Protegrity REST Container 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 REST 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 REST Container, which is based on the Application Protector form factor that Protegrity have been delivering for several years.
  - The AP REST Container is a standard Docker Container that is familiar and expected in cloud deployments.
  - The AP REST Container form factor makes the container a lightweight deployment of AP REST.
- 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 REST Container with Policy and instantiating a new one.
  - Changes to Policy or the AP REST Container itself 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 auto scale 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 auto scale when the node thresholds are reached.

- 3rd Party Integration
- ESA Connectivity Not Required: The important implication is that the ESA is no longer required to be up and running when the Kubernetes runtime has all the required components and can auto scale when needed.

1.3 Architecture and Components

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

The IAP-REST 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 AP-REST Container on a Kubernetes cluster, which uses the policy snapshot created in stage 1.

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

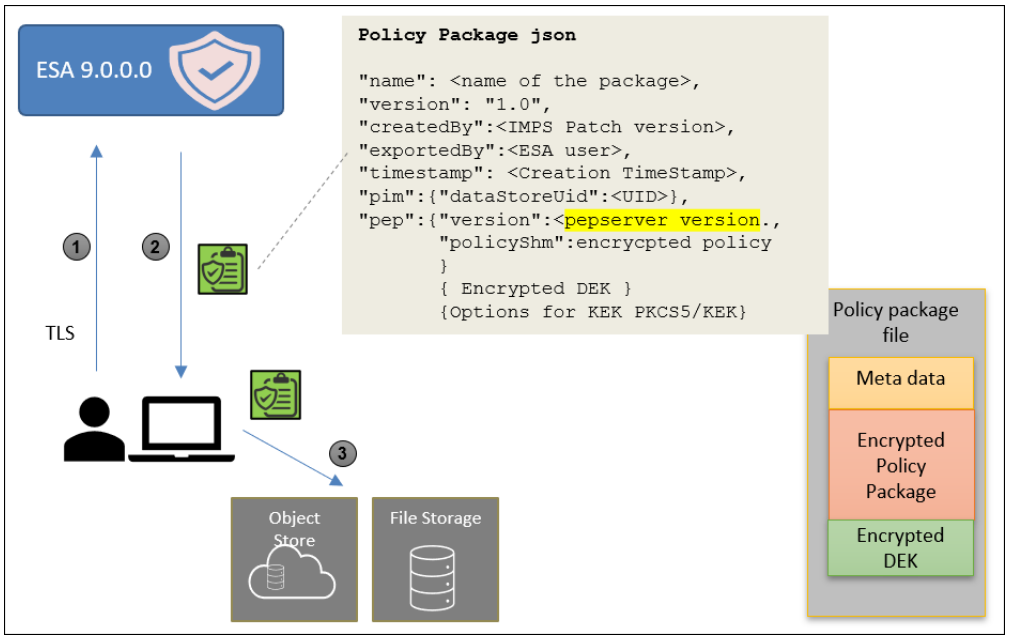


Figure 1-1: Stage 1: Deploying the Get ESA Policy Container

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 AP-REST Container on a Kubernetes cluster.

Stage 2: Deploying the AP-REST Container

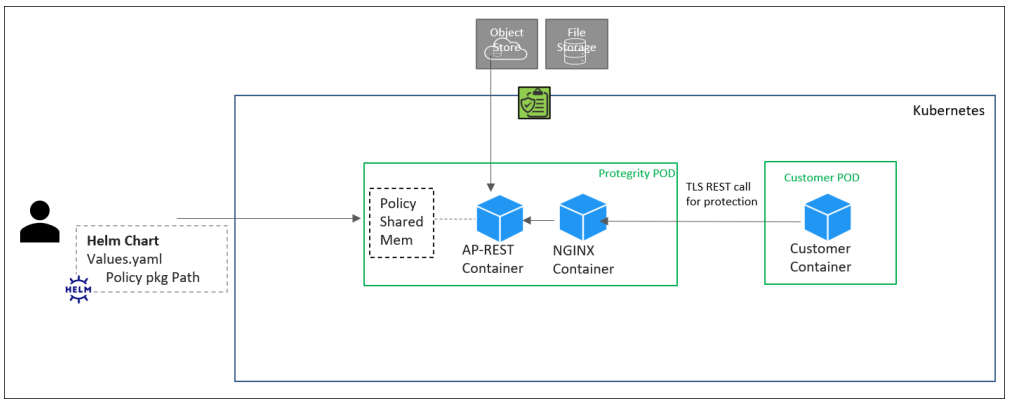


Figure 1-2: Pattern 1 - Communication between Pods in the same Kubernetes Cluster

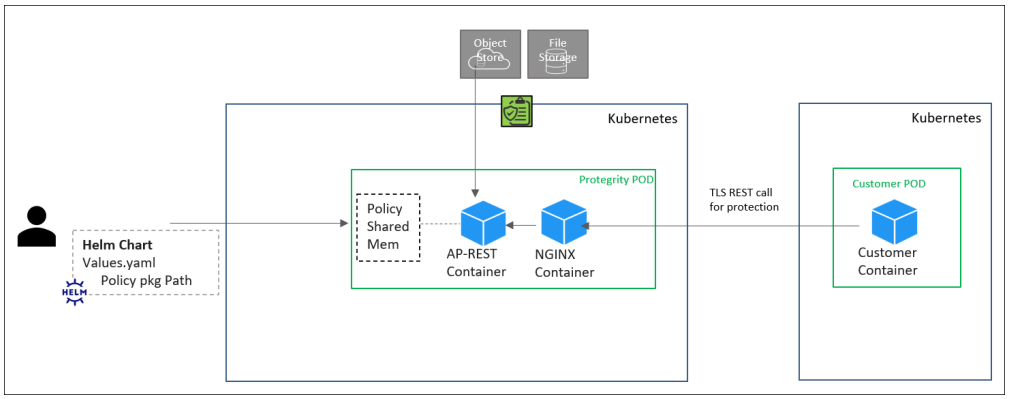


Figure 1-3: Pattern 2 - Communication between Pods in separate Kubernetes Clusters

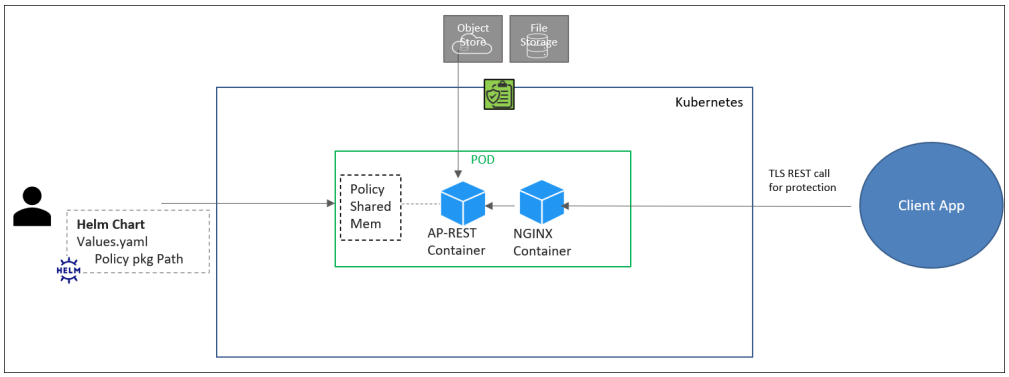


Figure 1-4: Pattern 3 - Communication with an External Application

The following steps describe the workflow of deploying the AP-REST container on a Kubernetes cluster.

1. The user runs the Helm chart to deploy the AP-REST to the Kubernetes cluster.
2. The AP-REST 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 AP-REST container over REST for protecting, unprotecting, and reprotecting data. The NGINX container is used to terminate the TLS connection from the client application. The client application and the NGINX container communicate with each other over a secure TLS channel, while the NGINX container and the AP-REST container communicate with each other over a non-TLS channel. You can have the following three patterns of communication between the customer application and the AP-REST container:
   - **Pattern 1**: The customer container and the AP-REST container are in different pods but on the same Kubernetes cluster. The AP-REST and the Customer pods are deployed in separate namespaces on the same Kubernetes cluster.
   - **Pattern 2**: The customer container and the AP-REST container are in different pods in separate Kubernetes clusters.

- **Pattern 3**: The AP-REST container receives the request for protecting, unprotecting, and reprotecting data from an external customer application, which is not deployed in a Kubernetes environment.

4. The AP-REST container returns the accurate value to the client application through the NGINX container.

Chapter 2

AP-REST Deployment Model

This section describes the AP-REST Reference Deployment model that customers can use to deploy the AP-REST container on a Kubernetes cluster.

2.1 Verifying the Prerequisites

Ensure that the following prerequisites are met:

Reference Solution Deployment

- *REST_RHUBI-ALL-64_x86-64_ALL.K8S_<AP-REST_version>.tgz* – Installation package for the IAP-REST Deployment. This package contains the following packages:
  - *REST-HELM_ALL-ALL-ALL_x86-64_K8S_<AP-REST_version>.tgz* – Package containing the Helm charts, which are used to deploy the Application Protector REST on the Kubernetes cluster.
  - *REST_RHUBI-ALL-64_x86-64_ALL.K8S_<AP-REST_version>.tar.gz* - Application Protector REST container image. This image is used to create the Application Protector REST container, which is used to perform security operations.
  - *REST-Samples_Linux-ALL-ALL_x86-64_<AP-REST_version>.tgz* - Package containing the sample application for testing the AP-REST containers with sample data.
  - *REST-SRC_<version_number>.tgz* - Package containing the Dockerfile that can be used to create a custom image for the AP-REST 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*.

  - *HOW-TO-BUILD* - Text file specifying how to build the Docker image.
  - *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

The following prerequisites for using the reference application need to be met:

- 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*.

- 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*.

- User application – For example, Pharmacy application, which contains the patient data that you want to protect using the Application Protector REST.

Additional Prerequisites

The following additional prerequisites need to be met:

- *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.
- Permission to create a Kubernetes cluster.
- Permission to create a persistent volume and 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.

- Permission to create a bucket in 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.
- *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 regarding a role and an instance profile, attach a policy to the specified IAM role, and so on. The following actions must be permitted on the IAM service:
  - GetInstanceProfile
  - GetRole
  - AddRoleToInstanceProfile
  - CreateInstanceProfile
  - CreateRole
  - PassRole
  - AttachRolePolicy
- 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

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 AP-REST 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 write permissions to upload the policy package to AWS S3 bucket. This permission is required if you want to the store the policy package on AWS S3.
- AWS EFS write permissions to upload the policy package to AWS EFS. This permission is required if you want to the store the policy package 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.

- IAM User 3 - Permission to create an AWS Cognito User Pool.

  For more information about the permissions required to create an AWS Cognito User Pool, refer to the section *Actions, resources, and condition keys for Amazon Cognito User Pools* in the *AWS Cognito documentation*.

- Access to Amazon Elastic Kubernetes Service (EKS) to create a Kubernetes cluster.
- Access to AWS Elastic Container Registry (ECR) to upload the Get ESA Policy, Init, and AP-REST container images.
- 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*.

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 *REST_RHUBI-ALL-64_x86-64_ALL.K8S_<AP-REST_version>.tgz* file on the Linux instance.
2. Run the following command to extract the files from the *REST_RHUBI-ALL-64_x86-64_ALL.K8S_<AP-REST_version>.tgz* file.

   ```
   tar -xvf REST_RHUBI-ALL-64_x86-64_ALL.K8S_<AP-REST_version>.tgz
   ```

   The following files are extracted:

   - *REST-HELM_ALL-ALL-ALL_x86-64_K8S_<AP-REST_version>.tgz*
   - *REST_RHUBI-ALL-64_x86-64_ALL.K8S_<AP-REST_version>.tar.gz*
   - *REST-Samples_Linux-ALL-ALL_x86-64_<AP-REST_version>.tgz*
   - *REST-SRC_<version_number>.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 AWS CLI, which provides a set of command line tools for the AWS Cloud Platform.

   For more information about installing AWS CLI on the Linux instance, refer to the section *Installing the AWS CLI* in the AWS documentation.
2. Configure 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 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 you 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 ekstcl* in the ekstcl 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 *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 AP-REST container.
   ```
   tar -xvf <Helm chart package>
   ```
   For example:
   ```
   tar -xvf REST-HELM_ALL-ALL-ALL_x86-64_K8S_<AP-REST_version>.tgz
   ```
   The extracted package contains the *iap-rest* directory, which has the following contents:

   - *Chart.yaml* – A YAML file that provides information regarding 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 NGINX container.

> **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 iaprest-ca.key -out iaprest-ca.crt -days 365 -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 iaprest-wildcard.key -out iaprest-
wildcard.csr -nodes -subj '/CN=*.example.com'

openssl x509 -req -sha256 -days 365 -in iaprest-wildcard.csr -CA iaprest-ca.crt
-CAkey iaprest-ca.key -set_serial 04 -out iaprest-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*.

> **Note:**
> If you want to send a request from a customer pod that is on the same Kubernetes cluster, but in a different namespace, then you to set the *CN* parameter to **.<Namespace>.svc.cluster.local*.

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 iaprest-client.key -out iaprest-
client.csr -nodes -subj '/CN=My IAP-REST Client'

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

4. Copy all the certificates to a common directory.
   For example, create a directory named *iaprest-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 Get ESA Policy, Init, and AP-REST images to the Container Repository.

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

➤ 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. Extract the installation package.

The AP-REST container image is extracted.

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

4. Perform the following steps to upload the AP-REST container image to Amazon ECR.
   a. Run the following command to load the AP-REST container image into Docker.
      ```
      docker load -i REST_RHUBI-ALL-64_x86-64_ALL.K8S_<AP-REST_version>.tar.gz
      ```
   b. Run the following command to list the AP-REST container image.
      ```
      docker images
      ```
   c. 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 ap-rest:AWS <aws_account_id>.dkr.ecr.us-east-1.amazonaws.com/ap-rest:AWS
      ```

      For more information regarding tagging an image, refer to the section *Pushing an image* in the AWS documentation.

   d. 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/ap-rest:AWS
      ```
5. Navigate to the directory where you have extracted the Helm charts packages for the AP-REST containers.
6. In the *values.yaml* file, update the appropriate path for the *iaprestImage* setting, along 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.

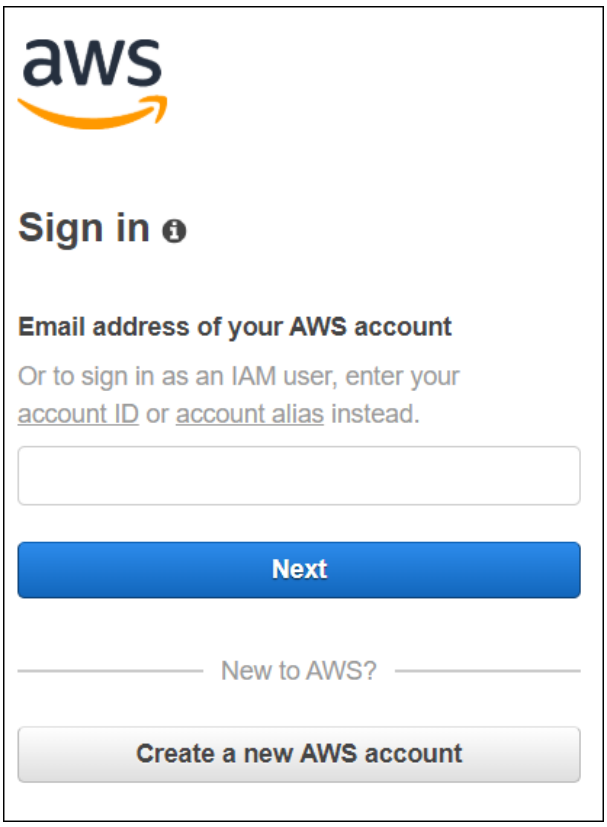


Figure 2-1: AWS Sign in screen

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

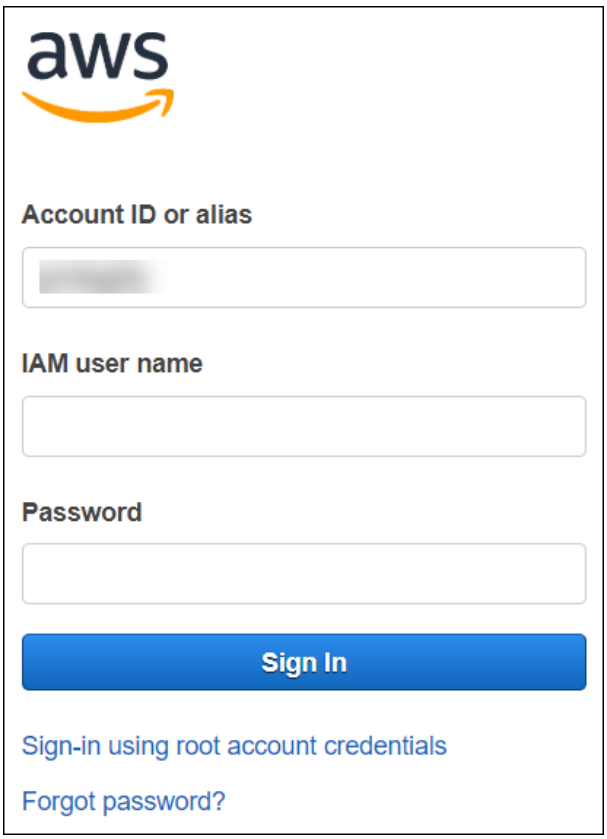


Figure 2-2: AWS login screen

5. Click **Sign in**.
   After successful authentication, the *AWS Management Console* screen appears.

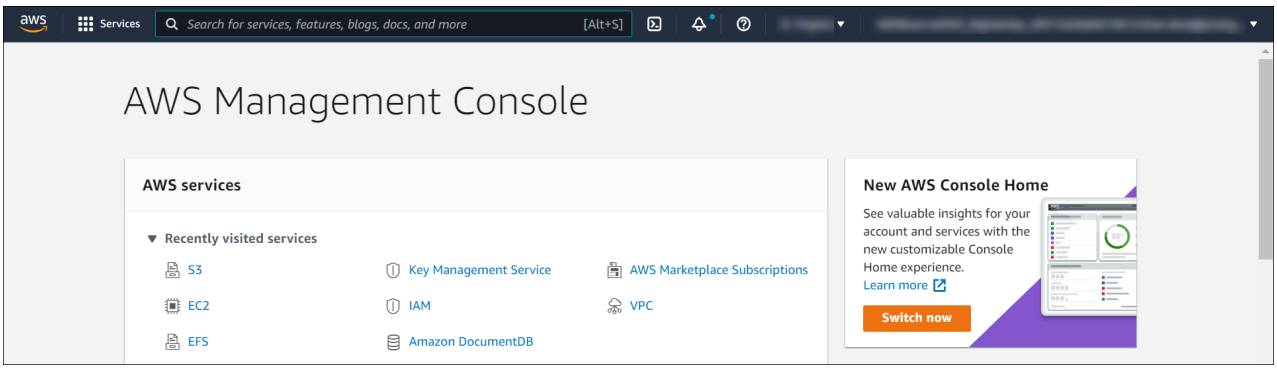


Figure 2-3: AWS Management Console Screen

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.

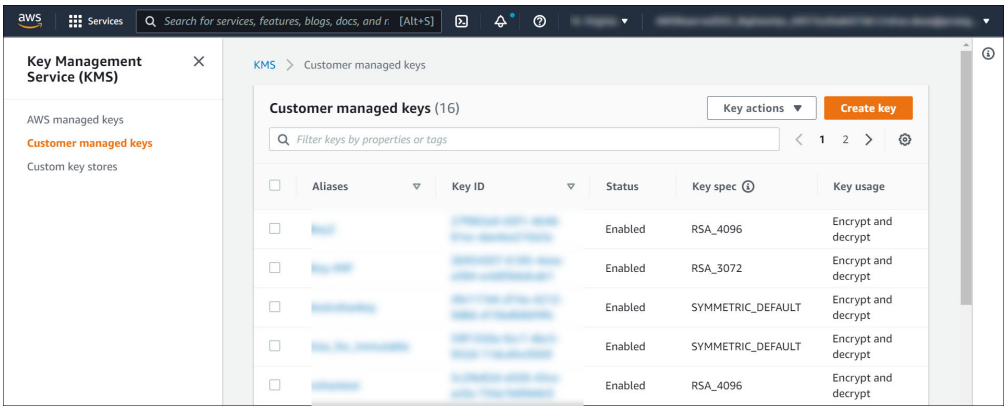


Figure 2-4: Customer Managed Keys Screen

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

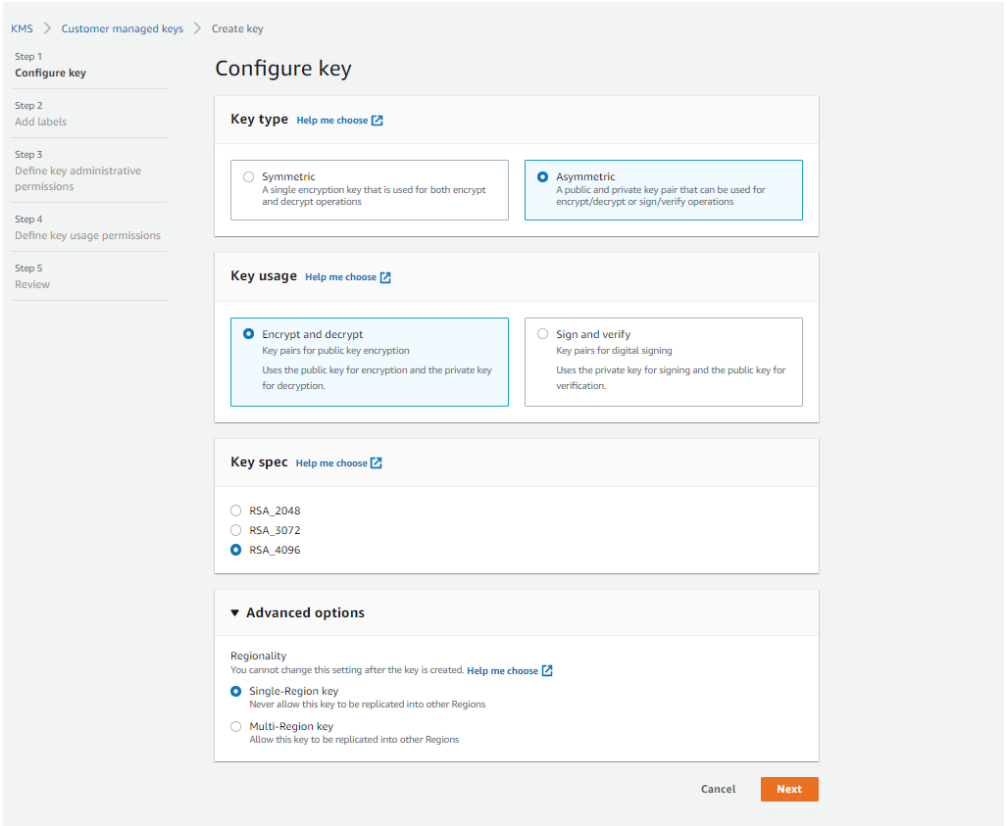
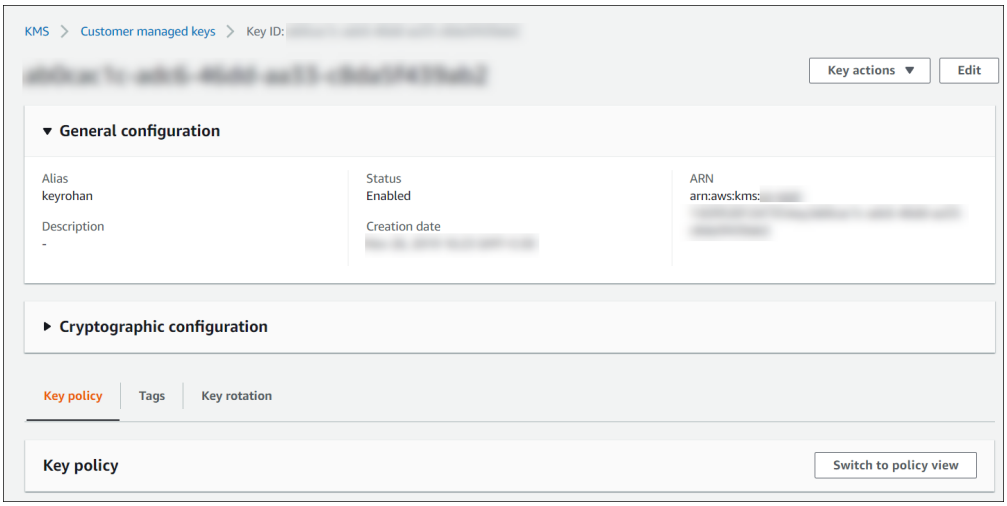


Figure 2-5: Configure Key Screen

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

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



13. In the **General Configuration** section, copy the value specified in the **ARN** field, and save it on your local machine.
    You need to attach the key to the *KMSDecryptAccess* policy. You also need to specify this ARN value in the command for creating a Kubernetes secret for the key.

14. Navigate to **Services** > **IAM**.
15. Click **Policies**.
    The **Policies** screen appears.
16. Select the **KMSDecryptAccess** policy.
    The **Permissions** tab appears.
17. Click **Edit policy** to edit the policy in JSON format.
18. Modify the policy to add the ARN of the key that you have copied in *step 15* to the Resource parameter.

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

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

2.6.1.3 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.

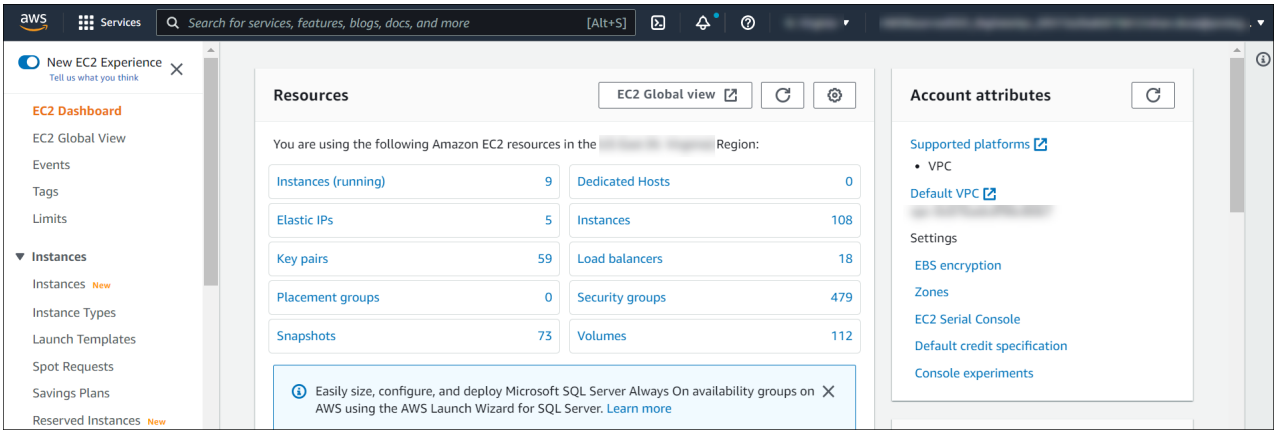


Figure 2-6: EC2 Dashboard Screen

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

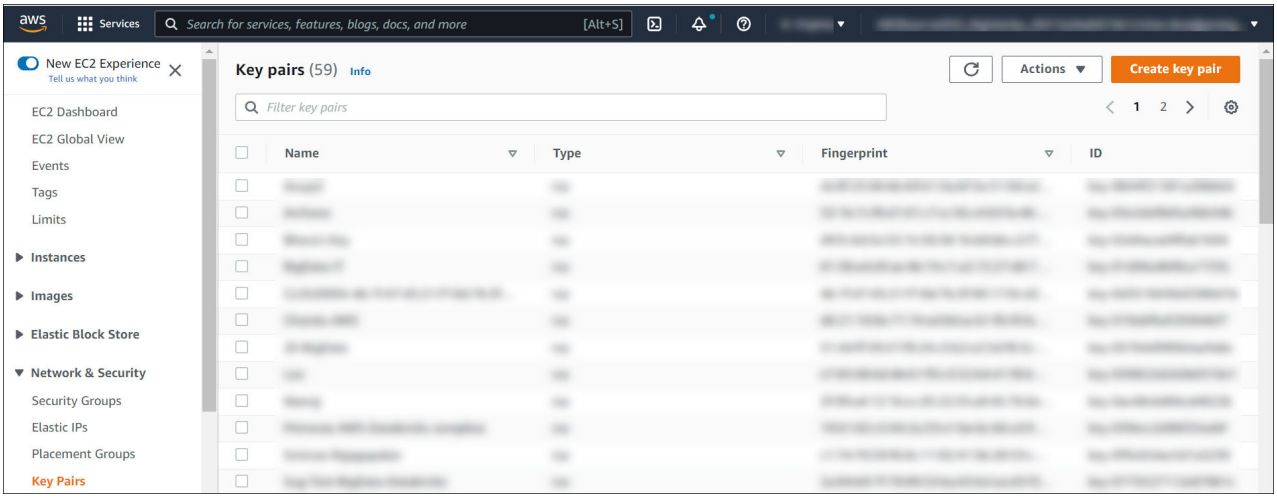


Figure 2-7: Key Pairs Screen

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

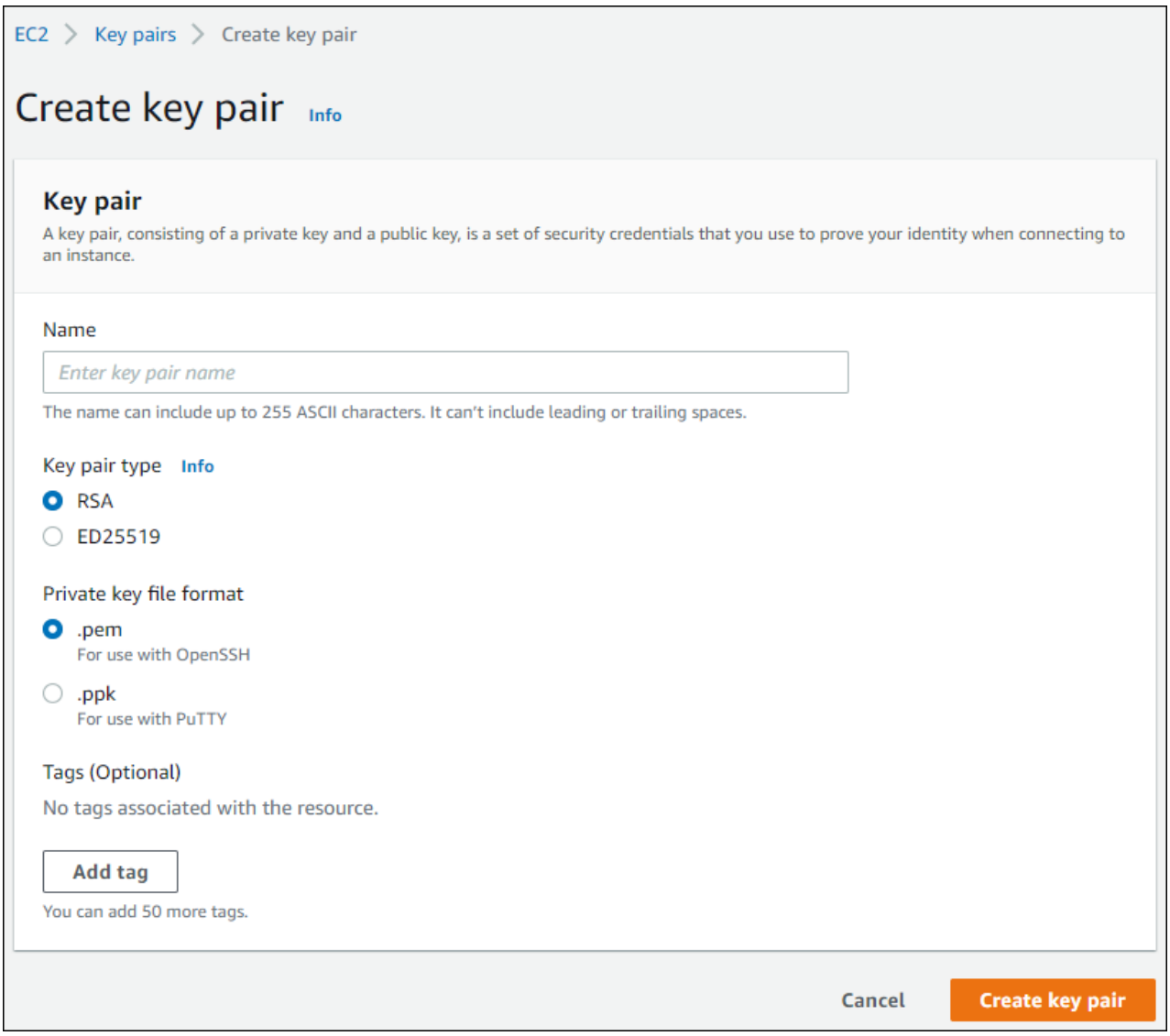


Figure 2-8: Create Key Pair Screen

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.

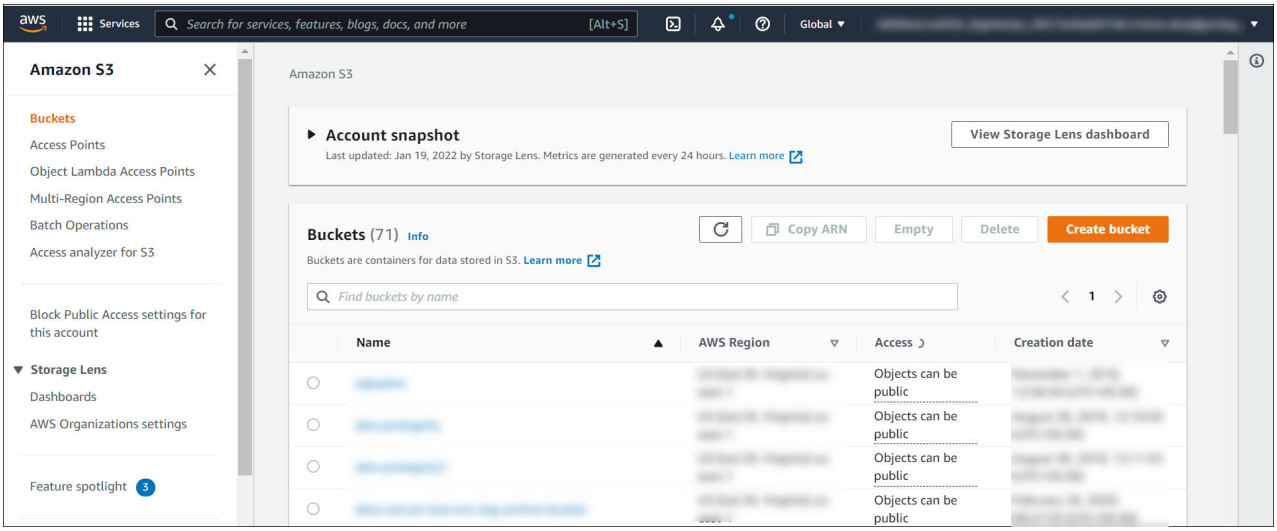


Figure 2-9: S3 Buckets Screen

4. Click **Create bucket**.
   The **Create bucket** screen appears.

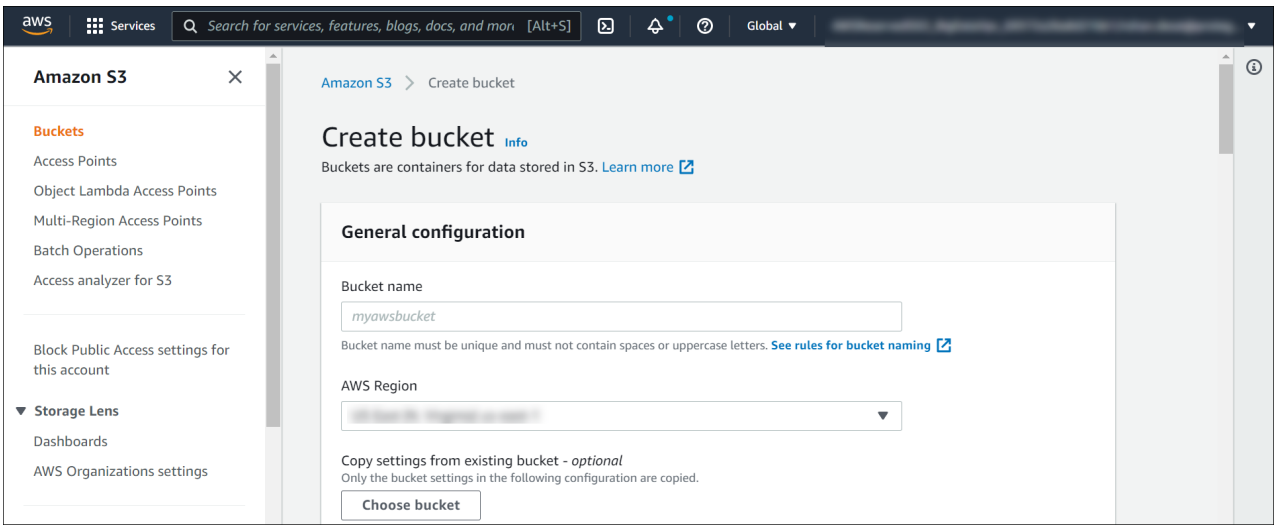


Figure 2-10: Create Bucket Dialog Box

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 AP-REST 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.

2.6.1.7 Enabling the Authentication Functionality

You can use the OAuth authentication functionality to authenticate the REST API requests sent from the client application to the AP-REST container. This functionality uses an access token, which is a JSON Web Token (JWT), to authenticate the requests that are sent from a REST API client to AP-REST instance for performing security operations. The access token is digitally signed by AWS Cognito and is then included in the authorization header of the request that is sent to the AP-REST instance.

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

➤ To enable the authentication functionality:

1. Perform the following steps to create and configure a user pool in AWS Cognito.
   a. Login to the AWS environment with IAM User 3.

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

   For more information about IAM User 3, refer to the section *Additional Prerequisites*.

   b. On the Linux instance, run the following command to create a user pool in AWS Cognito.

   aws cognito-idp create-user-pool --pool-name <Container pool name>

For example:

```
aws cognito-idp create-user-pool --pool-name container_pool_test_1
```

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

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

c. Navigate to **Services.** on AWS.
A list of AWS services appears.

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

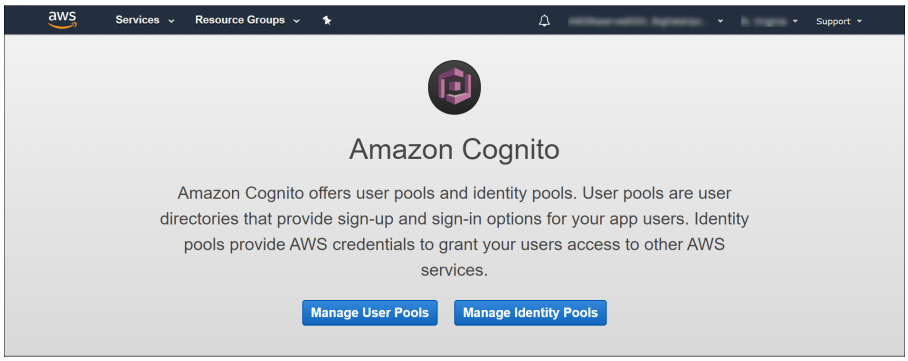


Figure 2-11: Amazon Congito Screen

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

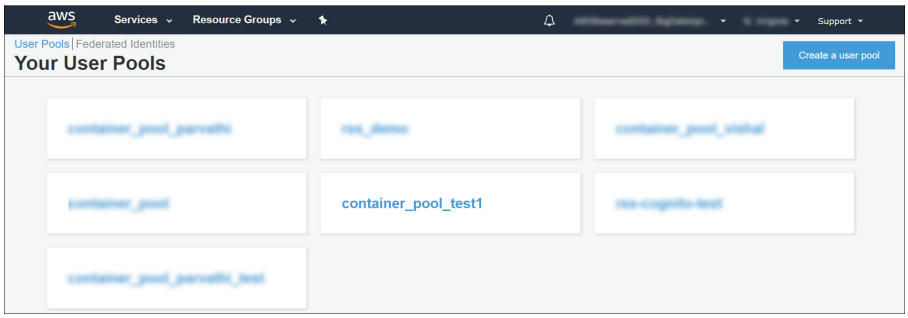


Figure 2-12: User Pools Screen

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

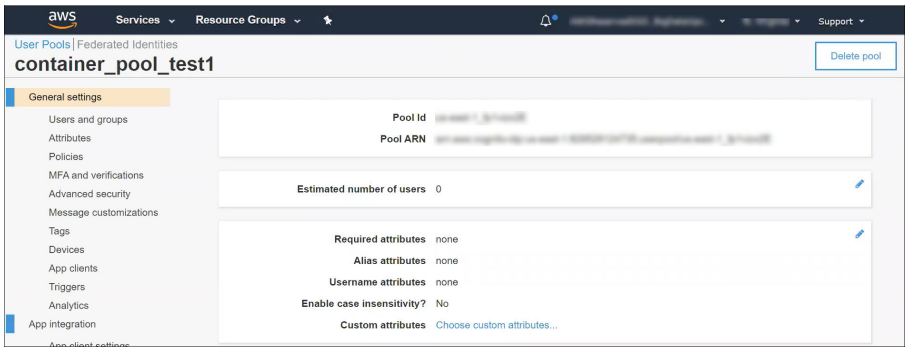


Figure 2-13: Select User Pool Details

Note the Pool Id of the user pool.

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

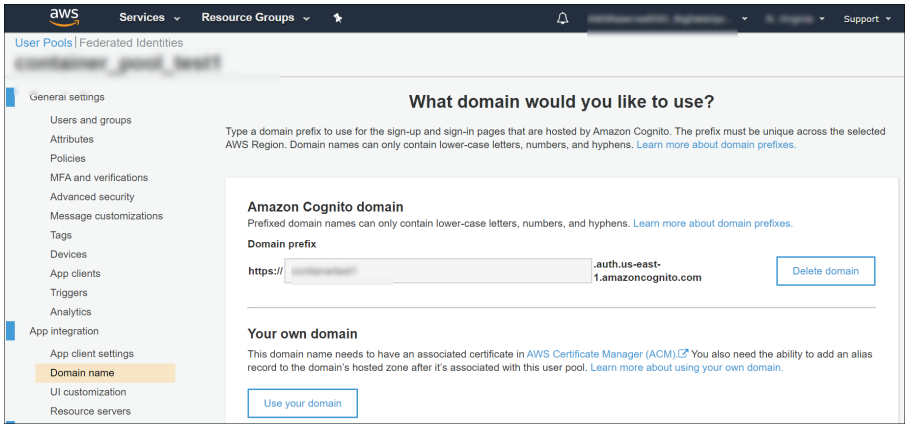


Figure 2-14: Domain Name Tab

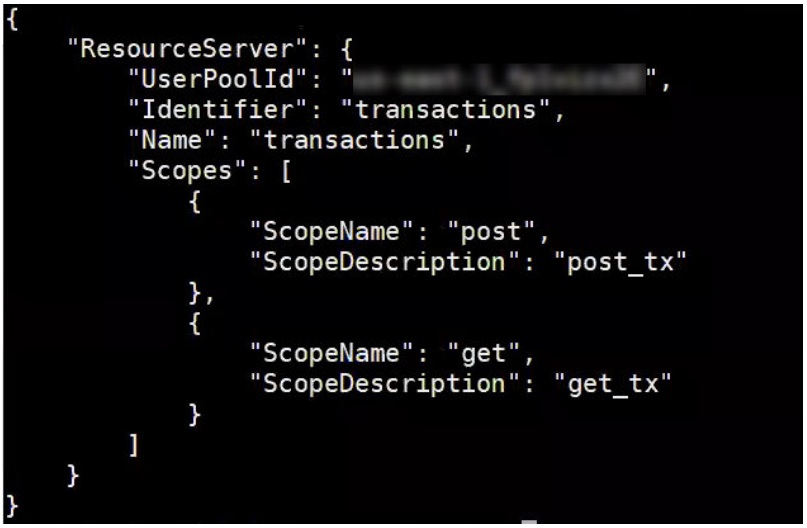
h. In the **Amazon Cognito Domain** > **Domain prefix** field, specify a prefix for the domain name that will be used to obtain the access token from AWS Cognito.

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

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

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

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



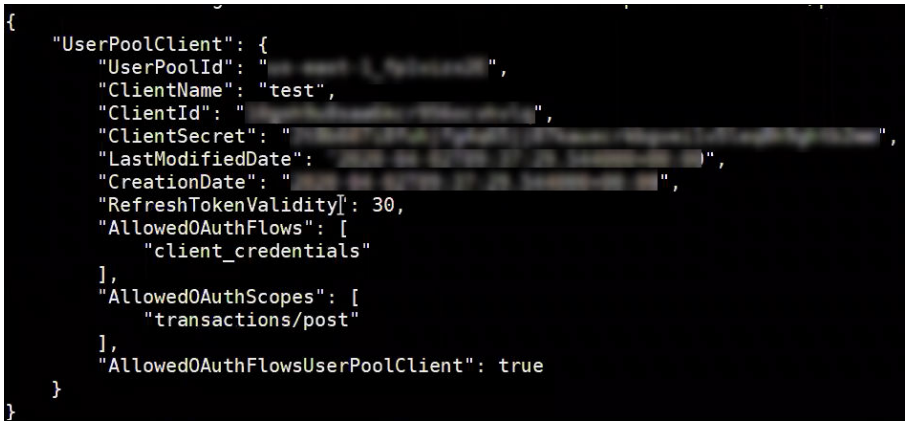
For more information about creating a resource server, refer to the section *Defining Resource Servers for Your User Pool* in the AWS documentation.

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

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

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

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



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

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

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

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

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

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

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

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

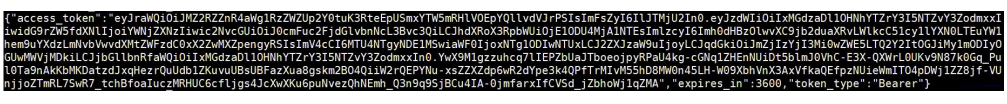
curl -X POST https://<Domain prefix>.auth.us-east-1.amazoncognito.com/oauth2/token -H 'authorization: Basic <Base64 encoded value of the client ID and client secret>' -H 'content-type: application/x-www-form-urlencoded' -d 'grant_type=client_credentials&scope=transactions%2Fpost'

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

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

The access token is generated.

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



You need to copy this token and paste it in the *Token* field of the authorization header in the REST API request that you send to the DSG instance.

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

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 AP-REST 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 AP-REST container.

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-rest
   ```
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 AP-REST container.
   a. Run the following command to create an OpenID Connect (OIDC) identity provider in the IAM console.
      ```
      eksctl utils associate-iam-oidc-provider --cluster <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 AP-REST 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 aprest-sa --namespace iap-rest --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 *aprest-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 AP-REST container.
   a. Run the following command to create an OpenID Connect (OIDC) identity provider in the IAM console.
   ```
   eksctl utils associate-iam-oidc-provider --cluster <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 AP-REST container.
   ```
   eksctl create iamserviceaccount --name <Service_account_name> --namespace
   <Namespace> --cluster <Kubernetes_cluster_name> --approve
   ```
   For example:

   ```
   eksctl create iamserviceaccount --name aprest-sa --namespace iap-rest --cluster
   cluster-name --approve
   ```

   > **Important:** Ensure that the name of the service account is *aprest-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 AP-REST container, and run the following command to apply the Protegrity PSP to the Kubernetes cluster.
   ```
   kubectl apply -f pty-psp.yaml -n <Namespace>
   ```
   For example:

   ```
   kubectl apply -f pty-psp.yaml -n iap-rest
   ```

   > **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 AP-REST container 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 AP-REST container.

   > **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 AP-REST container.

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

      For example:

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

> **Important:** Ensure that the name of the service account is *aprest-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 aprest-deployer --namespace iap-rest -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-rest -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 AP-REST 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-rest
```

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.json --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>.json --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.json --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 AP-REST Container

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

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

➤ To set up the environment for deploying the AP-REST 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-rest
```

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-aprest
--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 AP-REST Container on the Kubernetes Cluster

This section describes how to deploy the AP-REST container on the Kubernetes cluster by installing the Helm charts.

> **Important:** Use the *ap-rest-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 AP-REST 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 *aprest-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. Run the following command to create the *nginxCertsSecrets* secret, which stores the TLS certificates for the NGINX container.

```
kubectl create -n iap-rest secret generic nginx-certificates --from-file=tls.crt=iaprest-wildcard.crt --from-file=tls.key=iaprest-wildcard.key --from-file=ca.crt=iaprest-ca.crt
```

The TLS certificates are created using the procedure described in the section *Creating Certificates and Keys for TLS Authentication*.

6. Run the following command to create the *oAuth* secret, if you want to use OAuth authentication for authenticating the requests that are sent from a REST API client to the AP-REST instance for performing security operations.

```
kubectl create secret generic oauth --from-literal=auth.json='{jwks_url:"https://cognito-idp.us-east-1.amazonaws.com/us-east-1_fp1vizx2E/ .well-known/jwks.json", "user_claim":["cognito:username", "username", "client_id"], "aud":["4db69pm51siq5sddkuvakvbso4"], "iss":["https://cognito-idp.us-east-1.amazonaws.com/us-east-1_fp1vizx2E"]}'
```

Specify the following parameters.

Parameter Name	Description	Required / Optional
jwks_url	Specify the public keys that are used to digitally sign the access token. You can obtain the latest JSON Web Key Set	Required

Parameter Name	Description	Required / Optional
	(JWKS) from AWS Cognito by accessing the following URL: *https://www.cognito-idp.us-east-1.amazonaws.com/<User Pool Id>/.well-known/jwks.json*	
user_claim	Specify the user who will perform the security operations. This parameter is required only if you want to specify the username from the token. If you want to specfiy the user name from the payload, then this parameter is optional.	Optional
aud	Specify the audience	Optional
iss	Specify the issuer	Optional

For more information about the OAuth parameters, refer to the *Amazon Cognito documentation*.

7. 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 AP-REST 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 9*.

8. 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 AP-REST 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 9*.

9. On the Linux instance, navigate to the location where you have extracted the Helm charts to deploy the AP-REST container.

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

   The *iap-rest > values.yaml* file contains the default configuration values for deploying the AP-REST container 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

...
...
...


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

## Choose your private key helper. Currently we support [PKCS5, AWS_KMS, GCP_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

## k8s secret containing TLS certificates for nginx sidecar
nginxCertsSecrets: NGINX_SECRET_NAME

## enable the auth config for JWT Token based Authentication
## specify the secret containing the config for JWT Token validation & verification
authConfig:
  enabled: false
  secret: AUTH_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 iaprest Pod replicas
replicaCount: 1

...
..
.
```

```
## specify the ports exposed in your iaprest 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.
iaprestService:
  name: "iaprest"
  port: 8443
  targetPort: 8443

  # only applicable if ingress is disabled
  # allows you to configure service type: LoadBalancer or ClusterIP
  type: LoadBalancer

  # only apply, if the ingress is disabled
  # Specify service related annotations here
  annotations:
    #service.beta.kubernetes.io/aws-load-balancer-internal: "true"

## 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-aprest

 ## if required, specify a different Ingress Controller and related annotations here
  annotations:
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"

  ## Specify the host names and paths for Ingress rules
  ## 'prod' and 'staging' http paths can be used as optional fields.
  ## For accessing AP-REST service from a different namespace
  ## within the cluster, please ensure that the 'host' value matches
  ## the FQDN of the nginx ingress controller service name.
  ## The 'host' value should also match the CN provided in the certificates.
  hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8443
        prod: "/"
  ## Specify a staging host for routing traffic to the staging service
    ## of the blue-green deployment. The httpPath endpoint is 'staging' in
    ## this case. The staging hostname should also match the hostname
    ## provided in the CN of the certificates. You may specify the
    ## CN as "*.example.com" to match both the prod and staging hosts, while
    ## creating the certificates.
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8443
    #     staging: "/"
```

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

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

    For more information about Helm charts and their default values, refer to the section *Appendix A: AP-REST 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*.
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 7* that contains the passphrase and the salt that were used to generate key, which encrypted the policy. The AP-REST container uses the value specified in the *pbeSecrets* parameter to decrypt the policy.

Field	Description
	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 • The user should avoid commonly used passwords, such as, *123456789*, *11111111*, *password*, and *qwertyuiop*
privateKeySecret	Specify the Kubernetes secret created in *step 8* that contains the private key of the AWS Customer Master Key, which encrypted the policy. The AP-REST 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.
nginxCertsSecrets	Specify the Kubernetes secret that contains the TLS certificates for the NGINX container. The TLS certificates are created using the procedure described in the section *Creating Certificates and Keys for TLS Authentication*.
authConfig/enabled	Enable or disable the OAuth authentication. By default, this parameter is set to false. To enable the authentication, change the value of this parameter to true.
authConfig/secret	Specify the OAuth secret name
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:///<Directory in mount point>/xyz/imp*. In this case, a policy with the name as *imp* will be stored in the *<Directory in mount point>/xyz* directory in the required persistent volume. 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.
greenDeployment/enabled	Specify the value as *false*. While deploying Release 1, the *green* deployment strategy is disabled by default.
iaprestService/name	Specify a name for the tunnel to distinguish between ports.
iaprestService/port	Specify the port number on which you want to expose the Kubernetes service externally.
iaprestService/targetPort	Specify the port on which the AP-REST 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/ingressClassName	Specify the value of the external ingress controller, if any. For example, specify, *nginx-aprest* if you want to use the NGINX Ingress Controller.
ingress/annotations/nginx.ingress.kubernetes.io/ssl-passthrough	Specify whether you want to use the NGINX Ingress Controller as an SSL passthrough, to ensure end-to-end encrypted communication between the REST API client and the AP-REST pods. Set the value of this field to *true*.
ingress/annotations/nginx.ingress.kubernetes.io/backend-protocol	Set the value of this field to *HTTPS*, to ensure that the REST API client and the AP-REST pods can communicate with each other over an HTTPS channel.
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 AP-REST 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.

11. Run the following command to deploy the AP-REST container on the Kubernetes cluster.

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

    For example:

    ```
    helm install ap-rest --namespace iap-rest iap-rest
    ```

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

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

    For example:

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

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

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

    For example:

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

    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.

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 AP-REST.
   For more information about the extracted Helm charts, refer to the *step 8* of the section *Initializing the Linux Instance*.
   The *values.yaml* file contains the default configuration values for deploying the AP-REST 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

...
...
...


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

## Choose your private key helper. Currently we support [PKCS5, AWS_KMS, GCP_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

## k8s secret containing TLS certificates for nginx sidecar
nginxCertsSecrets: NGINX_SECRET_NAME

## enable the auth config for JWT Token based Authentication
## specify the secret containing the config for JWT Token validation & verification
authConfig:
  enabled: false
  secret: AUTH_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 iaprest Pod replicas
replicaCount: 1

...
..
```

```
.


## specify the ports exposed in your iaprest 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.
iaprestService:
  name: "iaprest"
  port: 8443
  targetPort: 8443

  # only applicable if ingress is disabled
  # allows you to configure service type: LoadBalancer or ClusterIP
  type: LoadBalancer

  # only apply, if the ingress is disabled
  # Specify service related annotations here
  annotations:
    #service.beta.kubernetes.io/aws-load-balancer-internal: "true"

## 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-aprest

 ## if required, specify a different Ingress Controller and related annotations here
  annotations:
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"

  ## Specify the host names and paths for Ingress rules
  ## 'prod' and 'staging' http paths can be used as optional fields.
  ## For accessing AP-REST service from a different namespace
  ## within the cluster, please ensure that the 'host' value matches
  ## the FQDN of the nginx ingress controller service name.
  ## The 'host' value should also match the CN provided in the certificates.
  hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8443
        prod: "/"
  ## Specify a staging host for routing traffic to the staging service
    ## of the blue-green deployment. The httpPath endpoint is 'staging' in
    ## this case. The staging hostname should also match the hostname
    ## provided in the CN of the certificates. You may specify the
    ## CN as "*.example.com" to match both the prod and staging hosts, while
    ## creating the certificates.
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8443
    #     staging: "/"
```

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: AP-REST Helm Chart Details*.

Field	Description
pvcName	Specify the name of the Persistent Volume Claim where you want to store the policy snapshot. Leave the value of this field blank if you want to use AWS S3 buckets for storing the policy snapshot.
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 AP-REST 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 AP-REST 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.
nginxCertsSecrets	Specify the Kubernetes secret that contains the TLS certificates for the NGINX container. The TLS certificates are created using the procedure described in the section *Creating Certificates and Keys for TLS Authentication*.
authConfig/enabled	Enable or disable the OAuth authentication. By default, this parameter is set to false. To enable the authentication, change the value of this parameter to true.
authConfig/secret	Specify the OAuth secret name
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:///<Directory in mount point>/xyz/imp*. In this case, a policy with the name as *imp* will be stored in the *<Directory in mount point>/xyz* directory in the required persistent volume.

Field	Description
	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.
iaprestService/name	Specify a name for the tunnel to distinguish between ports.
iaprestService/port	Specify the port number on which you want to expose the Kubernetes service externally.
iaprestService/targetPort	Specify the port on which the AP-REST 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/ingressClassName	Specify the value of the external ingress controller, if any. For example, specify, *nginx-aprest* if you want to use the NGINX Ingress Controller.
ingress/annotations/nginx.ingress.kubernetes.io/ssl-passthrough	Specify whether you want to use the NGINX Ingress Controller as an SSL passthrough, to ensure end-to-end encrypted communication between the REST API client and the AP-REST pods. Set the value of this field to *true*.
ingress/annotations/nginx.ingress.kubernetes.io/backend-protocol	Set the value of this field to *HTTPS*, to ensure that the REST API client and the AP-REST pods can communicate with each other over an HTTPS channel.
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 AP-REST 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

Field	Description
	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 AP-REST container> <Location of the directory that contains the Helm charts>
   ```

   For example:

   ```
   helm upgrade ap-rest --namespace iap-rest iap-rest
   ```

   Using this configuration ensures that the AP-REST 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 AP-REST 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 AP-REST container> <Location of the directory that contains the Helm charts>
   ```

   For example:

   ```
   helm upgrade ap-rest --namespace iap-rest iap-rest
   ```

   Using this configuration ensures that the AP-REST 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 shutdown 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 shutdown all the pods that were deployed as part of the *blue* deployment.

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

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

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

   For example:

```
helm upgrade ap-rest --namespace iap-rest iap-rest
```

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

If you want to modify any policy or subsequent releases, 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 AP-REST Container from v7.1 MR4 to v9.1.0.5

This section describes how to seamlessly upgrade the AP-REST container from v7.1 MR4 to v9.1.0.5 by upgrading the Helm charts.

➤ To upgrade AP-REST Container from v7.1 MR4 to v9.1.0.5:

Before you begin
Ensure that the following prerequisites are met before upgrading the AP-REST container 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 AP-REST 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 AP-REST container to decrypt the policy.
- Ensure that you are using the AP-REST container image or have created a custom image for the AP-REST 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 AP-REST container.

   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 AP-REST container 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

...
...
...


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

## Choose your private key helper. Currently we support [PKCS5, AWS_KMS, GCP_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

## k8s secret containing TLS certificates for nginx sidecar
nginxCertsSecrets: NGINX_SECRET_NAME

## enable the auth config for JWT Token based Authentication
## specify the secret containing the config for JWT Token validation & verification
authConfig:
  enabled: false
  secret: AUTH_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 iaprest Pod replicas
replicaCount: 1

...
..
.


## specify the ports exposed in your iaprest 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.
```

```
iaprestService:
  name: "iaprest"
  port: 8443
  targetPort: 8443

  # only applicable if ingress is disabled
  # allows you to configure service type: LoadBalancer or ClusterIP
  type: LoadBalancer

  # only apply, if the ingress is disabled
  # Specify service related annotations here
  annotations:
    #service.beta.kubernetes.io/aws-load-balancer-internal: "true"

## 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-aprest

 ## if required, specify a different Ingress Controller and related annotations here
  annotations:
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"

  ## Specify the host names and paths for Ingress rules
  ## 'prod' and 'staging' http paths can be used as optional fields.
  ## For accessing AP-REST service from a different namespace
  ## within the cluster, please ensure that the 'host' value matches
  ## the FQDN of the nginx ingress controller service name.
  ## The 'host' value should also match the CN provided in the certificates.
  hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8443
        prod: "/"
  ## Specify a staging host for routing traffic to the staging service
    ## of the blue-green deployment. The httpPath endpoint is 'staging' in
    ## this case. The staging hostname should also match the hostname
    ## provided in the CN of the certificates. You may specify the
    ## CN as "*.example.com" to match both the prod and staging hosts, while
    ## creating the certificates.
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8443
    #     staging: "/"
```

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: AP-REST 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:

Field	Description
	• *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 AP-REST 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 AP-REST 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:///<Directory in mount point>/xyz/imp*. In this case, a policy with the name as *imp* will be stored in the *<Directory in mount point>/xyz* directory in the required persistent volume. 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.
iaprestService/name	Specify a name for the tunnel to distinguish between ports.
iaprestService/port	Specify the port number on which you want to expose the Kubernetes service externally.
iaprestService/targetPort	Specify the port on which the AP-REST 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/ingressClassName	Specify the value of the external ingress controller, if any. For example, specify, *nginx-aprest* 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 AP-REST container.
ingress/annotations/nginx.ingress.kubernetes.io/ssl-passthrough	Specify whether you want to use the NGINX Ingress Controller as an SSL passthrough, to ensure end-to-end encrypted communication between the REST API client and the AP-REST pods. Set the value of this field to *true*.
ingress/annotations/nginx.ingress.kubernetes.io/backend-protocol	Set the value of this field to *HTTPS*, to ensure that the REST API client and the AP-REST pods can communicate with each other over an HTTPS channel.
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 AP-REST 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

Field	Description
	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 AP-REST container> <Location of the directory that contains the Helm charts>
   ```

   For example:

   ```
   helm upgrade ap-rest --namespace iap-rest iap-rest
   ```

   Using this configuration ensures that the AP-REST container 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 AP-REST 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 AP-REST container 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.json --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>.json --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.json --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 AP-REST Container on the Kubernetes Cluster

This section describes how to deploy the AP-REST 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-rest
   ```
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-aprest
--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
```

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 AP-REST container 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 Helm chart will be deployed> --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-rest --cluster ap-rest --attach-policy-arn arn:aws:iam::aws:policy/
AmazonS3ReadOnlyAccess --attach-policy-arn arn:aws:iam::829528124735:policy/
KMSDecryptAccess --approve
```

4. Run the following command to create the *nginxCertsSecrets* secret, which stores the TLS certificates for the NGINX container.

   ```
   kubectl create -n iap-rest secret generic nginx-certificates --from-file=tls.crt=iaprest-wildcard.crt --from-file=tls.key=iaprest-wildcard.key --from-file=ca.crt=iaprest-ca.crt
   ```

   The TLS certificates are created using the procedure described in the section *Creating Certificates and Keys for TLS Authentication*.

5. Run the following command to create the *oAuth* secret, if you want to use OAuth authentication for authenticating the requests that are sent from a REST API client to the AP-REST instance for performing security operations.

   ```
   kubectl create secret generic oauth --from-literal=auth.json='{jwks_url:"https://cognito-idp.us-east-1.amazonaws.com/us-east-1_fp1vizx2E/ .well-known/jwks.json", "user_claim":["cognito:username", "username", "client_id"], "aud":["4db69pm51siq5sddkuvakvbso4"], "iss":["https://cognito-idp.us-east-1.amazonaws.com/us-east-1_fp1vizx2E"]}'
   ```

   Specify the following parameters.

Parameter Name	Description	Required / Optional
jwks_url	Specify the public keys that are used to digitally sign the access token. You can obtain the latest JSON Web Key Set (JWKS) from AWS Cognito by accessing the following URL: *https://www.cognito-idp.us-east-1.amazonaws.com/<User Pool Id>/.well-known/jwks.json*	Required
user_claim	Specify the user who will perform the security operations. This parameter is required only if you want to specify the username from the token. If you want to specfiy the user name from the payload, then this parameter is optional.	Optional
aud	Specify the audience	Optional
iss	Specify the issuer	Optional

   For more information about the OAuth parameters, refer to the *Amazon Cognito documentation*.

6. 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 AP-REST 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*.

7. 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 AP-REST 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 13* 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*.

8. On the Linux instance, navigate to the location where you have extracted the Helm charts to deploy the AP-REST.

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

   The *iap-rest > values.yaml* file contains the default configuration values for deploying the AP-REST 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

...
...
...


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

## Choose your private key helper. Currently we support [PKCS5, AWS_KMS, GCP_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

## k8s secret containing TLS certificates for nginx sidecar
nginxCertsSecrets: NGINX_SECRET_NAME

## enable the auth config for JWT Token based Authentication
## specify the secret containing the config for JWT Token validation & verification
authConfig:
  enabled: false
  secret: AUTH_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 iaprest Pod replicas
replicaCount: 1

...
..
.


## specify the ports exposed in your iaprest 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.
iaprestService:
  name: "iaprest"
  port: 8443
  targetPort: 8443

  # only applicable if ingress is disabled
  # allows you to configure service type: LoadBalancer or ClusterIP
  type: LoadBalancer

  # only apply, if the ingress is disabled
  # Specify service related annotations here
  annotations:
    #service.beta.kubernetes.io/aws-load-balancer-internal: "true"

## 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-aprest

 ## if required, specify a different Ingress Controller and related annotations here
  annotations:
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"

  ## Specify the host names and paths for Ingress rules
  ## 'prod' and 'staging' http paths can be used as optional fields.
  ## For accessing AP-REST service from a different namespace
  ## within the cluster, please ensure that the 'host' value matches
```

```
  ## the FQDN of the nginx ingress controller service name.
  ## The 'host' value should also match the CN provided in the certificates.
  hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8443
        prod: "/"
  ## Specify a staging host for routing traffic to the staging service
    ## of the blue-green deployment. The httpPath endpoint is 'staging' in
    ## this case. The staging hostname should also match the hostname
    ## provided in the CN of the certificates. You may specify the
    ## CN as "*.example.com" to match both the prod and staging hosts, while
    ## creating the certificates.
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8443
    #     staging: "/"
```

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

9. 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: AP-REST 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*. If you want to use AWS EFS for storing and PBE for decrypting the policy snapshot, then 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.

Field	Description
	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 6* that contains the passphrase and the salt that were used to generate key, which encrypted the policy. The AP-REST 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 • The user should avoid commonly used passwords, such as, *123456789*, *11111111*, *password*, and *qwertyuiop*
privateKeySecret	Specify the Kubernetes secret created in *step 7* that contains the private key of the AWS Customer Master Key, which encrypted the policy. The AP-REST 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.
nginxCertsSecrets	Specify the Kubernetes secret that contains the TLS certificates for the NGINX container. The TLS certificates are created using the procedure described in the section *Creating Certificates and Keys for TLS Authentication*.
authConfig/enabled	Enable or disable the OAuth authentication. By default, this parameter is set to false. To enable the authentication, change the value of this parameter to true.
authConfig/secret	Specify the OAuth secret name
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.

Field	Description
	For example, if you are using AWS EFS as the persistent volume, then you can specify the file path as *file:///<Directory in mount point>/xyz/imp*. In this case, a policy with the name as *imp* will be stored in the *<Directory in mount point>/xyz* directory in the required persistent volume. 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.
iaprestService/name	Specify a name for the tunnel to distinguish between ports.
iaprestService/port	Specify the port number on which you want to expose the Kubernetes service externally.
iaprestService/targetPort	Specify the port on which the AP-REST 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/ingressClassName	Specify the value of the external ingress controller, if any. For example, specify, *nginx-aprest* if you want to use the NGINX Ingress Controller.
ingress/annotations/nginx.ingress.kubernetes.io/ssl-passthrough	Specify whether you want to use the NGINX Ingress Controller as an SSL passthrough, to ensure end-to-end encrypted communication between the REST API client and the AP-REST pods. Set the value of this field to *true*.
ingress/annotations/nginx.ingress.kubernetes.io/backend-protocol	Set the value of this field to *HTTPS*, to ensure that the REST API client and the AP-REST pods can communicate with each other over an HTTPS channel.
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 AP-REST application is running inside the Docker container.

Field	Description
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.

10. Run the following command to deploy the AP-REST container on the Kubernetes cluster.

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

    For example:

    ```
    helm install ap-rest --namespace iap-rest iap-rest
    ```

11. 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-rest
    ```

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

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

    For example:

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

    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.

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 AP-REST. For more information about the extracted Helm charts, refer to the *step 8* of the section *Initializing the Linux Instance*. The *values.yaml* file contains the default configuration values for deploying the AP-REST 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

...
...
...


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

## Choose your private key helper. Currently we support [PKCS5, AWS_KMS, GCP_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

## k8s secret containing TLS certificates for nginx sidecar
nginxCertsSecrets: NGINX_SECRET_NAME

## enable the auth config for JWT Token based Authentication
## specify the secret containing the config for JWT Token validation & verification
authConfig:
  enabled: false
  secret: AUTH_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 iaprest Pod replicas
replicaCount: 1

...
..
.


## specify the ports exposed in your iaprest 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.
iaprestService:
  name: "iaprest"
  port: 8443
  targetPort: 8443

  # only applicable if ingress is disabled
  # allows you to configure service type: LoadBalancer or ClusterIP
  type: LoadBalancer

  # only apply, if the ingress is disabled
  # Specify service related annotations here
  annotations:
    #service.beta.kubernetes.io/aws-load-balancer-internal: "true"

## 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-aprest

 ## if required, specify a different Ingress Controller and related annotations here
  annotations:
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"

  ## Specify the host names and paths for Ingress rules
  ## 'prod' and 'staging' http paths can be used as optional fields.
  ## For accessing AP-REST service from a different namespace
  ## within the cluster, please ensure that the 'host' value matches
  ## the FQDN of the nginx ingress controller service name.
  ## The 'host' value should also match the CN provided in the certificates.
  hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8443
        prod: "/"
```

```
  ## Specify a staging host for routing traffic to the staging service
    ## of the blue-green deployment. The httpPath endpoint is 'staging' in
    ## this case. The staging hostname should also match the hostname
    ## provided in the CN of the certificates. You may specify the
    ## CN as "*.example.com" to match both the prod and staging hosts, while
    ## creating the certificates.
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8443
    #     staging: "/"
```

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: AP-REST Helm Chart Details*.

Field	Description
pvcName	Specify the name of the Persistent Volume Claim where you want to store the policy snapshot. Leave the value of this field blank if you want to use AWS S3 buckets for storing the policy snapshot.
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 AP-REST 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 AP-REST 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.
nginxCertsSecrets	Specify the Kubernetes secret that contains the TLS certificates for the NGINX container. The TLS certificates are created using the procedure described in the section *Creating Certificates and Keys for TLS Authentication*.
authConfig/enabled	Enable or disable the OAuth authentication. By default, this parameter is set to false.

Field	Description
	To enable the authentication, change the value of this parameter to true.
authConfig/secret	Specify the OAuth secret name
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:///<Directory in mount point>/xyz/imp*. In this case, a policy with the name as *imp* will be stored in the *<Directory in mount point>/xyz* directory in the required persistent volume. 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.
iaprestService/name	Specify a name for the tunnel to distinguish between ports.
iaprestService/port	Specify the port number on which you want to expose the Kubernetes service externally.
iaprestService/targetPort	Specify the port on which the AP-REST 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/ingressClassName	Specify the value of the external ingress controller, if any. For example, specify, *nginx-aprest* if you want to use the NGINX Ingress Controller.
ingress/annotations/nginx.ingress.kubernetes.io/ssl-passthrough	Specify whether you want to use the NGINX Ingress Controller as an SSL passthrough, to ensure end-to-end encrypted communication between the REST API client and the AP-REST pods. Set the value of this field to *true*.

Field	Description
ingress/annotations/nginx.ingress.kubernetes.io/backend-protocol	Set the value of this field to *HTTPS*, to ensure that the REST API client and the AP-REST pods can communicate with each other over an HTTPS channel.
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 AP-REST 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 AP-REST container> <Location of the directory that contains the Helm charts>
   ```

   For example:

   ```
   helm upgrade ap-rest --namespace iap-rest iap-rest
   ```

   Using this configuration ensures that the AP-REST 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 AP-REST 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 AP-REST container> <Location of the directory that contains the Helm charts>
   ```

   For example:

   ```
   helm upgrade ap-rest --namespace iap-rest iap-rest
   ```

   Using this configuration ensures that the AP-REST 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 shutdown 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 shutdown all the pods that were deployed as part of the *blue* deployment.

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

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

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

   For example:

   ```
   helm upgrade ap-rest --namespace iap-rest iap-rest
   ```

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

   If you want to modify any policy or subsequent releases, 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 AP-REST Container from v7.1 MR4 to v9.1.0.5

This section describes how to seamlessly upgrade the AP-REST container from v7.1 MR4 to v9.1.0.5 by upgrading the Helm charts.

➤ To upgrade AP-REST Container from v7.1 MR4 to v9.1.0.5:

Before you begin
Ensure that the following prerequisites are met before upgrading the AP-REST container 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 AP-REST 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 AP-REST container to decrypt the policy.
- Ensure that you are using the AP-REST container image or have created a custom image for the AP-REST 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 AP-REST container.

   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 AP-REST container 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

...
...
...


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

## Choose your private key helper. Currently we support [PKCS5, AWS_KMS, GCP_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

## k8s secret containing TLS certificates for nginx sidecar
nginxCertsSecrets: NGINX_SECRET_NAME

## enable the auth config for JWT Token based Authentication
## specify the secret containing the config for JWT Token validation & verification
authConfig:
  enabled: false
  secret: AUTH_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 iaprest Pod replicas
replicaCount: 1

...
..
.


## specify the ports exposed in your iaprest 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.
iaprestService:
  name: "iaprest"
  port: 8443
  targetPort: 8443

  # only applicable if ingress is disabled
  # allows you to configure service type: LoadBalancer or ClusterIP
  type: LoadBalancer

  # only apply, if the ingress is disabled
  # Specify service related annotations here
  annotations:
    #service.beta.kubernetes.io/aws-load-balancer-internal: "true"

## 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-aprest

 ## if required, specify a different Ingress Controller and related annotations here
  annotations:
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"

  ## Specify the host names and paths for Ingress rules
  ## 'prod' and 'staging' http paths can be used as optional fields.
  ## For accessing AP-REST service from a different namespace
  ## within the cluster, please ensure that the 'host' value matches
  ## the FQDN of the nginx ingress controller service name.
  ## The 'host' value should also match the CN provided in the certificates.
  hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8443
        prod: "/"
  ## Specify a staging host for routing traffic to the staging service
    ## of the blue-green deployment. The httpPath endpoint is 'staging' in
    ## this case. The staging hostname should also match the hostname
    ## provided in the CN of the certificates. You may specify the
    ## CN as "*.example.com" to match both the prod and staging hosts, while
    ## creating the certificates.
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8443
    #     staging: "/"
```

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: AP-REST 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 AP-REST 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 AP-REST 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:///<Directory in mount point>/xyz/imp*. In this case, a policy with the name as *imp* will

Field	Description
	be stored in the *<Directory in mount point>/xyz* directory in the required persistent volume. 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.
iaprestService/name	Specify a name for the tunnel to distinguish between ports.
iaprestService/port	Specify the port number on which you want to expose the Kubernetes service externally.
iaprestService/targetPort	Specify the port on which the AP-REST 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/ingressClassName	Specify the value of the external ingress controller, if any. For example, specify, *nginx-aprest* 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 AP-REST container.
ingress/annotations/nginx.ingress.kubernetes.io/ssl-passthrough	Specify whether you want to use the NGINX Ingress Controller as an SSL passthrough, to ensure end-to-end encrypted communication between the REST API client and the AP-REST pods. Set the value of this field to *true*.
ingress/annotations/nginx.ingress.kubernetes.io/backend-protocol	Set the value of this field to *HTTPS*, to ensure that the REST API client and the AP-REST pods can communicate with each other over an HTTPS channel.

Field	Description
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 AP-REST 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 AP-REST container> <Location of the directory that contains the Helm charts>
```

For example:

```
helm upgrade ap-rest --namespace iap-rest iap-rest
```

Using this configuration ensures that the AP-REST container 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 AP-REST 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.

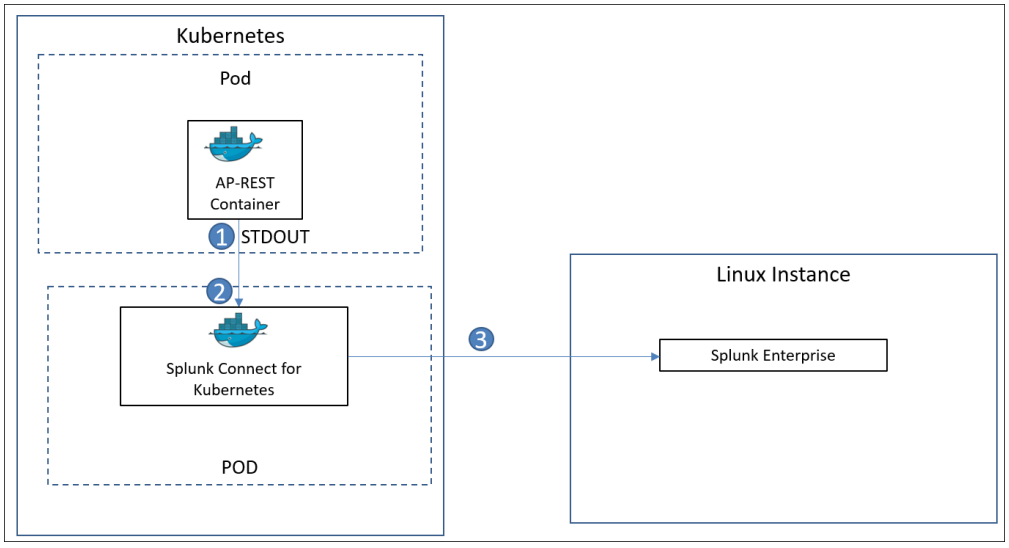


Figure 3-1: Sample Logging Workflow

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

1. The AP-REST 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 Kuberntes* page on Github.
3. The Splunk Connect for Kubernetes then forwards the logs to the Splunk Enterprise, which is used to view the logs.

3.2 Setting Up Splunk

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

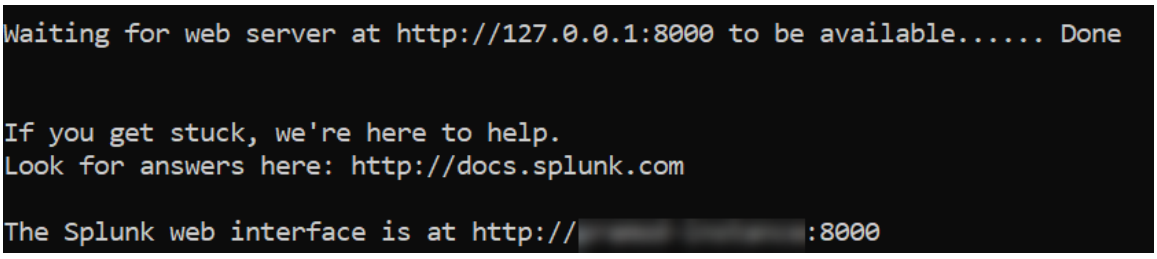
➤ To set up Splunk:

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.

   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 AP-REST and Container logs.

➤ To configure Splunk Enterprise:

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

   The Splunk Enterprise screen appears.

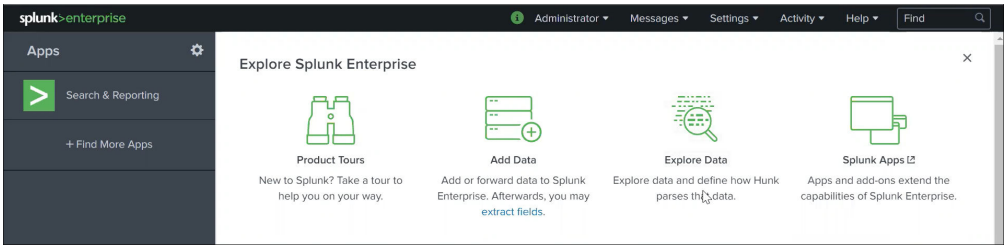


Figure 3-2: Splunk Enterprise Web UI

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.

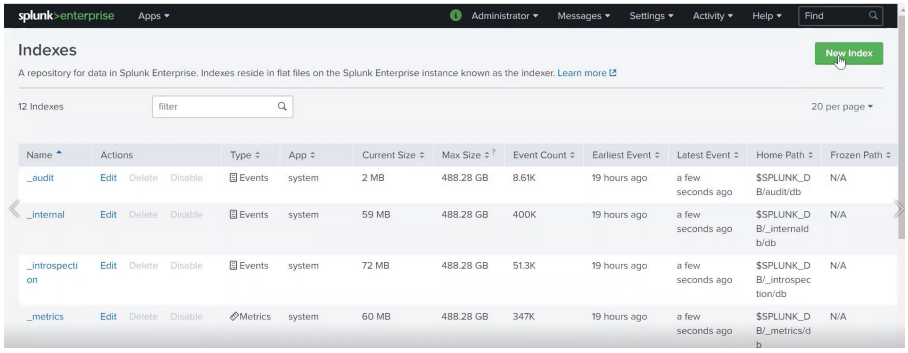


Figure 3-3: Indexes Screen

   b. Click **New Index**.

      The **New Index** dialog box appears.

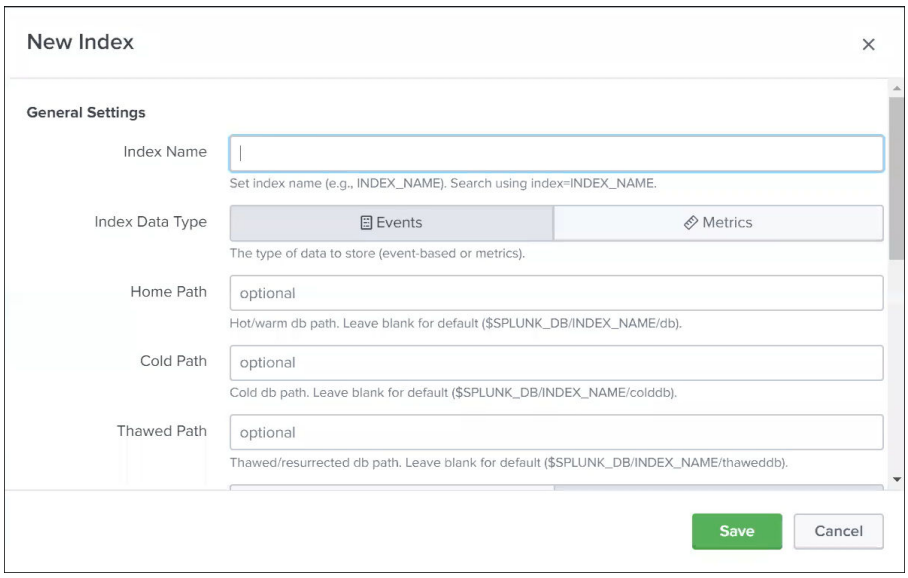


Figure 3-4: New Index Dialog Box

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 AP-REST 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 AP-REST and Container logs sent from Kubernetes.

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

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

The **Data inputs** screen appears.

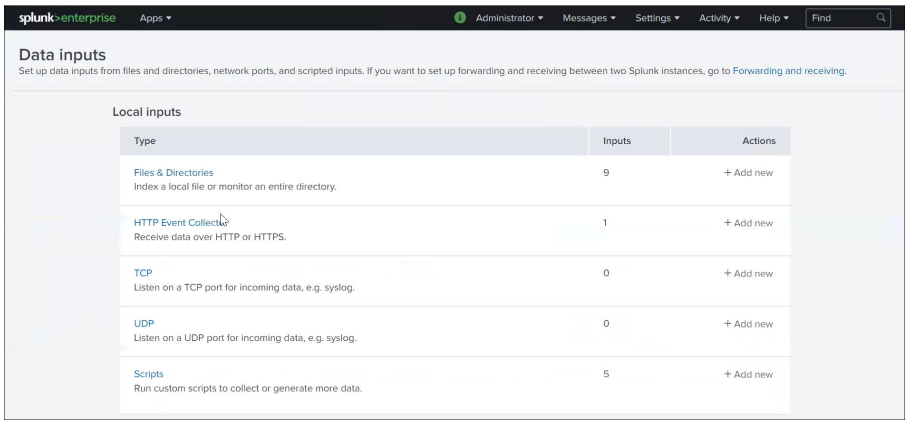


Figure 3-5: Data inputs Screen

b. Click **HTTP Event Collector**.

The **HTTP Event Collector** screen appears.

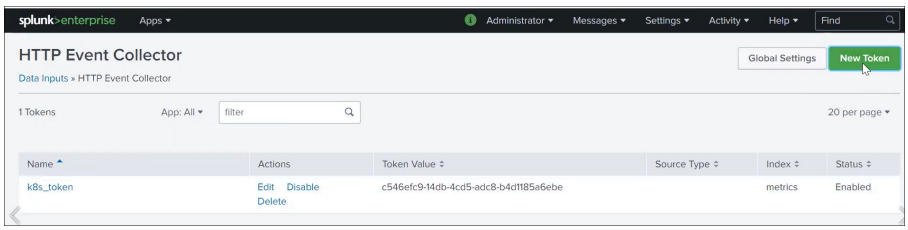


Figure 3-6: HTTP Event Collector Screen

c. Click **New Token**.

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

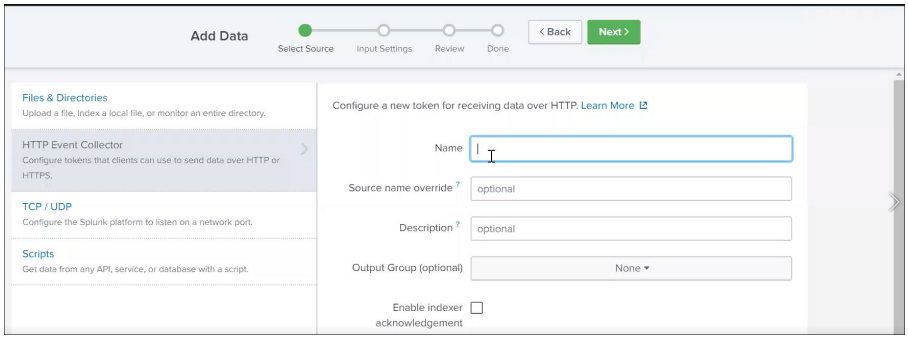


Figure 3-7: Select Source Screen

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.

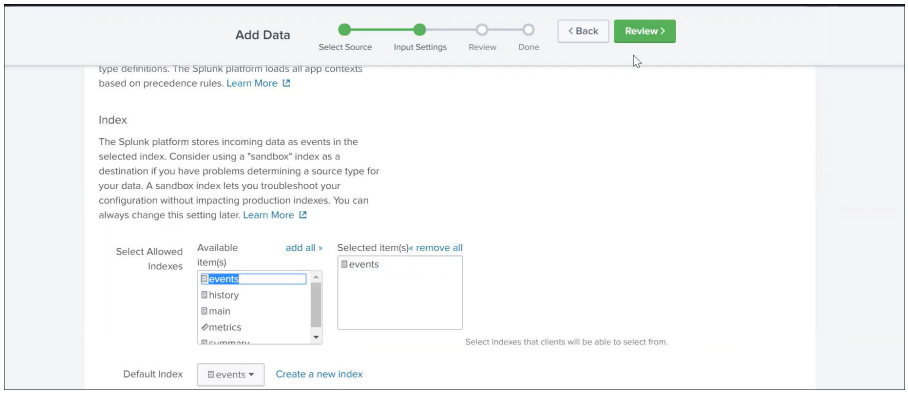


Figure 3-8: Input Settings Screen

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.

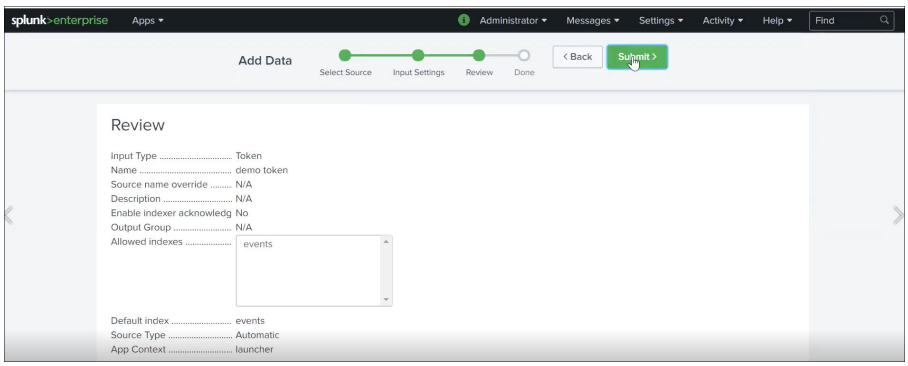


Figure 3-9: Review Screen

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.

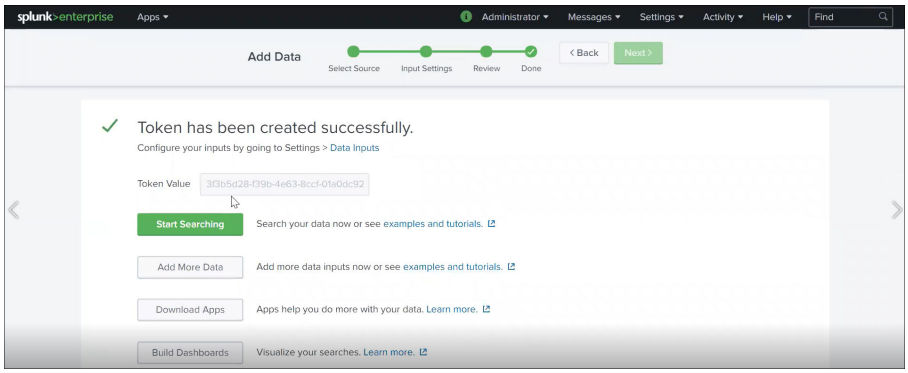


Figure 3-10: Done Screen

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.

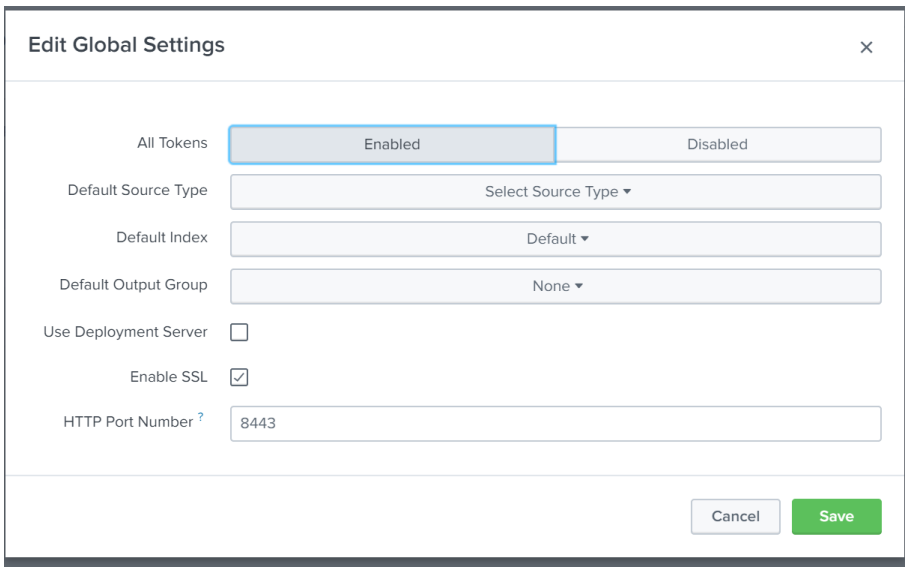


Figure 3-11: Edit Global Settings Dialog Box

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 AP-REST container.

➤ To deploy the Splunk Connect for Kubernetes Helm chart:

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

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

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

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.

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

Chapter 4

Protecting Data Using the AP-REST Container Deployment

This section describes how to protect data using the AP-REST 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 AP-REST.

4.1 Running Security Operations

This section describes how you can use the AP-REST 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 communicating with the AP-REST container from outside the Kubernetes cluster, then send the following CURL request from the Linux instance:

```
curl -v https://prod.example.com/rest-v1/protect -H 'Content-Type: application/
json' --data '{ "protect": { "policyusername": "user1", "dataelementname":
"Alphanum", "bulk": { "id": 1, "data": [ { "id": 1, "content": "<Data encoded
in base64>" }, { "id": 2, "content": "Data encoded inbase64" } ] } } }' --cacert
iaprest/certs/iaprest-ca.crt --cert iaprest/certs/iaprest-client.crt --key iaprest/
certs/iaprest-client.key
```

> **Important:** The input data must be Base64 encoded.

The AP-REST container instance returns the following protected output.

```
{"protect":{"bulk":{"id":1,"returntype":"success","data":[{"id":1,"returncode":"/rest-v1/
returncodes/id/
6","returntype":"success","content":"AHMAVABLAGMARwBSAHEAbQBxAEMAMQAyADMANAA1"},
{"id":2,"returncode":"/rest-v1/returncodes/id/
6","returntype":"success","content":"AHIASABGAE4AdgBVAFIARgBEAFMAWgBmAGYAZABPAEsAcg=="}]}}
}
```

If you are using the OAuth functionality to authenticate the REST API requests, then in the Authorization Header of the cURL command, you need to specify the access token that you have generated in *step 2* of the section *Enabling the Authentication Functionality*.

```
curl -v https://prod.example.com/rest-v1/protect -H 'Content-Type: application/
json' -H "Authorization: Bearer {token}" --data '{ "protect":
{"policyusername":"user1", "dataelementname":"Alphanum", "bulk": {"id": 1, "data":
[ {"id": 1,"content":"<Data encoded in base64>"}, {"id": 2,"content":"Data encoded
in base64"} ] } } }' --cacert iaprest/certs/iaprest-ca.crt --cert iaprest/certs/
iaprest-client.crt --key iaprest/certs/iaprest-client.key
```

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

```
curl -v POST https://prod.example.com/rest-v1/unprotect -H 'Content-
Type:application/json' --data'{ "unprotect": { "policyusername": "user1",
"dataelementname": "Alphanum", "bulk": { "id": 1, "data": [ { "id": 1, "content":
"<Data encoded in base64>" }, { "id": 2, "content": "<Data encoded in
base64>" } ] } } }' --cacert iaprest/certs/iaprest-ca.crt --cert iaprest/certs/
iaprest-client.crt --key iaprest/certs/iaprest-client.key
```

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

```
curl -v POST https://prod.example.com/rest-v1/reprotect -H 'Content-
Type:application/json' --data '{ "reprotect": { "policyusername": "user1",
"olddataelementname": "Alphanum", "newdataelementname": "Alphanum1", "bulk":
{ "id": 1, "data": [ { "id": 1, "content": "<Dataencoded in base64>" }, { "id":
2, "content": "<Data encoded in base64>" } ] } } }' --cacert iaprest/certs/
iaprest-ca.crt --cert iaprest/certs/iaprest-client.crt --key iaprest/certs/iaprest-
client.key
```

You can use these commands with the following patterns:

- External customer application communicating with the AP-REST container.
- Customer container and the AP-REST container are in different pods in different Kubernetes clusters.

> **Note:**
> You need to obtain the IP address of the NGINX ingress controller using the following command, and then specify the IP address as the Alias Target in *step 2* of the section *Appendix C: Creating a DNS Entry for the ELB Hostname in Route53*.
>
> ```
> kubectl get ingress -n <namespace>
> ```

> **Note:** 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 C: 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 C: Creating a DNS Entry for the ELB Hostname in Route53*.

2. If the AP-REST container and the Customer container are in separate pods, but on the same Kubernetes cluster, then send the following CURL request from the Linux instance.

```
curl -v https://<Kubernetes service name of AP-REST container>.<Namespace
of AP-REST container>.svc.cluster.local:8443/rest-v1/protect -H 'Content-
Type: application/json' --data '{ "protect": { "policyusername": "user1",
"dataelementname": "Alphanum", "bulk": { "id": 1, "data": [ { "id": 1,
"content": "<Data encoded in base64>" }, { "id": 2, "content": "Data encoded
inbase64" } ] } } }' --cacert iaprest/certs/iaprest-ca.crt --cert iaprest/certs/
iaprest-client.crt --key iaprest/certs/iaprest-client.key
```

> **Important:** The input data must be Base64 encoded.

The AP-REST container instance returns the following protected output.

```
{"protect":{"bulk":{"id":1,"returntype":"success","data":[{"id":1,"returncode":"/rest-v1/
returncodes/id/
6","returntype":"success","content":"AHMAVABLAGMARwBSAHEAbQBxAEMAMQAyADMANAA1"},
{"id":2,"returncode":"/rest-v1/returncodes/id/
6","returntype":"success","content":"AHIASABGAE4AdgBVAFIARgBEAFMAWgBmAGYAZABPAEsAcg=="}]}}
}
```

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

```
curl -v POST https://<Kubernetes service name of AP-REST container>.<Namespace
of AP-REST container>.svc.cluster.local:8443/rest-v1/unprotect -H 'Content-
Type: application/json' --data'{ "unprotect": { "policyusername": "user1",
"dataelementname": "Alphanum", "bulk": { "id": 1, "data": [ { "id": 1, "content":
"<Data encoded in base64>" }, { "id": 2, "content": "<Data encoded in
base64>" } ] } } }' --cacert iaprest/certs/iaprest-ca.crt --cert iaprest/certs/
iaprest-client.crt --key iaprest/certs/iaprest-client.key
```

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

```
curl -v POST https://<Kubernetes service name of AP-REST container>.<Namespace
of AP-REST container>.svc.cluster.local:8443/rest-v1/reprotect -H 'Content-
Type:application/json' --data '{ "reprotect": { "policyusername": "user1",
"olddataelementname": "Alphanum", "newdataelementname": "Alphanum1", "bulk":
{ "id": 1, "data": [ { "id": 1, "content": "<Dataencoded in base64>" }, { "id":
2, "content": "<Data encoded in base64>" } ] } } }' --cacert iaprest/certs/
iaprest-ca.crt --cert iaprest/certs/iaprest-client.crt --key iaprest/certs/iaprest-
client.key
```

4.2 Viewing Logs in Splunk

This section describes how you can view the AP-REST and Container logs in Splunk Enterprise.

➤ To view logs:

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

   The Splunk Enterprise screen appears.

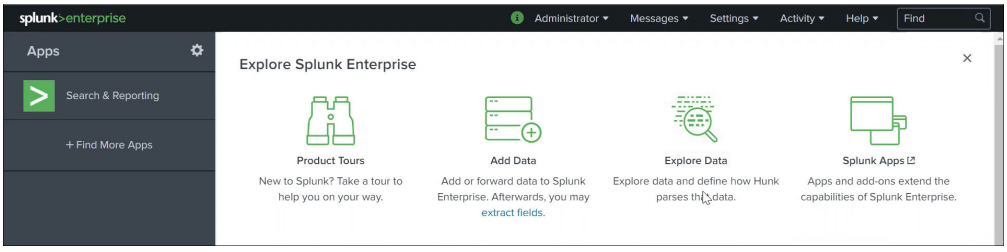


Figure 4-1: Splunk Enterprise Web UI

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.

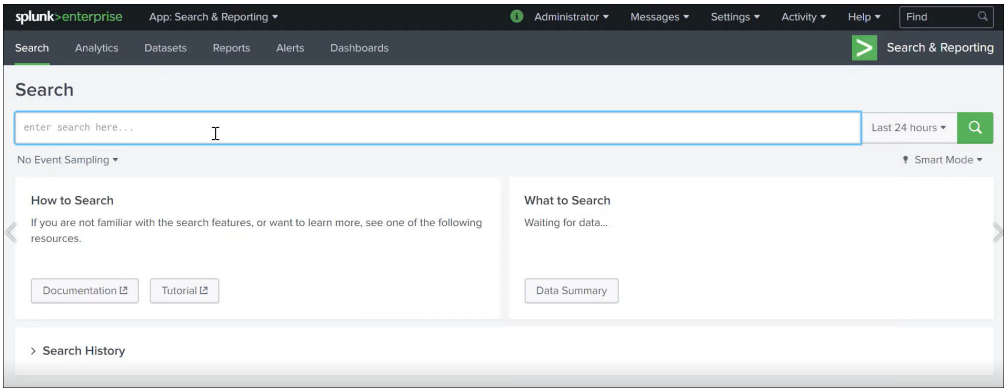


Figure 4-2: Search Screen

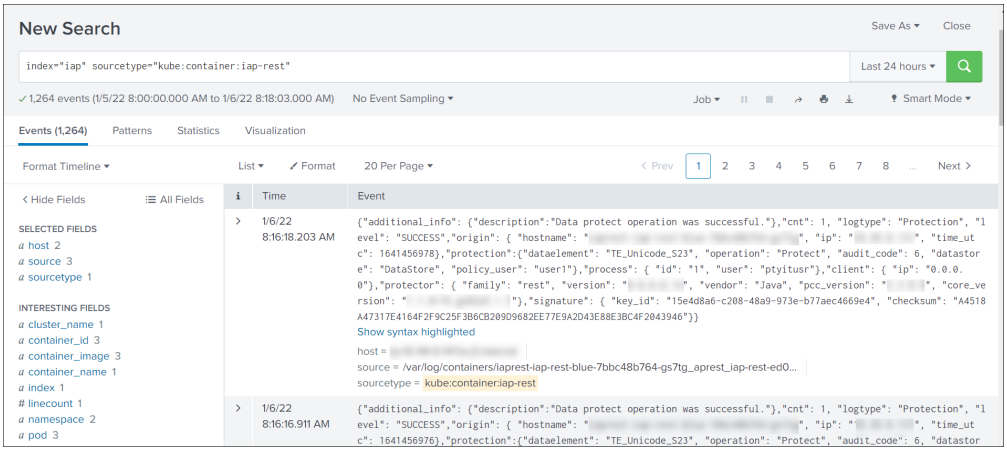
4. In the **Search** field, type the following text to filter the logs.

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

   For example:

```
index="events" sourcetype="kube:container:iap-rest"
```

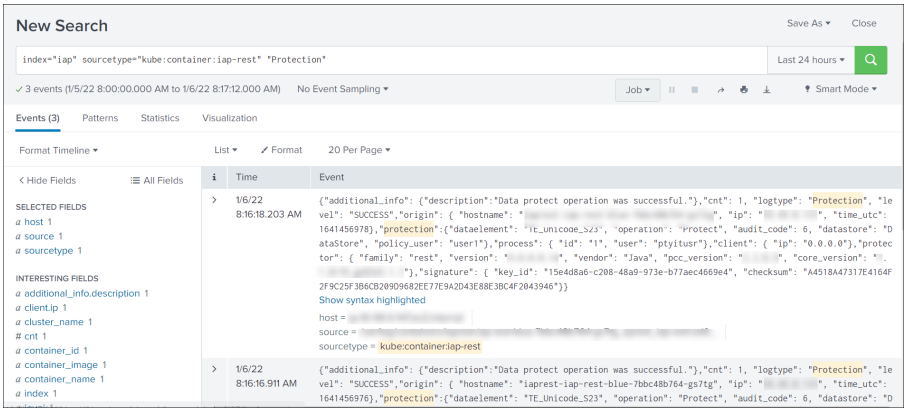
5. Press **Enter**.
   The search results appear in the **Events** tab. The search results display all the STDOUT logs from the specified container.



6. If you want to view only the logs that are related to the AP-REST, 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:iap-rest" "Protection"
```

7. Press **Enter**.
   The search results display only the logs that are related to the AP-REST.



4.3 Autoscaling the Protegrity Reference Deployment

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

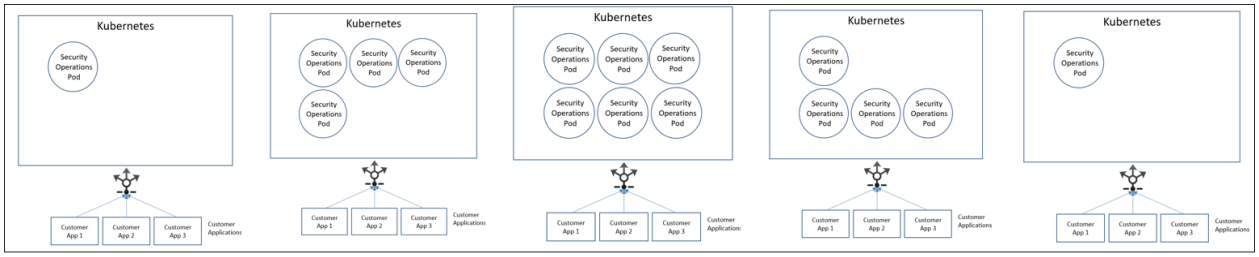


Figure 4-3: Autoscaling Kubernetes Cluster

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

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

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

Chapter 5

Appendix A: AP-REST Helm Chart Details

This section describes the details of the AP-REST 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 regarding Helm versions.

These values can be left as default.

```
apiVersion: v1
appVersion: "1.0"
description: Immutable Application Protector REST Chart for Kubernetes
name: iap-rest
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.

```
iaprestImage:
  repository: IAPREST_IMAGE_REPOSITORY
  tag: IAPREST_IMAGE_TAG
  pullPolicy: Always

# Docker Hub Image (Root User): docker.io/nginx:stable
# To use nginx image that runs with non-root permissions
# Ref. https://hub.docker.com/r/nginxinc/nginx-unprivileged
nginxImage:
  repository: NGINX_IMAGE_REPOSITORY
  tag: NGINX_IMAGE_TAG
  pullPolicy: Always
```

The following list provides an overview for the parameters in the AP_REST and NGINX container images:

- *iaprestImage* – Refers to details for the AP-REST container image:
  - *repository* – Refers to the name of the registry

    > **Note:**
    > Ensure that you only add the Container registry path as the repository value. Do not include the tag name in this value. You need to add tag name as the value in the *tag* field.

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

  > **Note:**
  > If you want to use the NGINX image that runs with root user, then download the NGINX image from the Docker hub *docker.io/nginx:stable*.

> If you want to use NGINX image that runs with non-root permissions, then refer to the website *Docker Hub for NGINX Docker Images*.

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

iaprestResources:
  limits:
    cpu: 500m
    memory: 512Mi
  requests:
    cpu: 200m
    memory: 200Mi

nginxResources:
  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 more resources than the limit you set.
- *requests* - Specify the resource request for the containers in a pod. This information determines the nodes for deploying the pods.

For more about the resource parameters, refer to the section *Managing Resources for Containers* in the Kubernetes documentation.

5.2.5 Pod Service Account

This section specifies the name of the service account that you have created for the pod in the Kubernetes Cluster. Do not edit the field if not applicable.

```
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 provides information for the intervals required to run health checks for the AP-REST container images. Users need not change these settings.

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

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

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 AP-REST and NGINX containers.

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

## set the nginx Container's security context object
## leave the field empty if not applicable
nginxContainerSecurityContext:
  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.

```
#Name of persistent volume claim to be used for this deployment.
pvcName: PVC_NAME
```

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

5.2.10 PBE Secrets

This section uses the value specified in the *pbeSecrets* parameter to decrypt the policy in the AP-REST container.

```
pbeSecrets: PBE_SECRET_NAME
```

5.2.11 NGINX Secrets

This section describes the *nginxCertsSecrets* Kubernetes secret. The *nginxCertsSecrets* Kubernetes secret contains the TLS certificates for the NGINX container.

```
## k8s secret containing TLS certificates for nginx sidecar
nginxCertsSecrets: NGINX_SECRET_NAME
```

5.2.12 OAuth Secrets

This section describes the configuration to enable the OAuth authentication fucntionality.

```
## enable the auth config for JWT Token based Authentication
## specify the secret containing the config for JWT Token validation & verification
authConfig:
  enabled: false
  secret: AUTH_SECRET_NAME
```

Set the *authConfig/enabled* parameter to *true* to enable the OAuth authentication functionality.

The *authConfig/secret* Kubernetes secret stores the configuration for authenticating the requests that are sent from a REST API client to the AP-REST instance for performing security operations.

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 iaprest Pod replicas
replicaCount: 1

## HPA configuration
autoScaling:
  minReplicas: 1
  maxReplicas: 10
  targetCPU: 50
```

The following list provides information for the parameters that can be used for autoscaling of pods in a cluster:

- *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 specifies the configurations for the REST service that needs to be used on the cluster running the AP-REST containers.

```
## specify the ports exposed in your iaprest 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.
iaprestService:
  name: "iaprest"
  port: 8443
  targetPort: 8443

  # only applicable if ingress is disabled
  # allows you to configure service type: LoadBalancer or ClusterIP
  type: LoadBalancer

  # only apply, if the ingress is disabled
```

```
  # Specify service related annotations here
  annotations:
    #service.beta.kubernetes.io/aws-load-balancer-internal: "true"
```

The following is a list of parameters that can be configured on the REST service:

- *name* - Specifies the name of the Kubernetes service. The name must contain only lowercase alphanumeric characters, which is based on standard Kubernetes naming convention.
- *port* - Specifies the port on which you want to expose the service externally.
- *targetPort* - Specifies the port on which the AP-REST is running inside the Docker container.
- *type* - Specifies whether you want to use the *LoadBalancer* of the *ClusterIP* service type. This parameter is only applicable if the ingress is disabled.

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-aprest

 ## if required, specify a different Ingress Controller and related annotations here
  annotations:
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"

  ## Specify the host names and paths for Ingress rules
  ## 'prod' and 'staging' http paths can be used as optional fields.
  ## For accessing AP-REST service from a different namespace
  ## within the cluster, please ensure that the 'host' value matches
  ## the FQDN of the nginx ingress controller service name.
  ## The 'host' value should also match the CN provided in the certificates.
  hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8443
        prod: "/"
  ## Specify a staging host for routing traffic to the staging service
    ## of the blue-green deployment. The httpPath endpoint is 'staging' in
    ## this case. The staging hostname should also match the hostname
    ## provided in the CN of the certificates. You may specify the
    ## CN as "*.example.com" to match both the prod and staging hosts, while
    ## creating the certificates.
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8443
    #     staging: "/"
```

In this case, the term NGINX is being referred to as 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-aprest* to indicate that the NGINX Ingress Controller is used.

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

 ## if required, specify a different Ingress Controller and related annotations here
  annotations:
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
```

This ensures that the NGINX Ingress Controller works like SSL passthrough and allows encrypted communication between the REST API client and the AP-REST Container through HTTPS.

```
## Specify the host names and paths for Ingress rules
  ## 'prod' and 'staging' http paths can be used as optional fields.
  ## For accessing AP-REST service from a different namespace
  ## within the cluster, please ensure that the 'host' value matches
  ## the FQDN of the nginx ingress controller service name.
  ## The 'host' value should also match the CN provided in the certificates.
  hosts:
    - host: prod.example.com
      httpPaths:
      - port: 8443
        prod: "/"

    ## Specify a staging host for routing traffic to the staging service
    ## of the blue-green deployment. The httpPath endpoint is 'staging' in
    ## this case. The staging hostname should also match the hostname
    ## provided in the CN of the certificates. You may specify the
    ## CN as "*.example.com" to match both the prod and staging hosts, while
    ## creating the certificates.
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8443
    #     staging: "/"
```

The above snippet refers to URLs and ports referred to for *blue green* deployments. Following is a list of parameters that can be configured:

- *port* – Refers to the port on which the AP-REST container is running inside the Docker container field.
- *prod* – Refers to the URI used for pointing to the current production deployment. This the URI on which the production data traffic is diverted.
- *staging* – Refers to the URI used for pointing to the current deployment under test.

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

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

Using the hostname option, you have *prod* URLs running on one hostname and *staging* URLs running on another hostname.

Chapter 6

Appendix B: Using the Samples

This section explains details and usage of the components included in the *REST-Samples_Linux-ALL-ALL_x86-64_<AP-REST_version>.tgz* archive.

6.1 Overview

Protegrity delivers a sample application as part of the AP-REST Container installation package. The sample application constitutes a canned policy (to get imported to the ESA workbench), sample Postman collection (to test AP-REST Container Pods serving deployed IMP) and an autoscaling script (to push more load to the Kubernetes cluster to force autoscaling of the AP-REST Container Pods).

On consuming the deliverables in the AP-REST Container installation package, we expect customers to run this sample application end-to-end, as a sanity test, to confirm the installation was completed accurately. In this section, we are providing details on the exact steps that the customer needs to follow to run the sample application end-to-end. This section explains details and usage of the components included in *REST-Samples_Linux-ALL-ALL_x86-64_<AP-REST_version>.tgz* archive.

The following components are included in the *REST-Samples_Linux-ALL-ALL_x86-64_<AP-REST_version>.tgz* archive.

6.1.1 Policy Sample

Package – *Sample_App_Policy_V2.tgz*

This component consists of the sample policy that can be imported on the ESA 9.1.0.5 for getting started with the AP-REST Containers use case. The following are the details for the policy.

Policy Name - Sample_policy

Token Type	Data Element Name
Alphanumeric	Alphanum
Alphanumeric	Alphanum1

6.1.2 Autoscaling Script

Package – *Sample_App_autoscale.sh*

Script for making 10,000 REST calls to AP-REST. This script can be triggered to test the autoscaling of the pods.

6.1.3 PostMan Collection

Package – *Sample_App_PostMan_Collection_V2.json*

This collection can be used to make REST calls to AP-REST for protecting the data. The JSON file contains the following collections:

- Release 1 protect request
- Release 1 unprotect request
- Release 1 reprotect request
- Release 2 protect request

Release 1 protect request

Post Request Path: - **`https://prod.example.com/rest-v1/protect`**

Release 1 unprotect request

Post Request Path: - **`https://prod.example.com/rest-v1/unprotect`**

Release 1 protect request

Post Request Path: - **`https://prod.example.com/rest-v1/reprotect`**

Release 2 protect request

Post Request Path: - **`https://prod.example.com/rest-v1/protect`**

> **Note:**
> The POST requests will change depending on the deployment pattern used.
>
> For more information about the deployment patterns, refer to the section *Architecture and Components*.
>
> For more information about the POST requests for each deployment patterns, refer to the section *Running Security Operations*.

6.2 Using the Samples

1. Ensure that the prerequisites mentioned in the subsection *Additional Pre-Requisite* within the section *Verifying the Prerequisites* are followed
2. A Kubernetes environment is created. For more information about creating the cloud runtime environment, refer to the section *Creating the Cloud Runtime Environment*.

The user must perform the following tasks.

6.2.1 Importing Policy Sample on the ESA

The user needs to perform the following steps to import the *Sample_App_Policy_V2.tgz* file on the ESA.

➤ To import policy sample on the ESA:

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

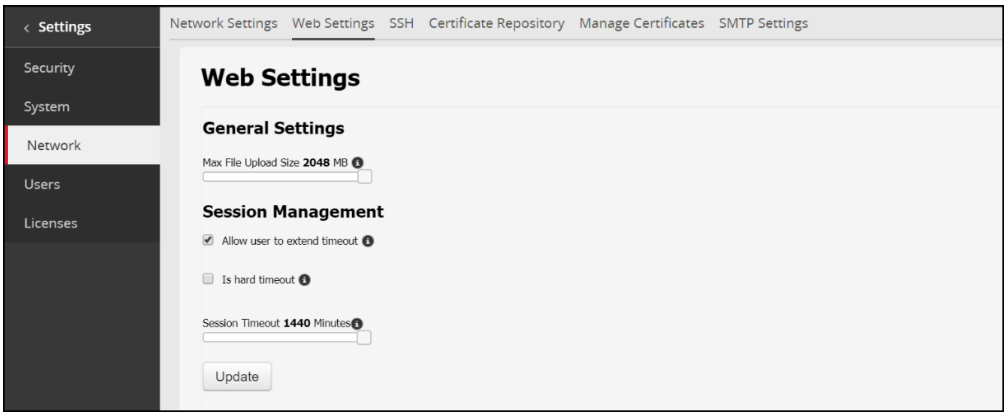


Figure 6-1: ESA Web Settings

3. In the **General Settings** section, change the **Max File Upload Size** value to the maximum value.
4. In the **Session Management** section, change the **Session Timeout** value to the maximum value.
5. Click **Update**.
6. Navigate to **Settings** > **System** > **File Upload**.
7. Click **Choose File** to select the *Sample_App_Policy_V2.tgz* file that you want to upload.
8. Enter the admin password.
9. Navigate to **System** > **Backup and Restore** > **Import**, and select the *Sample_App_Policy_V2.tgz* file from the drop down list and click **Import**.
10. Provide the password as **`admin1234`** and click **Import**..

    After successful import, the *Sample_policy* should be available in the *Policies* section.

6.2.2 Creating IMP Packages

For Release 1

Set the following values for creating IMP package for release 1 and create IMP packages:

- policy/filePath: *test/release1/policy*

6.2.3 Importing Certificates to the Postman Client

This section describes the steps to import the CA certificate, the client certificates, and keys to the Postman client, if you want to ensure secure communication between the Postman client and the NGINX container using TLS.

➤ To import certificates to the Postman client:

1. Open the Postman client.
2. In the header of the Postman client, click , and then select **Settings**.

   The **SETTINGS** dialog box appears.

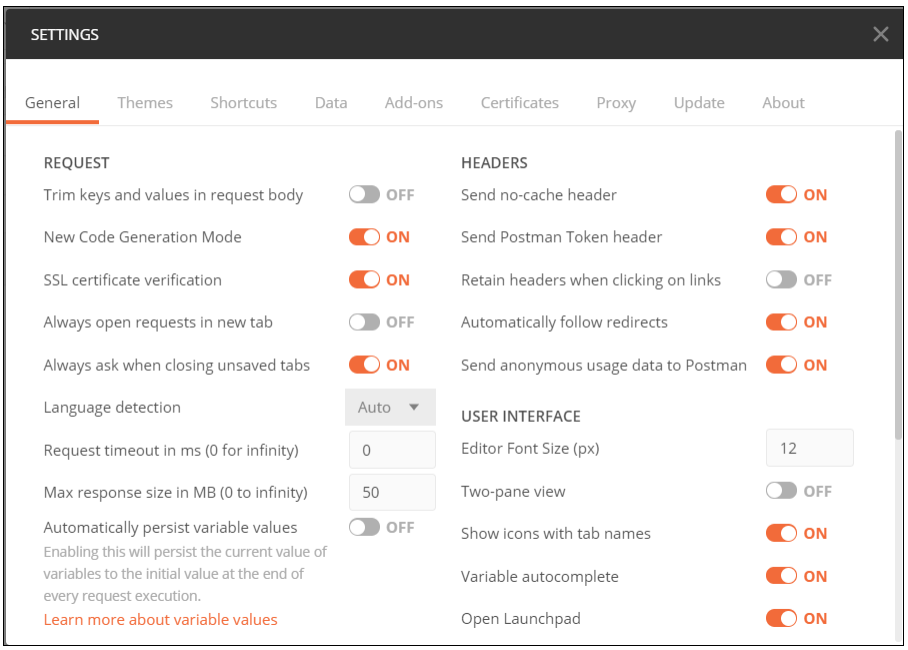


Figure 6-2: SETTING Dialog Box

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

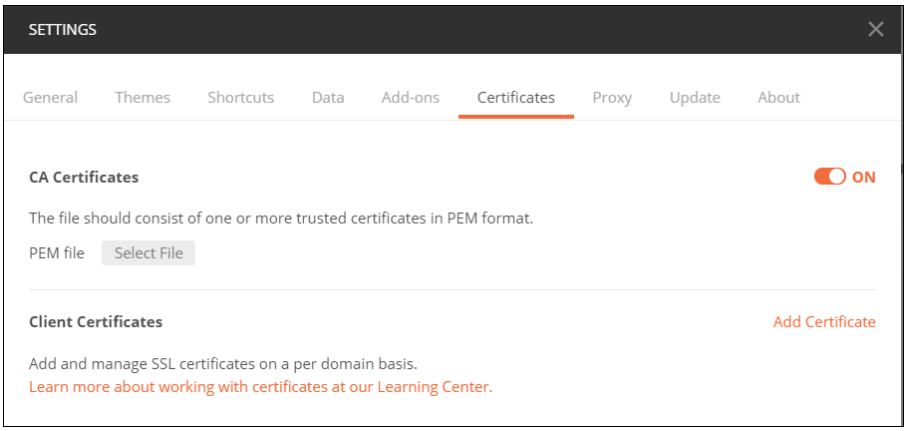


Figure 6-3: Certificates Tab

4. In the **CA Certificates** section, click **Select File** and browse for the *iap-ca.crt* file in the *iap-certs* directory that you have created in the section *Creating Certificates and Keys for TLS Authentication*.
5. In the **Client Certificates** section, click **Add Certificate**.

   The **Add Certificate** screen appears.

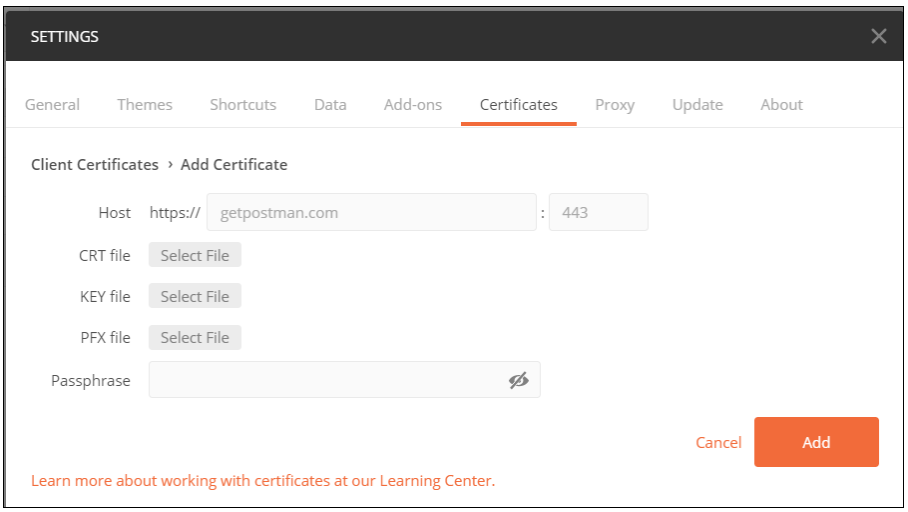


Figure 6-4: Add Certificate Screen

6. In the **Host** field, specify the value as *prod.example.com*.
   You need to specify the ingress port number, which is *8443* by default.
7. In the **CRT file** field, click **Select File** to browse for the *iap-client.crt* file in the *iap-certs* directory.
8. In the **KEY file** field, click **Select File** to browse for the *iap-client.key* file in the *iap-certs* directory.
9. Click **Add** to add the client certificate and key to the Postman client.
10. Repeat steps 5 to 9 for adding client certificates for the host *staging.example.com*.

6.2.4 Deploying Release 1

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

```
blueDeployment:
  enabled: true
  policy:
    filepath: test/release1/policy
```

2. Deploy Release 1 on the cluster.

   For more information about deploying the AP-REST Container on the Kubernetes Cluster with RBAC and PSP, refer to the section *Deploying the AP-REST Container on the Kubernetes Cluster*.

   For more information about deploying the AP-REST Container on the Kubernetes Cluster without RBAC and PSP, refer to the section *Deploying the AP-REST Container on the Kubernetes Cluster*.

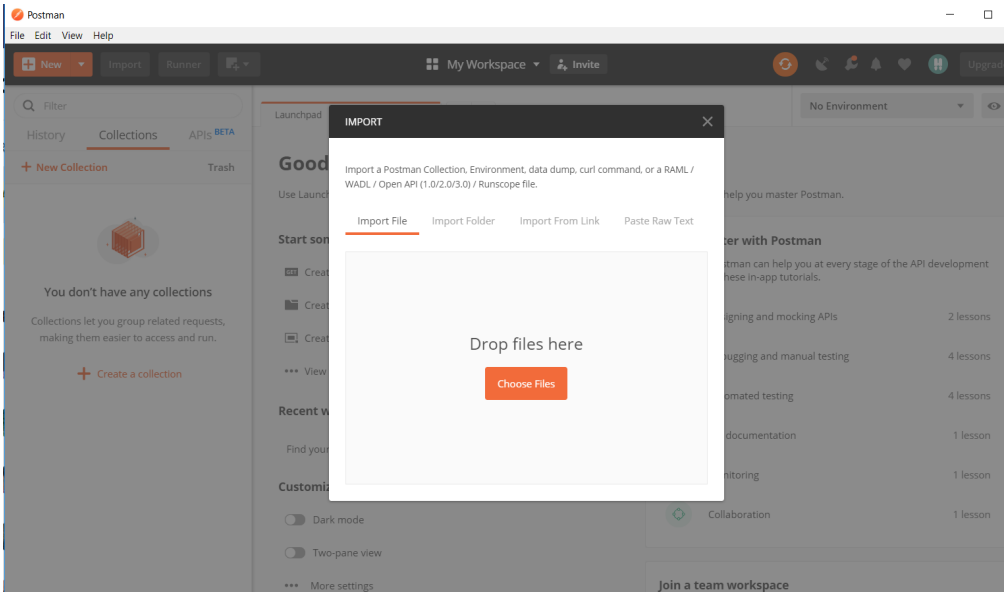
6.2.5 Running the Postman Collection

This section describes the steps for protecting data for Release 1 and Release 2.

The component consists of the Postman JSON file to generate the REST request for protecting data.

For protecting data with Release 1

1. Import the Postman collection.



2. After import, the following four collections should be available:
   - Release 1 protect request
   - Release 1 unprotect request
   - Release 1 reprotect request
   - Release 2 protect request
3. If you have enabled the OAuth authentication functionality, then navigate to the **Authorization** tab.

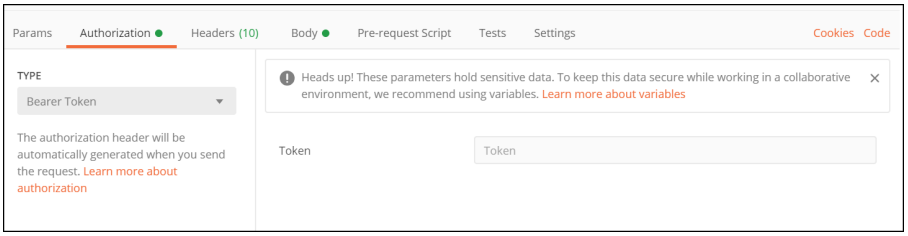


Figure 6-5: Authorization Tab

4. Select *Bearer Token* as the type of authorization header from the **Type** drop-down list.
   The **Token** field appears.
5. In the **Token** field, specify the ID token that you have generated in *step 2* of the section *Enabling the Authentication Functionality* .
6. Select **Release 1 protect request** in AP_REST SAMPLE and click **Send**.
   The user should get response as successful *200 OK* and receive protected data.
7. Select **Release 1 unprotect request** in AP_REST SAMPLE and click **Send**.
   The user should get response as successful *200 OK* and receive unprotected data.
8. Select **Release 1 protect request** in AP_REST SAMPLE and click **Send**.
   The user should get response as successful *200 OK* and receive reprotected data.

For protecting data with Release 2

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

For Release 2

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

- securityConfigDestination: *VolumeMount*
- policy/filePath: *test/release2/policy*

1. Run Helm upgrade for switching to Release 2.

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

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

```
greenDeployment:
enabled: true
securityConfigSource: VolumeMount
policy:
filePath: test/release2/policy
```

3. Select **Release 2 protect request** in AP_REST SAMPLE and click **Send**.

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

6.2.6 Running the Autoscaling Script

This section provides an overview on the Autoscaling script.

The Autoscaling script is used to issue continuous requests on the AP-REST containers. When the number of REST requests hitting the container are increased, with Horizontal Pod Autoscaling, Kubernetes automatically scales the number of pods based on the CPU utilization observed.

The user can run the autoscaling script from any Linux node, which can connect the external IP for the deployment.

Usage

```
./Sample_App_autoscale.sh <Ingress address> <CA certificate path> <Client certificate path> <Client key path>
```

After running this script, the user can observe that new pods are created to handle to the incoming traffic.

Chapter 7

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

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

1. . Configure Route53 for DNS resolution.
   - Create a private hosted zone in 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 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. The Alias Target must be set as the IP address of the NGINX Ingress Controller
3. Save the record Create Inbound endpoint for DNS queries from a network to hosted VPC used in Kubernetes.
   - Select Configure endpoints in Route53 Resolver service.
   - Select Inbound Only endpoint.
   - Give a name to the endpoint.
   - Select the VPC use in k8s cluster & Route53 private hosted zone.
   - Select the availability zone as per the subnet.
   - Review and Create the endpoint.
   - Note down the IP addresses from the Inbound endpoint page.

     Specify this IP address in the /etc/resolv.conf file in Linux in *step 1* 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 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 AP-REST 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 AP-REST container, using the Dockerfile provided with the installation package.

➤ To create custom image:

1. Download the installation package.

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

   > **Important:** The dependency packages required for building the Docker images are specified in the *HOW-TO-BUILD* file, 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 AP-REST container.

   a. Run the following command to extract the files from the *REST-SRC_<version_number>.tgz* file to a directory.

   ```
   tar -C <dir> REST-SRC_<version_number>.tgz
   ```

   The following files are extracted:

   - *ImmutableApplicationProtectorRESTLinux_x64_<version_number>.tgz*
   - *PolicyUtil_Linux_x64_<version_number>.tgz*
   - *REST_RHUBI_DOCKERFILE_<version_number>*
   - *docker-entrypoint.sh*

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

   ```
   ARG BASE_DOCKER_IMAGE=RHEL-MINIMAL-UBI
   ```

c. Run the following command in the directory where you have extracted the contents of the *REST-SRC_<version_number>.tgz* file.

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

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

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

d. Run the following command to list the AP-REST container image.

```
docker images
```

e. Push the AP-REST 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*.

Chapter 9

Appendix E: Application Protector API on REST

This section describes the AP REST protector APIs that are available for protection and unprotection of data. In addition, it lists the error handling capabilities provided by the AP API on REST.

9.1 AP REST APIs

This section describes the AP REST APIs available for protection and unprotection of data.

9.1.1 HTTP GET version

This API displays the version of the AP REST protector API being used.

Request

Method	URI
GET	https://hostname/rest-v1/version

Parameters

Hostname	Resource
Hostname as defined in the AP-REST deployment	/rest-v1/version

Result

This function returns the current version of the AP REST protector API.

Response

Status	Response
200	`{"version":"9.1.0.5.13","components": {"jpepVersion":"9.1.0.5.15","coreVersion":"1.1.0+76.ge82e5.1.1"}}`

Example

```
$ curl 'https://<HostName>/rest-v1/version' --cacert iap-rest-ca.crt --cert iap-rest-client.crt  --key iap-rest-client.key
```

9.1.2 HTTP POST protect

This API returns protected data.

URI

https://hostname/rest-v1/protect

Method

POST

Parameters

Hostname: Host name of the endpoint, as defined in the AP-REST deployment

Resource: The resource to be used, which is */rest-v1/protect*

Result

This API returns protected data.

Example 1 - without external IV and external tweak

```
$ curl --location --request POST 'https://<hostname>/rest-v1/protect' \
--connect-to  "<hostname>:443:<AWS LoadBalancer>:443"  \
--header 'Content-Type: application/json' \
 --cacert iap-rest-ca.crt --cert iap-rest-client.crt  --key iap-rest-client.key  --data
'{
  "protect": {
    "policyusername": "Uername",
    "dataelementname": "DataElement1",
    "bulk":{
      "id": 1,
      "data": [
        {
          "id": 1,
          "content": "AFAAcgBvAHQAZQBnAHIAaQB0AHkAMQAyADMANA=="
        },
               {
          "id": 2,
          "content": "AFAAcgBvAHQAZQBnAHIAaQB0AHkAMQAyADMANA=="
        }
      ]
    }
  }
}'
```

Response 1 - without external IV and external tweak

The following response appears for the status code 200, if the API is invoked successfully.

```
{
   "protect":{
      "bulk":{
         "id":1,
         "returntype":"success",
         "data":[
            {
               "id":1,
               "returncode":"/rest-v1/returncodes/id/6",
               "returntype":"success",
               "content":"AGoAZABzAHIAdQBlAGMAagBaAEMAMQAyADMANA=="
            },
            {
               "id":2,
               "returncode":"/rest-v1/returncodes/id/6",
               "returntype":"success",
               "content":"AGoAZABzAHIAdQBlAGMAagBaAEMAMQAyADMANA=="
            }
         ]
      }
```

```
   }
}
```

Example 2 - with external IV

```
$ curl --location --request POST 'https://<hostname>/rest-v1/protect' \
--connect-to  "<hostname>:443:<AWS LoadBalancer>:443"  \
--header 'Content-Type: application/json' \
 --cacert iap-rest-ca.crt --cert iap-rest-client.crt  --key iap-rest-client.key  --data
'{
  "protect": {
    "policyusername": "Uername",
    "dataelementname": "DataElement1",
    "externaliv": "ZXh0ZXJuYWpdg=="
    "bulk":{
      "id": 1,
      "data": [
        {
          "id": 1,
          "content": "RW5eEN2RGZZaw=="
        },
              {
          "id": 2,
          "content": "cmZBcnJTRg=="
        }
      ]
    }
  }
}'
```

Response 2 - with external IV

The following response appears for the status code 200, if the API is invoked successfully.

```
{
   "protect":{
      "bulk":{
         "id":1,
         "returntype":"success",
         "data":[
            {
               "id":1,
               "returncode":"/rest-v1/returncodes/id/6",
               "returntype":"success",
               "content":"OG8xZW0QlQ3MQ=="
            },
            {
               "id":2,
               "returncode":"/rest-v1/returncodes/id/6",
               "returntype":"success",
               "content":"blg2Qm5Ddg=="
            }
         ]
      }
   }
}
```

Example 3 - with external tweak

```
$ curl --location --request POST 'https://<hostname>/rest-v1/protect' \
--connect-to  "<hostname>:443:<AWS LoadBalancer>:443"  \
--header 'Content-Type: application/json' \
 --cacert iap-rest-ca.crt --cert iap-rest-client.crt  --key iap-rest-client.key  --data
'{
  "protect": {
    "policyusername": "Uername",
    "dataelementname": "DataElement2_FPE",
    "externaltweak": "ZXh0ZXJuYWpdg=="
    "bulk":{
      "id": 1,
      "data": [
```

```
        {
          "id": 1,
          "content": "RW5eEN2RGZZaw=="
        },
              {
          "id": 2,
          "content": "cmZBcnJTRg=="
        }
      ]
    }
  }
}'
```

Response 3 - with external tweak

The following response appears for the status code 200, if the API is invoked successfully.

```
{
   "protect":{
      "bulk":{
         "id":1,
         "returntype":"success",
         "data":[
            {
               "id":1,
               "returncode":"/rest-v1/returncodes/id/6",
               "returntype":"success",
               "content":"MHM4OVpsRndIbA=="
            },
            {
               "id":2,
               "returncode":"/rest-v1/returncodes/id/6",
               "returntype":"success",
               "content":"VzFsNmd1Ng=="
            }
         ]
      }
   }
}
```

9.1.3 HTTP POST unprotect

This API unprotects the protected data.

URI

https://hostname/rest-v1/unprotect

Method

POST

Parameters

Hostname: Host name of the endpoint, as defined in the AP-REST deployment

Resource: The resource to be used, which is */rest-v1/unprotect*

Result

This API returns unprotected data.

Example 1 - without external IV and external tweak

```
$ curl --request POST 'https://<hostname>/rest-v1/unprotect' \
--connect-to  "<hostname>:443:<AWS LoadBalancer>:443"  \
--header 'Content-Type: application/json' \
 --cacert iap-rest-ca.crt --cert iap-rest-client.crt  --key iap-rest-client.key  --data
'{
  "unprotect": {
    "policyusername": "UserName",
    "dataelementname": "DataElement1",
    "bulk":{
```

```
      "id": 1,
      "data": [
        {
          "id": 1,
          "content": "AFAAcgBvAHQAZQBnAHIAaQB0AHkAMQAyADMANA=="
        },
              {
          "id": 2,
          "content": "AFAAcgBvAHQAZQBnAHIAaQB0AHkAMQAyADMANA=="
        }
      ]
    }
  }
}'
```

Response 1 - without external IV and external tweak

The following response appears for the status code 200, if the API is invoked successfully.

```
{
   "unprotect":{
      "bulk":{
         "id":1,
         "returntype":"success",
         "data":[
            {
               "id":1,
               "returncode":"/rest-v1/returncodes/id/8",
               "returntype":"success",
               "content":"AGwATgBWAEwATAByAFIAUAB2AGcAMQAyADMANA=="
            },
            {
               "id":2,
               "returncode":"/rest-v1/returncodes/id/8",
               "returntype":"success",
               "content":"AGwATgBWAEwATAByAFIAUAB2AGcAMQAyADMANA=="
            }
         ]
      }
   }
}
```

Example 2 - with external IV

```
$ curl --request POST 'https://<hostname>/rest-v1/unprotect' \
--connect-to  "<hostname>:443:<AWS LoadBalancer>:443"  \
--header 'Content-Type: application/json' \
 --cacert iap-rest-ca.crt --cert iap-rest-client.crt  --key iap-rest-client.key  --data
'{
  "unprotect": {
    "policyusername": "UserName",
    "dataelementname": "DataElement1",
    "externaliv": "ZXh0ZXJuYWpdg=="
    "bulk":{
      "id": 1,
      "data": [
        {
          "id": 1,
          "content": "OG8xZW0QlQ3MQ=="
        },
              {
          "id": 2,
          "content": "blg2Qm5Ddg=="
        }
      ]
    }
  }
}'
```

Response 2 - with external IV

The following response appears for the status code 200, if the API is invoked successfully.

```
{
   "unprotect":{
      "bulk":{
         "id":1,
         "returntype":"success",
         "data":[
            {
               "id":1,
               "returncode":"/rest-v1/returncodes/id/8",
               "returntype":"success",
               "content":"RW5eEN2RGZZaw=="
            },
            {
               "id":2,
               "returncode":"/rest-v1/returncodes/id/8",
               "returntype":"success",
               "content":"cmZBcnJTRg=="
            }
         ]
      }
   }
}
```

Example 3 - with external tweak

```
$ curl --request POST 'https://<hostname>/rest-v1/unprotect' \
--connect-to  "<hostname>:443:<AWS LoadBalancer>:443"  \
--header 'Content-Type: application/json' \
 --cacert iap-rest-ca.crt --cert iap-rest-client.crt  --key iap-rest-client.key  --data
'{
  "unprotect": {
    "policyusername": "UserName",
    "dataelementname": "DataElement2_FPE",
    "externaltweak": "ZXh0ZXJuYWpdg=="
    "bulk":{
      "id": 1,
      "data": [
        {
          "id": 1,
          "content": "MHM4OVpsRndIbA=="
        },
              {
          "id": 2,
          "content": "VzFsNmd1Ng=="
        }
      ]
    }
  }
}'
```

Response - with external tweak

The following response appears for the status code 200, if the API is invoked successfully.

```
{
   "unprotect":{
      "bulk":{
         "id":1,
         "returntype":"success",
         "data":[
            {
               "id":1,
               "returncode":"/rest-v1/returncodes/id/8",
               "returntype":"success",
               "content":"RW5eEN2RGZZaw=="
            },
            {
               "id":2,
```

```
                "returncode":"/rest-v1/returncodes/id/8",
                "returntype":"success",
                "content":"cmZBcnJTRg=="
             }
          ]
       }
    }
}
```

9.1.4 HTTP POST reprotect

This API reprotects the data.

URI

https://hostname/rest-v1/reprotect

Method

POST

Parameters

Hostname: Host name of the endpoint, as defined in the AP-REST deployment

Resource: The resource to be used, which is */rest-v1/reprotect*

Result

This API reprotects the data.

Example 1 - without external IV and external tweak

```
$ curl --request POST 'https://<hostname>/rest-v1/reprotect' \
--connect-to  "<hostname>:443:<AWS LoadBalancer>:443"  \
--header 'Content-Type: application/json' \
 --cacert iap-rest-ca.crt --cert iap-rest-client.crt  --key iap-rest-client.key  --data
'{
  "reprotect": {
    "policyusername": "UserName",
    "olddataelementname": "DataElement1", "newdataelementname": "DataElement2",
    "bulk":{
      "id": 1,
      "data": [
        {
          "id": 1,
          "content": "AFAAcgBvAHQAZQBnAHIAaQB0AHkAMQAyADMANA=="
        },
              {
          "id": 2,
          "content": "AFAAcgBvAHQAZQBnAHIAaQB0AHkAMQAyADMANA=="
        }
      ]
    }
  }
}'
```

Response 1 - without external IV and external tweak

The following response appears for the status code 200, if the API is invoked successfully.

```
{
   "reprotect":{
      "bulk":{
         "id":1,
         "returntype":"success",
         "data":[
            {
               "id":1,
               "returncode":"/rest-v1/returncodes/id/6",
               "returntype":"success",
               "content":"AFAAcgBvAHQAZQBnAHIAaQB0AHkAMQAyADMANA=="
            },
```

```
            {
               "id":2,
               "returncode":"/rest-v1/returncodes/id/6",
               "returntype":"success",
               "content":"AFAAcgBvAHQAZQBnAHIAaQB0AHkAMQAyADMANA=="
            }
         ]
      }
   }
}
```

Example 2 - with external IV

```
curl --location --request POST 'https://<hostname>/rest-v1/reprotect' \
--connect-to  "<hostname>:443:<AWS LoadBalancer>:443"  \
--header 'Content-Type: application/json' \
 --cacert iap-rest-ca.crt --cert iap-rest-client.crt --key iap-rest-client.key  --data
'{
  "reprotect": {
    "policyusername": "UserName",
    "olddataelementname": "DataElement1",
    "newdataelementname":
"DataElement2","oldexternaliv":"MTIzNDVhYmNzIyQlXiM2Nzg5MFMrTlNBQkNTRA=","newexternaliv"
:"MTIzNDVhYmNzIyQlXiM2Nzg5MFMrTlNBQkNTRA="
    "bulk":{
      "id": 1,
      "data": [
        {
          "id": 1,
          "content": "MTA1MTYwNTk1MjE5OTY3OTU="
        },
              {
          "id": 2,
          "content": "MTA1MTYwNTk1MjE5OTY3OTU="
        }
      ]
    }
  }
}'
```

Response 2 - with external IV

The following response appears for the status code 200, if the API is invoked successfully.

```
{
   "reprotect":{
      "bulk":{
         "id":1,
         "returntype":"success",
         "data":[
            {
               "id":1,
               "returncode":"/rest-v1/returncodes/id/6",
               "returntype":"success",
               "content":"Q09udGFpbmVyVGVhbTEyMzQ1Njc="
            },
            {
               "id":2,
               "returncode":"/rest-v1/returncodes/id/6",
               "returntype":"success",
               "content":"AFAAcgBvAHQAZQBnAHIAaQB0AHkAMQAyADMANAA1"
            }
         ]
      }
   }
}
```

Example 3 - with external tweak

```
curl --location --request POST 'https://<hostname>/rest-v1/reprotect' \
--header 'Host: <hostname>' \
```

```
--connect-to  "<hostname>:443:<AWS LoadBalancer>:443"  \
--header 'Content-Type: application/json' \
 --cacert iap-rest-ca.crt --cert iap-rest-client.crt --key iap-rest-client.key  --data
'{
  "reprotect": {
    "policyusername": "UserName",
    "olddataelementname": "DataElement1",
    "newdataelementname":
"DataElement2","oldexternaltweak":"MTIzNDVhYmNzIyQlXiM2Nzg5MFMrTlNBQkNTRA=","newexternal
tweak":"MTIzNDVhYmNzIyQlXiM2Nzg5MFMrTlNBQkNTRA="
    "bulk":{
      "id": 1,
      "data": [
        {
          "id": 1,
          "content": "MTA1MTYwNTk1MjE5OTY3OTU="
        },
                {
          "id": 2,
          "content": "MTA1MTYwNTk1MjE5OTY3OTU="
        }
      ]
    }
  }
}'
```

Response 3 - with external tweak

The following response appears for the status code 200, if the API is invoked successfully.

```
{
   "reprotect":{
      "bulk":{
         "id":1,
         "returntype":"success",
         "data":[
            {
               "id":1,
               "returncode":"/rest-v1/returncodes/id/6",
               "returntype":"success",
               "content":"AFAAYQByAGgAbQBoAFAAawBMAGcAZQBaAFgAaABtAGEAcg"
            },
            {
               "id":2,
               "returncode":"/rest-v1/returncodes/id/6",
               "returntype":"success",
               "content":"ADEAMgAzADQANQA2ADcAOAA5ADA"
            }
         ]
      }
   }
}
```

9.1.5 HTTP Headers

The client should send the required HTTP headers to the server to specify the type of data being sent in the payload. The content type also specifies the type of result being sent by the server to the client.

To send a JSON request and get a JSON response, specify the following HTTP header:

Content-Type: application/json

9.2 Error Handling

For record error handling, the *bulk id* and *data id* fields are used, which enable tracking of the errors from the client side.

The following table lists the record error handling status codes, which are sent from the server to the client.

Table 9-1: Record Error Handling Status Codes

Status Code	Responses

Success	{
	"bulk":{
	"id":1,
	"returntype":"success",
	"data":[
	{
	"id":1,
	"returncode":"/rest-v1/returncodes/id/6",
	"returntype":"success",
	"content":"AGoAZABzAHIAdQBlAGMAagBaAEMAMQAyADMANA=="
	},
	{
	"id":2,
	"returncode":"/rest-v1/returncodes/id/6",
	"returntype":"success",
	"content":"AGoAZABzAHIAdQBlAGMAagBaAEMAMQAyADMANA=="
	}
	]
	}
	}

Success, with warning	{
	"bulk":{
	"id":1,
	"returntype":"warning",
	"data":[
	{
	"id":1,
	"returntype":"warning",
	"content":null
	},
	{
	"id":2,
	"returntype":"warning",
	"content":null
	}
	]
	}
	}

Error type of log return code	{
	"bulk":{
	"id":1,
	"returntype":"error",
	"data":[
	{
	"id":1,
	"message":"Data is too short to be protected/unprotected.",
	"returncode":"/rest-v1/returncodes/id/22",
	"returntype":"error"
	},
	{
	"id":2,
	"message":"Data is too short to be protected/unprotected.",
	"returncode":"/rest-v1/returncodes/id/22",
	"returntype":"error"
	}
	]
	}
	}

Status Code	Responses

Error type of log return code (different)	{
	"bulk":{
	"id":1,
	"returntype":"error",
	"data":[
	{
	"id":1,
	"message":"Data is too short to be protected/unprotected.",
	"returncode":"/rest-v1/returncodes/id/22",
	"returntype":"error"
	},
	{
	"id":2,
	"returncode":"/rest-v1/returncodes/id/6",
	"returntype":"success",
	"content":"AGoAZABzAHIAdQBlAGMAagBaAEMAMQAyADMANA=="
	}
	]
	}
	}

9.3 AP REST API Return Codes

When you develop an application using the Application Protector (AP) REST APIs, you may encounter the errors described in this section. You can avoid most of the errors if you accurately use the API.

For more information, refer to the section *AP REST APIs*.

9.3.1 AP REST API Log Return Error Codes

This section lists the log return error codes returned by the AP REST API as a result of policy exceptions. Audits are generated in the ESA for these errors.

Table 9-2: AP REST API Log Return Codes

Code	Error Message	Error Description
1	1, No such user.	The user name could not be found in the policy residing in the shared memory.
2	2, Data element not found.	The data element could not be found in the policy residing in the shared memory.
3	3, Permission denied.	The user does not have the required permissions to perform the requested operation.
5	5, Integrity check failed.	The data integrity check failed when decrypting using a Data Element with CRC enabled.
6	6, Protect success.	The operation to protect the data was successful.
7	7, Protect failed.	The operation to protect the data failed.
8	8, Unprotect success.	The operation to unprotect the data was successful.
9	9, Unprotect failed.	The operation to unprotect the data failed.
10	10, Policy check OK.	The user has the required permissions to perform the requested operation. This return code ensures a verification and no data is protected or unprotected.

Code	Error Message	Error Description
11	11, Inactive key id used.	The operation to unprotect the data was successful using an inactive Key ID.
12	12, Invalid parameter.	Input parameters are either NULL or not within allowed limits.
13	13, Internal error.	Internal error occurring in a function call after the PEP provider has been opened. For example: - ***failed to get mutex / semaphore, - unexpected null param***.
14	14, Failed to load key.	A key for a data element could not be loaded from shared memory into the Crypto engine.
15	15, Tweak input is too long.	Tweak input is too long.
17	17, Init failed.	The PEP server failed to initialize, which is a fatal error.
19	19, Unsupported tweak action for the specified fpe dataelement	The external tweak is not supported for the specified FPE data element.
20	20, Out of memory.	Failed to allocate memory.
21	21, Buffer too small.	The input or output buffer is very small.
22	22, Input too short.	The data is too short to be protected or unprotected.
23	23, Input too long.	The data is too long to be protected or unprotected.
25	25, User name too long.	The user name is longer than the maximum supported length of the user name that can be used for protect or unprotect operations.
26	26, Unsupported.	The algorithm or action for the specific data element is unsupported.
31	31, Empty policy.	The policy residing in the shared memory is empty.
39	39, Policy locked.	The policy residing in the shared memory is locked. This error can be caused by a *Disk Full* alert.
40	40, License expired.	The license is invalid or the current date is beyond the license expiry date.
41	41, Method restricted.	The use of the Protection method is restricted by the license.
42	42, Invalid license.	The license is invalid or the time is prior to the start of the license tenure.
44	44, Invalid format.	The content of the input data is invalid.

9.3.2 AP REST API PEP Result Codes

This section lists the PEP result codes returned by AP REST APIs as a result of system exceptions. Audits are not generated in the ESA for these errors.

Table 9-3: AP REST API PEP Result Codes

Code	Error Message	Error Description
-1	-1, Invalid parameter	The parameter is invalid.
-7	-7, Error when parsing contents, e.g.	The error occurred when the contents were parsed.
-8	-8, Not found!	The search operation was not successful.
-16	-16, Buffer is too small	The buffer size is very small.
-43	-43, Invalid format	The format is invalid.
-46	-46, Requesting service/function on an object that is not initialized	The service requested or function is performed on an object that is not initialized.
-47	-47, Policy locked for some reason	The Policy is locked.

9.4 AP REST HTTP Response Codes

This section lists the response codes generated for the HTTP REST requests sent to the AP REST APIs. It also specifies the corresponding audit code generated in the logs.

Table 9-4: AP REST HTTP Response Codes

Scenario	Operation	*Audit Code in Logs*	HTTP Response Code
Failed to decode Base64	• Protect • Unprotect • Reprotect	No audit code generated	400
User name not present in policy	Protect	1	401
User name not present in policy	Unprotect	1	200
User name not present in policy	Reprotect	1	401
Data element not found in policy	• Protect • Unprotect	2	400
Data element not found in policy	Reprotect	2	400
No access to data element	Protect	3	403
No access to data element	Unprotect (The API returns an Exception)	3	403
No access to data element	Unprotect (The API returns the protected value)	3	200
No access to data element	Unprotect (The API returns a Null value)	3	200
No access to data element	Reprotect	3	403
Integrity check failed	Unprotect	5	400
Invalid parameter	• Protect • Unprotect • Reprotect	12	400
Failed to load the data encryption key	Unprotect	14	400
Input too short	• Protect • Unprotect • Reprotect	22	200
Input too long	• Protect • Unprotect • Reprotect	23	200
Unsupported algorithm or unsupported action for the specific data element	Unprotect	26	400
Invalid format	• Protect • Unprotect • Reprotect	44	200
Any other *policy exception errors* apart from the ones already listed in this table **Important:** This scenario is only applicable for the 9.1.0.0 release and	Any	Any	400

Scenario	Operation	*Audit Code in Logs*	HTTP Response Code
not for the 7.1 MR4 and 9.0.0.0 releases.			

Important: The HTTP response codes generated for the 9.1.0.0 release differ from the ones that are generated for the 7.1 MR4 and 9.0.0.0 releases.