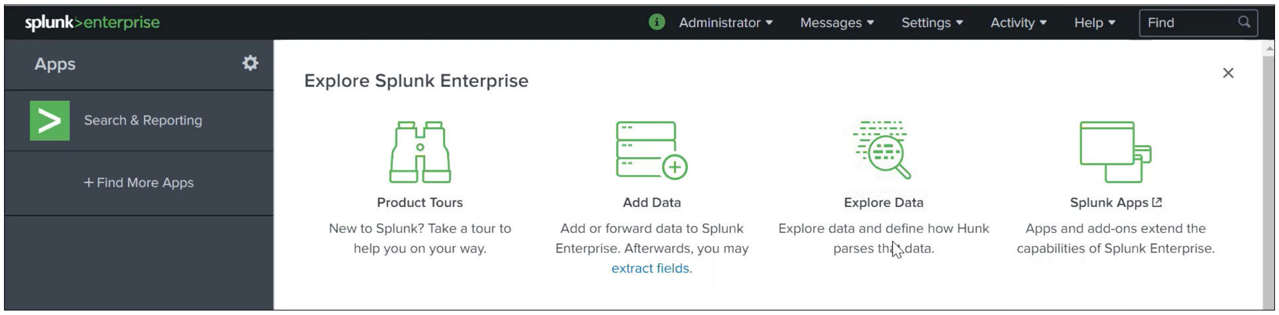
3.3 Configuring Splunk

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

To configure Splunk Enterprise:

1. Login to 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.



New to Splunk? Take a tour to help you on your way.

Add or forward data to Splunk Enterprise. Afterwards, you may extract fields.

Explore data and define how Hunk parses the data.

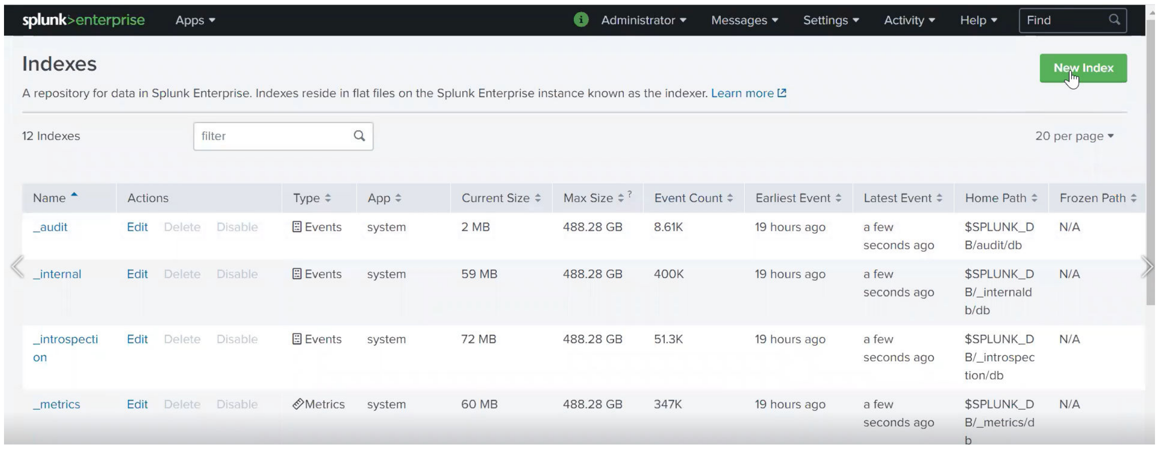
Apps and add-ons extend the capabilities of Splunk Enterprise.

Figure 3-2: Splunk Enterprise Web UI

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

3. Perform the following steps to create an index, which is a repository for storing the Splunk Enterprise data.
   a. Navigate to **Settings** > **Index**.

      The **Indexes** screen appears.



Indexes

A repository for data in Splunk Enterprise. Indexes reside in flat files on the Splunk Enterprise instance known as the indexer. Learn more

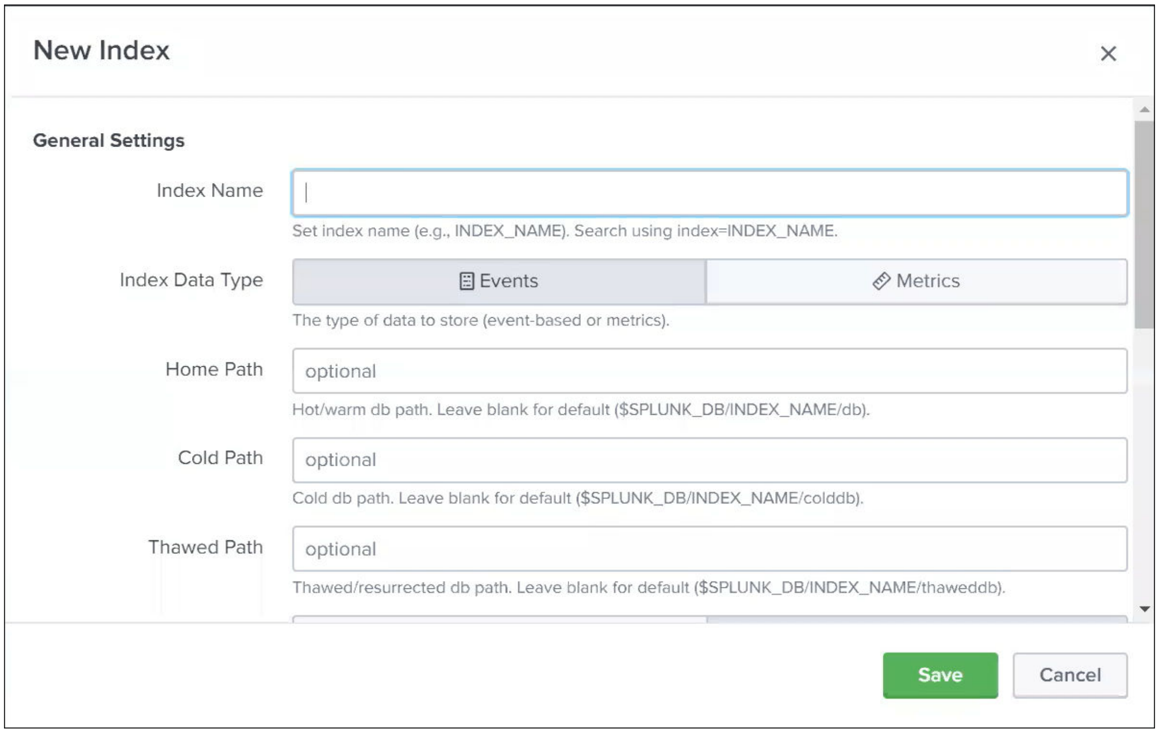
12 Indexes filter 20 per page

Name	Actions	Type	App	Current Size	Max Size	Event Count	Earliest Event	Latest Event	Home Path	Frozen Path
_audit	Edit Delete Disable	Events	system	2 MB	488.28 GB	8.61K	19 hours ago	a few seconds ago	$SPLUNK_DB/audit/db	N/A
_internal	Edit Delete Disable	Events	system	59 MB	488.28 GB	400K	19 hours ago	a few seconds ago	$SPLUNK_DB/_internaldb/db	N/A
_introspection	Edit Delete Disable	Events	system	72 MB	488.28 GB	51.3K	19 hours ago	a few seconds ago	$SPLUNK_DB/_introspection/db	N/A
_metrics	Edit Delete Disable	Metrics	system	60 MB	488.28 GB	347K	19 hours ago	a few seconds ago	$SPLUNK_DB/_metrics/db	N/A

Figure 3-3: Indexes Screen

   b. Click **New Index**.

      The **New Index** dialog box appears.



New Index

General Settings

Index Name:

Set index name (e.g., INDEX_NAME). Search using index=INDEX_NAME.

Index Data Type: Events | Metrics

The type of data to store (event-based or metrics).

Home Path: optional

Hot/warm db path. Leave blank for default ($SPLUNK_DB/INDEX_NAME/db).

Cold Path: optional

Cold db path. Leave blank for default ($SPLUNK_DB/INDEX_NAME/colddb).

Thawed Path: optional

Thawed/resurrected db path. Leave blank for default ($SPLUNK_DB/INDEX_NAME/thaweddb).

Figure 3-4: 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 Sample Application and Container logs.

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

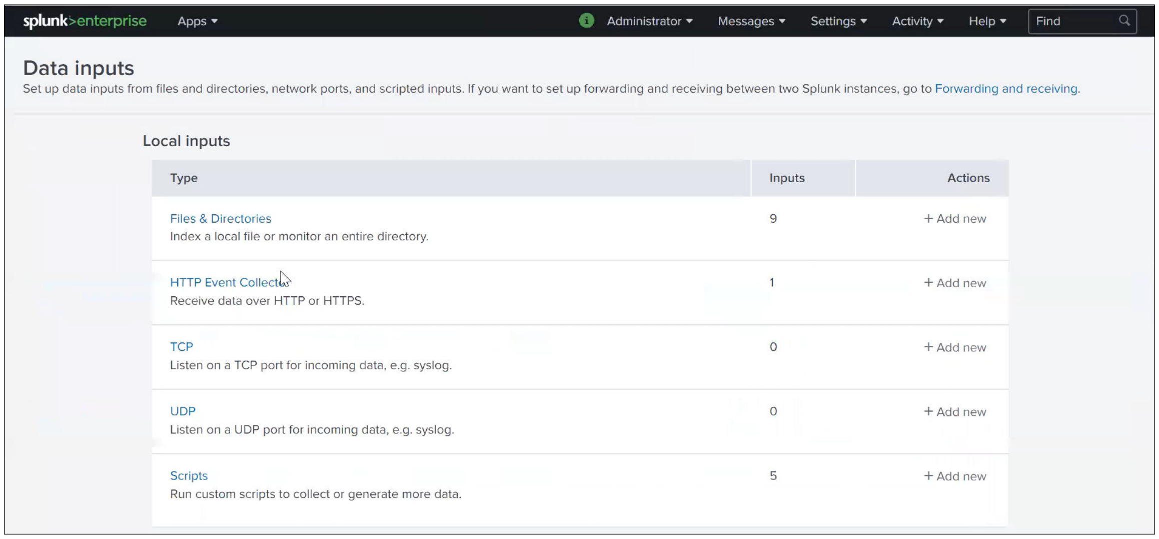
For more information about creating an index in Splunk Enterprise, refer to the section *Create custom indexes* in the Splunk documentation.

4. Perform the following steps to set up HTTP Event Collector (HEC) in Splunk Web. The HEC enables you to receive the Sample Application 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.



Data inputs

Set up data inputs from files and directories, network ports, and scripted inputs. If you want to set up forwarding and receiving between two Splunk instances, go to Forwarding and receiving.

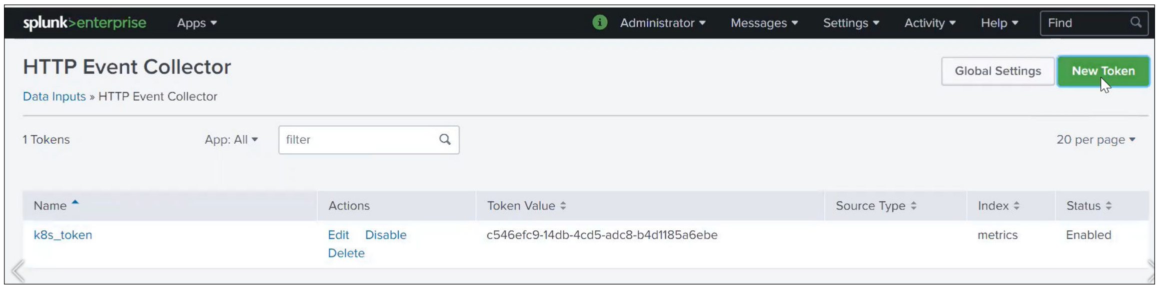
Local inputs

Type	Inputs	Actions
Files & Directories Index a local file or monitor an entire directory.	9	+ Add new
HTTP Event Collector Receive data over HTTP or HTTPS.	1	+ Add new
TCP Listen on a TCP port for incoming data, e.g. syslog.	0	+ Add new
UDP Listen on a UDP port for incoming data, e.g. syslog.	0	+ Add new
Scripts Run custom scripts to collect or generate more data.	5	+ Add new

Figure 3-5: Data inputs Screen

   b. Click **HTTP Event Collector**.

      The **HTTP Event Collector** screen appears.



HTTP Event Collector

Data Inputs » HTTP Event Collector

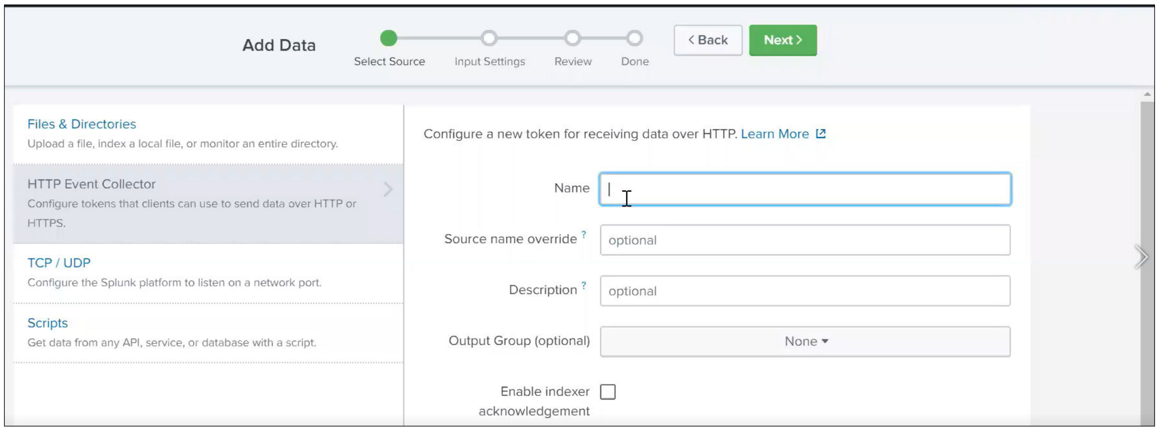
1 Tokens

Name	Actions	Token Value	Source Type	Index	Status
k8s_token	Edit Disable Delete	c546efc9-14db-4cd5-adc8-b4d1185a6ebe		metrics	Enabled

Figure 3-6: HTTP Event Collector Screen

   c. Click **New Token**.

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


Add Data
Select Source
Input Settings
Review
Done
< Back
Next >
Files & Directories
Upload a file, index a local file, or monitor an entire directory.
HTTP Event Collector
Configure tokens that clients can use to send data over HTTP or HTTPS.
TCP / UDP
Configure the Splunk platform to listen on a network port.
Scripts
Get data from any API, service, or database with a script.

Configure a new token for receiving data over HTTP. Learn More

Name

Source name override: optional

Description: optional

Output Group (optional): None

Enable indexer acknowledgement

Figure 3-7: Select Source Screen

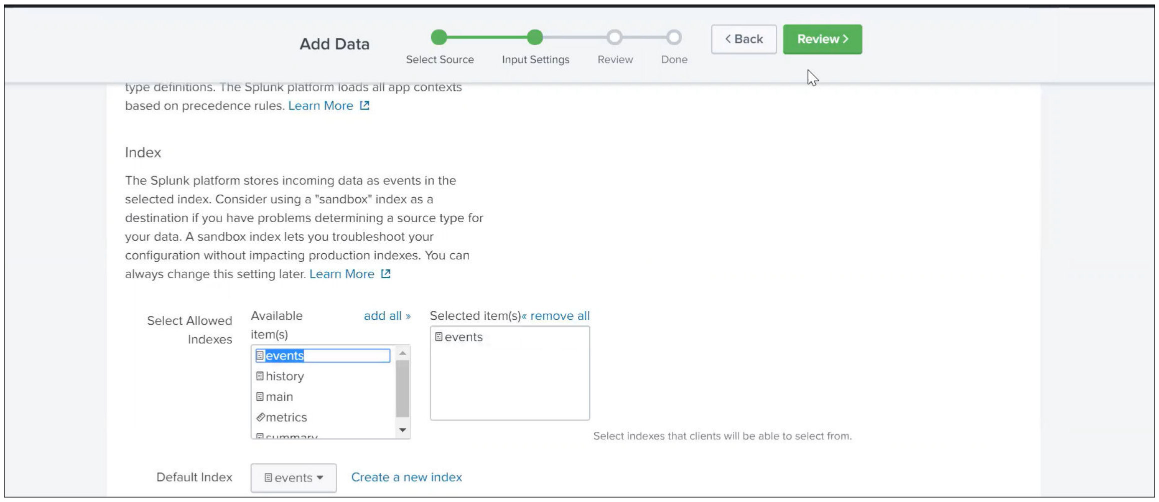
d. In the **Name** field, type a name for the token.

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.



type definitions. The Splunk platform loads all app contexts based on precedence rules. Learn More

Index

The Splunk platform stores incoming data as events in the selected index. Consider using a "sandbox" index as a destination if you have problems determining a source type for your data. A sandbox index lets you troubleshoot your configuration without impacting production indexes. You can always change this setting later. Learn More

Select Allowed Indexes — Available item(s): events, history, main, metrics, summary (add all ») — Selected item(s): events (« remove all)

Select indexes that clients will be able to select from.

Default Index: events — Create a new index

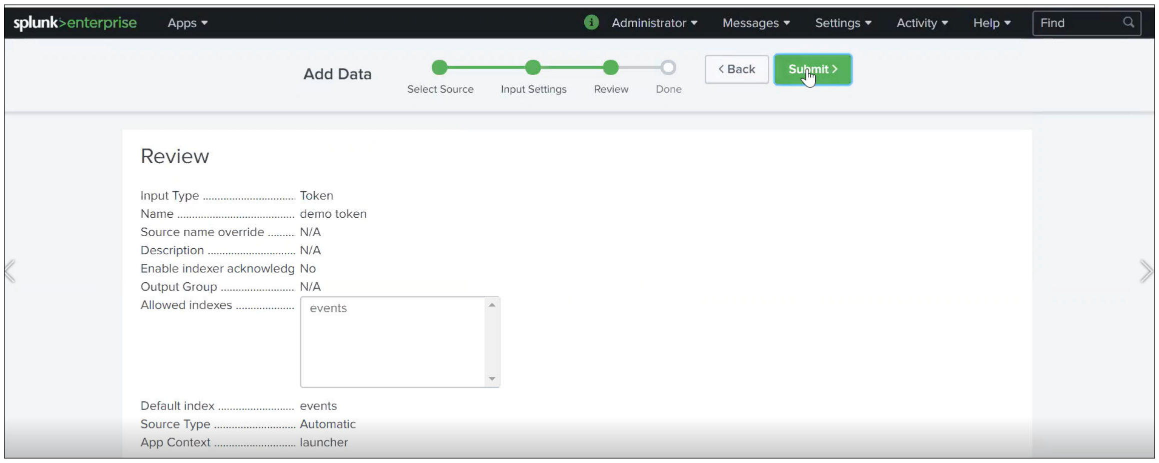
Figure 3-8: Input Settings Screen

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.



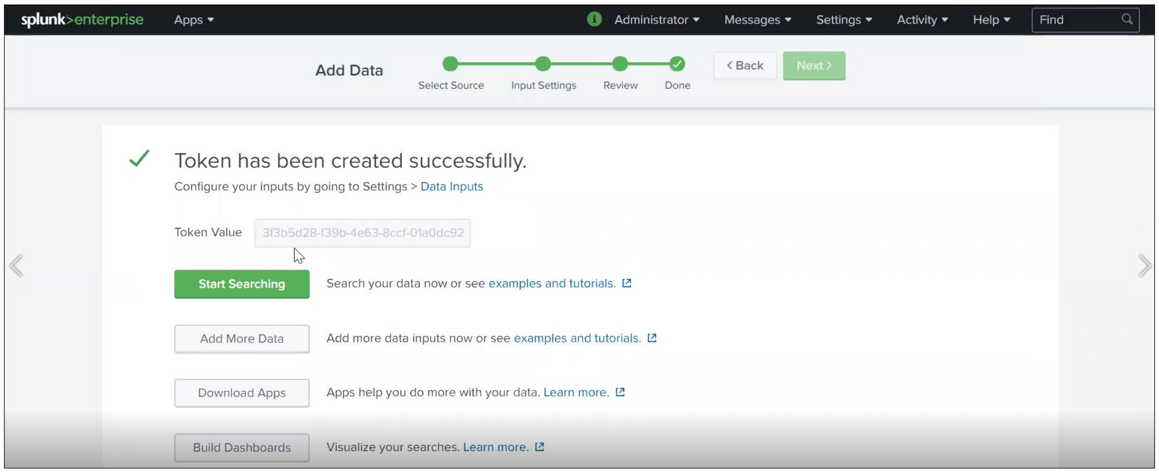
Review

Input Type	Token
Name	demo token
Source name override	N/A
Description	N/A
Enable indexer acknowledg	No
Output Group	N/A
Allowed indexes	events
Default index	events
Source Type	Automatic
App Context	launcher

Figure 3-9: 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.



Token has been created successfully.

Configure your inputs by going to Settings > Data Inputs

Token Value 3f3b5d28-f39b-4e63-8ccf-01a0dc92

Start Searching — Search your data now or see examples and tutorials.

Add More Data — Add more data inputs now or see examples and tutorials.

Download Apps — Apps help you do more with your data. Learn more.

Build Dashboards — Visualize your searches. Learn more.

Figure 3-10: Done Screen

5. Perform the following steps to configure the global settings for the HEC.

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

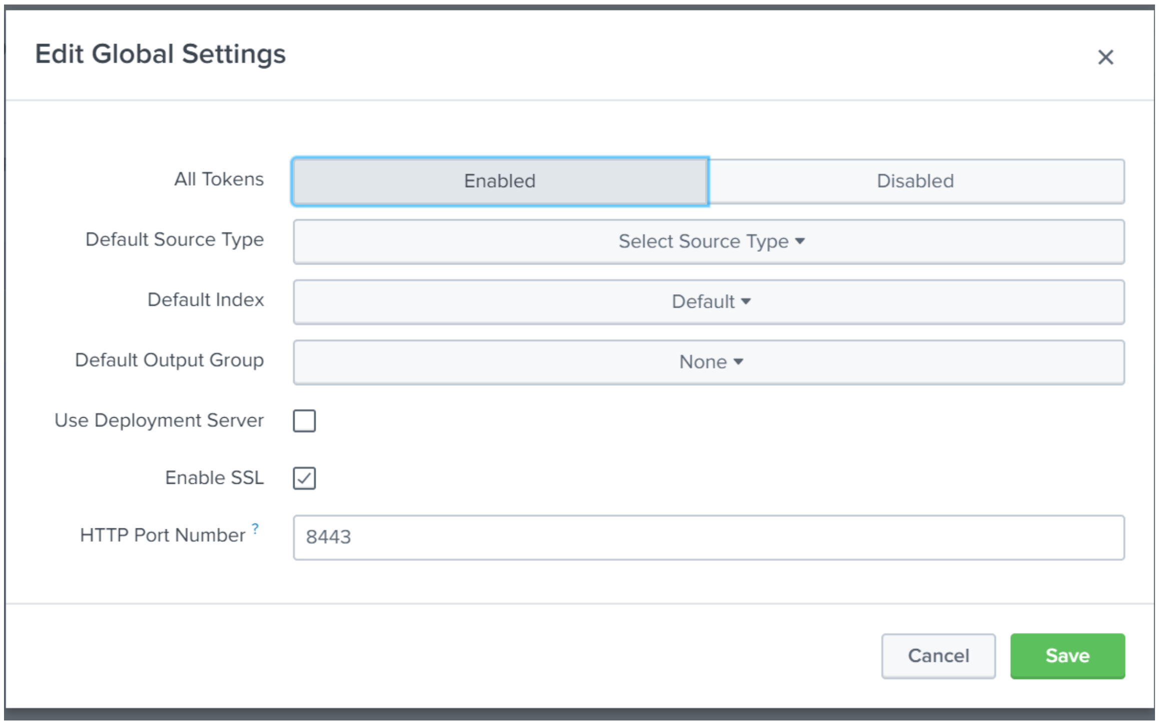
      The **Data inputs** screen appears.

   b. Click **HTTP Event Collector**.

      The **HTTP Event Collector** screen appears.

   c. Click **Global Settings**.

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


Edit Global Settings

All Tokens	Enabled / Disabled
Default Source Type	Select Source Type
Default Index	Default
Default Output Group	None
Use Deployment Server	☐
Enable SSL	✓
HTTP Port Number	8443

Figure 3-11: 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.

3.4 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 same Kubernetes cluster where you have deployed the Sample Application container.

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

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 displays a sample *custom-values.yaml* file.

```
global:
  splunk:
    hec:
      protocol: https
      insecureSSL: true
      token: <Token value>
```

```
          host: 10.0.0.100
          port: 8443
          indexName: <Index name>
```

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
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.

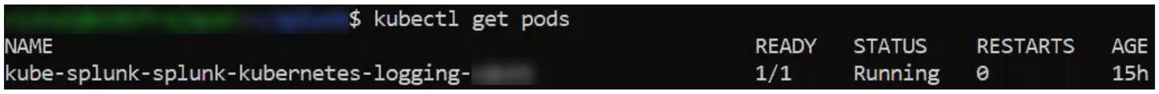
3. Run the following command to deploy the Splunk Connect for Kubernetes Helm chart on the Kubernetes cluster.

```
helm install kube-splunk -n kube-system -f custom-values.yaml https://github.com/splunk/splunk-connect-for-kubernetes/releases/download/1.4.2/splunk-kubernetes-logging-1.4.2.tgz
```

4. 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.



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

Chapter 4

Protecting Data Using the Sample Application Container Deployment

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

4.1 Running Security Operations

This section describes how you can use the Sample Application instances running on the Kubernetes cluster to protect the data that is sent by a REST API client.

➤ To run security operations:

1. If you are not using TLS authentication between the Sample Application instance and the REST API client, then send the following cURL request from the Linux instance.

```
curl --location --request POST 'http://<IP address of Ingress>/protect' --header
'Host: prod.example.com' --header 'Content-Type: application/json' --data-raw
'{ "dataElement": "Alphanum", "policyUser": "user1", "input": [ "SampleAppJava",
"OnKubernetes" ] }'
```

   You can obtain the IP address of the ingress after you run the *kubectl get ingress* command, after deploying the sample application on the Kubernetes cluster.

   For more information about the IP address associated with the ingress object, refer to *step 7b* in the section *Deploying the Sample Application Container on the Kubernetes Cluster*.

   The Sample Application container instance internally sends this request to the Application Protector Java, and returns the protected value.

   If you want to unprotect the data, then you can run the following command.

```
curl --location --request POST 'http://<IP address of Ingress>/unprotect' --header
'Host: prod.example.com' --header 'Content-Type: application/json' --data-raw
'{ "dataElement": "Alphanum", "policyUser": "user1", "input": [ "iaDDNBdH6EI8U",
"9jB7cRSuk98B" ] }'
```

   If you want to reprotect the data, then you can run the following command.

```
curl --location --request POST 'http://<IP address of Ingress>/reprotect' --
header 'Host: prod.example.com' --header 'Content-Type: application/json' --data-
raw '{ "dataElement": "Alphanum", "newDataElement": "Alphanum1", "policyUser":
"user1", "input": [ "iaDDNBdH6EI8U", "9jB7cRSuk98B" ] }'
```

2. If you are using TLS authentication between the Sample Application instance and the REST API client, then perform the following steps.

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

      In Windows, in the DNS Server Settings, specify the IP address of the Inbound Endpoint that you have created in *step 3* of the section *Appendix B: 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 B: Creating a DNS Entry for the ELB Hostname in Route53*.

   b. Send the following cURL request from the Linux instance.

      ```
      curl --location --request POST 'https://<host name>/protect' --header 'Content-
      Type: application/json' --data-raw '{ "dataElement": "Alphanum", "policyUser":
      "user1", "input": [ "SampleAppJava", "OnKubernetes" ] }' --cacert ./iap-ca.crt --
      cert ./iap-client.crt --key ./iap-client.key
      ```

      In the above example, the *<host name>* value refers to the host name created using the Route53 service.

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

      The Sample Application container instance internally sends this request to the Application Protector Java, and returns the protected value.

      If you want to unprotect the data, then you can run the following command.

      ```
      curl --location --request POST 'https://<host name>/unprotect' --header 'Content-
      Type: application/json' --data-raw '{ "dataElement": "Alphanum", "policyUser":
      "user1", "input": [ "iaDDNBdH6EI8U", "9jB7cRSuk98B" ] }' --cacert ./iap-ca.crt --
      cert ./iap-client.crt --key ./iap-client.key
      ```

      If you want to reprotect the data, then you can run the following command.

      ```
      curl --location --request POST 'https://<host name>/reprotect' --header
      'Content-Type: application/json' --data-raw '{ "dataElement": "Alphanum",
      "newDataElement": "Alphanum1", "policyUser": "user1", "input": [ "iaDDNBdH6EI8U",
      "9jB7cRSuk98B" ] }' --cacert ./iap-ca.crt --cert ./iap-client.crt --key ./iap-
      client.key
      ```

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

4.2 Viewing Logs in Splunk

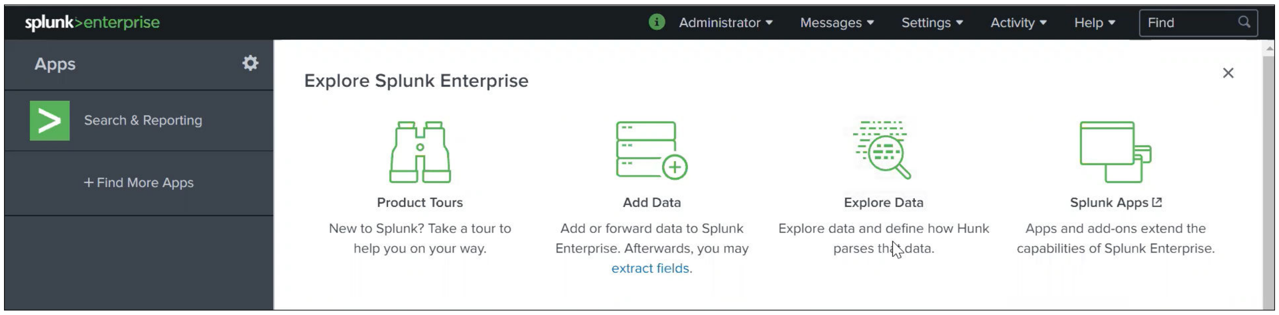
This section describes how you can view the Application Protector Java and Container logs in Splunk Enterprise.

Important: You require a non-privileged role to perform this task.

➤ 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.

   New to Splunk? Take a tour to help you on your way.

   Add or forward data to Splunk Enterprise. Afterwards, you may extract fields.

   Explore data and define how Hunk parses the data.

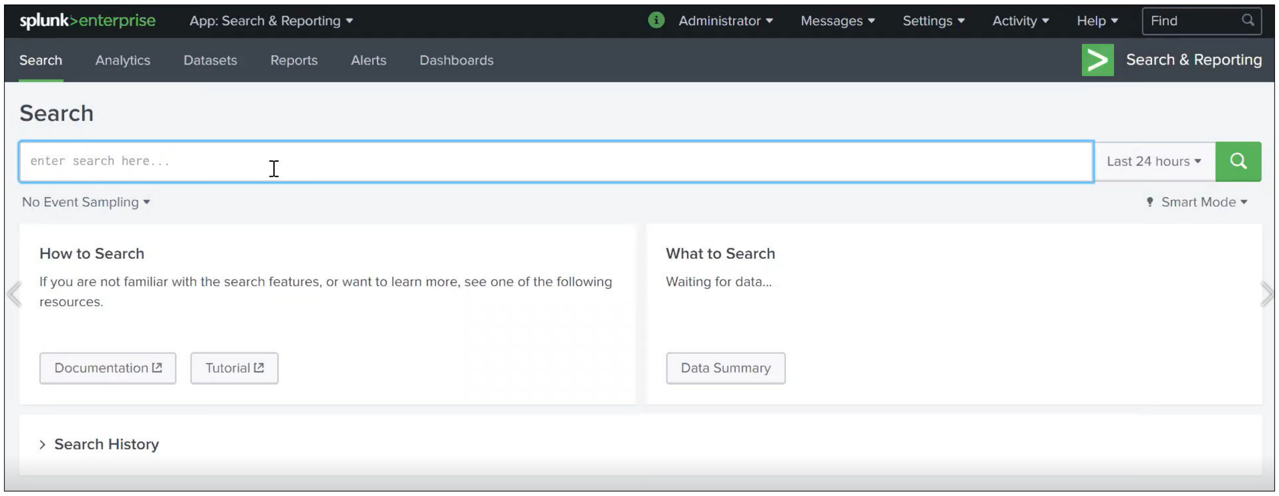
   Apps and add-ons extend the capabilities of Splunk Enterprise.

   Figure 4-1: 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.

   If you are not familiar with the search features, or want to learn more, see one of the following resources.

   Waiting for data...

   Figure 4-2: Search Screen

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

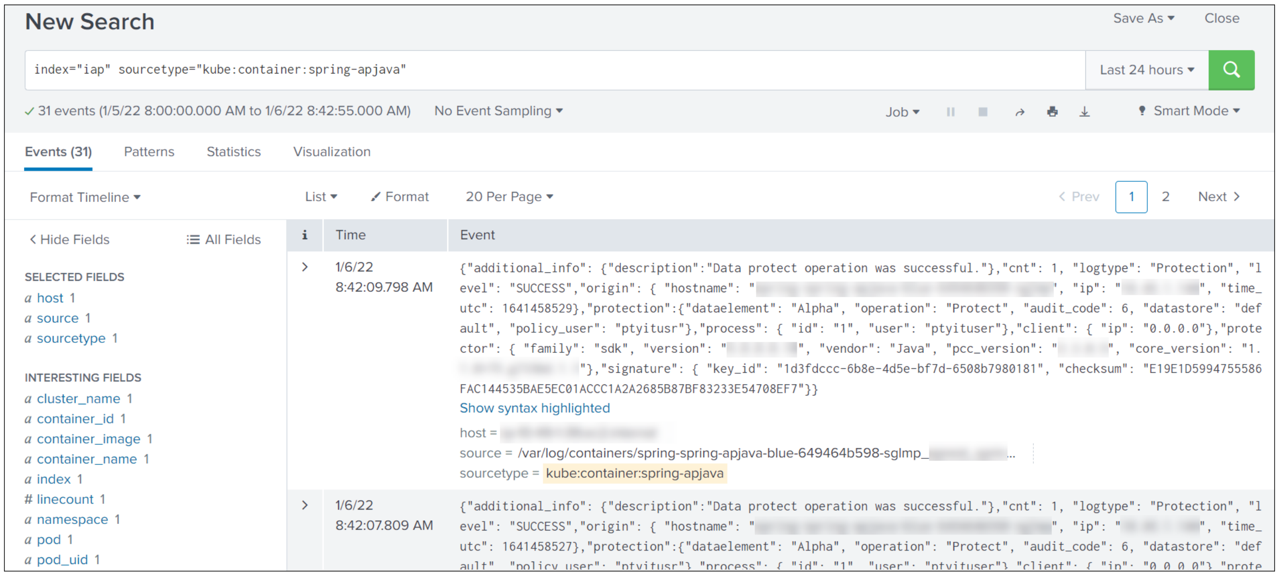
```
index="<Index_name>" sourcetype="kube:container:<Container_name>"
```

   For example:

```
index="events" sourcetype="kube:container:spring-app-java"
```

5. Press **Enter**.

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



```
index="iap" sourcetype="kube:container:spring-apjava"
```

✓ 31 events (1/5/22 8:00:00.000 AM to 1/6/22 8:42:55.000 AM)

SELECTED FIELDS
a host 1
a source 1
a sourcetype 1

INTERESTING FIELDS
a cluster_name 1
a container_id 1
a container_image 1
a container_name 1
a index 1
# linecount 1
a namespace 1
a pod 1
a pod_uid 1

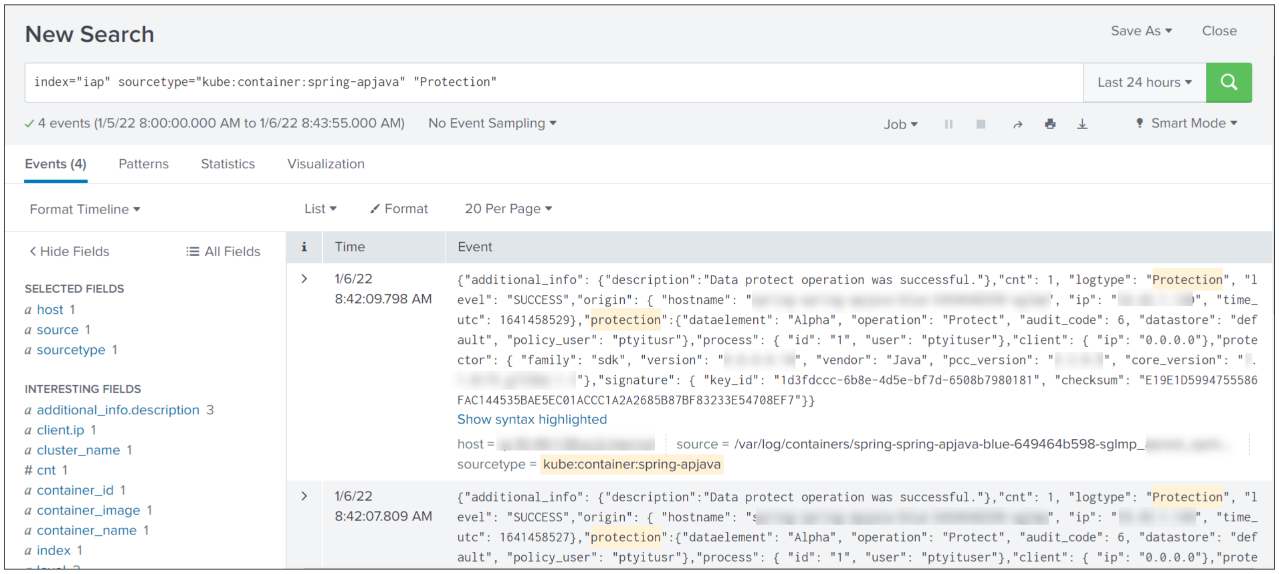
```
1/6/22
8:42:09.798 AM
{"additional_info": {"description":"Data protect operation was successful."},"cnt": 1, "logtype": "Protection", "level": "SUCCESS","origin": { "hostname": "[redacted]", "ip": "[redacted]", "time_utc": 1641458529},"protection":{"dataelement": "Alpha", "operation": "Protect", "audit_code": 6, "datastore": "default", "policy_user": "ptyitusr"},"process": { "id": "1", "user": "ptyituser"},"client": { "ip": "0.0.0.0"},"protector": { "family": "sdk", "version": "[redacted]", "vendor": "Java", "pcc_version": "[redacted]", "core_version": "1.[redacted]"},"signature": { "key_id": "1d3fdccc-6b8e-4d5e-bf7d-6508b7980181", "checksum": "E19E1D5994755586FAC144535BAE5EC01ACCC1A2A2685B87BF83233E54708EF7"}}
host = [redacted]
source = /var/log/containers/spring-spring-apjava-blue-649464b598-sglmp_[redacted]...
sourcetype = kube:container:spring-apjava

1/6/22
8:42:07.809 AM
{"additional_info": {"description":"Data protect operation was successful."},"cnt": 1, "logtype": "Protection", "level": "SUCCESS","origin": { "hostname": "[redacted]", "ip": "[redacted]", "time_utc": 1641458527},"protection":{"dataelement": "Alpha", "operation": "Protect", "audit_code": 6, "datastore": "default", "policy_user": "ptyitusr"},"process": { "id": "1", "user": "ptyituser"},"client": { "ip": "0.0.0.0"},"prote
```

6. If you want to view only the logs that are related to the Application Protector Java, 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:spring-app-java" "Protection"
```

7. Press **Enter**.

   The search results display only the logs that are related to the Application Protector Java.



```
index="iap" sourcetype="kube:container:spring-apjava" "Protection"
```

✓ 4 events (1/5/22 8:00:00.000 AM to 1/6/22 8:43:55.000 AM)

SELECTED FIELDS
a host 1
a source 1
a sourcetype 1

INTERESTING FIELDS
a additional_info.description 3
a client.ip 1
a cluster_name 1
# cnt 1
a container_id 1
a container_image 1
a container_name 1
a index 1

```
1/6/22
8:42:09.798 AM
{"additional_info": {"description":"Data protect operation was successful."},"cnt": 1, "logtype": "Protection", "level": "SUCCESS","origin": { "hostname": "[redacted]", "ip": "[redacted]", "time_utc": 1641458529},"protection":{"dataelement": "Alpha", "operation": "Protect", "audit_code": 6, "datastore": "default", "policy_user": "ptyitusr"},"process": { "id": "1", "user": "ptyituser"},"client": { "ip": "0.0.0.0"},"protector": { "family": "sdk", "version": "[redacted]", "vendor": "Java", "pcc_version": "[redacted]", "core_version": "[redacted]"},"signature": { "key_id": "1d3fdccc-6b8e-4d5e-bf7d-6508b7980181", "checksum": "E19E1D5994755586FAC144535BAE5EC01ACCC1A2A2685B87BF83233E54708EF7"}}
host = [redacted]   source = /var/log/containers/spring-spring-apjava-blue-649464b598-sglmp_[redacted]
sourcetype = kube:container:spring-apjava

1/6/22
8:42:07.809 AM
{"additional_info": {"description":"Data protect operation was successful."},"cnt": 1, "logtype": "Protection", "level": "SUCCESS","origin": { "hostname": "[redacted]", "ip": "[redacted]", "time_utc": 1641458527},"protection":{"dataelement": "Alpha", "operation": "Protect", "audit_code": 6, "datastore": "default", "policy_user": "ptyitusr"},"process": { "id": "1", "user": "ptyituser"},"client": { "ip": "0.0.0.0"},"prote
```

4.3 Autoscaling the Protegrity Reference Deployment

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

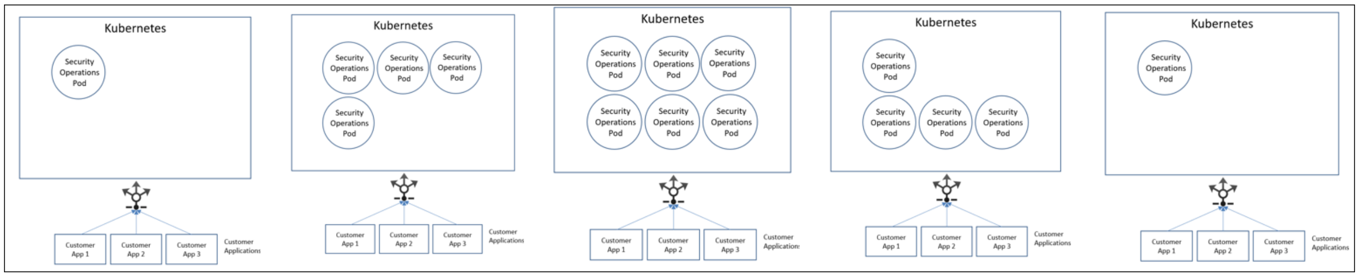


Figure 4-3: Autoscaling Kubernetes Cluster

As illustrated in this figure, the Customer Applications experience an increase and a decrease in workload as required by the businesses. 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 Sample Application appliances to meet the business need and then remove them if not required. Kubernetes performs the task of provisioning the Sample Application instances automatically.

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

Chapter 5

Appendix A: Sample Application Helm Chart Details

This section describes the details of the Sample Application Helm chart structure and the entities that the user needs to modify for adapting the chart for their environments.

5.1 Chart.yaml

The *Chart.yaml* file contains information about Helm versions. These values can be left as default.

```
apiVersion: v1
description: A Spring boot application chart for Kubernetes
name: spring-apjava
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 specifics of each environment.

The following sections will explain the details of each field that needs to be replaced.

5.2.1 Image Pull Secrets

This section specifies the credentials that are 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.

Important: Both privileged and non-privileged roles can perform this task.

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

This section specifies the values that indicate how naming for releases are managed for deployment.

> **Note:** 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 a combination of the ***`nameOverride`*** parameter value and the Helm chart release name.

If both the values are kept 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.

```
springappImage:
  repository: SPRING_APJAVA_IMAGE_REPOSITORY
  tag: SPRING_APJAVA_IMAGE_TAG
  pullPolicy: Always
  debug: false # change debug to true to get container debug logs on STDOUT.
```

The following list provides an overview of the *springappImage* parameter, which refers to the details for the Sample 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 the tag name as the value in the *tag* field.

- *tag* – Refers to the tag mentioned at the time of image upload.
- *debug* - Sets the value of this field to *true*, if you want debug logs for the Sample container. By default, the value of this field is set to *false*.

> **Note:** If you are deploying a Customer Application, then you must specify the name of the Customer Application image.

5.2.4 Resources

This section specifies the resource limits that can be set for the containers in a pod.

```
initContainerResources:
# Important: Set the resource based on the policy metadata dump size.
# Default is set for a policy metadata dump size upto ~350Mb.
  limits:
    cpu: 300m
    memory: 3096Mi
  requests:
    cpu: 200m
    memory: 512Mi

springContainerResources:
  limits:
    cpu: 500m
    memory: 512Mi
  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 information about the resource parameters, refer to the section *Managing Resources for Containers* in the Kubernetes documentation.

5.2.5 Pod Service Account

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

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

If you want to use AWS S3 to store or AWS KMS to decrypt the policy snapshot, then you must specify the name of the service account that you have created for the pod in the Kubernetes Cluster. This service account is mapped to the following policies:

- *AmazonS3ReadOnlyAccess* policy to ensure that only the pod, where the Spring Boot application has been deployed, is able to download the policy snapshot from the S3 bucket.
- *KMSDecryptAccess* policy to allow the user to decrypt the policy packages that have been encrypted using AWS Customer Master Key

If you want to use AWS EFS to store and PBE to decrypt the policy snapshot and you also applying a PSP, then you must specify the name of the corresponding EFS service account that you have created for the pod in the Kubernetes Cluster. Otherwise, leave the value blank.

5.2.6 Liveliness and Readiness Probe Settings

This section describes the values that reflect the intervals required to run health checks for the Sample Application container images. Users are recommended not to 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
```

5.2.7 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.

5.2.8 Container Security Context

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

```
## set the Spring App Container's security context object
## leave the field empty if not applicable
springContainerSecurityContext:
  capabilities:
    drop:
    - ALL
  allowPrivilegeEscalation: false
  privileged : false
  runAsNonRoot : true
  readOnlyRootFilesystem: true
```

The default values in the Container Security Context ensure that the container is not run in privileged mode.

Note: It is recommended not to edit the default values.

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.

5.2.9 Persistent Volume Claim

This section specifies the value of the persistent volume claim that will be used to store the immutable policy package.

```
## persistent volume name e.g. nfs-pvc
pvcName: PVC_NAME
```

Specify the value of the persistent volume claim that will be used to store the policy dump.

Important: You require a non-privileged role to perform this task.

For more information about persistent volumes and persistent volume claims, refer to the section *Persistent Volumes* in the Kubernetes documentation.

5.2.10 Private Key Source

This section provides information about the method used for encrypting the policy package.

```
## Choose your private key helper. Currently we support [PKCS5, AWS_KMS]
privateKeySource: PKCS5
```

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

5.2.11 PBE Secrets

This section provides information for the passphrase and the salt that are used to generate a key.

```
## k8s secret containing passphrase for decrypting policy
pbeSecrets: PBE_SECRET_NAME
```

The Sample Application container uses the value specified in the *pbeSecrets* parameter to decrypt the policy.

Note: By default, the Passphrase-Based Encryption is used to encrypt the policy package. If you want to use the Azure Key Vault for encrypting the policy package, then you must comment out the *pbeSecrets* entry in the *values.yaml* file or leave the value of the *pbeSecrets* parameter as blank.

Important: Ensure that the password complexity meets the following requirements:

- The password must contain a minimum of 10 characters and a maximum of 128 characters
- The user should be able to specify Unicode characters, such as Emoji, in the password
- The user should avoid commonly used passwords, such as, *123456789*, *11111111*, *password*, and *qwertyuiop*

5.2.12 Private Key Secret

This section provides information for the AWS private key secret.

```
privateKeySecret: PKEY_SECRET_NAME
```

The Sample Application container uses the value specified in the *privateKeySecret* parameter to decrypt the policy.

5.2.13 Deployment

This section provides an overview on the configurations that refer to the policy package information for the current release. The first release is considered to be *Blue* deployment.

When the ***`blueDeployment/enabled: true`*** parameter is enabled, the configurations mentioned under the policy section are deployed.

```
blueDeployment:
  enabled: true
```

```
### Policy metadata for Blue Deployments
blueDeployment:
  enabled: true
  securityConfigSource: VolumeMount
  policy:
    # absolute path in PV where the policy dump file will be stored. <eg. /test/xyz/policy >
    filePath: ABSOULTE_POLICY_PATH

## Policy metadata for Green Deployments
greenDeployment:
  enabled: false
  securityConfigSource: VolumeMount
  policy:
    # absolute path in PV where the policy dump file will be stored. <eg. /test/xyz/policy >
    filePath: ABSOULTE_POLICY_PATH
```

The users are required to update the file path where the policy package has been uploaded. Note that all entries are case-sensitive.

5.2.14 Autoscaling

This section provides an overview for the parameters used for autoscaling of pods on a cluster.

```
## specify the initial no. of springapp 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 at the start of the deployment.
- *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* setting.

5.2.15 Service Configuration

This section provides configurations for the REST service to be used on the cluster running the Sample Application containers.

```
## specify the ports exposed in your springapp 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.
springappService:

  # 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/azure-load-balancer-internal: "true"

  name: "restapi"
  port: 8080
  targetPort: 8080
```

You can specify the value of the *springappService/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 following is a list of parameters that can be configured for the REST service:

- *name* - Specifies the name of the Kubernetes service. The name must contain only lowercase alphanumeric characters, which is based on the standard Kubernetes naming convention.
- *port* - Specifies the port on which you want to expose the service externally.
- *targetPort* - Specifies the port on which the Sample application is running inside the Docker container.

5.2.16 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-iap

  ## if required, specify a different Ingress Controller and related annotations here
  annotations:
    ## Enable client certificate authentication
    #nginx.ingress.kubernetes.io/auth-tls-verify-client: "on"

    ## Name of the Secret along with the Namespace, where its created,
    ## that contains the full CA chain ca.crt,
    ## that is enabled to authenticate against this Ingress
    #nginx.ingress.kubernetes.io/auth-tls-secret: "<NAMESPACE>/ca-secret"

  ## specify the host names and paths for Ingress rules
  ## prod and staging http paths can be used as optional fields.
  hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8080
        prod: "/"
    ## 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: 8080
    #     prod: "/"
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8080
    #     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: example-tls
  #    hosts:
  #      - prod.example.com
```

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

```
ingress:
  enabled: true
```

When the parameter is set to *enabled*: *true*, the ingress controller will be based on the value used in the annotation *ingress.class*. 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 Application Protector Java services.

By default, the value of the *ingressClassName* parameter is set to *nginx-iap* to indicate that the NGINX Ingress Controller is used.

```
 # specify ingressClass cluster resource. It associates which controller will implement the
resource.
  ingressClassName: nginx-iap

  ## if required, specify a different Ingress Controller and related annotations here
  annotations:
    ## Enable client certificate authentication
    #nginx.ingress.kubernetes.io/auth-tls-verify-client: "on"

    ## Name of the Secret along with the Namespace, where its created,
    ## that contains the full CA chain ca.crt,
    ## that is enabled to authenticate against this Ingress
    #nginx.ingress.kubernetes.io/auth-tls-secret: "<NAMESPACE>/ca-secret"
```

If you want to enable TLS mutual authentication between the REST API client and the AWS load balancer, then you must uncomment the above code snippet as follows:

```
## Enable client certificate authentication
    nginx.ingress.kubernetes.io/auth-tls-verify-client: "on"
    ## Name of the Secret along with the Namespace, where its created,
    ## that contains the full CA chain ca.crt,
    ## that is enabled to authenticate against this Ingress
    nginx.ingress.kubernetes.io/auth-tls-secret: "<NAMESPACE>/ca-secret"
```

This ensures that when the client sends a request to the AWS load balancer, the client is also authenticated.

You also need to run the following command to generate the CA secret.

```
kubectl create -n iap secret generic ca-secret --from-file=ca.crt=iap-ca.crt
```

You then need to specify the name of the generated CA secret as the value of the *nginx.ingress.kubernetes.io/auth-tls-secret* field. In addition, ensure that you specify the namespace in the Kubernetes cluster where you have generated the CA certificate secret.

For example, if you have generated the CA certificate secret in the same namespace where you have deployed the Sample Application container, then you must set the namespace as shown in the following code snippet:

```
nginx.ingress.kubernetes.io/auth-tls-secret: iap/ca-secret"
```

```
## specify the host names and paths for Ingress rules
  ## prod and staging http paths can be used as optional fields.
  hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8080
        prod: "/"
    ## 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: 8080
    #     prod: "/"
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8080
    #     staging: "/"
```

The above section refers to URLs and ports referred to for *blue green* deployments.

- *port* – Refers to the port on which the Sample Application container is running inside the Docker container 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: "/"
    # - host: "randomhostname"
    #   httpPaths:
    #   - port: 8080
    #     prod: "/"
```

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 the *prod* URLs running on one hostname and the *staging* URLs running on another hostname.

If you do not the specify hostname in the *host* "" field, then the ingress can be accessed on the host name of the AWS Elastic Load Balancer directly.

```
## If TLS is terminated on the Ingress Controller Load Balancer,
  ## K8s TLS Secret containing the certificate and key must also be provided
  # tls:
  #  - secretName: example-tls
  #    hosts:
  #      - prod.example.com
```

If you are terminating the TLS connection on the NGINX Ingress Controller, then you must uncomment the above code snippet as follows.

```
## If TLS is terminated on the Ingress Controller Load Balancer,
  ## K8s TLS Secret containing the certificate and key must also be provided
    tls:
     - secretName: example-tls
       hosts:
         - prod.example.com
```

You also need to run the following command to generate the TLS secret.

```
kubectl create -n iap secret generic example-tls --from-file=tls.crt=server.crt --
from-file=tls.key=server.key --from-file=ca.crt=iap-ca.crt
```

Chapter 6

Appendix B: 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.

Important: You require a non-privileged role to perform this task.

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 & 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.

      Specify this IP address in the */etc/resolv.conf* file in Linux in *step 2a* 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.

Chapter 7

Appendix C: Applying the NSS Wrapper to the Customer Application Container Image

If you run the Customer Application container image with a random user ID, then the */etc/password* file does not contain an entry for the user ID. As a result, the Application Protector Java cannot determine the user name of the application. You can fix this issue by installing the *nss_wrapper* library on the Customer Application container image, and create a file that contains the mapping between the user name and the user ID, as part of the startup command of the image.

This section describes how you can apply the NSS wrapper to the Customer Application container image.

For more information about the *nss_wrapper* library, refer to the section *The nss_wrapper library*.

To apply the NSS wrapper to the Customer Application container image:

1. Create a file named *docker-entrypoint.sh*, which is used as the default argument for the *ENTRYPOINT* instruction in the Dockerfile of the container image.

   Include the following content in the *docker-entrypoint.sh* file.

   ```
   #!/bin/bash
   export USER_ID=$(id -u)
   export GROUP_ID=$(id -g)
   envsubst < /etc/passwd.template > /tmp/passwd
   export LD_PRELOAD=/usr/lib64/libnss_wrapper.so
   export NSS_WRAPPER_PASSWD=/tmp/passwd
   export NSS_WRAPPER_GROUP=/etc/group

   exec java \
   -cp <APPLICATION_LIBS>:/opt/protegrity/applicationprotector/java/lib/* \
        <APPLICATION_NAME>
   ```

   This command creates file named *passwd.template*.

   The *ENTRYPOINT* instruction specifies the startup command for executing the container image.

   For more information about Dockerfiles, refer to the *Docker* documentation.

2. Modify the *passwd.template* file to include the mapping between the user name of the container application and the user ID that is used to run the container application.

   The following snippet shows a sample *passwd.template* file that has been used for the Sample Application.

   ```
   root:x:0:0:root:/root:/bin/bash
   bin:x:1:1:bin:/bin:/sbin/nologin
   daemon:x:2:2:daemon:/sbin:/sbin/nologin
   adm:x:3:4:adm:/var/adm:/sbin/nologin
   lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
   sync:x:5:0:sync:/sbin:/bin/sync
   shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
   ```

```
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
systemd-network:x:192:192:systemd Network Management:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
ptyituser:x:${USER_ID}:${GROUP_ID}:ptyituser:/home/ptyituser:/bin/bash
```

You need to replace the *ptyituser* value with the user name of your Customer Application.

3. Modify the Dockerfile of your Customer Application container image to install the *nss_wrapper* and *gettext* packages, copy the *docker-entrypoint.sh* and *passwd.template* files to the image, and specify the *docker-entrypoint.sh* file as a default argument to the *ENTRYPOINT* instruction.

```
...

# Install the nss_wrapper and gettext packages
RUN yum -y install nss_wrapper gettext

# Copy the passwd.template file to the container image
COPY passwd.template /etc/passwd.template

# Copy the docker-entrypoint.sh file to the container image
COPY docker-entrypoint.sh /opt/docker-entrypoint.sh
...

# Specify the docker-entrypoint.sh file as a default argument for the ENTRYPOINT
instruction
ENTRYPOINT [ "/opt/docker-entrypoint.sh" ]
...
```

Chapter 8

Appendix D: Using the Dockerfile to Build Custom Images

This section explains how to use the Dockerfiles, which are provided as part of the installation package, to build custom image for the Sample Application container. This enables you to use a base image of your choice, instead of using the default RHEL Universal Base Image, which is used as the default base image for the Protegrity images.

8.1 Creating Custom Images Using Dockerfile

This section describes the steps for creating a custom image for the Sample Application container, using the Dockerfile provided with the installation package.

Before you begin

Ensure that Maven 3.2 or later is installed on the machine on which you are creating the custom Docker image.

To create custom image:

1. Download the installation package.

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

   Important: The dependency packages required for building the Docker images are specified in the *HOW-TO-BUILD*, which is a part of the installation package. You must ensure that these dependency packages can be downloaded either from the Internet or from your internal repository.

   For more information about the *HOW-TO-BUILD* file, refer to the section *Verifying the Prerequisites*.

2. Perform the following steps to build a Docker image for the Sample Application container.

   a. Run the following command to extract the files from the *IAP-SAMPLE-APP_SRC_<version_number>.tgz* file to a directory.

   ```
   tar -xvf IAP-SAMPLE-APP_SRC_<version_number>.tgz -C <dir>
   ```

   The following files are extracted:

   - *IAP_RHUBI_SAMPLE-APP_DOCKERFILE_<version_number>*
   - *apjava-springboot-<version_number>.jar*
   - *docker-entrypoint.sh*
   - *passwd.template*

b. Switch to the directory where you have extracted the *IAP-SAMPLE-APP_SRC_<version_number>.tgz* package.

c. Execute the following command in the directory.

```
mvn clean install
```

The *apjava-springboot-0.1.0.jar* file appears in the *./target* directory.

d. Run the following command to create an artifacts directory where you have extracted the *IAP-SAMPLE-APP_SRC_<version_number>.tgz* package.

```
mkdir artifacts
```

e. Copy the *APJavaIMSetup_<OS>_x64_<App_Protector_version>.tgz* and *PolicyUtil_Linux_x64_<version>.tgz* from the installation package into the artifacts directory that you have created in *step 2d*.

f. Edit the *IAP_RHUBI_SAMPLE-APP_DOCKERFILE_<version_number>* file and in the following code snippet, replace the placeholder *RHEL-MINIMAL-UBI* with the name of the repository from where you want to download the custom RHEL UBI.

```
FROM RHEL-MINIMAL-UBI
```

g. Run the following command in the directory where you have extracted the contents of the *IAP-SAMPLE-APP_AWS_SRC_<version_number>.tar.gz* file.

```
docker build -t <image-name>:<image-tag> -f IAP_RHUBI_SAMPLE-
APP_DOCKERFILE_<version_number> .
```

For more information the Docker build command, refer to the *Docker* documentation.

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

h. Run the following command to list the Sample Application container image.

```
docker images
```

i. Push the Sample Application container image to your preferred Container Repository.

For more information about pushing an image to the repository, refer to the section *Uploading the Images to the Container Repository*.