Protegrity Application Protector Java Immutable Policy User Guide for Azure 9.1.0.0

Created on: Nov 19, 2024

Copyright

Table of Contents

Chapter 1

Protegrity Security Operations Cluster using Kubernetes

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

1.1 Business Problem

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

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

1.2 Protegrity Solution

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

- Cloud standard Application Protector Java form factor:
  - The delivery form factor for cloud deployments is an SDK and a supporting Dockerfile. Customers can use this Dockerfile to build an AP Java Container, which is based on the Application Protector form factor that Protegrity been delivering for several years.
  - The AP Java Container is a standard Docker Container that is familiar and expected in cloud deployments. Customers can create their own container image by integrating their applications with the AP Java libraries.
  - The AP Java Container is a lightweight deployment of the AP Java.
- Immutable Deployment:
  - Cloud deployments or containers do not change dynamically – they are immutable. In other words, when there is a change in Policy, the change is carried out by terminating the existing AP Java Container with the Policy and instantiating a new one.
  - Changes to Policy or the AP Java Container happen through special deployment strategies available through Kubernetes. For Protegrity deployments, the recommended deployment strategy is a *Blue / Green* strategy.
- Auto Scalability:
  - Kubernetes can be set up to autoscale based on setting a configuration on CPU thresholds.

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

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

1.3 Architecture and Components

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

The IAP deployment consists of the following two stages:

- Stage 1: Deploying the Get ESA Policy Container on a Kubernetes cluster to fetch the ESA policy and create the policy package.
- Stage 2: Deploying the Sample Application on a Kubernetes cluster, which uses the policy package created in stage 1.

The following describes the architecture diagram and the steps for Stage 1 - Deploying the Get ESA Policy container on a Kubernetes cluster.

The following describes the architecture diagram and the steps for deploying the Get ESA Policy container on a Kubernetes cluster.

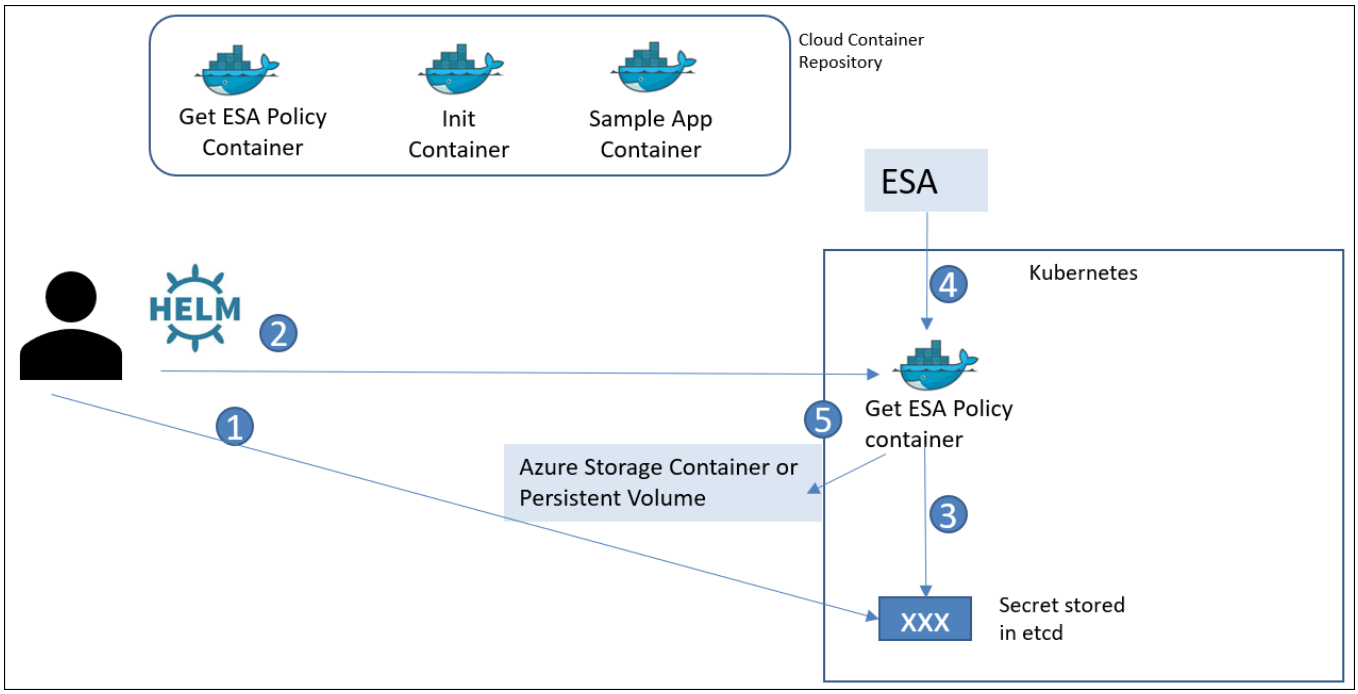


Figure 1-1: Stage 1: Deploying the Get ESA Policy Container

1. The user stores the ESA credentials in Kubernetes secret. The user also stores the passphrase and the salt for encrypting the policy in the Kubernetes secret. The user can also use the Azure Key Management System (KMS) for encrypting the policy.
2. The user runs the Helm chart to deploy the Get ESA Policy container as a Kubernetes job.
3. The Get ESA Policy container retrieves the policy from the ESA, creates a snapshot of the immutable policy, and encrypts it.
4. The Get ESA Policy container then uploads the policy package to an Azure storage container or a persistent volume.
5. The Kubernetes job is completed.

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

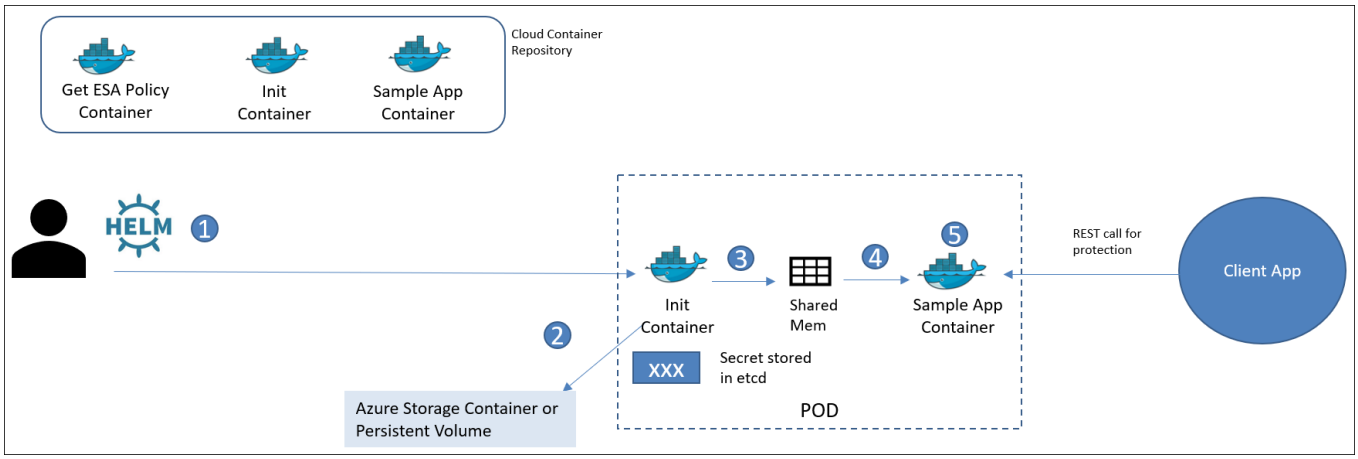


Figure 1-2: Stage 2: Deploying the Sample Application Container

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

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

2. The Init Container is initialized first. It reads the immutable policy package from the Azure storage container or a persistent volume, decrypts it, and uploads the policy in the shared memory of the Pod and then exits.
3. The Sample Application container is then initialized. It uses the policy on the shared memory.
4. The Client Application sends a request to the Sample application over REST for protecting, unprotecting, and reprotecting data.
5. The Sample Application internally sends the request to the Application Protector Java, and then returns the accurate value to the client application.

Chapter 2

Immutable Application Protector (IAP) Deployment Model

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

2.1 Verifying the Prerequisites

Ensure that the following prerequisites are met:

Reference Solution Deployment

- *IAP_RHEL-ALL-64_x86-64_AZURE.K8S.JRE-1.7_<IAP_version>.tgz* – Installation package for the IAP Deployment. This package contains the following packages:
  - *IAP-GET-ESA-POLICY-HELM_ALL-ALL-ALL_x86-64_AZURE.K8S_<version_number>.tgz* - Package containing the Helm charts, which are used to deploy the Get ESA Policy application on the Kubernetes cluster. Helm is a package manager for Kubernetes.
  - *IAP-GET-ESA-POLICY_RHEL-ALL-64_x86-64_AZURE.K8S_<version_number>.tar.gz* - Get ESA Policy container image. This image is used to create the Get ESA Policy container, which is used to generate an immutable snapshot of the policy and upload this snapshot to the cloud storage in the Cloud Runtime Environment.
  - *IAP-INIT-CONTAINER_RHEL-ALL-64_x86-64_AZURE.K8S_<version_number>.tar.gz* - Init container image. This image is used to create the Init container, which is used to retrieve the policy snapshot stored in the persistent volume and make the snapshot available to the Sample Application container.
  - *APJavaIMSetup_<OS>_x64_<App_Protector_version>.tgz* - Immutable AP Java installation package.
  - *IAP-SAMPLE-APP-HELM_ALL-ALL-ALL_x86-64_K8_<version_number>.tgz* – Package containing the Helm charts, which are used to deploy the sample application on the Kubernetes cluster.
  - *IAP-GET-ESA-POLICY_AZURE_SRC_<version_number>.tgz* - Package containing the Dockerfile that can be used to create a custom image for the Get ESA Policy 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*.

- *IAP-INIT-CONTAINER_AZURE_SRC_<version_number>.tgz* - Package containing the Dockerfile that can be used to create a custom image for the Init 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*.

- *IAP-SAMPLE-APP_AZURE_SRC_<version_number>.tgz* - Package containing the Dockerfile that can be used to create a custom image for the Sample Application container and the associated binary files.

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

- *HOW-TO-BUILD-DOCKER-IMAGES* - Text file specifying how to build the Sample Application and Docker images.
- *manifest.json* - File specifying the name of the product and its components, the version number of the protector, and the build number of product.

Reference Application

The following prerequisites are required for using the reference application:

- Policy – Ensure that you have defined the security policy in the ESA 9.1.0.0.

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

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

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

  > **Important:** If you are deploying a Customer Application container, then you must apply the NSS wrapper, which is a *nss-wrapper* library for the Name Service Switch (NSS) API, to the Customer Application container image.
  >
  > For more information about applying the NSS wrapper to the Customer Application container image, refer to the section *Appendix C: Applying the NSS Wrapper to the Customer Application Container Image*.
  >
  > For more information about the *nss_wrapper* library, refer to the section *The nss_wrapper library*.

  For more information about setting up a Trusted Application, refer to the section *Working with Trusted Applications* in the *Policy Management Guide 9.1.0.0*.

- Non-admin ESA user - Create a non-admin ESA user that will be used by the Get ESA Policy container to retrieve the security policy and the certificates from the ESA. Ensure that the user is assigned the *Export Certificates* and the *Appliance CLI Viewer* roles.

  For more information about assigning roles, refer to the section *Managing Roles* in the *Appliances Overview Guide 9.1.0.0*.

Additional Prerequisites

The following additional prerequisites are required:

- *Linux instance* - This instance can be used to communicate with the Kubernetes cluster. This instance can be on-premise or on Azure. Ensure that Helm is installed on this Linux instance. You can also choose to install Docker on this Linux instance to communicate with the Container Registry, where you want to upload the Docker images.
- *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 Azure.
- Access to an Azure subscription and a valid Azure Active Directory user for creating the following Azure services:

- Resource Group - Used to organize resources. All the remaining Azure services are created within a Resource Group.
- Azure Storage Account for creating the Azure Storage Container or the Azure File Share, where the immutable policy package will be uploaded.
- Azure File Share, if you want to use Azure File Share for uploading the immutable policy package, instead of using an Azure Storage Container.
- Azure Storage Container - This prerequisite is optional, and is required only if you want to use the Azure Storage Container for storing the policy package, instead of the Azure File Share.
- Two service principals:
  - **Service principal 1** with *Write* permissions on the Azure Storage Account and *Get* permission on Key Management of the Azure Key Vault
  - **Service principal 2** with *Read* permissions on the Azure Storage Account and *Decrypt* permission on Key Management of the Azure Key Vault

> **Note:** A service principal is created when you register an application on the Azure portal

- Access to Azure Kubernetes Service (AKS) to create a Kubernetes cluster
- Access to Azure Container Registry to upload the Get ESA Policy, Init, Sample Application container images

2.2 Downloading the Installation Package

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

To download the installation package:

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

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

   The following files are extracted:
   - *IAP-GET-ESA-POLICY-HELM_ALL-ALL-ALL_x86-64_AZURE.K8S_<version_number>.tgz*
   - *IAP-GET-ESA-POLICY_RHEL-ALL-64_x86-64_AZURE.K8S_<version_number>.tar.gz*
   - *IAP-INIT-CONTAINER_RHEL-ALL-64_x86-64_AZURE.K8S_<version_number>.tar.gz*
   - *APJavaIMSetup_<OS>_x64_<App_Protector_version>.tgz*
   - *IAP-SAMPLE-APP-HELM_ALL-ALL-ALL_x86-64_K8_<version_number>.tgz*
   - *IAP-GET-ESA-POLICY_AZURE_SRC_<version_number>.tgz*
   - *IAP-INIT-CONTAINER_AZURE_SRC_<version_number>.tgz*
   - *IAP-SAMPLE-APP_AZURE_SRC_<version_number>.tgz*
   - *HOW-TO-BUILD-DOCKER-IMAGES*
   - *manifest.json*

2.3 Initializing the Linux Instance

This section describes how you can initialize the Linux instance.

To initialize the Linux instance:

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

   For more information about installing the Azure CLI on the Linux instance, refer to the section *Install the Azure CLI* in the Microsoft Azure documentation.
2. Start the Azure CLI with the following command.

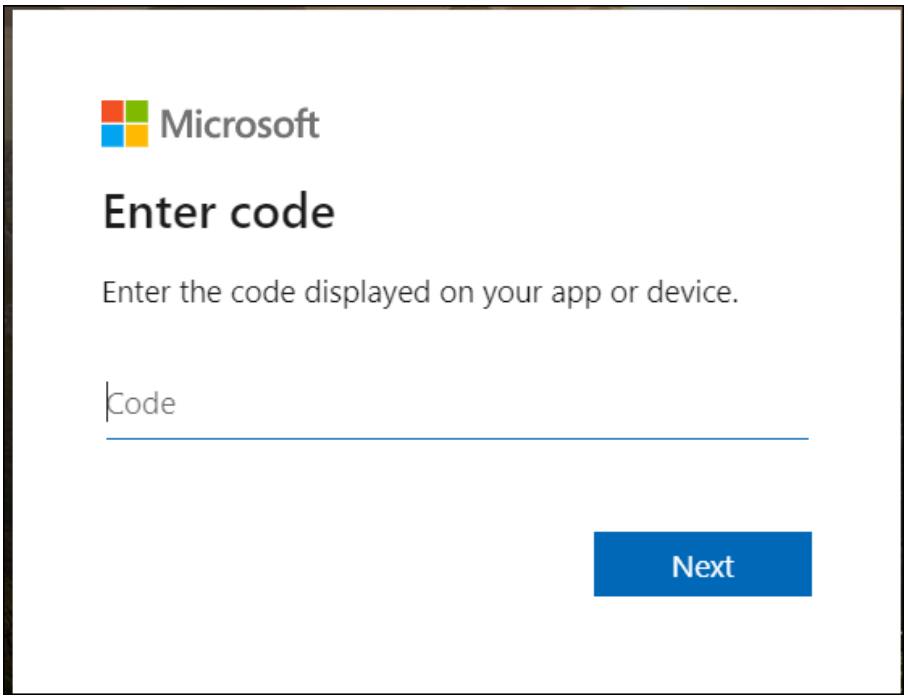
   ```
   az login
   ```

   If the CLI is able to open your default browser, then the Azure sign-in page appears.

   If the CLI is unable to open your default browser, then a verification URL appears.

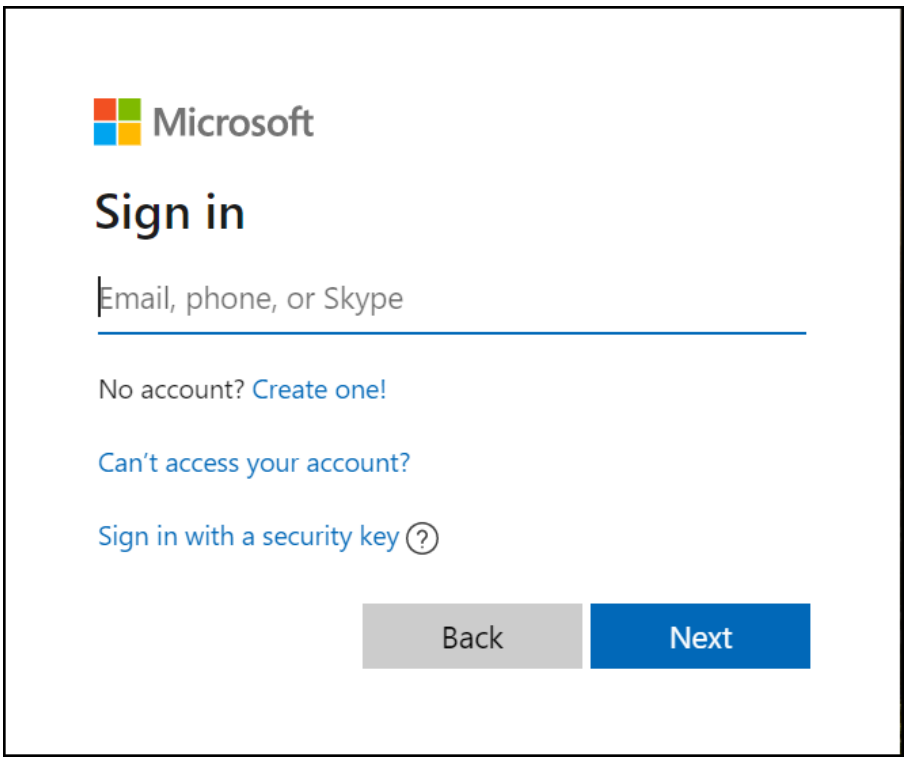
   ```
   Not able to launch a browser to log you in, falling back to device code...
   To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code G79EK67W6 to authenticate.
   ```
3. Copy the URL and paste it in a browser, and then press **Enter**.

   You are prompted to enter the verification code that appears on the console of the Linux instance.



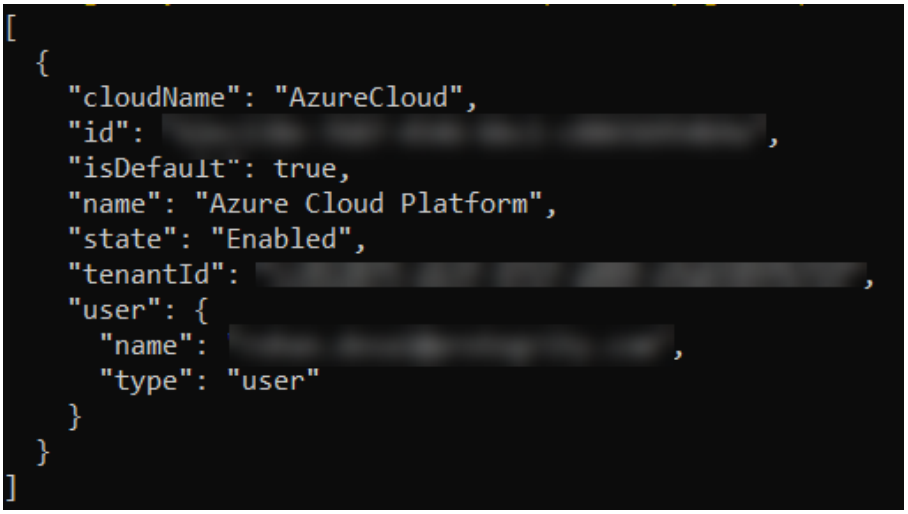
4. Enter the verification code, and then click **Next**.

   You are prompted to sign in to a Microsoft account that will be used to login to the Microsoft Azure CLI on your Linux instance.



5. Enter the email address, and then click **Next**.
   A dialog box appears that prompts you to enter your password.
6. Enter your password required for logging into Azure CLI, and then click **Sign in**.
   A screen appears that informs you have signed in to the Azure CLI on the Linux instance.

If you navigate to the Linux instance, then the subscription information appears on the console.



7. Install Kubectl, which is the command line interface for Kubernetes using the following command.

   ```
   az aks install-cli
   ```

   Kubectl enables you to run commands from the Linux instance so that you can communicate with the Kubernetes cluster.

8. 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
   ```

9. Run the following command to extract the *.tgz* package containing the Helm charts for deploying the Get ESA Policy container.

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

   For example:

   ```
   tar -xvf IAP-GET-ESA-POLICY-HELM_ALL-ALL-ALL_x86-64_AZURE.K8S_<version_number>.tgz
   ```

   The extracted package contains the *get-esa-policy* directory, which has the following contents:

   - *Chart.yaml* – A YAML file that provides information about the chart
   - *encryption-config.yaml* – Configuration file used for encrypting the immutable policy
   - *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-rbac.yaml* – The values for configuring the Protegrity Role Based Access Control (RBAC) for the pods

- *credentials.json* – Credentials file for the service prinicpal 1, which is required for encrypting the policy packages that are uploaded on Azure. By default, the *credentials.json* file contains placeholders, which need to be replaced with the actual values applicable for the service principal 1.

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

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

    For example:

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

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

    - *Chart.yaml* – A YAML file that provides information about the chart
    - *templates* – A directory that contains the templates required to generate the Kubernetes manifest files. Kubernetes uses the manifest files to deploy the application on the Kubernetes cluster.
    - *values.yaml* – The default values for configuring the Helm chart
    - *credentials.json* – Credentials file for the service prinicpal 2, which is required for decrypting the policy packages that are uploaded on Azure. By default, the *credentials.json* file contains placeholders, which need to be replaced with the actual values applicable for the service principal 2.
    - *decryption-config.yaml* – Configuration file used for decrypting the immutable policy
    - *pty-rbac.yaml* – The values for configuring the Protegrity Role Based Access Control (RBAC) for the pods.

    > **Note:** Ignore the remaining files in the *spring-apjava* directory as they are not required for deploying the Application Protector Java on Azure.

2.4 Creating Certificates and Keys for TLS Authentication

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

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

Before you begin
Ensure that OpenSSL is installed on the Linux instance to create the required certificates.

➤ To initialize the Linux instance:

1. On the Linux instance, run the following command to create a CA certificate and a private key for the certificate.

   ```
   openssl req -x509 -sha256 -newkey rsa:2048 -keyout iap-ca.key -out iap-ca.crt -days 356 -nodes -subj '/CN=IAP Certificate Authority'
   ```

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

2. On the Linux instance, create a server certificate and a private key that have been signed using the private key of the CA created in *step 1*.

```
openssl req -new -newkey rsa:2048 -keyout iap-wildcard.key -out iap-wildcard.csr
-nodes -subj '/CN=*.example.com'

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

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

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

3. On the Linux instance, create a client certificate and key that have been signed using the private key of the CA created in *step 1*.

```
openssl req -new -newkey rsa:2048 -keyout iap-client.key -out iap-client.csr -nodes
-subj '/CN=My IAP Client'

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

4. Copy all the certificates to a common directory.

   For example, create a directory named *iap-certs* and copy all the certificates that have been created to this directory.

2.5 Uploading the Images to the Container Repository

This section describes how you can upload the Get ESA Policy, Init, and Sample Application images to the Container Repository.

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

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

> **Important:** If you are deploying a Customer Application container, instead of the Sample Application container, then ensure that the Init container and the Customer Application container have the same user ID. This ensures that the Customer Application container can access the shared memory.

> **Note:** Before deploying the Customer Application container, ensure that you have applied the NSS wrapper to the Customer Application container image.
>
> For more information about applying the NSS wrapper to the Customer Application container image, refer to the section *Appendix C: Applying the NSS Wrapper to the Customer Application Container Image*.

➤ To upload the images to the Container Repository:

1. Install Docker on the Linux instance.

   For more information about installing Docker on a Linux machine, refer to the *Docker documentation*.
2. Login to the Azure Container Registry using the following command.

```
docker login <Container Registry Name>.azurecr.io
```

3. Extract the installation package.

   The Get ESA Policy, Init, and Sample Application container images are 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 Get ESA Policy image to Azure Container Registry.
   a. Run the following command to load the Get ESA Policy container image on Docker.
   ```
   docker load -i IAP-GET-ESA-POLICY_RHEL-
   ALL-64_x86-64_AZURE.K8S_<IAP_version>.tar.gz
   ```
   b. Run the following command to list the Get ESA Policy container image.
   ```
   docker images
   ```
   c. Tag the image to the Azure Container Registry by running the following command.
   ```
   docker tag <Container image>:<Tag> <Container registry path>/<Container
   image>:<Tag>
   ```
   For example:

   ```
   docker tag get-esa-policy:AZURE <Container Registry Name>.azurecr.io/get-esa-
   policy:AZURE
   ```

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

   d. Push the tagged image to the Azure Container Registry by running the following command.
   ```
   docker push <Container registry path>/<Container image>:<Tag>
   ```
   For example:

   ```
   docker push <Container Registry Name>.azurecr.io/get-esa-policy:AZURE
   ```

5. Repeat *step 4* for pushing the Init and Sample Application container images to the Azure Container Registry.
6. Navigate to the directory where you have extracted the Helm charts packages for the Get ESA Policy and Spring Boot Application containers.
7. In the *values.yaml* file, update the required path for the *getEsaPolicyImage*, *initImage* and *springappImage* settings, with their tags.

2.6 Creating the Cloud Runtime Environment

This section describes how to create the Cloud runtime environment.

2.6.1 Creating the Azure Runtime Environment

This section describes how to create the Azure runtime environment.

Prerequisites

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

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

Audience

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

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

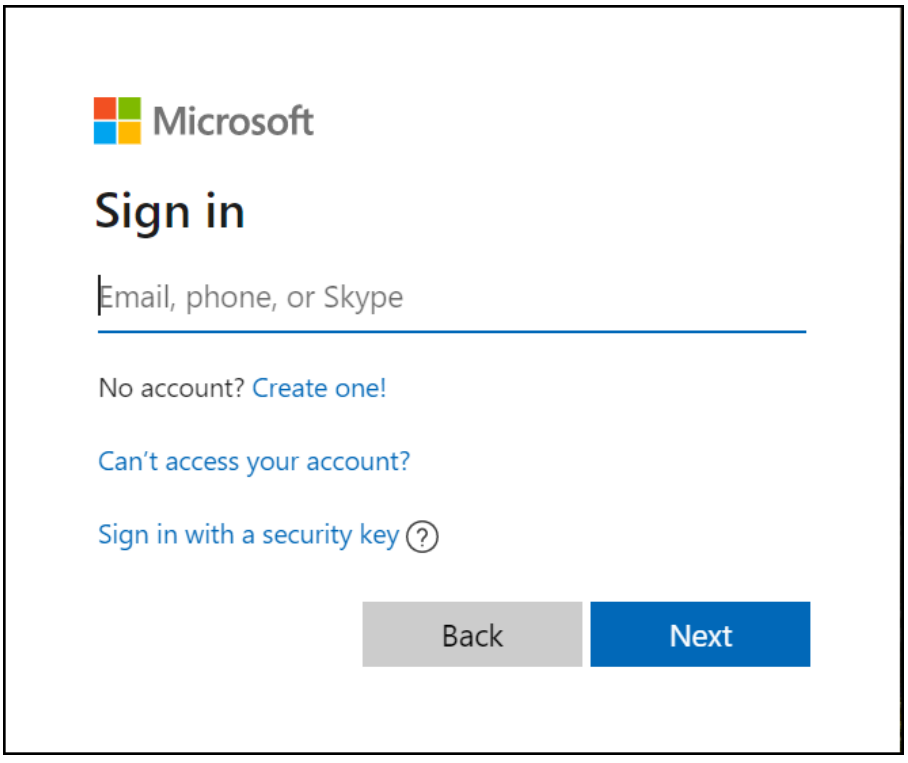
2.6.1.1 Logging in to the Azure Environment

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

➤ To login to the Azure environment:

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

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



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

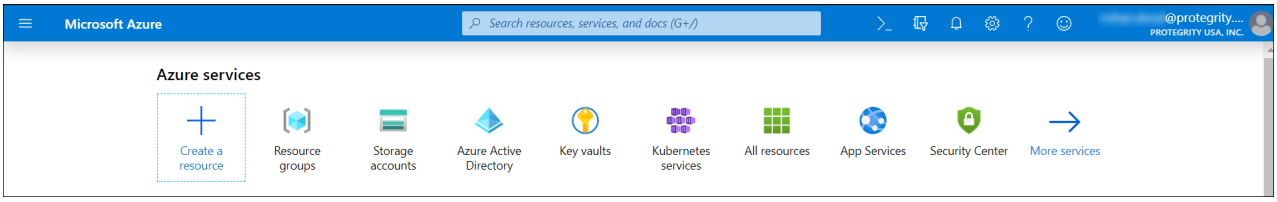
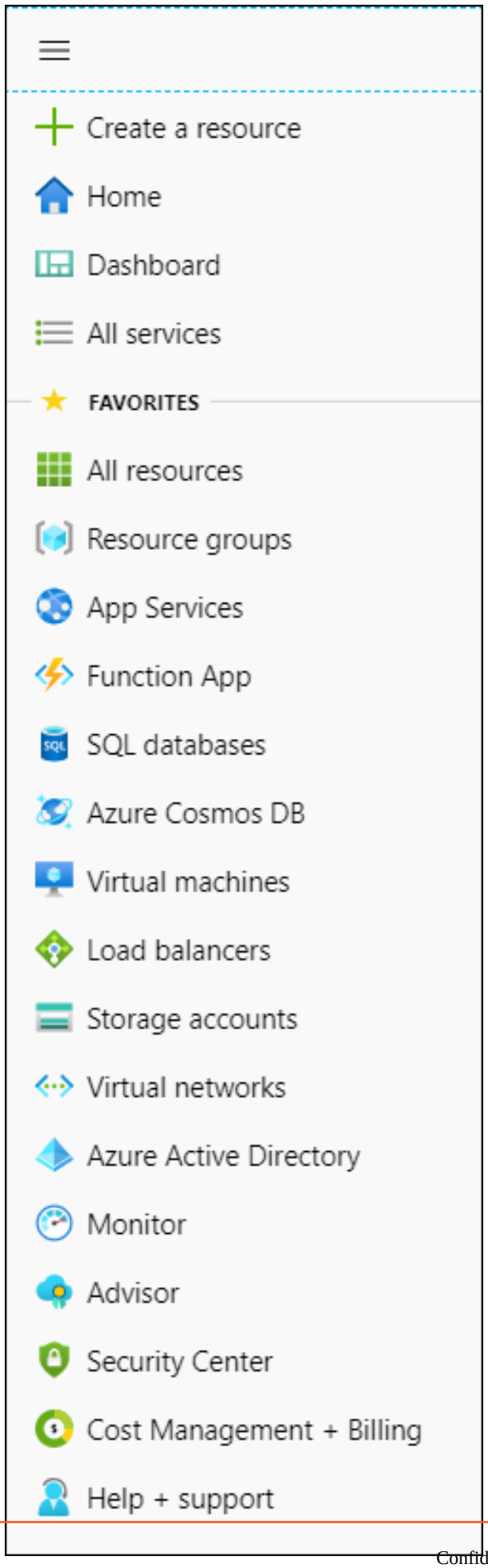


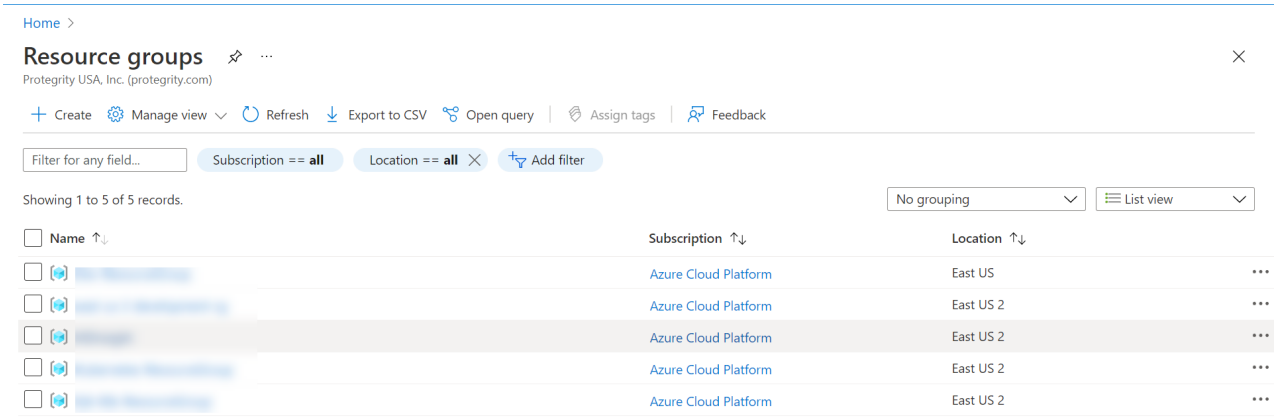
Figure 2-1: Azure Portal Home Screen

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

   The **Portal** menu appears.


Create a resource
Home
Dashboard
All services
FAVORITES
All resources
Resource groups
App Services
Function App
SQL databases
Azure Cosmos DB
Virtual machines
Load balancers
Storage accounts
Virtual networks
Azure Active Directory
Monitor
Advisor
Security Center
Cost Management + Billing
Help + support

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



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

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

2.6.1.2 Registering an Application

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

To register an application:

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

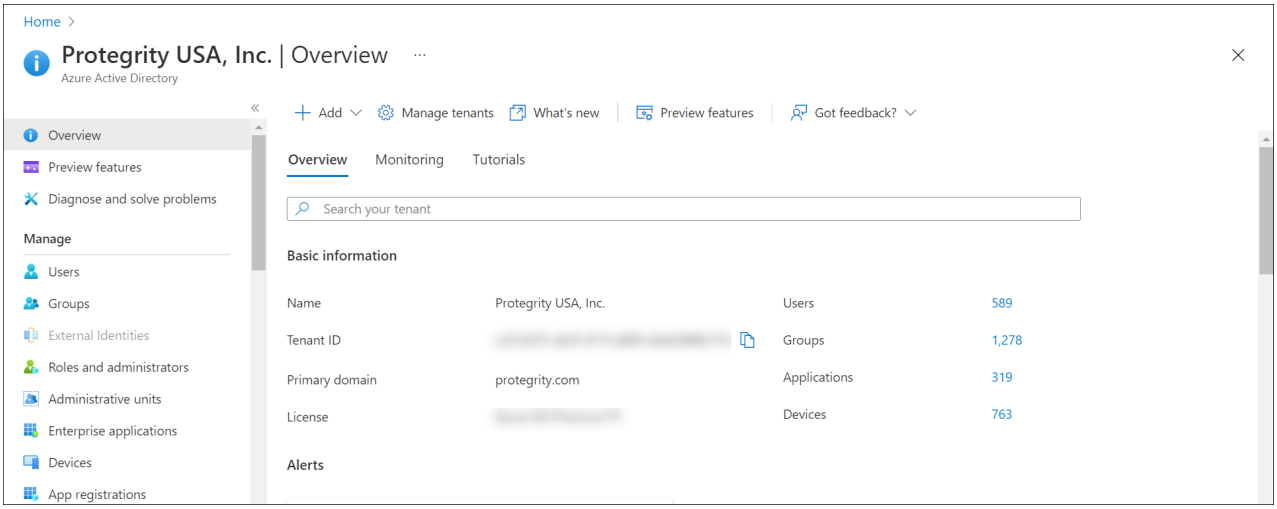


Figure 2-2: Azure Active Directory Overview Screen

4. Navigate to **App registrations**.

   The **App Registrations** screen appears.

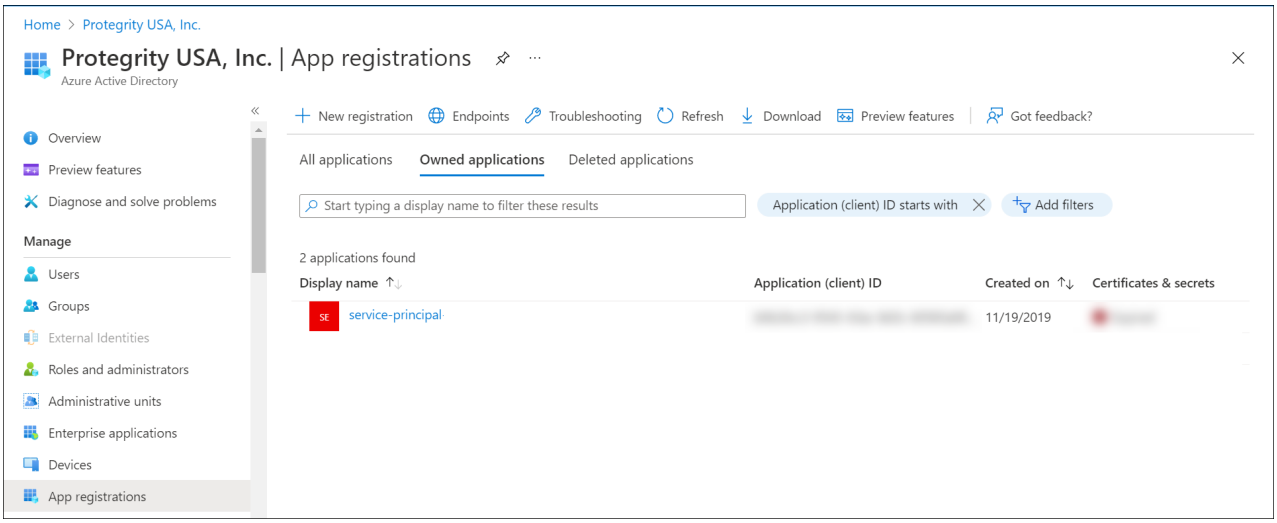


Figure 2-3: App Registrations Screen

5. Click **New registration**.

   The **Register an application** screen appears.

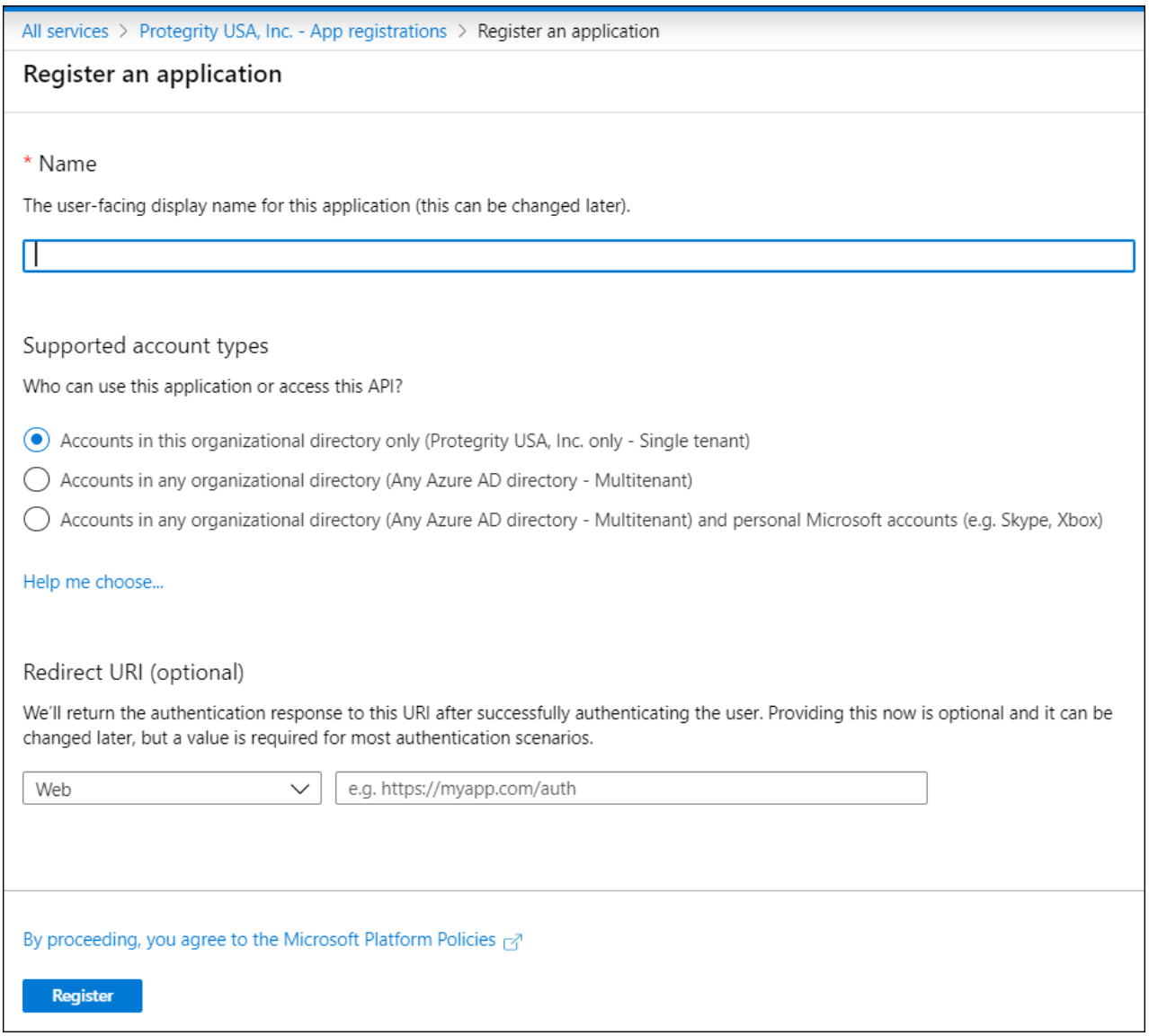
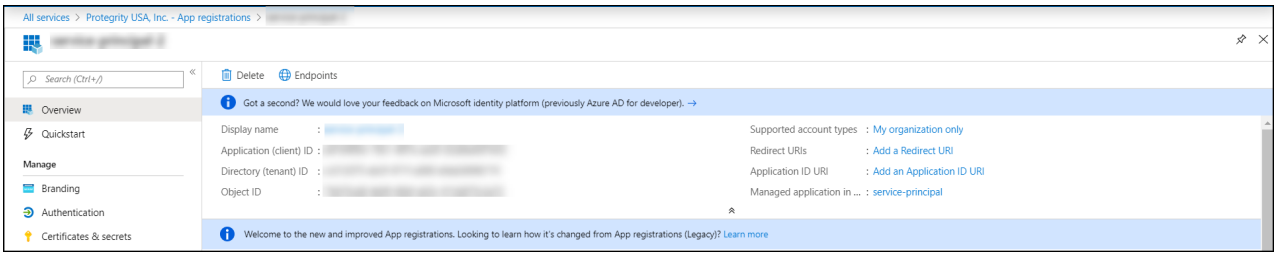


Figure 2-4: Register an Application Screen

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



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

> For more information about the *credentials.json* file, refer to the section *Credentials.json*.

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

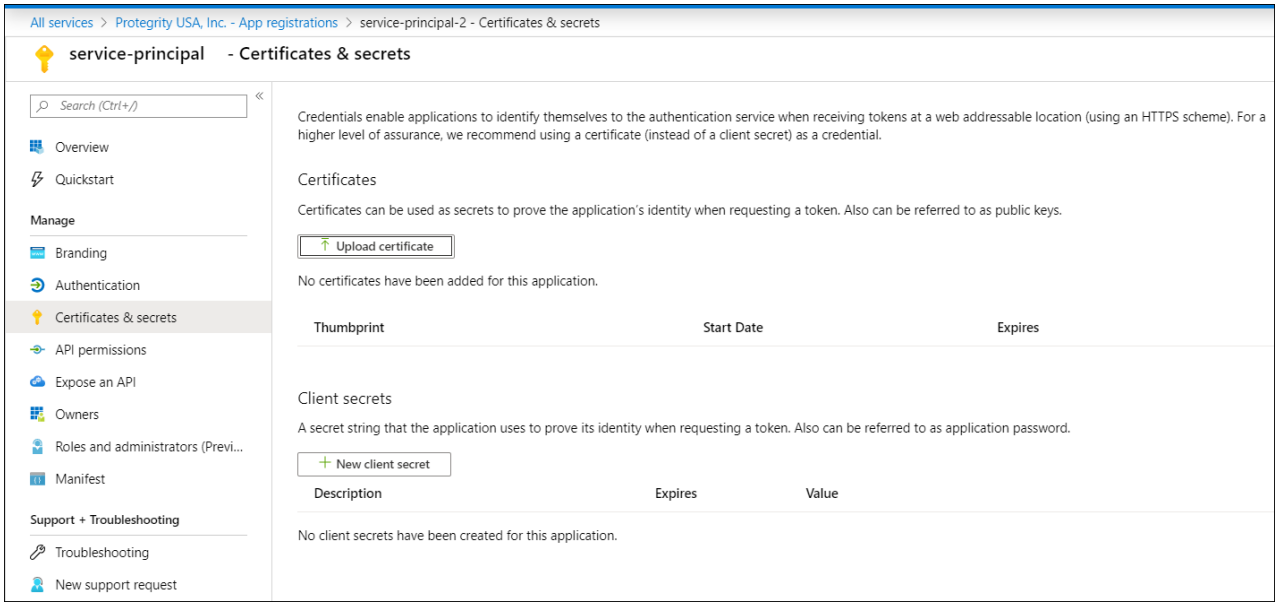


Figure 2-5: Certificates & Secrets Screen

   b. Click **New client secret**.
      The **Add a client secret** screen appears.

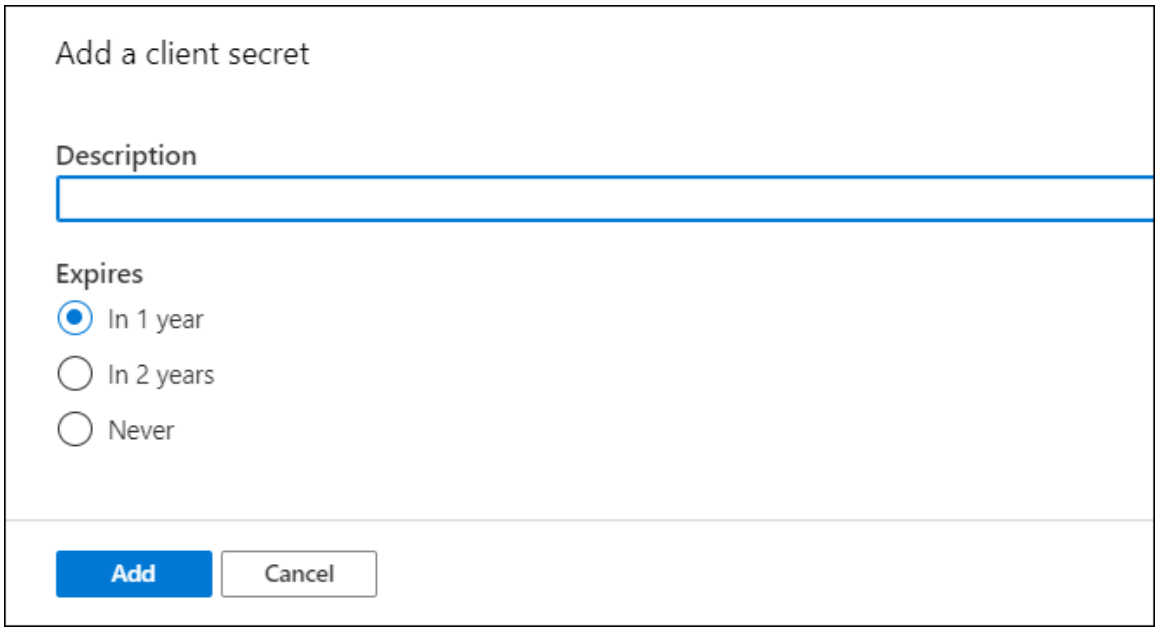
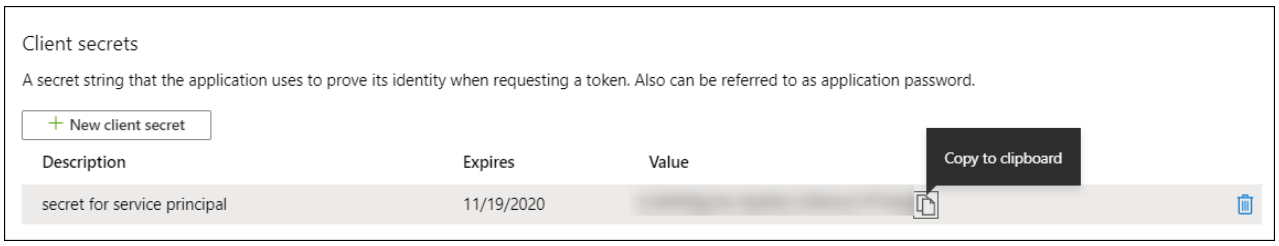


Figure 2-6: Add a Client Secret Scren

   c. In the **Description** field, specify a description for the client secret.
   d. In the **Expires** option, retain the default value, **In 1 year**.
   e. Click **Add**.
      The client secret for the registered application appears in the **Client secrets** section.



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

> **Important:** Copy the client secret value to a secure location. You need to specify this value in the *credentials.json* file for the corresponding service principal.
>
> For more information about the *credentials.json* file, refer to the section *Credentials.json*.

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

2.6.1.3 Creating an Azure Key

This section describes how to create an Azure key, which is used to encrypt the policy package. The policy package is encrypted before it is uploaded to the Azure Storage Container or the Azure File Share.

➤ To create an Azure Key:

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

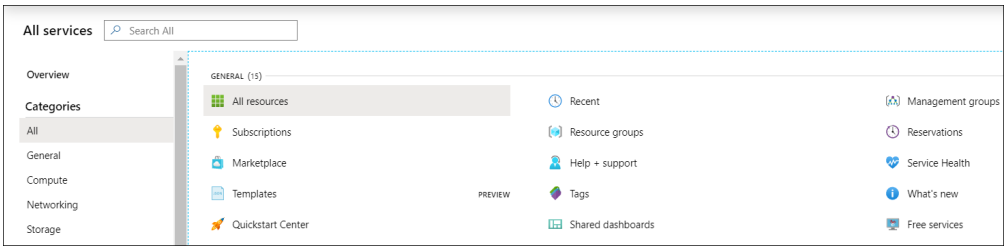


Figure 2-7: All Services Screen

4. Select **Key vaults** under the **SECURITY** section.
   The **Key vaults** screen appears.

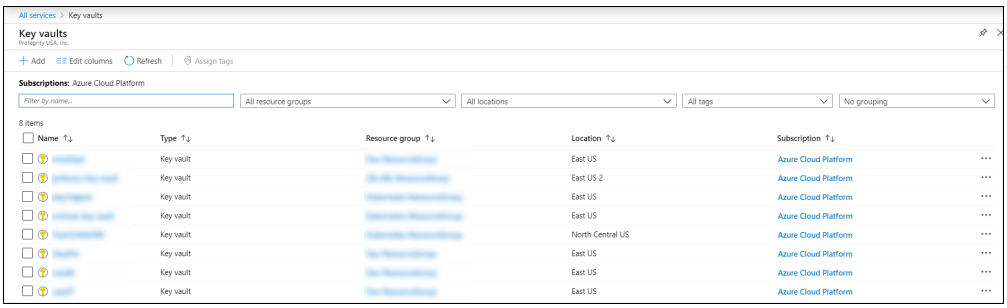
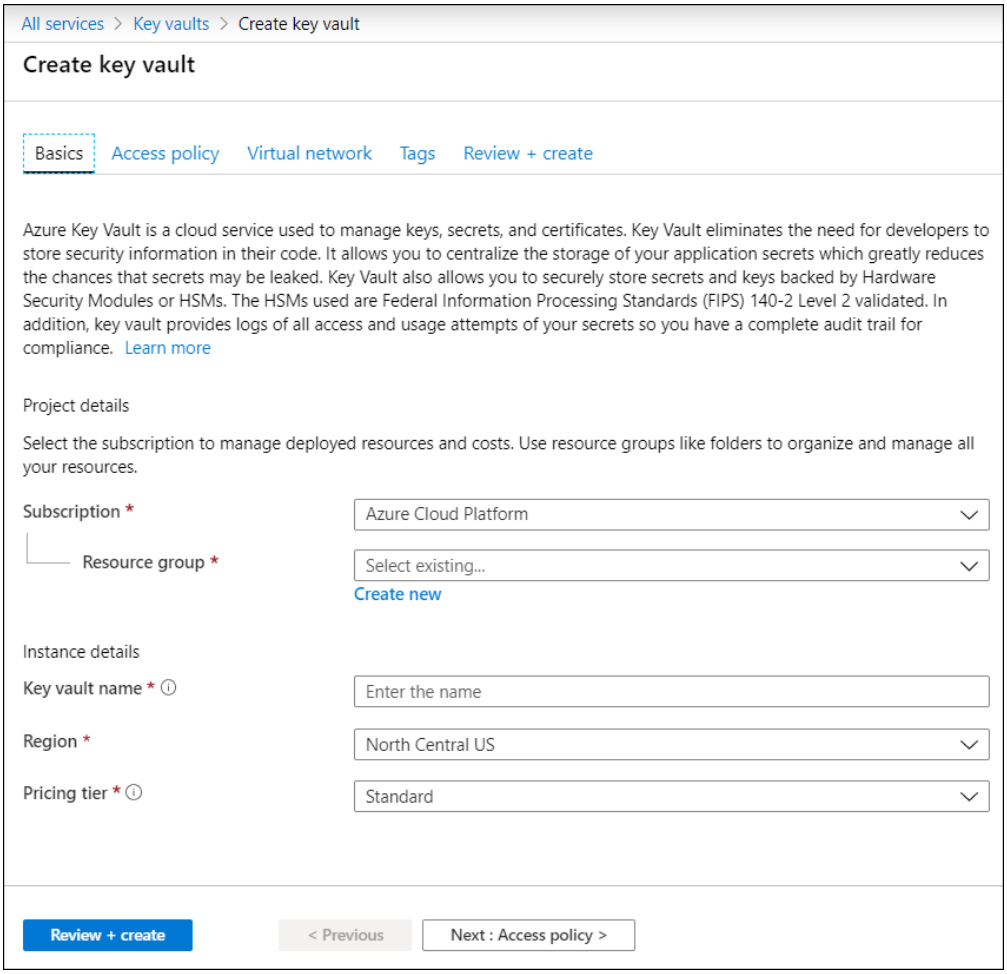


Figure 2-8: Key Vaults Screen

5. Click **Add**.

   The **Create key vault** screen appears.



6. On the **Basics** tab, enter the following details:

Field	Description
Subscription	By default, *Azure Cloud Platform* is selected.
Resource group	Select the required resource group.
Key vault name	Enter a name for your key vault.
Region	Select a region where the key vault will be created.
Pricing tier	Select a pricing tier. For example, *Standard* or *Premium*.

7. Click **Next** to navigate to the **Access policy** tab.

   The **Access policy** tab appears.

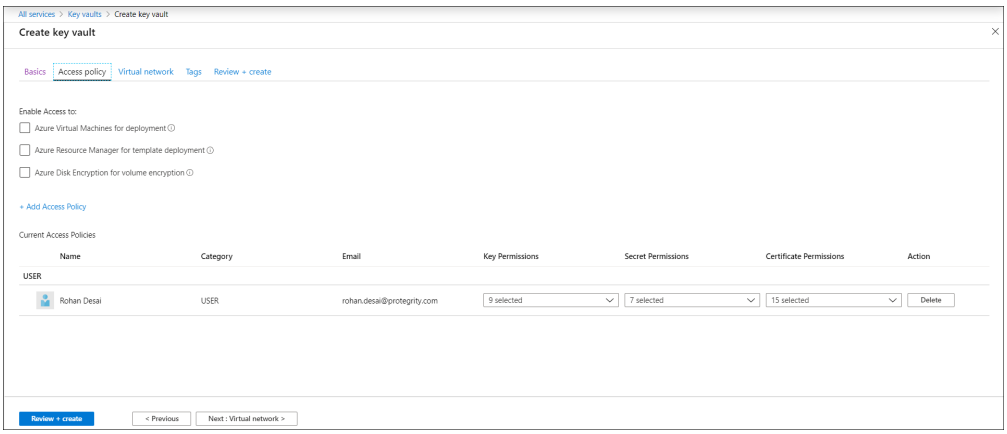


Figure 2-9: Access Policy Tab

8. Perform the following steps to provide the required permissions to the service principals that have been created.
   a. Click **Add Access Policy**.
      The **Add access policy** screen appears.

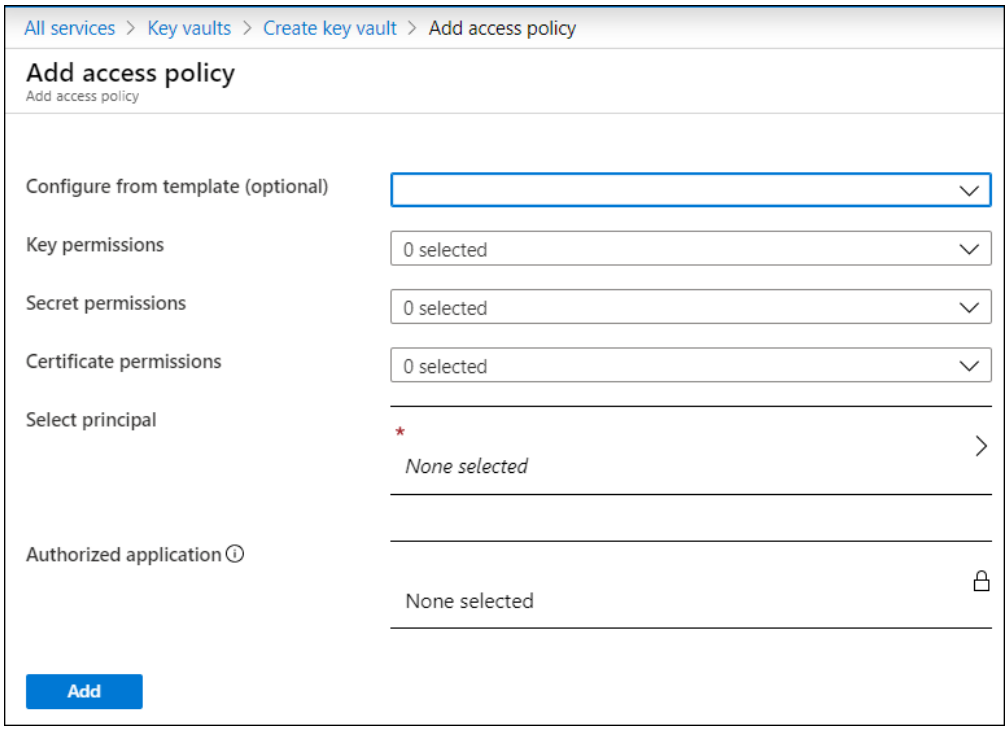


Figure 2-10: Add Access Policy Screen

   b. In the **Key permissions** list, select the **Get** check box under the **Key Management Operations** section.
      You need to assign the *Get* permission to the service principal 1 so that it can get the key from the Azure Key Vault for encrypting the policy package.

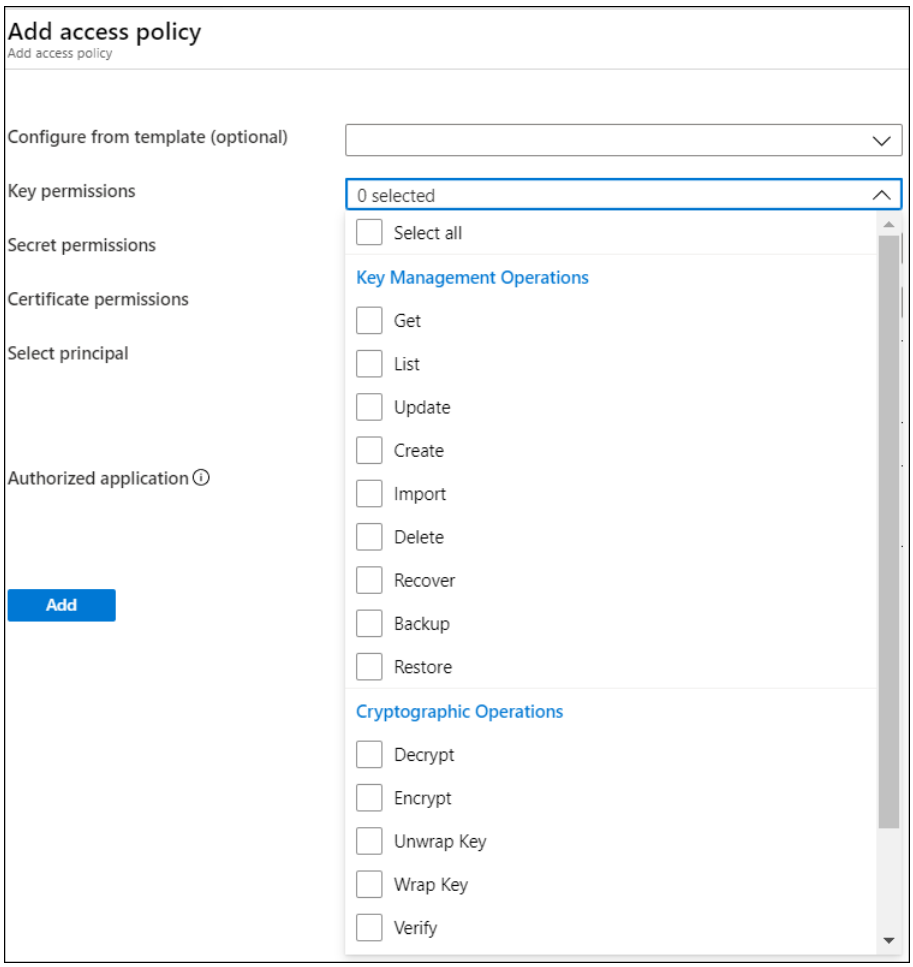


Figure 2-11: Key Permissions List

c. Click the **Select principal** field.
The **Principal** dialog box appears.

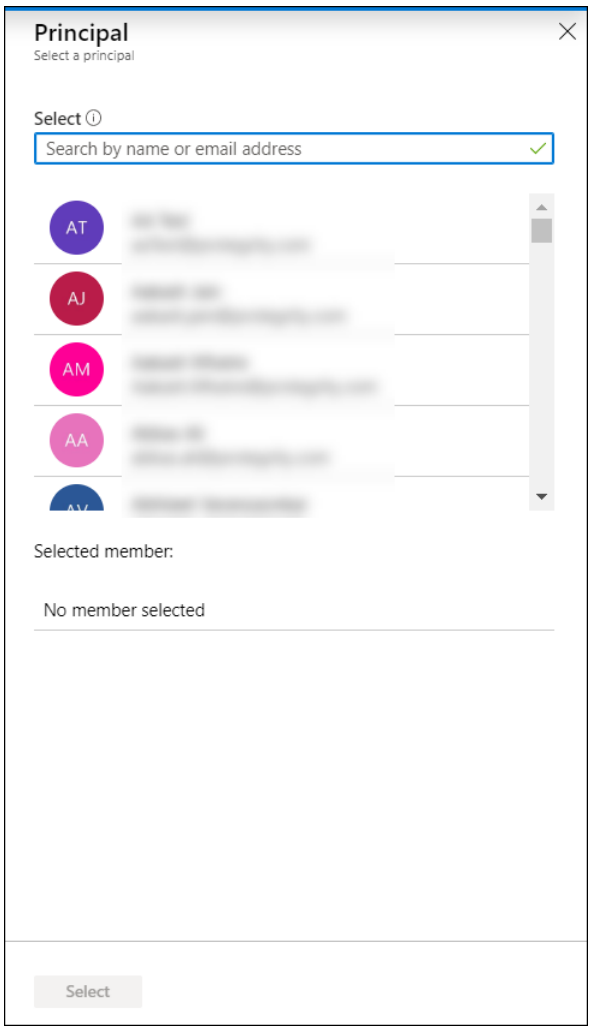


Figure 2-12: Principal Dialog Box

d. In the **Select** field, search for and then select the required service principal.

   For example, search for and select the service principal 1 that you have created in the section *Registering an Application*.

e. Click **Select** to add the service principal.

f. Click **Add** to assign the required permission to the selected service principal.

g. Repeat substeps a to d to assign the *Decrypt* permission to the service principal 2. In step b, you need to select the **Decrypt** check box under the **Cryptographic Operations** section, while in step c, you need to select the service principal 2.

   The *Decrypt* permission is required to decrypt the key that has been used to encrypt the policy package.

h. In the **Select** field, search for and then select the required service principal.

   For example, search for and select the service principal that you have created in the section *Registering an Application*.

9. Click **Next** to navigate to the **Virtual network** tab.

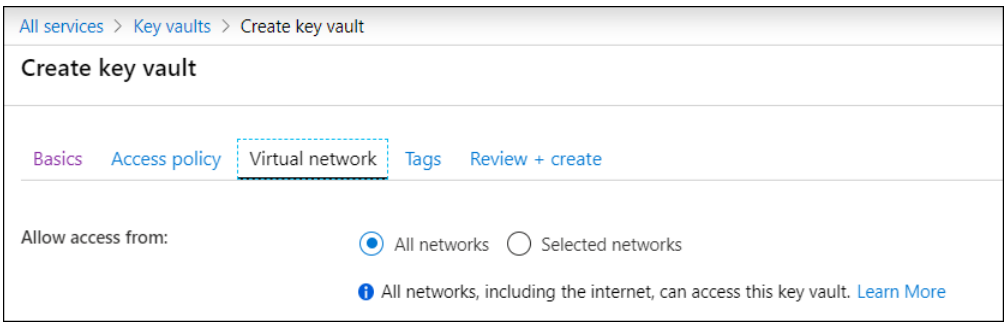


Figure 2-13: Virtual Network Tab

10. In the **Allow access from** field, select **All networks**.
    This option is required if the ESA is on-premise.
11. Click **Next** to navigate to the **Tags** tab.

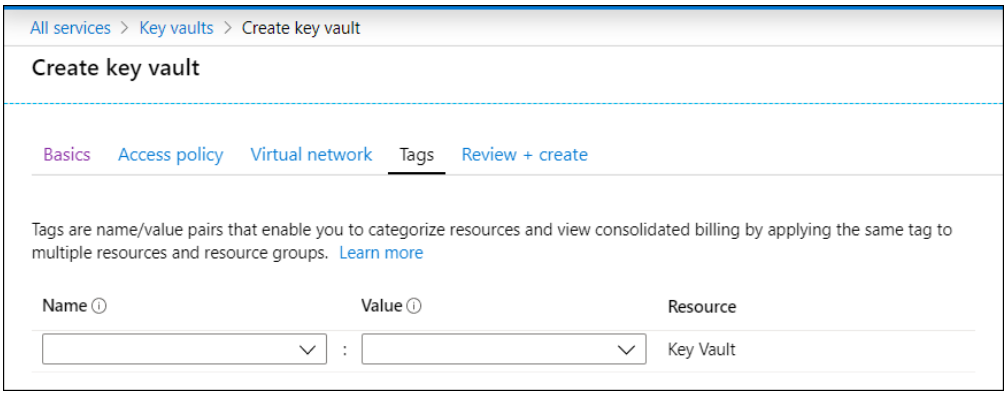


Figure 2-14: Tags Tab

12. Add tags to the key vault, if required.
    Tags are name-value pairs for categorizing the key vault resource.
13. Click **Review + create** to validate the key vault configuration.
    Azure validates the information that you have entered.

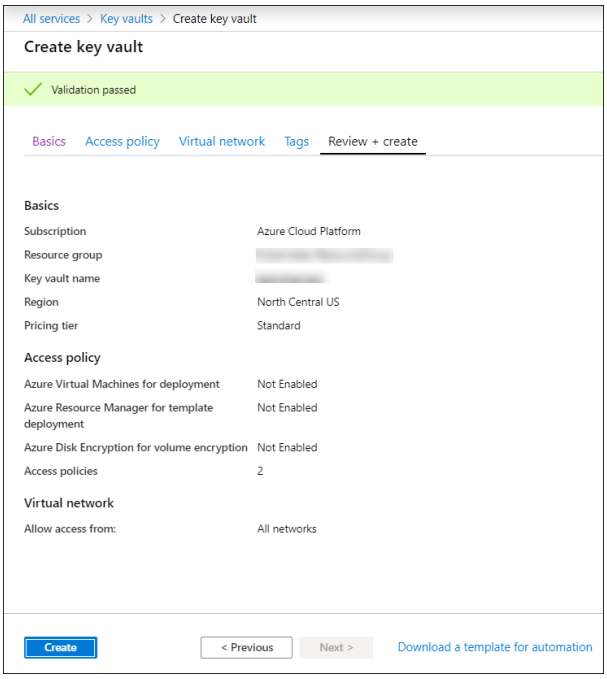
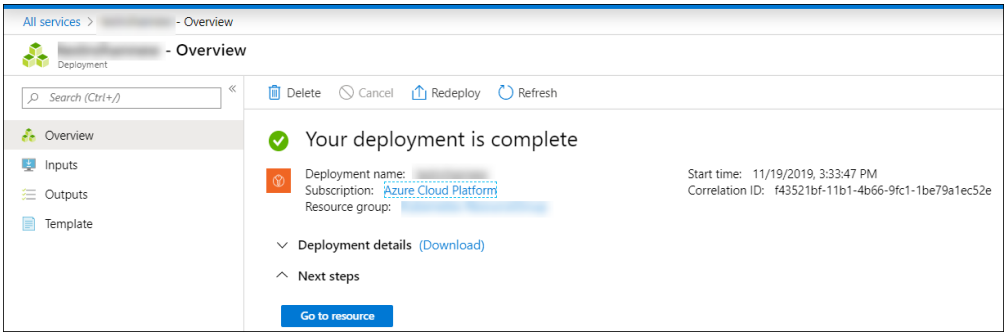


Figure 2-15: Review + create Tab

14. Click **Create** to create the required key vault.
    The key vault is created and the following screen appears.



15. Click **Go to resource**.
    The **Overview** screen appears displaying information about the key vault that you have created.

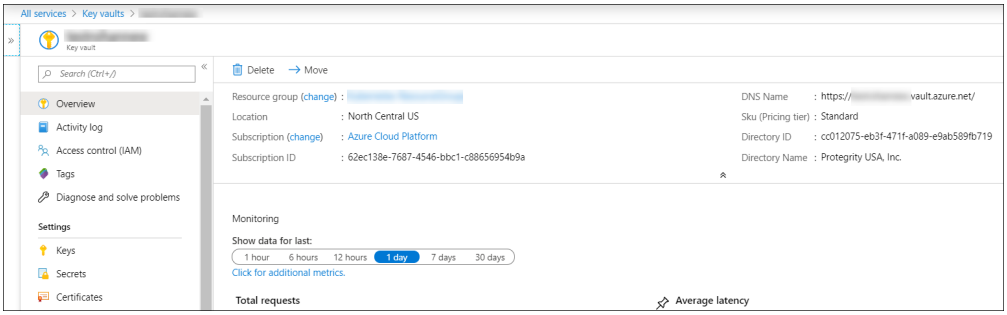


Figure 2-16: Key Vaults Overview Screen

16. On the left pane, under the **Settings** section, click **Keys**.
    The **Keys** screen appears.

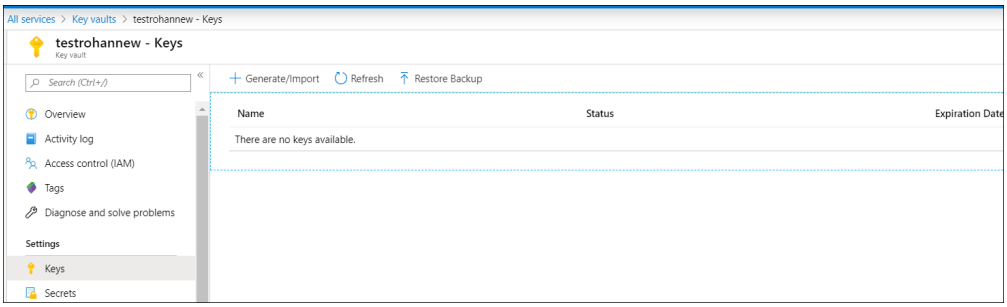


Figure 2-17: Key Screen

17. Click **Generate/Import** to create a key.
    The **Create a key screen** appears.

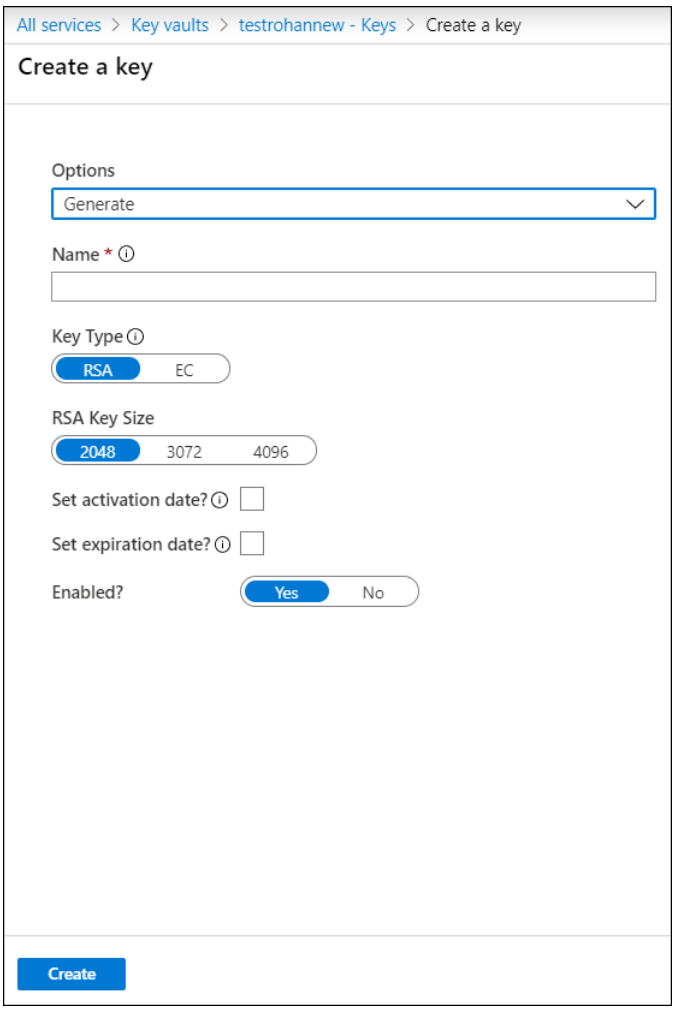
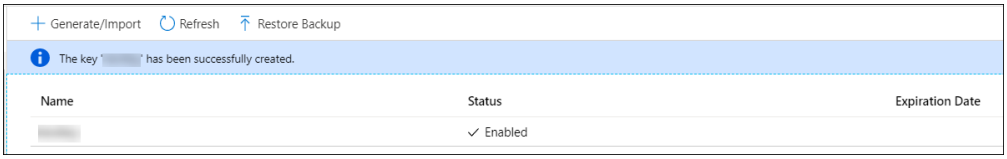


Figure 2-18: Create a Key Screen

18. Enter the following details for the key.

Field	Description
Options	Select **Generate** to generate the key.
Name	Enter the name for your Azure key.
Key Type	Specify the key type. Select *RSA*.
RSA Key Size	Specify the key size for RSA. Select the value as *3072*.
Enabled	Select *Yes* to enable the key.

19. Click **Create**.
    The key is created and the following message appears.



20. Click the key name.
    The **Versions** screen for the key appears. It lists all the versions for the key.

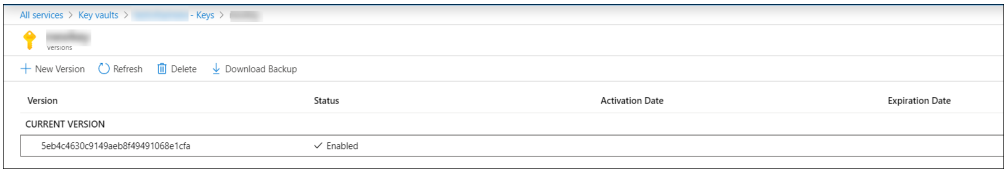


Figure 2-19: Versions Screen

21. Click the specific key version.

    The **Key Version** screen appears. This screen displays the properties and settings for the specific key version.

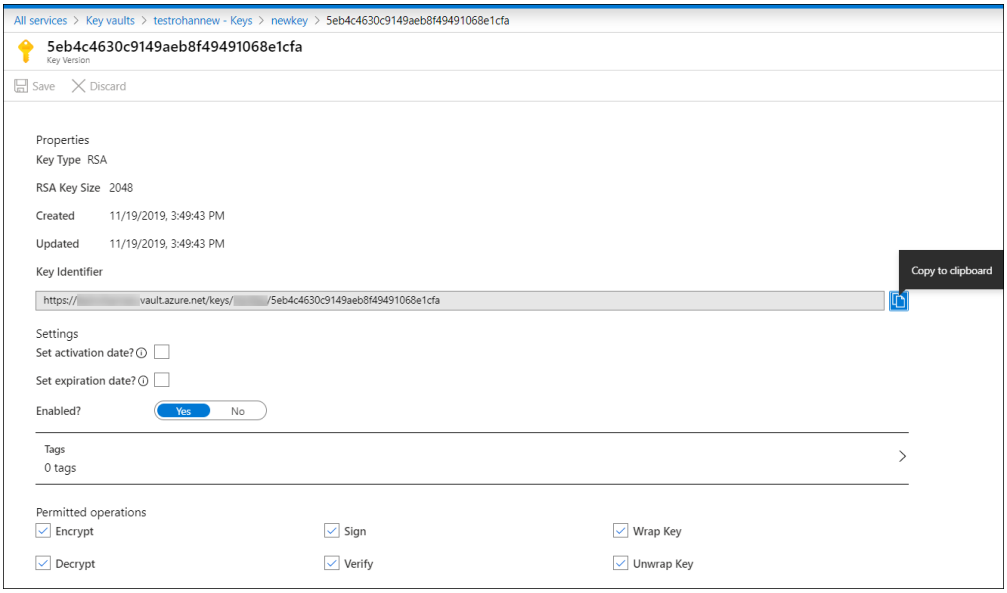


Figure 2-20: Key Version Screen

22. Click the icon next to the **Key Identifier** field to copy the field value to the clipboard.

    A message appears informing that the identifier has been copied.

    The following is an example of an Azure key identifier:

    https://<Key_Vault_Name>.vault.azure.net/keys/<Key_Name>/0a5e4ea190a24707983ffd0efcc03c8a

    You need to paste this identifier as the value of the *azureKeyId* parameter in the *encryption-config.yml* file in the *get-esa-policy* directory, where you have extracted the Helm charts for deploying the Get ESA Policy application.

    For more information about the extracted Helm charts, refer to the *step 10* of the section *Initializing the Linux Instance*.

    For more information about updating the value of the *azureKeyId* parameter in the *encryption-config.yaml* file for encrypting the policy on a container with an Azure Policy and RBAC, refer to *step 6* in section *Deploying the Get ESA Policy Container on the Kubernetes Cluster*.

    For more information about updating the value of the *azureKeyId* parameter in the *encryption-config.yaml* file for encrypting the policy on a container without an Azure Policy and RBAC, refer to *step 9* in section *Deploying the Get ESA Policy Container on the Kubernetes Cluster*.

    You also need to paste this identifier as the value of the *azureKeyId* parameter in the *decryption-config.yaml* file in the *spring-apjava* directory, where you have extracted the Helm charts to deploy the Sample application.

    For more information about the extracted Helm charts, refer to the *step 10* of the section *Initializing the Linux Instance*.

For more information about updating the value of the *azureKeyId* parameter in the *decryption-config.yaml* file for decrypting the policy on a container with an Azure Policy and RBAC, refer to *step 3* in section *Deploying the Sample Application Container on the Kubernetes Cluster*.

For more information about updating the value of the *azureKeyId* parameter in the *decryption-config.yaml* file for decrypting the policy on a container without an Azure Policy and RBAC, refer to *step 6* in section *Deploying the Sample Application Container on the Kubernetes Cluster*.

23. In the **Permitted operations** section, select the **Decrypt** check box and clear all the remaining check boxes.

Figure 2-21: Permitted Operations Section

This ensures that only the **Decrypt** operation is permitted for the Azure key.

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

2.6.1.4 Creating an Azure Storage Container

This section describes how to create an Azure Storage Container.

Important: This procedure is optional, and is required only if you want to use the Azure Storage Container for storing the policy package, instead of using the Azure File Share.

To create an Azure Storage Container:

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

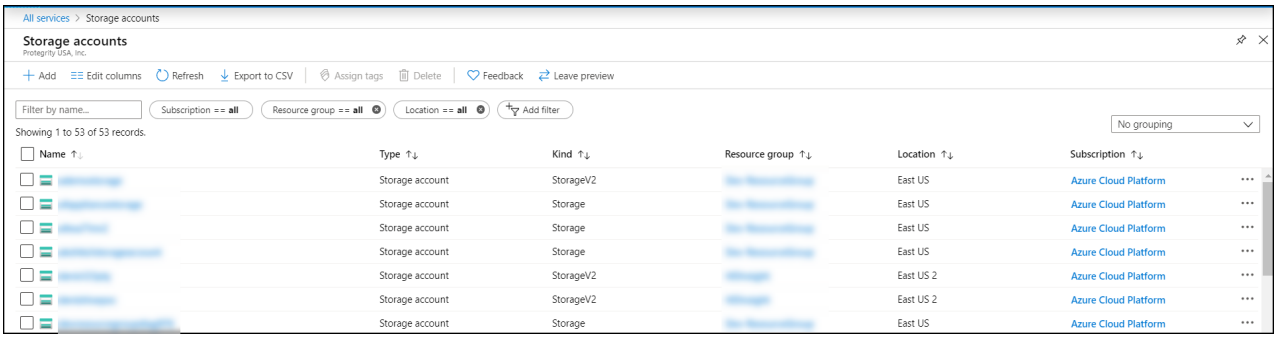


Figure 2-22: Storage Accounts Screen

4. Click **Add**.
   The **Create storage account** screen appears. By default, the **Basics** tab appears.

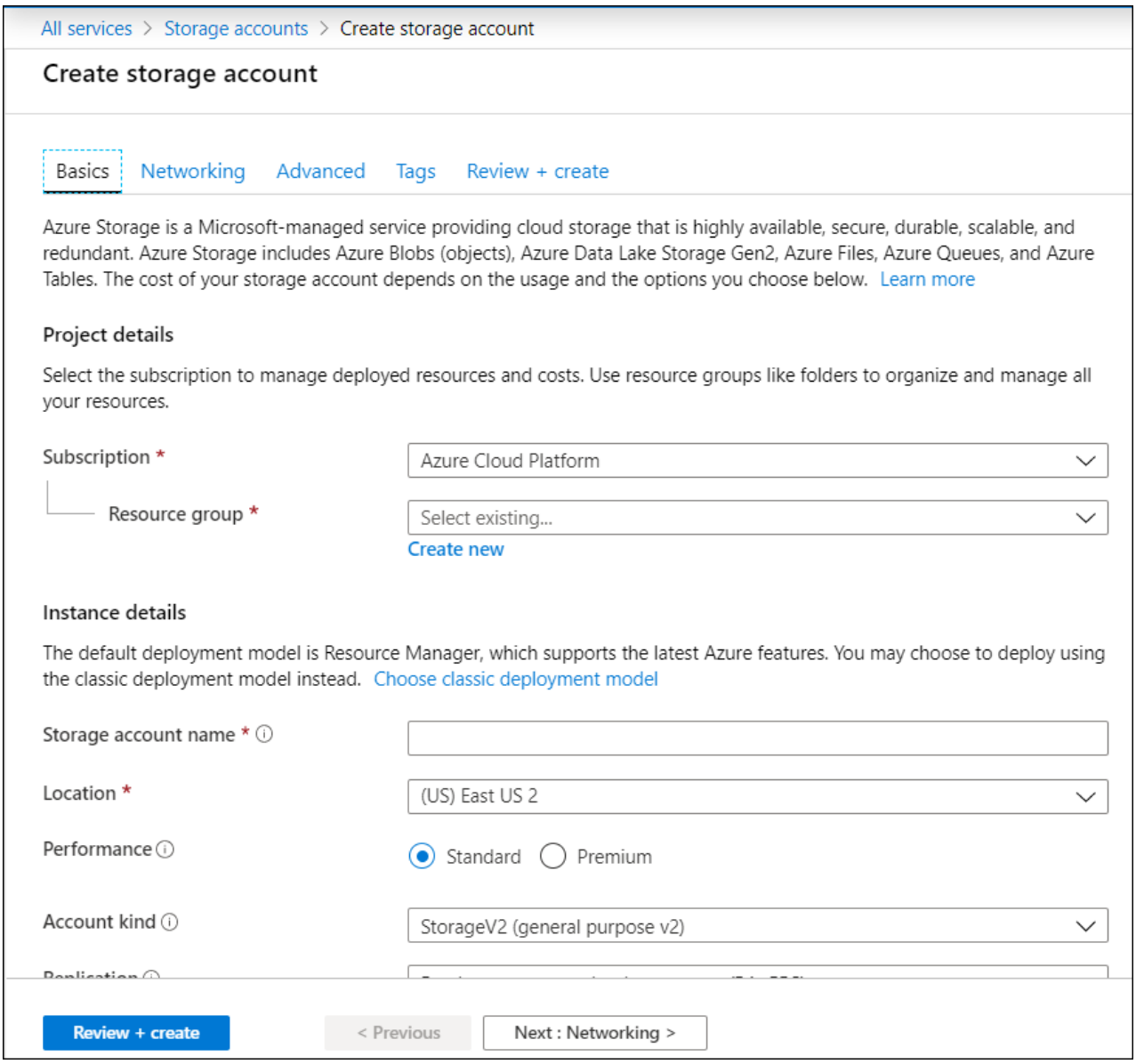


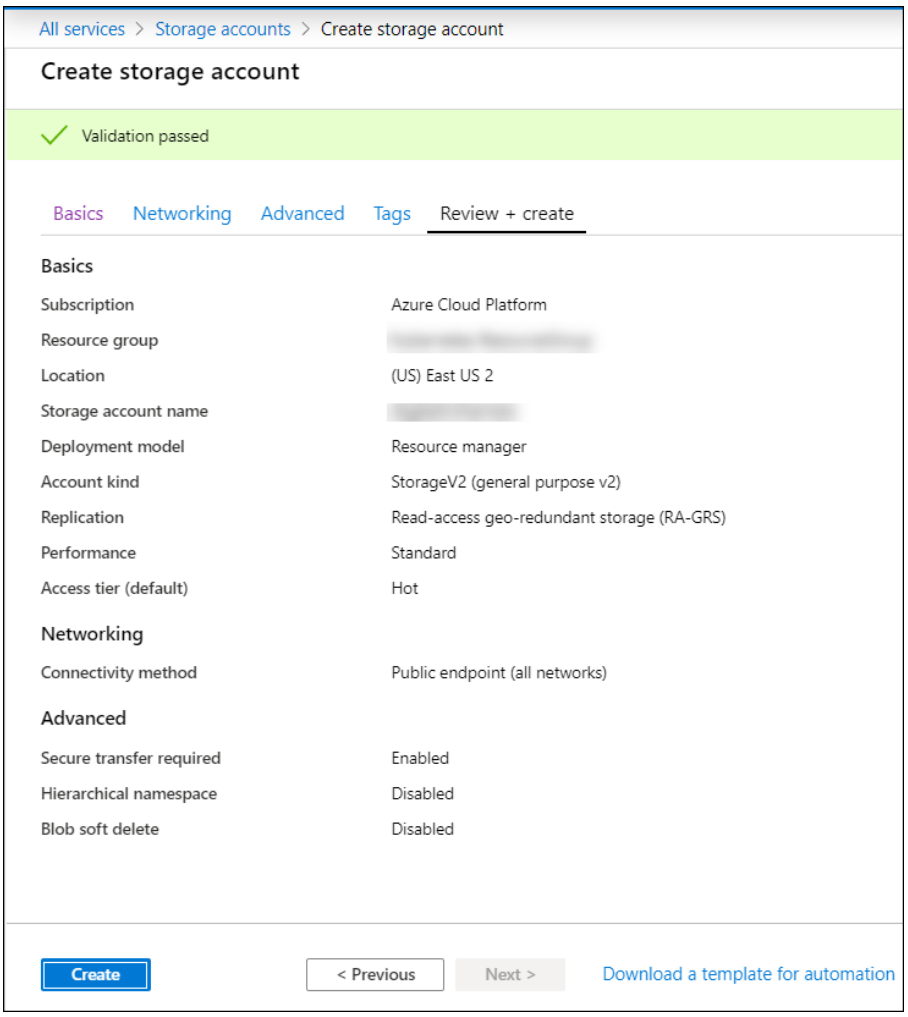
Figure 2-23: Create Storage Account Screen

5. In the **Basics** tab, enter the following mandatory details.

Field	Description
Subscription	By default, this value is set to *Azure Cloud Platform*.
Resource group	Select the required resource group.
Location	Select a region where the Azure storage account will be created. For example, select *(US) US East 2*.

> **Note:** Retain the default values for all the remaining tabs on the **Create storage account screen**.

6. Click **Review + create** to validate the entered information.
   Azure validates the information that you have entered.



7. Click **Create** to create the storage account.
   The storage account is created and the following screen appears.

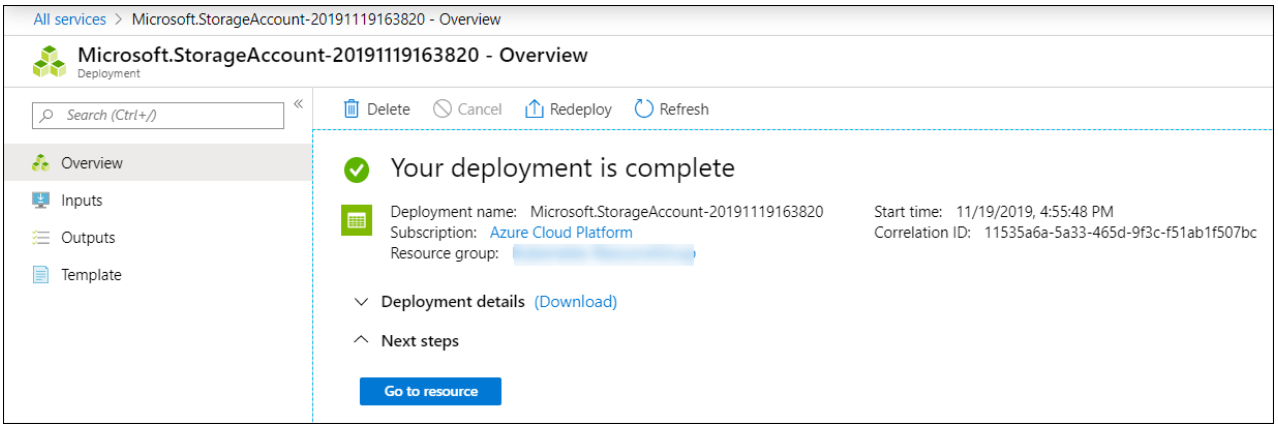


Figure 2-24: Storage Account Overview Screen

8. Click **Go to resource**.
   The **Overview** screen appears, displaying the details of your storage account.

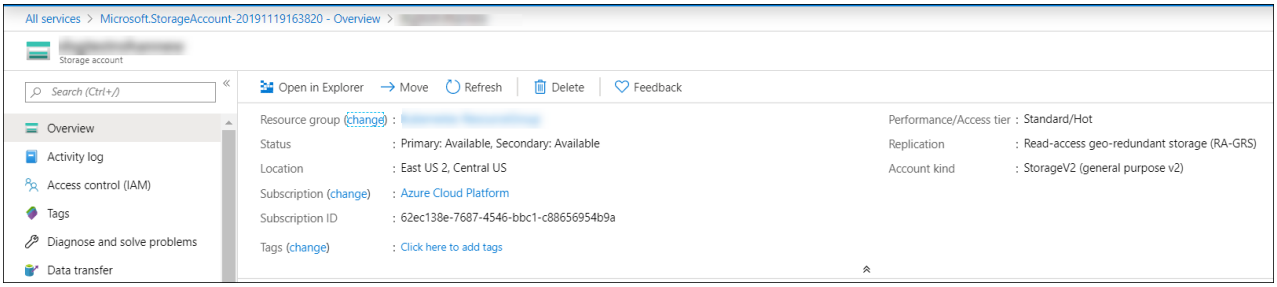
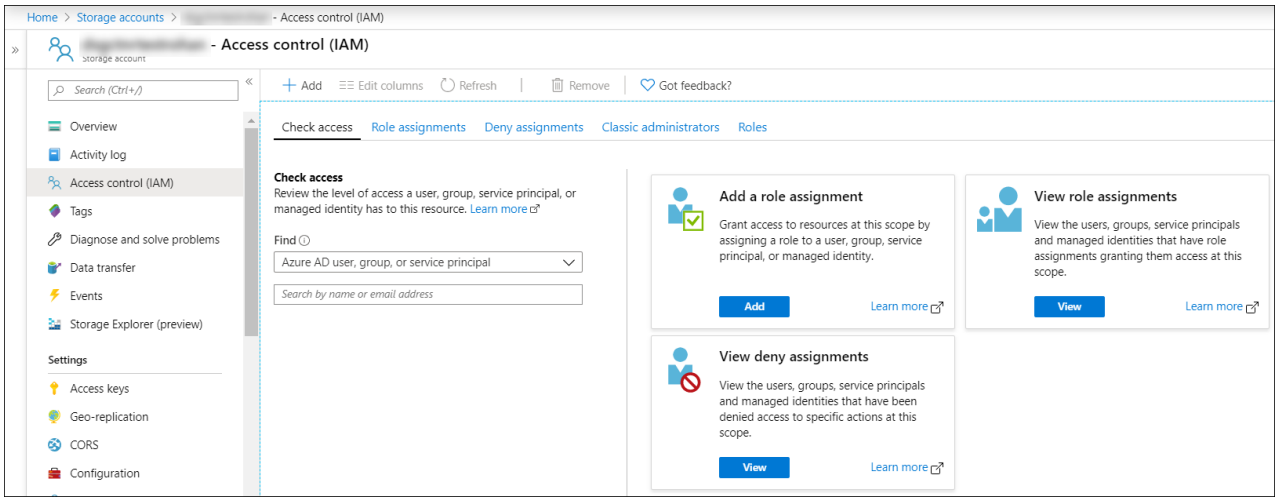
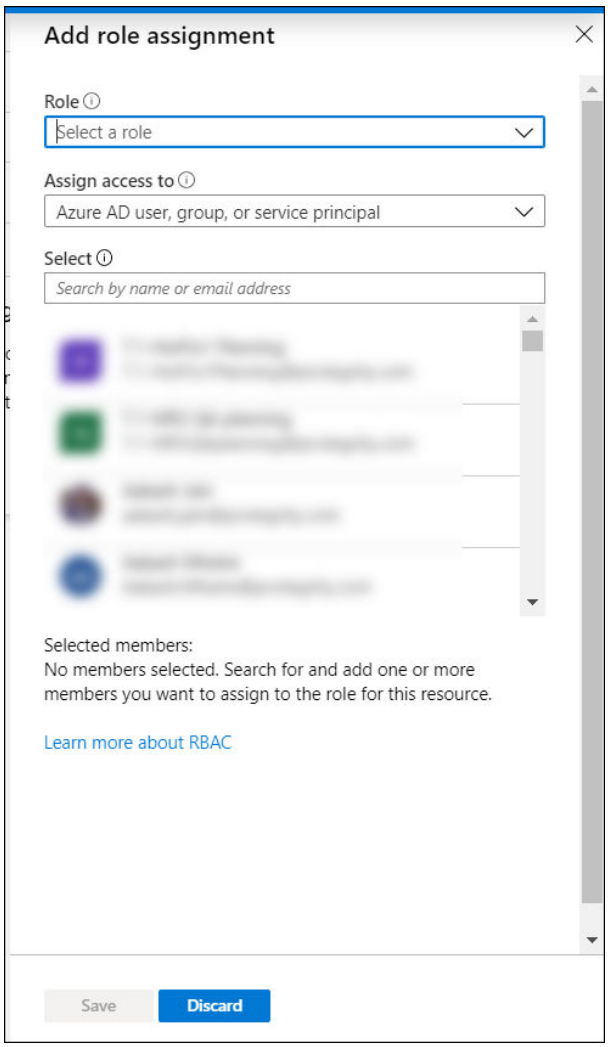


Figure 2-25: Storage Account Overview Screen

9. Perform the following steps to assign a role to the service principals for accessing the storage account.
   a. On the left pane, click **Access control (IAM)**.
      The **Access control (IAM)** screen appears.



   b. Click **Add**.
   c. Select **Add role assignment**.
      The **Add role assignment** dialog box appears.



d. In the **Role** list, select the required role.

   For example, for service principal 1, select the role as *Storage Blob Data Contributor*.

e. In the **Assign access to** list, retain the default value *Azure AD user, group, or service principal*.

f. In the **Select** list, select service principal 1.

   The service principal has the same name as that of the application that you have registered in the section *Registering an Application*.

g. Click **Save** to assign the role to the service principal.

h. Repeat *step b* to *step g* to assign the *Storage Blob Data Reader* role to service principal 2.

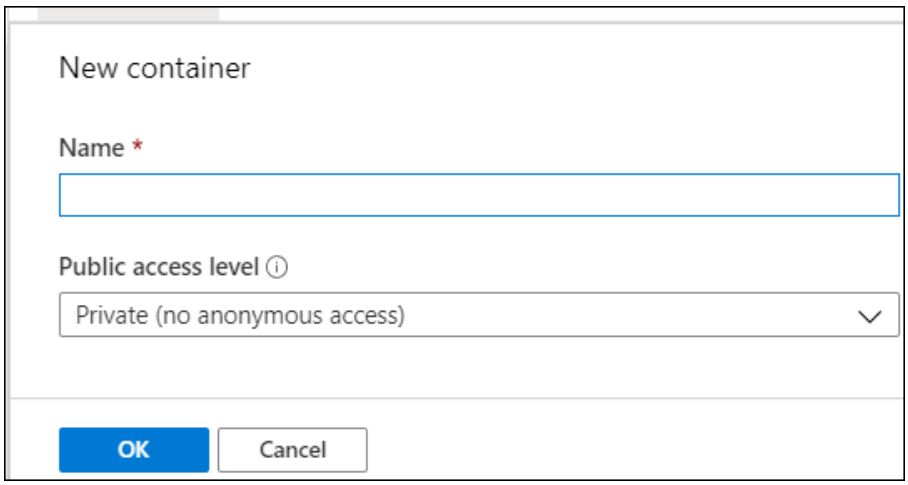
10. Perform the following steps to create a storage container.

    a. On the left pane, click **Blob service** > **Containers**.

       The **Containers** screen appears.

    b. Click **Container**.

       The **New Container** screen appears.



c. Specify a name for the container in the **Name** field.

d. Retain the default value for the **Public access level** list.
The default value is set to *Private (no anonymous access)*.

e. Click **OK** to create the container.

2.6.1.5 Creating an Azure File Share

This section describes how to create an Azure File Share.

➤ To create an Azure File Share:

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

3. Click **Storage accounts** .
The **Storage accounts** screen appears.

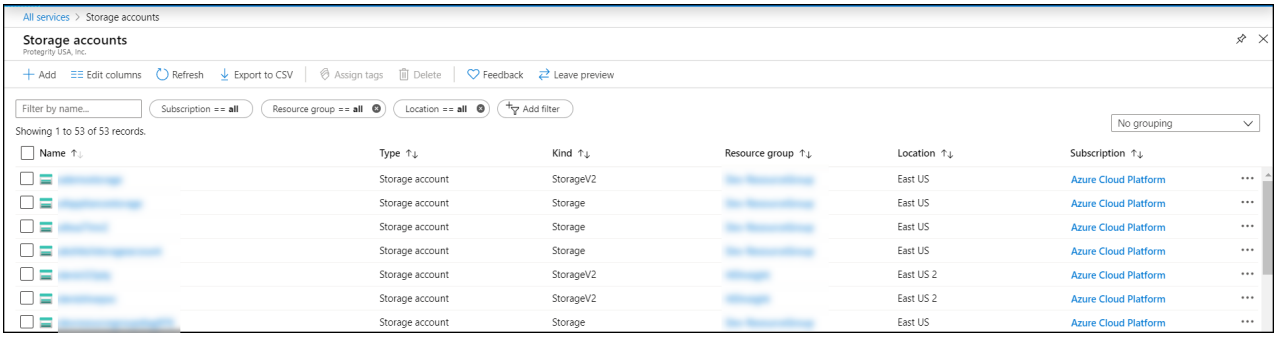


Figure 2-26: Storage Accounts Screen

4. Click **Add**.
The **Create storage account** screen appears. By default, the **Basics** tab appears.

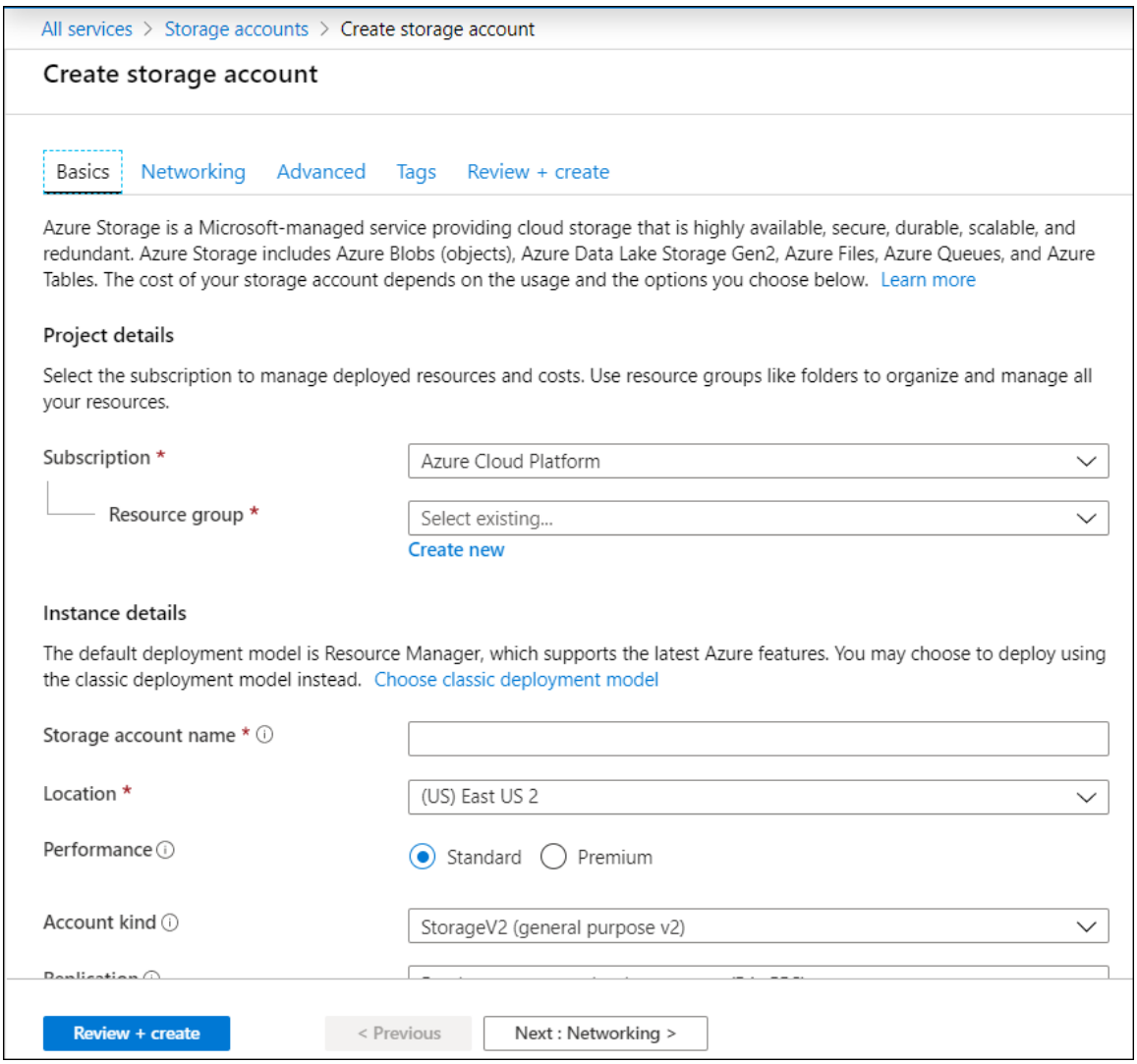


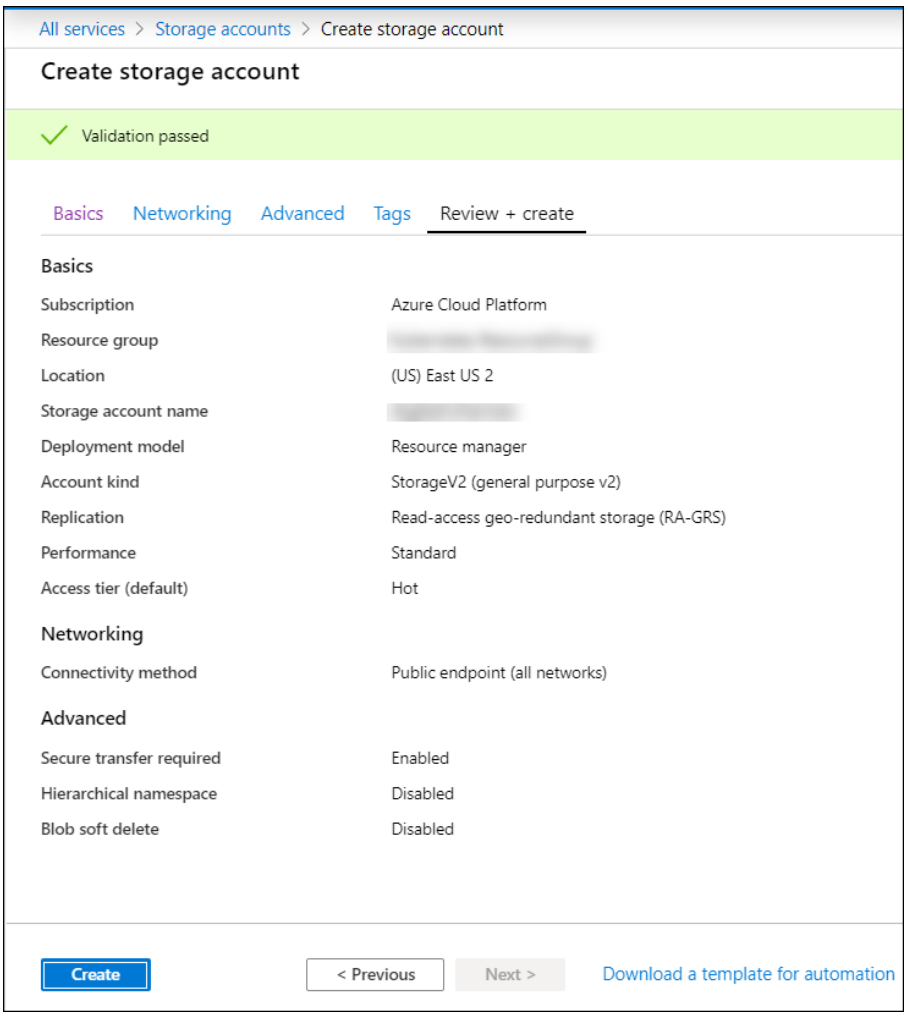
Figure 2-27: Create Storage Account Screen

5. In the **Basics** tab, enter the following mandatory details.

Field	Description
Subscription	By default, this value is set to Azure Cloud Platform.
Resource group	Select the required resource group.
Location	Select a region where the Azure storage account will be created. For example, select *(US) US East 2*.

> **Note:** Retain the default values for all the remaining tabs on the **Create storage account screen**.

6. Click **Review + create** to validated the entered information.
   Azure validates the information that you have entered.



7. Click **Create** to create the storage account.
   The storage account is created and the following screen appears.

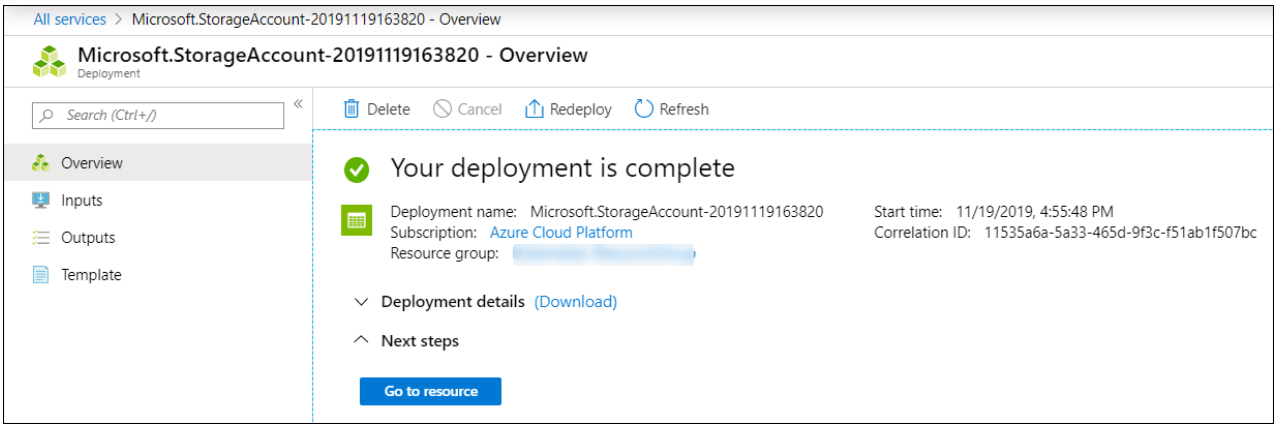


Figure 2-28: Storage Account Overview Screen

8. Click **Go to resource**.
   The **Overview** screen appears, displaying the details of your storage account.

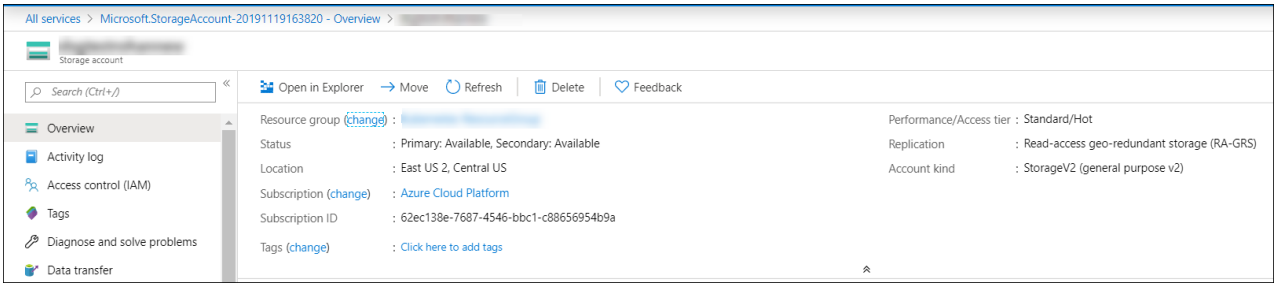


Figure 2-29: Storage Account Overview Screen

9. Navigate to **Settings** > **Access keys**.

   The **Access Keys** screen appears.

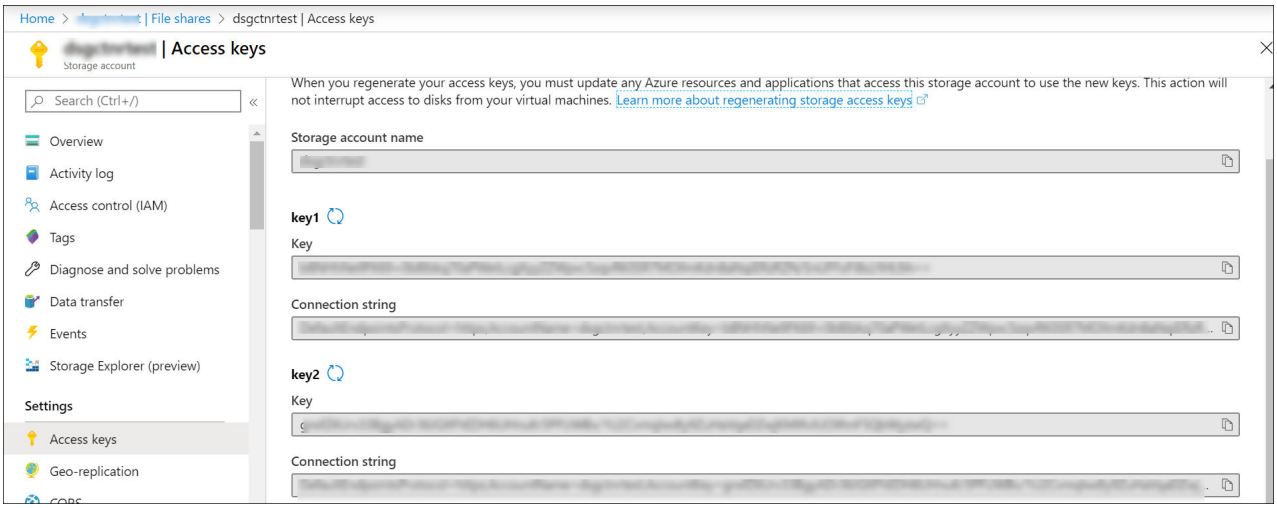


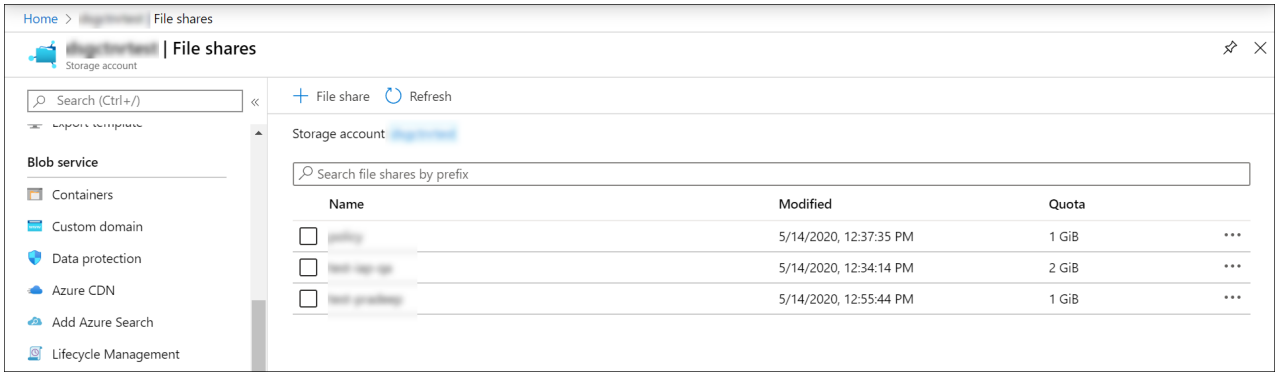
Figure 2-30: Access Keys Screen

Note the storage access keys, key 1, and key 2. These keys are created automatically when you create an Azure Storage Account. You can use any one of these keys as the value of the *azurestorageaccountkey* parameter while creating the *azure-secret* in the section *Setting up the Environment for Deploying the Sample Application Container*.

10. Perform the following steps to create a file share.

    a. On the left pane, click **File Service** > **File shares**.

       The **File Shares** screen appears.



    b. Click **File share**.

       The **New file share** screen appears.

c. Specify a name for the file share in the **Name** field.
   You need to specify this name as the value of the *azureFile/shareName* parameter in the *pv.yaml* file.
d. Specify the storage size in the **Quota** field.
   You can specify a maximum value of **`5120`** GB.
e. Click **Create** to create the file share.

2.6.1.6 Creating a Kubernetes Cluster

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

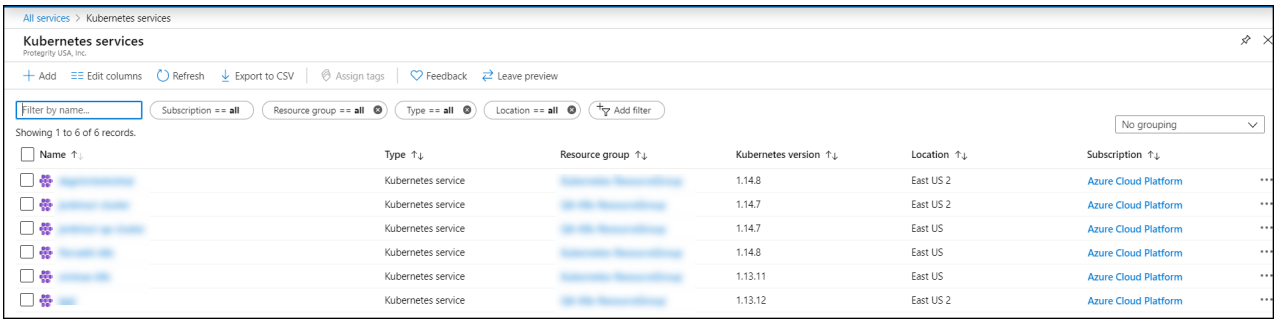
> **Note:** The steps listed in this procedure for creating a Kubernetes cluster are for reference use. If you have an existing Kubernetes cluster or want to create a Kubernetes cluster based on your requirements, then you can directly navigate to *step 9* to connect your Kubernetes cluster and the Linux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

➤ To create a Kubernetes cluster:

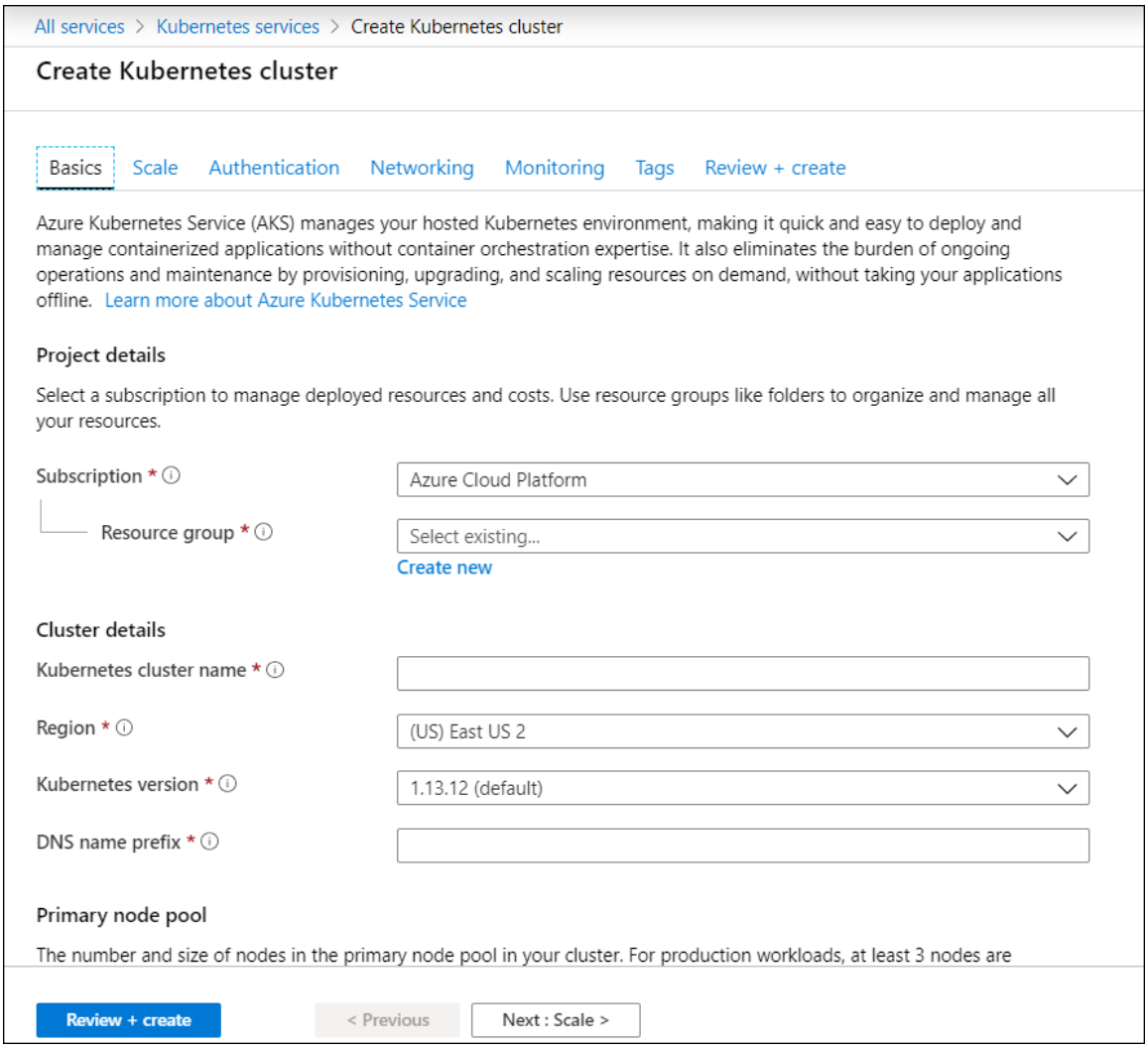
1. Login to the Azure environment.
   For more information about logging in to the Azure environment, refer to the section *Logging in to the Azure Environment*.
2. Click the **Portal** menu icon ( ).

The **Portal** menu appears.

3. Navigate to **All Services** > **Kubernetes services**.
   The **Kubernetes Services** screen appears.



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



5. In the **Resource group** field, select the required resource group.
6. In the **Kubernetes cluster name** field, specify a name for your Kubernetes cluster.
   Retain the default values for the remaining settings.
7. Click **Review + create** to validate the configuration.
8. Click **Create** to create the Kubernetes cluster.

The Kubernetes cluster is created.

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

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

   The Linux instance is now connected with the Kubernetes cluster. You can now run commands using the Kubernetes command line interface (kubectl) to control the nodes on the Kubernetes cluster.

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

    ```
    kubectl get nodes
    ```

    The command lists the Kubernetes nodes available in your cluster.

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

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

    For more information about deploying the containers without an Azure Policy and RBAC, refer to the section *Deploying the Containers without an Azure Policy and RBAC*.

2.7 Deploying the Containers with Azure Policy and Role-Based Access Control (RBAC)

This section describes the steps that you need to perform for deploying the Get ESA Policy, Init, and Sample Application containers with an Azure Policy and Role Based Access Control (RBAC). An Azure Policy determines the security policies for running a pod.

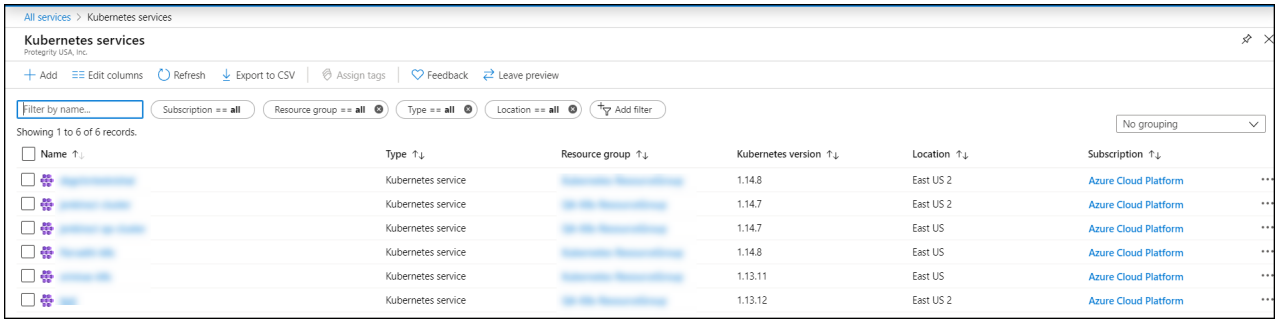
The user with the *cluster-admin* role creates the RBAC and applies the Azure Policy. This user also creates two Kubernetes service accounts that will install the Helm charts for deploying the Get ESA Policy and the Sample Application containers.

2.7.1 Applying an Azure Policy

This section describes how to apply an Azure Policy.

➤ To apply an Azure Policy:

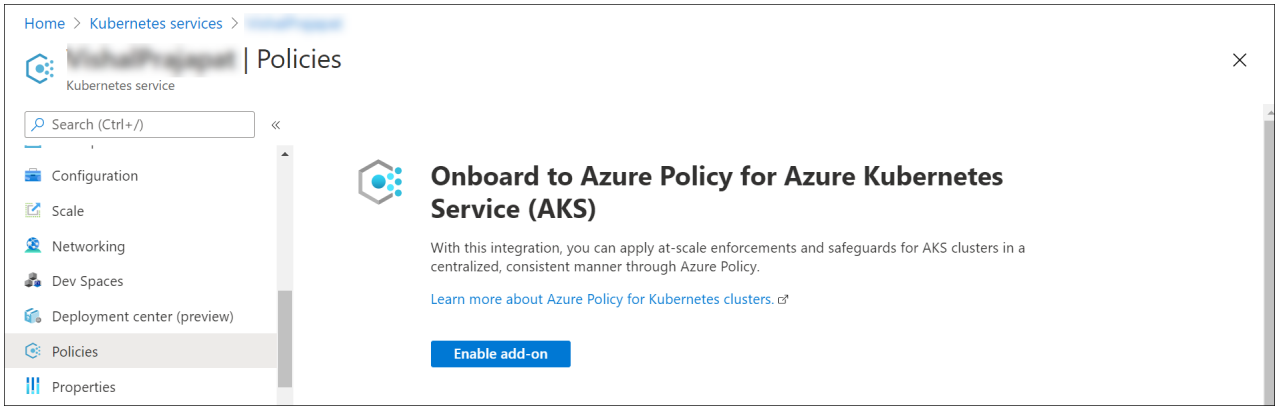
1. Login to the Azure environment.
   For more information about logging in to the Azure environment, refer to the section *Logging in to the Azure Environment*.
2. Click the **Portal** menu icon ( ).
   The **Portal** menu appears.
3. Perform the following steps to enable the application of Azure Policies on the Kubernetes cluster.
   a. Navigate to **All ServicesKubernetes services**.
      The **Kubernetes services** screen appears.



b. Select the Kubernetes cluster that you have created in the section *Creating a Kubernetes Cluster*.

c. Navigate to **Policies**.

The **Onboard to Azure Policy for Azure Kubernetes Service (AKS)** screen appears.



d. Click **Enable add-on**.

This allows you to assign Azure Policies to the Kubernetes cluster.

After you enable the policy application, the following pods are created in the Kubernetes cluster:

- *azure-policy-<ID>* - This pod is created in the kube-system namespace.
- *azure-policy-webhook-<ID>* - This pod is created in the kube-system namespace.
- *gatekeeper-audit-<ID>* - This pod is created in the gatekeeper-system namespace.
- *gatekeeper-controller-<ID>* - This pod is created in the gatekeeper-system namespace.

e. To verify the new pods created, run the following commands.

```
kubectl get pods -nkube-system
```

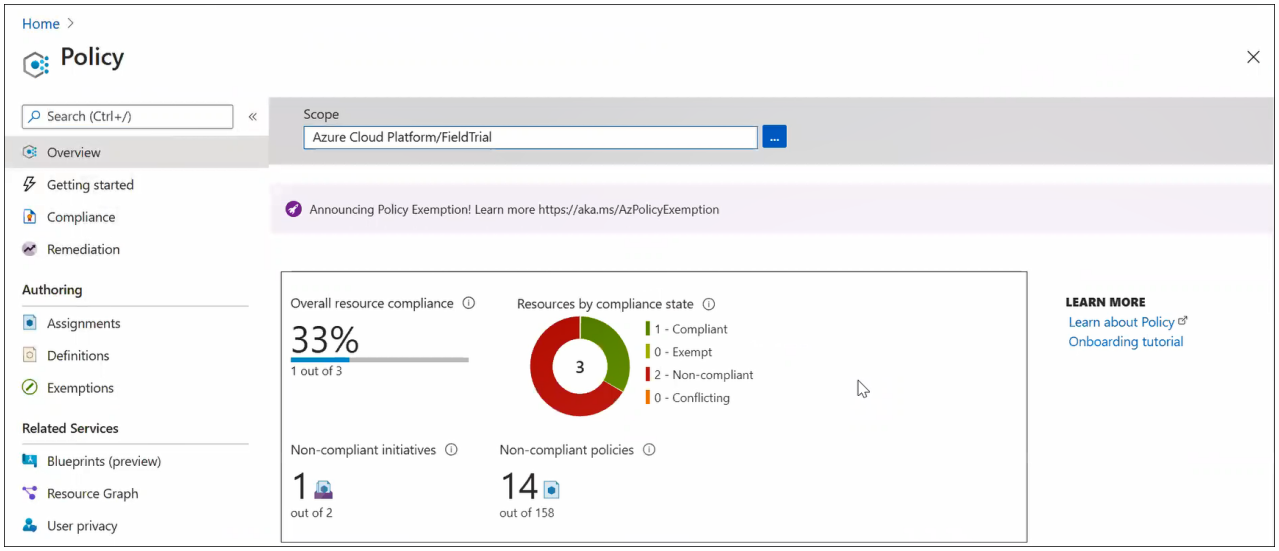
```
kubectl get pods -ngatekeeper-system
```

4. Perform the following steps to assign an existing Azure Policy to your Kubernetes cluster.

For more information about assigning a policy, refer to the section *Assign a built-in policy definition* in the Azure documentation.

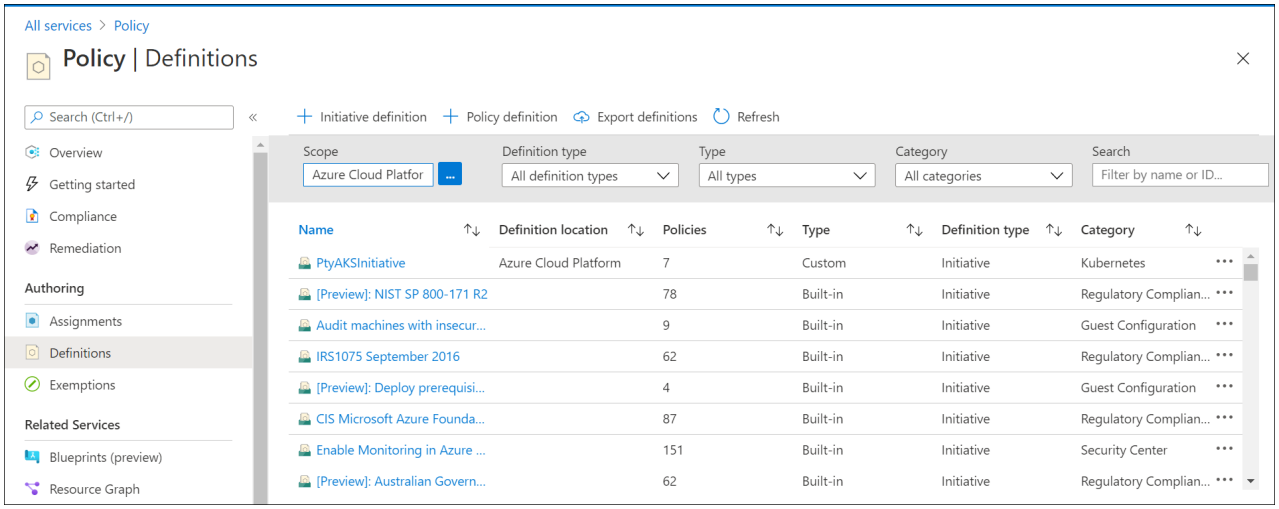
a. Navigate to **All Services** > **Policy**.

The **Policy** screen appears.



b. On the left pane, click **Definitions**.

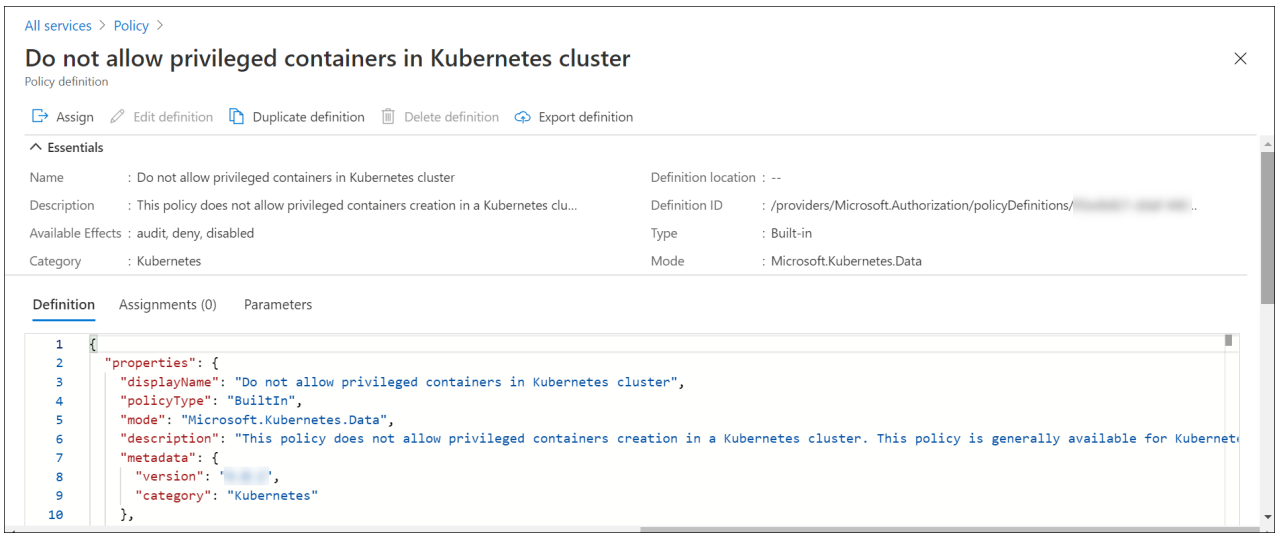
The **Definitions** screen appears.



c. From the **Category** drop-down list, clear the **Select all** check box, and then select the **Kubernetes** check box.

d. Select the required policy definition.

The policy definition details screen appears.



You need to apply the following Azure Policies to the Kubernetes cluster:

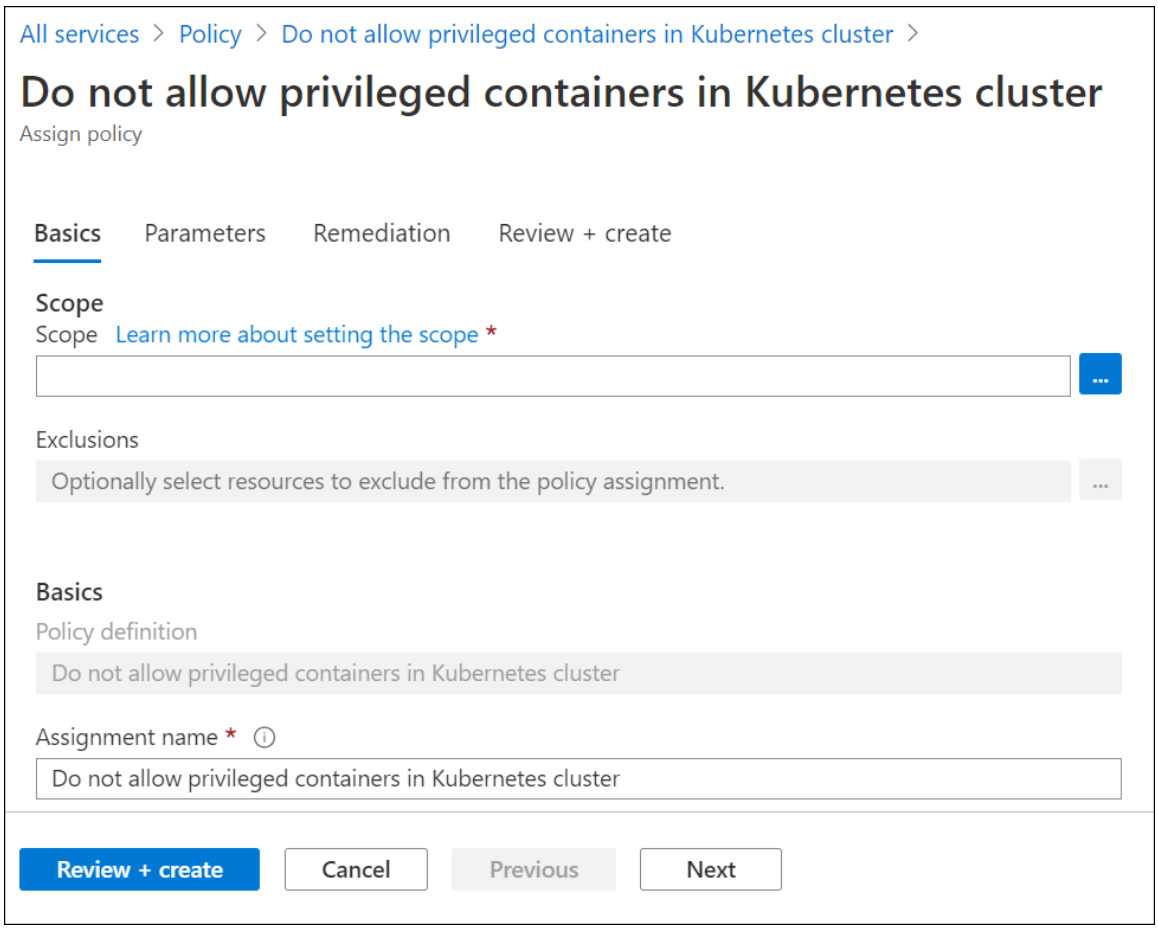
- *Do not allow privileged containers in Kubernetes cluster*
- *Kubernetes cluster containers should not share host process ID or host IPC namespace*
- *Kubernetes cluster containers should only use allowed capabilities*
- *Kubernetes cluster pods and containers should only run with approved user and group IDs*
- *Kubernetes cluster pods should only use allowed volume types*
- *Kubernetes cluster pods should only use approved host network and port range*
- *Kubernetes clusters should not allow container privilege escalation*
- *Kubernetes cluster containers should run with a read-only root file system*

For more information about each of the Azure Policies, refer to the section *Azure Policy built-in definitions for Azure Kubernetes Service* in the Azure documentation.

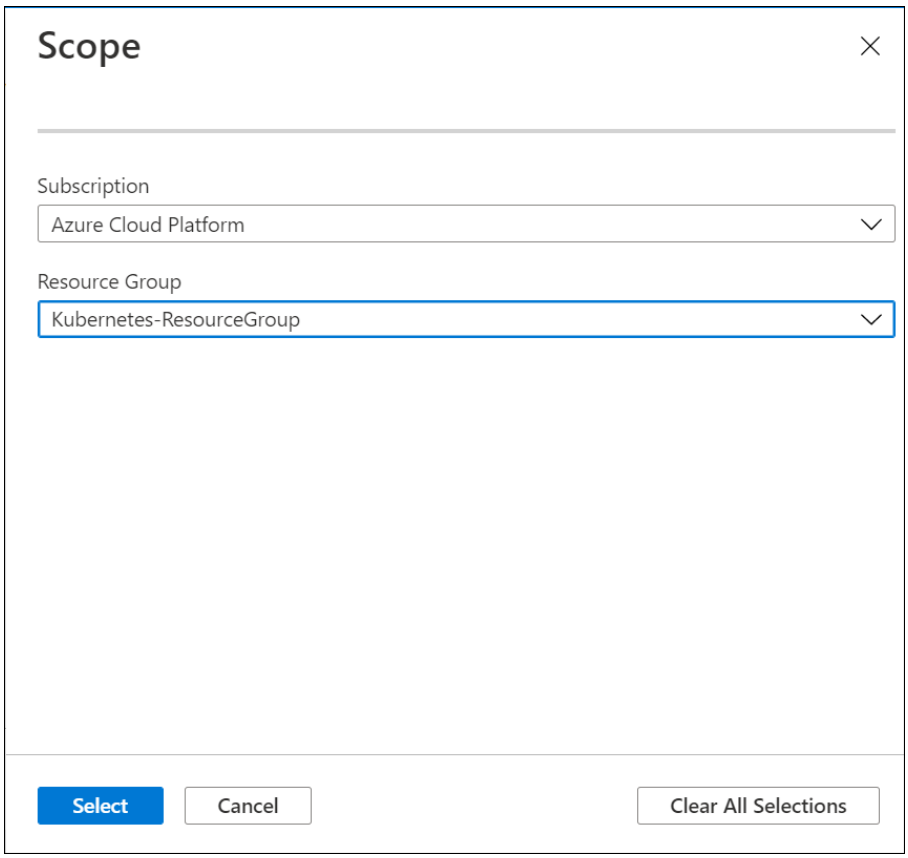
Repeat *step 4d* to *4m* for each of the in-built policies.

e. Click **Assign**.

The **Basics** tab of the **Assign policy** screen appears.



f. In the **Scope** section, click the ... icon to launch the scope selector.
The **Scope** screen appears.

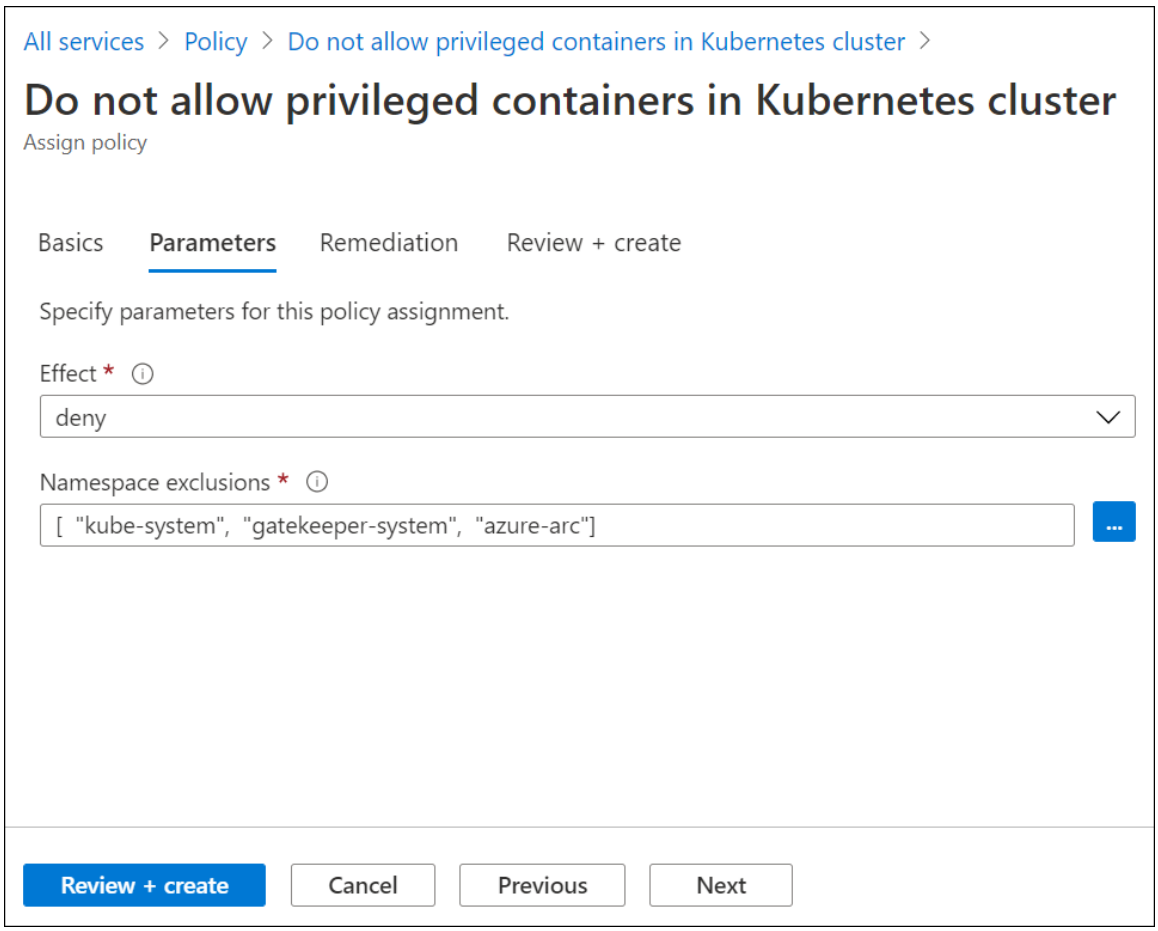


g. In the **Subscription** list, select your subscription.

h. In the **Resource Group** field, select the resource group of your Kubernetes cluster.
This ensures that the Azure Policy is applicable only to the resources that are a part of the resource group.

i. Click **Select** to specify the scope.

j. Navigate to the **Parameters** tab.
The **Parameters** tab appears.



The list of parameters depend on the specific policy that you have selected.

k. In the **Effect** list, specify one of the following values:
   - *audit* - The policy will allow any non-compliant resource to be created or updated. However, it will mark the resource as non-compliant.
   - *deny* - The policy will not allow any non-compliant resource to be created or updated.
   - *disabled* - Turns off the policy.

   Specify the value as *deny* for all the in-built policies.

   > **Note:** This parameter is applicable to all the in-built policies.

l. In the **Namespace** exclusions field, specify the namespaces that should not be assigned the Azure Policy.
   Specify the following namespaces for which the Azure Policy is not applicable:
   - *kube-system*
   - *gatekeeper-system*
   - *azure-arc*
   - *nginx*

   Enclose each namespace value in double quotes, and separate individual namespace values with a comma.

   > **Note:** This parameter is applicable to all the in-built policies.

m. If you want to assign the *Kubernetes cluster containers should only use allowed capabilities* policy, then specify the values of the following parameters:

- **Allowed capabilities** - Specify the value as `[]`, to ensure that no capability can be added to a container.
- **Required drop capabilities** - Specify the value as `["ALL"]` to ensure that the container drops all the capabilities.

n. If you want to assign the *Kubernetes cluster pods and containers should only run with approved user and group IDs* policy, then specify the values of the following parameters:

- **Run as user rule**: Specify the value as `MustRunAsNonRoot`
- **Allowed user ID ranges**: Specify the value as `{ "ranges:[]"}`
- **Run as group rule**: Specify the value as `MustRunAs`
- **Allowed group ID ranges**: Specify the value as `{ "ranges": [ { "min": 1, "max": 65535 } ]}`
- **Supplemental group rule**: Specify the value as `MustRunAs`
- **Allowed supplemental group ID ranges**: Specify the value as `{ "ranges": [ { "min": 1, "max": 65535 } ]}`
- **File system group rule**: Specify the value as `MustRunAs`
- **Allowed file system group ID ranges**: `{ "ranges": [ { "min": 1, "max": 65535 } ]}`

o. If you want to assign the *Kubernetes cluster pods should only use allowed volume types* policy, then specify the values of the following parameter:

- **Allowed volume types**: Specify the value as `[ "configMap", "secret", "persistentVolumeClaim", "emptyDir"]`

p. If you want to assign the *Kubernetes cluster pods should only use approved host network and port range* policy, then specify the values of the following parameters:

- **Allow host network usage**: Specify the value as `false`
- **Min host port**: Specify the value as `0`
- **Max host port**: Specify the value as `0`

q. Click **Review + create** to create the policy.

r. Repeat *step 4d* to *4q* for each Azure Policy.

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 Get ESA Policy, Init, and Sample Application containers.

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

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

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

   For example:

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

> **Important:** Ensure that the name of the service account is *get-esa-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 get-esa-deployer --namespace iap -o
jsonpath='{.secrets[0].name}'
```

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

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

For example:

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

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

You can obtain the context by running the following command:

```
kubectl config current-context
```

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

```
TOKEN=`echo -n <Token from step3> | base64 -d`
```

e. Repeat *step 1a* to *step 1d* for creating a service account to create a service account that can be used to deploy the Sample Application container.

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

2. Navigate to the directory where you have extracted the Helm charts for deploying the Get ESA Policy application, and run the following command to apply the Protegrity RBAC to the Kubernetes cluster.

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

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

For example:

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

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

2.7.3 Setting up the Environment for Deploying the Get ESA Policy Container

This section describes how to setup the environment for deploying the Get ESA Policy container on the Kubernetes cluster.

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

To setup the environment for deploying the Get ESA Policy container:

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

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

   For example:

   ```
   kubectl create namespace iap
   ```

2. If you want to authenticate the communication between the Kubernetes cluster and the Azure Container Registry by using your existing Docker credentials, then perform the following steps.

   a. Run the following command to login to Docker.

      ```
      docker login
      ```

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

   b. Run the following command to specify a secret for pulling the Get ESA Policy container images from the Azure Container Registry to Kubernetes.

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

      For example:

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

      For more information about creating secrets, refer to the section *Pull an Image from a Private Registry* in the Kubernetes documentation.

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

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

3. If you want to use an Azure Storage Container for storing the policy package instead of the Azure File Share, then modify the *credentials.json* file to update the credentials for the service principal 1.

   The *credentials.json* file has the following format.

   ```
   {
     "storage_account_name" : "AZURE_STORAGE_ACCOUNT_NAME",
     "tenant_id" : "AZURE_TENANT_ID",
     "client_id" : "AZURE_CLIENT_ID",
     "client_secret": "AZURE_CLIENT_SECRET"
   }
   ```

   For more information about the values for the *tenant_id*, *client_id*, and *client_secret* keys, refer to the section *Registering an Application*.

   For more information about the structure of the *credentials.json* file, refer to the section *Credentials.json*.

4. If you want to use an Azure Storage Container for storing the policy package, instead of the Azure File Share, then run the following command to create the *create-obj-access* secret, which is used by the Get ESA Policy container to upload the immutable policy package to the Azure storage container.

```
kubectl create secret generic create-obj-access --from-file=get-esa-policy/
credentials.json --namespace <NAMESPACE>

kubectl create secret generic create-obj-access --from-file=get-esa-policy/
credentials.json --namespace iap
```

For more information about the *credentials.json* file, refer to the section *Credentials.json*.

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

For more information about the *storageAccessSecrets* parameters, refer to the section *Storage Access Secrets*.

5. If you want to use Azure File Share for storing the policy package, instead of the Azure Storage Container, then run the following command to create the *azure-secret* secret, which is used by the Get ESA Policy container to upload the immutable policy package to the Azure File Share in the Azure Storage Account.

```
kubectl create secret generic azure-secret --from-
literal=azurestorageaccountname='<Storage Account Name>' --from-
literal=azurestorageaccountkey='<Storage Account Key>' --namespace <Namespace>

kubectl create secret generic azure-
secret --from-literal=azurestorageaccountname='test' --from-
literal=azurestorageaccountkey='<Storage Account Key>' --namespace iap
```

In the *azurestorageaccountname* parameter, specify the value of the Azure Storage Account that you have created in *step 7* of the section *Creating an Azure File Share*.

In the *azurestorageaccountkey* parameter, specify the value of any one of the storage access keys for the Azure Storage Account, as specified in *step 9* of the section *Creating an Azure File Share*.

> **Note:** Ensure that you specify the name of the secret as the value of the *azureFile/secretName* field in the *pv.yaml* file in *step 6a*.

6. Perform the following steps if you want to use an Azure File Share for storing the policy dump instead of the Azure Storage Container.
   a. Create a file named *pv.yaml* for creating a persistent volume resource.

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

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: azurefile
  labels:
    target: policy
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  storageClassName: azurefile
  azureFile:
    secretName: azure-secret
    shareName: <AZURE_SHARE_NAME>
    readOnly: false
  mountOptions:
  - dir_mode=0744
  - file_mode=0644
  - uid=1000
  - gid=1000
```

```
    #- mfsymlinks
    #- nobrl
```

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

In the *storageClassName* parameter, ensure that you specify *azurefile*, for using the Azure File Share as the persistent volume.

In the *azureFile/secretName* parameter, ensure that you specify name of the secret that you have created in *step 5*.

For example, specify the value of the *azureFile/secretName* parameter as *azure-secret*.

In the *azureFile/shareName* parameter, ensure that you specify name of the Azure File Share that you have created in *step 10* of the section *Creating an Azure File Share*.

Ensure that the value specified in the *metadata/labels/target* is also specified as the value of the *selector/matchLabels/target* parameter in the *claim.yaml* file in *step 5a*.

You can specify the access control settings for the files and directories on the Azure File Share by using the *mountOptions* parameter.

For more information about the mount option parameters, refer to the section *Use Azure Files with Linux* in the Azure documentation.

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

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

A persistent volume resource is created.

2.7.4 Deploying the Get ESA Policy Container on the Kubernetes Cluster

This section describes how to deploy the Get ESA Policy container on the Kubernetes cluster by installing the Helm charts.

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

➤ To deploy the Get ESA Policy container on the Kubernetes cluster:

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

```
az aks get-credentials --name <Name of the Kubernetes cluster in the kube-config file> --resource-group <Name of the Kubernetes Resource Group>
```

2. Perform the following steps to setup a Kubernetes context for the *get-esa-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 in the kube-config file> --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>
```

3. Run the following command to create a passphrase secret, which is used by the Get ESA Policy container to encrypt the policy.

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

This command is used to implement the Password-Based Encryption (PBE) for encrypting the policy.

You need to provide a passphrase and a salt as a Kubernetes secret to the Get ESA Policy Container. 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.

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

4. Run the following command to create the *esa-creds* secret, which is used by the Get ESA Policy container to access the ESA using the credentials of a *non-admin ESA user*.

```
kubectl create secret generic esa-creds --from-literal=esa_user=<ESA user> --from-
literal=esa_pass=<ESA user password> --namespace <NAMESPACE>

kubectl create secret generic esa-creds --from-literal=esa_user=non-admin --from-
literal=esa_pass=<ESA user password> --namespace iap
```

5. Perform the following steps if you want to use an Azure File Share for storing the policy dump instead of the Azure Storage Container.

a. Create a file named *claim.yaml* for creating a claim on the persistent volume that you have created in *step 6b* in the section *Setting up the Environment for Deploying the Get ESA Policy Container*.

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

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: azurefile
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: azurefile
  resources:
    requests:
      storage: 1Gi
  selector:
    matchLabels:
      target: policy
```

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

In the *storageClassName* parameter, ensure that you specify *azurefile*, for using the Azure File Share as the persistent volume.

In the *selector/matchLabels/target* parameter, ensure that you specify the value that was defined in the *metadata/labels/target* parameter in the *pv.yaml* file in *step 6a*.

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

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

For example:

```
kubectl apply -f claim.yaml -n iap
```

A persistent volume claim is created.

6. On the Linux instance, replace the value of the *azureKeyId* parameter in the *encryption-config.yaml* file with the resource ID of the Azure key that you have created in the section *Creating an Azure Key*.

   The Azure key is used to encrypt the policy package.

   The *encryption-config.yaml* file is part of the Get Policy Helm chart package.

   For more information about the extracted Helm charts, refer to the *step 10* of the section *Initializing the Linux Instance*.

> **Note:** This step is optional, and is required only if you want to encrypt the policy package using the Azure Key Vault.

7. Run the following command to create a ConfigMap for storing the Azure encryption configuration in YAML.

```
kubectl create configmap encryption-config --from-file=<Path of the Azure
encryption configuration file> --namespace <Namespace name>
```

   For example:

```
kubectl create configmap encryption-config --from-file=get-esa-policy/encryption-
config.yaml --namespace iap
```

   The ConfigMap is used to encrypt the policy package.

   Ensure that the name of the ConfigMap matches the value of the *encryptionConfigMap* parameter in the *values.yaml* file.

> **Note:** This step is optional, and is required only if you want to encrypt the policy package using the Azure Key Vault.

8. On the Linux instance, navigate to the location where you have extracted the Helm charts for deploying the Get ESA Policy application.

   For more information about the extracted Helm charts, refer to the *step 10* of the section *Initializing the Linux Instance*.

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

```
...
..
.

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

```
  fsGroup: 1000
  supplementalGroups: 1000

....
....
....

# k8s secret for storing ESA credentials
esaCredSecrets: ESA_CREDENTIAL_SECRET_NAME

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

# encryption configmap name
# create a configmap from the file 'encryption-config.yaml' as given below:
# kubectl create configmap encryption-config --from-file=encryption-
config.yaml=encryption-config.yaml --namespace <SAME_AS_OF_CHART_DEPLOYMENT>
# Enable below if you want to enable envelope encryption.
encryptionConfigMap: ENCRYPTION_CONFIGMAP

## choose your storage destination. Currently we support <VolumeMount | ObjectStore>
securityConfigDestination: VolumeMount

## If securityConfigDestination is Volume Mount,
## specify the name of persistent volume claim to be used for this deployment.
pvcName: PVC_NAME

# k8s secret for storing credentials with write access to az storage
storageAccessSecrets: WRITE_ACCESS_CREDENTIAL_SECRET_NAME

policy:
  esaIP: ESA_IP

  ## absolute path in PV or ObjectStore where the policy dump file will be stored. <eg. /
test/xyz/policy >
  ## If destination is Object Store, make sure first name in path is bucket name i.e.
test will be our existing bucket
  filePath: ABSOLUTE_POLICY_PATH

  ## time (in mins) to wait till the policies get downloaded
  ## default is 15 (mins)
  timeout: 15
```

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: Get ESA Policy Helm Chart Details*.

Field	Description
podSecurityContext	Specify the privilege and access control settings for the pod, if you want to use Azure Storage Container for storing the policy snapshot. The default values are set as follows: • *runAsUser* - 1000 • *runAsGroup* - 1000 • *fsGroup* - 1000 • *supplementalGroups* - 1000 **Note:** Ensure that at any given point of time, the value of at least one of the four parameters must be *1000*. This ensures that the pod has the required permissions to access the *pty* directories in the containers.

Field	Description
	For more information about the Pod Security Context, refer to the section *Configure a Security Context for a Pod or Container* in the Kubernetes documentation. For more information about the parameters, refer to the section *PodSecurityContext v1 Core* in the Kubernetes API documentation.
esaCredSecrets	Specify the value of the secret that is used to store the ESA credentials. This is used by the Get ESA Policy container to retrieve the policy from the ESA.
pbeSecrets	Specify the value of the secret that is used to store the passphrase and salt. The Get ESA Policy container encrypts the policy using the key generated by using the passphrase and salt. **Important:** Ensure that the passphrase has a minimum length of 12 characters, with atleast 1 uppercase letter, 2 digits, 2 special characters, and 2 non-letters (such as, digits or special characters). The passphrase must also have a minimum value of entropybits as *30* and a minimum strength of *0.66*. Entropybits defines the randomness of the password and the strength defines the complexity of the password. For more information about the password strength, refer to the section *Password Strength*.
encryptionConfigMap	Specify the name of the ConfigMap that you have created in *step 7* to encrypt the policy package using Azure Key Vault. **Note:** This field is optional, and is required only if you want to encrypt the policy package using the Azure Key Vault. **Note:** By default, the Passphrase-Based Encryption is used to encrypt the policy package. If you want to use the Azure Key Vault for encrypting the policy package, then you must comment out the *pbeSecrets* entry in the *values.yaml* file or leave the value of the *pbeSecrets* parameter as blank.
securityConfigDestination	Specify one of the following options: • *VolumeMount* - Specify this value if you want to store the policy package on a volume that is mounted on the pod. Use this option if you are using Azure File Share for storing the policy package. • *ObjectStore* - Specify this value if you want to store the policy package in the Azure Storage Container. By default, the value is set to *VolumeMount*.
storageAccessSecrets	Specify the value of the secret that is used to store the credentials with write access to the Azure storage container. This is used by the

Field	Description
	Get ESA Policy container to upload the policy package to the Azure storage container. If you want to use the Azure File Share for storing the policy package, then leave the value of this field blank. **Important:** This field is required if you want to store the immutable policy package in the Azure Storage Container.
pvcName	Specify the name of the Persistent Volume Claim where you want to store the policy package. If you want to use the Azure Storage Container for storing the policy package, then leave the value of this field blank. **Important:** This field is required if you want to store the immutable policy package in the Azure File Share.
policy/esaIP	Specify the IP address of the ESA from where you want to download the policy.
policy/filePath	Specify the path in the persistent volume or the object store where you want to store the immutable policy package. For example, if you are using Azure File Share as the persistent volume, then you can specify the file path as */<Mount point>/xyz/policy*. In this case, an immutable policy package with the name as *policy* will be stored in the */<Mount point>/xyz* directory in the required persistent volume. If you want to store the policy package in an Object Store, such as an Azure Storage Container, then you can specify the container name as the first name in the *filePath* field. For example, you can specify the value of the *filePath* field, as *test/LOB/policy*, where *test* is the container name, *LOB* is the directory inside the container, and *policy* is the name of the immutable policy package. In this example, the container name is specified as the first name of the file path. **Important:** Use only uppercase and lowercase letters, numbers, dot, and underscore in the file path. Do not use special characters in the file path. **Note:** This value is case-sensitive.
policy/timeout	Specify the time for which the Get ESA Policy container tries to communicate with the ESA for fetching the policies, after which the connection times out. By default, this value is set to *15*. The unit of the timeout parameter is minutes.

10. Run the following command to deploy the Get ESA Policy application on the Kubernetes cluster.

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

For example:

```
helm install get-policy-esa --namespace iap get-esa-policy
```

11. Run the following command to check the deployment status.

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

After the Get ESA Policy container is up, it retrieves the policy from the ESA and uploads it to the Azure storage container specified in the *values.yaml* file. After uploading the policy package, the Kubernetes job is completed.

2.7.5 Verifying the IAP Policy Package on the Azure Storage Container

The Get ESA Policy container retrieves the policy from the ESA and uploads a snapshot of this immutable policy to the Azure storage container. This section describes how to verify the policy package in the Azure environment.

> **Important:** This procedure is optional, and is required only if you want to use the Azure Storage Container storing the policy package, instead of using the Azure File Share.

➤ To verify the policy package on the Azure Storage Container:

Perform the following task to verify whether the immutable policy package has been uploaded to the Azure storage container.

1. Login to the Azure environment.

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

2. Click the **Portal** menu icon ( ).

   The **Portal** menu appears.

3. Click **Storage accounts**.

   The **Storage accounts** screen appears.

4. Navigate to the path that you have specified as the value of the *filePath* parameter in the *values.yaml* file in *step 7* of the section *Deploying the Get ESA Policy Container on the Kubernetes Cluster*.

   The immutable policy package appears.

5. Click the immutable policy package to view additional details about the file.

2.7.6 Verifying the IAP Policy Package on the Azure File Share

The Get ESA Policy container retrieves the policy from the ESA and uploads a snapshot of this immutable policy to the Azure file share. This section describes how to verify the policy package in the Azure environment.

➤ To verify the policy package on the Azure File Share:

Perform the following task to verify whether the policy package file has been uploaded to the Azure file share.

1. Login to the Azure environment.
   For more information about logging in to the Azure environment, refer to the section *Logging in to the Azure Environment*.
2. Click the **Portal** menu icon ( ).
   The **Portal** menu appears.
3. Click **Storage accounts**.
   The **Storage accounts** screen appears.
4. Navigate to **File service** > **File shares**.
   The **File shares** screen appears.
5. Navigate to the path that you have specified as the value of the *filePath* parameter in the *values.yaml* file in *step 7* of the section *Deploying the Get ESA Policy Container on the Kubernetes Cluster*.
   The immutable policy package appears.
6. Click the immutable policy package to view additional details about the file.

2.7.7 Setting up the Environment for Deploying the Sample Application Container

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

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

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

1. On the Linux instance, run the following command to create the namespace required for Helm deployment.
   ```
   kubectl create namespace <Namespace name>
   ```
   For example:
   ```
   kubectl create namespace iap
   ```
2. Perform the following steps if you want to use NGINX as the Ingress Controller.

   > **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 <NGINX
      Namespace name> --set controller.replicaCount=2 ingress-
      ```

```
nginx/ingress-nginx --set controller.publishService.enabled=true
--set controller.ingressClassResource.name=nginx-iap --
set controller.service.annotations."service\.beta\.kubernetes\.io/azure-load-
balancer-internal"="true"
```

For example:

```
helm install nginx-ingress --namespace nginx --set controller.replicaCount=2
ingress-nginx/ingress-nginx --set controller.publishService.enabled=true
--set controller.ingressClassResource.name=nginx-iap --
set controller.service.annotations."service\.beta\.kubernetes\.io/azure-load-
balancer-internal"="true"
```

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 using the following command:

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

For example:

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

3. If you want to authenticate the communication between the Kubernetes cluster and the Azure Container Registry by using your existing Docker credentials, then perform the following steps.

a. Run the following command to login to Docker.

```
docker login
```

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

b. Run the following command to specify a secret for pulling the Sample Application container image from the Azure Container Registry to Kubernetes.

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

For example:

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

For more information about creating secrets, refer to the section *Pull an Image from a Private Registry* in the Kubernetes documentation.

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

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

4. If you want to use an Azure Storage Container for storing the policy package, instead of the Azure File Share, then perform the following steps.

a. Modify the *credentials.json* file, which is present inside the directory where the Helm charts for the Sample Application container have been extracted, to update the credentials for the service principal 2.

The following snippet contains the format of the *credentials.json* file.

```
{
  "storage_account_name" : "AZURE_STORAGE_ACCOUNT_NAME",
  "tenant_id" : "AZURE_TENANT_ID",
```

```
  "client_id" : "AZURE_CLIENT_ID",
  "client_secret": "AZURE_CLIENT_SECRET"
}
```

For more information about the values for the *tenant_id*, *client_id*, and *client_secret* keys, refer to the section *Registering an Application*.

For more information about the structure of the *credentials.json* file, refer to the section *credentials.json*.

b. Run the following command to create the *read-obj-access* secret, which is used by the Sample Application container to download the immutable policy package from the Azure storage container to the shared memory.

```
kubectl create secret generic read-obj-access --from-file=spring-apjava/
credentials.json --namespace <NAMESPACE>

kubectl create secret generic read-obj-access --from-file=spring-apjava/
credentials.json --namespace iap
```

For more information about the *credentials.json* file, refer to the section *credentials.json*.

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

For more information about the *azAccessSecrets* parameters, refer to the section *Azure Access Secrets*.

You must also perform this step if you want to use the Azure Key Vault for encrypting the policy package.

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

```
kubectl create secret generic key-secret --from-literal='privatekey'='<Azure Key
Identifier>' -n iap
```

For more information about the KeyID for the Azure Key Vault, refer to *step 22* of the section *Creating an Azure Key*.

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

For more information about the *privateKeySecret* parameters, refer to the section *Private Key Secret*.

6. If you want to use Azure File Share for storing the policy package, instead of the Azure Storage Container, then run the following command to create the *azure-secret* secret, which is used by the Sample Application container to pull the Policy packages.

```
kubectl create secret generic azure-secret --from-
literal=azurestorageaccountname='<Storage Account Name>' --from-
literal=azurestorageaccountkey='<Storage Account Key>' --namespace <Namespace>

kubectl create secret generic azure-
secret --from-literal=azurestorageaccountname='test' --from-
literal=azurestorageaccountkey='<Storage Account Key>' --namespace iap
```

In the *azurestorageaccountname* parameter, specify the value of the Azure Storage Account that you have created while creating the Azure File Share.

In the *azurestorageaccountkey* parameter, specify the value of any one of the storage access keys of the Azure Storage Account that you have created in the section *Creating an Azure File Share*.

> **Note:**

> Ensure that you specify the name of the secret as the value of the *azureFile/secretName* field in the *pv.yaml* file in *step 7a*.

7. Perform the following steps if you want to use an Azure File Share for storing the policy dump instead of the Azure Storage Container.
   a. Create a file named *pv.yaml* for creating a persistent volume resource.

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

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: azurefile
  labels:
    target: policy
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  storageClassName: azurefile
  azureFile:
    secretName: azure-secret
    shareName: <AZURE_SHARE_NAME>
    readOnly: false
  mountOptions:
  - dir_mode=0744
  - file_mode=0644
  - uid=1000
  - gid=1000
    #- mfsymlinks
    #- nobrl
```

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

      In the *storageClassName* parameter, ensure that you specify *azurefile*, for using the Azure File Share as the persistent volume.

      In the *azureFile/secretName* parameter, ensure that you specify name of the secret that you have created in *step 6*.

      For example, specify the value of the *azureFile/secretName* parameter as *azure-secret*.

      In the *azureFile/shareName* parameter, ensure that you specify name of the Azure File Share that you have created in *step 10* in the section *Creating an Azure File Share*.

      Ensure that the value specified in the *metadata/labels/target* is also specified as the value of the *selector/matchLabels/target* parameter in the *claim.yaml* file in *step 8c*.

      You can specify the access control settings for the files and directories on the Azure File Share by using the *mountOptions* parameter.

      For more information about the mount option parameters, refer to the section *Use Azure Files with Linux* in the Azure documentation.

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

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

      A persistent volume resource is created.

   c. Create a file named *claim.yaml* for creating a claim on the persistent volume that you have created in *step 8b*.

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

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: azurefile
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: azurefile
  resources:
    requests:
      storage: 1Gi
  selector:
    matchLabels:
      target: policy
```

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

In the *storageClassName* parameter, ensure that you specify *azurefile*, for using the Azure File Share as the persistent volume.

In the *selector/matchLabels/target* parameter, ensure that you specify the value that was defined in the *metadata/labels/target* parameter in the *pv.yaml* file in *step 8a*.

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

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

For example:

```
kubectl apply -f claim.yaml -n iap
```

A persistent volume claim is created.

2.7.8 Deploying the Sample Application Container on the Kubernetes Cluster

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

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

Important: If you are deploying a Customer Application container instead of the Sample Application container, then ensure that the Init container and the Customer Application container have the same user ID. This ensures that the Customer Application container can access the shared memory.

➤ To deploy a release on the Kubernetes cluster:

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

```
az aks get-credentials --name <Name of the Kubernetes cluster in the kube-config file> --resource-group <Name of the Kubernetes Resource Group>
```

2. Perform the following steps to set up a Kubernetes context for the *iap-deployer* service account.

a. Run the following command to create user credentials in the *kube-config* file, using the token created in *step 1d* of the section *Applying RBAC*.

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

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

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

3. On the Linux instance, replace the value of the *azureKeyId* parameter in the *decryption-config.yaml* file with the resource ID of the Azure key that you have created in the section *Creating an Azure Key*.

   The Azure key is used to decrypt the policy package.

   The *decryption-config.yaml* file is part of the Helm chart package.

   For more information about the extracted Helm charts, refer to the *step 11* of the section *Initializing the Linux Instance*.

   > **Note:** This step is optional, and is required only if you want to encrypt the policy package using the Azure Key Vault.

4. Run the following command to create a ConfigMap map for storing the Azure decryption configuration in YAML:

   ```
   kubectl create configmap decryption-config --from-file=<Path of the Azure decryption configuration file> --namespace <Namespace name>
   ```

   For example:

   ```
   kubectl create configmap decryption-config --from-file=spring-apjava/decryption-config.yaml --namespace dsg
   ```

   The ConfigMap is used to decrypt the policy package.

   Ensure that the name of the ConfigMap matches the value of the *decryptionConfigMap* parameter in the *values.yaml* file.

   > **Note:** This step is optional, and is required only if you want to encrypt the policy package using the Azure Key Vault.

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

   For more information about the extracted Helm charts, refer to the *step 10* of the section *Initializing the Linux Instance*.

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

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

# If Volume Mount, name of persistent volume claim to be used for this deployment.
```

```
pvcName: PVC_NAME

# k8s secret for storing credentials with read access to az storage
storageAccessSecrets: READ_ACCESS_CREDENTIAL_SECRET_NAME

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

# create a configmap from the file 'decryption-config.yaml' as given below:
# kubectl create configmap decryption-config --from-file=decryption-
config.yaml=decryption-config.yaml --namespace <SAME_AS_OF_CHART_DEPLOYMENT>
# applicable if policy dump was encrypted using envelope encryption.
decryptionConfigMap: DECRYPTION_CONFIGMAP

## start with blue deployment first
deploymentInUse: blue

## Policy metadata for Blue Deployments
blueDeployment:
  enabled: true
  # Choose your storage source. Currently we support <VolumeMount | ObjectStore>
  securityConfigSource: VolumeMount
  policy:
    # absolute path in PV or ObjectStore where the policy dump file will be stored. <eg. /
test/xyz/policy >
    # If Object Store make sure first name in path is bucket name i.e. /test will be our
existing bucket
    filePath: ABSOULTE_POLICY_PATH

## Policy metadata for Green Deployments
greenDeployment:
  enabled: false
  # Choose your storage source. Currently we support <VolumeMount | ObjectStore>
  securityConfigSource: VolumeMount
   policy:
    # absolute path in PV or ObjectStore where the policy dump file will be stored. <eg. /
test/xyz/policy >
    # If Object Store make sure first name in path is bucket name i.e. /test will be our
existing bucket
    filePath: ABSOULTE_POLICY_PATH

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

...
..
.


## specify the ports exposed in your springapp configurations where,
## name - distinguishes between different ports.
## port - the port on which you wan't to expose the service externally.
## targetPort - the port no. configured while creating Tunnel.
springappService:
    name: "restapi"
    port: 8080
    targetPort: 8080

## Ingress enables testing of Green (V2/Staging) Deployments alongside Blue ones
simultaneously.
## This is realized by exposing both deployments on the same External IP but on different
URL Paths.
## If you don't want expose both the deployments at the same time, keep 'ingress.enabled'
as false.
## This means that only Blue deployment gets exposed on the Load Balancer's External IP
via prod service.
ingress:
  enabled: true

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

```
  ## specify external Ingress Controller if any,
  ## else keep the 'ingress.class' field empty
  ## to use cloud native Ingress Controller.
  annotations:
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"

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

  ## specify the host names and paths for Ingress rules
  ## prod and staging http paths can be used as optional fields.
  hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8080
        prod: "/"
    ## Add host section with the hostname used as CN while creating server certificates.
    ## For accessing the staging end-points also, we need hostname as that of CN in
server certs.
    ## While creating the certificates you can use *.example.com as CN for the below
example
    # - host: "prod.example.com"
    #   httpPaths:
    #   - port: 8080
    #     prod: "/"
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8080
    #     staging: "/"
    #
  ## If TLS is terminated on the Ingress Controller Load Balancer,
  ## then uncomment the tls section below and,
  ## provide K8s TLS Secret containing the certificate and key.
  # tls:
  #  - secretName: example-tls
  #    hosts:
  #      - prod.example.com
```

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

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

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

Field	Description
podSecurityContext	Specify the privilege and access control settings for the pod. The default values are set as follows: • *runAsUser* - 1000 • *runAsGroup* - 1000 • *fsGroup* - 1000 • *supplementalGroups* - 1000 **Note:** Ensure that at any given point of time, the value of at least one of the four parameters must be *1000*. This ensures that the pod has the required permissions to access the *pty* directories in the containers.

Field	Description
	For more information about the Pod Security Context, refer to the section *Configure a Security Context for a Pod or Container* in the Kubernetes documentation. For more information about the parameters, refer to the section *PodSecurityContext v1 Core* in the Kubernetes API documentation.
pvcName	Specify the name of the Persistent Volume Claim where you want to store the policy package. If you want to use the Azure Storage Container for storing the policy package, then leave the value of this field blank. **Important:** This field is required if you want to store the immutable policy package in the Azure File Share.
storageAccessSecrets	Specify the value of the secret that is used to store the credentials with read access to the Azure storage container. This is used by the Sample Application container to download the policy package from the Azure storage container to the shared memory. If you want to use the Azure File Share for storing the policy package, then leave the value of this field blank. **Important:** This field is required if you want to store the immutable policy package in the Azure Storage Container.
pbeSecrets	Specify the Kubernetes secret that contains the passphrase and the salt that were used to generate the key, which encrypted the policy. The Sample Application container uses the value specified in the *pbeSecrets* parameter to decrypt the policy. **Important:** Ensure that the passphrase has a minimum length of 12 characters, with atleast 1 uppercase letter, 2 digits, 2 special characters, and 2 non-letters (such as, digits or special characters). The passphrase must also have a minimum value of entropybits as *30* and a minimum strength of *0.66*. Entropybits defines the randomness of the password and the strength defines the complexity of the password. For more information about the password strength, refer to the section *Password Strength*.
decryptionConfigMap	Specify the value as *decryption-config*. This indicates the name of the ConfigMap that will be created from the *decryption-config.yaml* file using the `kubectl` command. The *decryption-config.yaml*

Field	Description
	file includes the *azureKeyId*, which contains the resource ID of the Azure key. **Note:** This field is optional, and is required only if you want to encrypt the policy package using the Azure Key Vault. **Note:** By default, the Passphrase-Based Encryption is used to encrypt the policy package. If you want to use the Azure Key Vault for encrypting the policy package, then you must comment out the *pbeSecrets* entry in the *values.yaml* file or leave the value of the *pbeSecrets* parameter as blank.
deploymentInUse	Specify the value as *blue*. This indicates that the *blue* deployment strategy is in use.
blueDeployment/enabled	Specify the value as *true* to enable the *blue* deployment strategy.
blueDeployment/securityConfigSource	Specify one of the following options: • *VolumeMount* - Specify this value if you want to store the policy package on a volume that is mounted on the pod. Use this option if you are using Azure File Share for storing the policy package. • *ObjectStore* - Specify this value if you want to store the policy package in the Azure Storage Container. By default, the value is set to *VolumeMount*.
blueDeployment/policy/filePath	Specify the path in the persistent volume or the object store from where you want to retrieve the immutable policy package. Ensure that you specify the same value that you have specified as the value of the *filePath* parameter in the *values.yaml* file for the Get ESA Policy container in *step 7* of the section *Deploying the Get ESA Policy Container on the Kubernetes Cluster*. **Important:** Use only uppercase and lowercase letters, numbers, dot, and underscore in the file path. Do not use special characters in the file path. **Note:** This value is case-sensitive.
greenDeployment/enabled	Specify the value as *false*. While deploying Release 1, the *green* deployment strategy is always disabled.
springappService/name	Specify a name for the tunnel to distinguish between ports.
springappService/port	Specify the port number on which you want to expose the Kubernetes service externally.
springappService/targetPort	Specify the port on which the Sample application is running inside the Docker container.

Field	Description
ingress/enabled	Specify this value as *true* to enable the ingress resource in Kubernetes. The ingress resource acts as a load balancer. It enables the simultaneous deployment of *blue* and *green* strategy by exposing both the deployments on the same external IP address, but on different URL paths.
ingress/ingressClassName	Specify the value of the external ingress controller, if any. For example, specify, *nginx-iap*, if you want to use the NGINX Ingress Controller.
nginx.ingress.kubernetes.io/auth-tls-verify-client	Set the value of this field to on, if you want to enable TLS mutual authentication between the REST API client and the Azure load balancer. By default, this field is *disabled*.
nginx.ingress.kubernetes.io/auth-tls-secret	Specify a value for the CA secret, if you have enabled the TLS mutual authentication between the REST API client and the Azure load balancer. By default, this field is *disabled*. For more information about the value of this field, refer to the section *Ingress Configuration*.
ingress/hosts/host	Specify the hostname of the ingress resource. If you are using the external IP address of the ingress resource for performing the security operations, then leave this field blank.
ingress/hosts/httpPaths/port	Specify the port on which the Sample application is running inside the Docker container.
ingress/hosts/httpPaths/prod	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	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.

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

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

   For example:

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

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

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

      For example:

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

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

kubectl get ingress -n <namespace>

For example:

kubectl get ingress -n iap

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

2.7.9 Upgrading the Helm Charts

This section describes the steps for upgrading the Helm charts.

> **Note:** Protegrity recommends using the *Blue-Green* method of upgrading the Helm charts. In this method, you use two modes of deploying the application, production mode and staging mode. In the staging mode, you can test the changes to the application in a staging environment, without impacting the production environment. After you have tested the application in the staging environment, you can switch the environments by making the staging environment as the new production environment and the production environment as the new staging environment.
>
> In this way, you can seamless divert the customer traffic to the modified application without any production downtime. After your modified application is running on the new production environment, you can shutdown the application that is running on the staging environment. In the *Blue-Green* method of deploying applications, only one Release is active at a given point of time.

➤ To upgrade the Helm charts:

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

   For more information about the extracted Helm charts, refer to the *step 11* of the section *Initializing the Linux Instance*.

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

```
...
..
## set the Pod's security context object
## leave the field empty if not applicable
podSecurityContext:
  runAsUser: 1000
  runAsGroup: 1000
  fsGroup: 1000
  supplementalGroups: 1000
...
...
...
# If Volume Mount, name of persistent volume claim to be used for this deployment.
pvcName: PVC_NAME

# k8s secret for storing credentials with read access to az storage
storageAccessSecrets: READ_ACCESS_CREDENTIAL_SECRET_NAME

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

# create a configmap from the file 'decryption-config.yaml' as given below:
# kubectl create configmap decryption-config --from-file=decryption-
config.yaml=decryption-config.yaml --namespace <SAME_AS_OF_CHART_DEPLOYMENT>
# applicable if policy dump was encrypted using envelope encryption.
decryptionConfigMap: DECRYPTION_CONFIGMAP

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

```
## Policy metadata for Blue Deployments
blueDeployment:
  enabled: true
  # Choose your storage source. Currently we support <VolumeMount | ObjectStore>
  securityConfigSource: VolumeMount
  policy:
    # absolute path in PV or ObjectStore where the policy dump file will be stored. <eg. /
test/xyz/policy >
    # If Object Store make sure first name in path is bucket name i.e. /test will be our
existing bucket
    filePath: ABSOULTE_POLICY_PATH

## Policy metadata for Green Deployments
greenDeployment:
  enabled: false
  # Choose your storage source. Currently we support <VolumeMount | ObjectStore>
  securityConfigSource: VolumeMount
  policy:
    # absolute path in PV or ObjectStore where the policy dump file will be stored. <eg. /
test/xyz/policy >
    # If Object Store make sure first name in path is bucket name i.e. /test will be our
existing bucket
    filePath: ABSOULTE_POLICY_PATH

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

...
..
.


## specify the ports exposed in your springapp configurations where,
## name - distinguishes between different ports.
## port - the port on which you wan't to expose the service externally.
## targetPort - the port no. configured while creating Tunnel.
springappService:
    name: "restapi"
    port: 8080
    targetPort: 8080

## Ingress enables testing of Green (V2/Staging) Deployments alongside Blue ones
simultaneously.
## This is realized by exposing both deployments on the same External IP but on different
URL Paths.
## If you don't want expose both the deployments at the same time, keep 'ingress.enabled'
as false.
## This means that only Blue deployment gets exposed on the Load Balancer's External IP
via prod service.
ingress:
  enabled: true

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

  ## specify external Ingress Controller if any,
  ## else keep the 'ingress.class' field empty
  ## to use cloud native Ingress Controller.
  annotations:
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"

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

  ## specify the host names and paths for Ingress rules
  ## prod and staging http paths can be used as optional fields.
  hosts:
```

```
    - host: "prod.example.com"
      httpPaths:
      - port: 8080
        prod: "/"
    ## Add host section with the hostname used as CN while creating server certificates.
    ## For accessing the staging end-points also, we need hostname as that of CN in
server certs.
    ## While creating the certificates you can use *.example.com as CN for the below
example
    # - host: "prod.example.com"
    #   httpPaths:
    #   - port: 8080
    #     prod: "/"
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8080
    #     staging: "/"
    #
  ## If TLS is terminated on the Ingress Controller Load Balancer,
  ## then uncomment the tls section below and,
  ## provide K8s TLS Secret containing the certificate and key.
  # tls:
  #  - secretName: example-tls
  #    hosts:
  #      - prod.example.com
```

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

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

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

Field	Description
podSecurityContext	Specify the privilege and access control settings for the pod. The default values are set as follows: • *runAsUser* - 1000 • *runAsGroup* - 1000 • *fsGroup* - 1000 • *supplementalGroups* - 1000 **Note:** Ensure that at any given point of time, the value of at least one of the four parameters must be *1000*. This ensures that the pod has the required permissions to access the *pty* directories in the containers. For more information about the Pod Security Context, refer to the section *Configure a Security Context for a Pod or Container* in the Kubernetes documentation. For more information about the parameters, refer to the section *PodSecurityContext v1 Core* in the Kubernetes API documentation.
pvcName	Specify the name of the Persistent Volume Claim where you want to store the policy package.

Field	Description
	If you want to use the Azure Storage Container for storing the policy package, then leave the value of this field blank. **Important:** This field is required if you want to store the immutable policy package in the Azure File Share.
storageAccessSecrets	Specify the value of the secret that is used to store the credentials with read access to the Azure storage container. This is used by the Sample Application container to download the policy package from the Azure storage container to the shared memory. If you want to use the Azure File Share for storing the policy package, then leave the value of this field blank. **Important:** This field is required if you want to store the immutable policy package in the Azure Storage Container.
pbeSecrets	Specify the Kubernetes secret that contains the passphrase and the salt that were used to generate the key, which encrypted the policy. The Sample Application container uses the value specified in the *pbeSecrets* parameter to decrypt the policy.
decryptionConfigMap	Specify the value as *decryption-config*. This indicates the name of the ConfigMap that will be created from the *decryption-config.yaml* file using the `kubectl` command. The *decryption-config.yaml* file includes the *azureKeyId*, which contains the resource ID of the Azure key. **Note:** This field is optional, and is required only if you want to encrypt the policy package using the Azure Key Vault. **Note:** By default, the Passphrase-Based Encryption is used to encrypt the policy package. If you want to use the Azure Key Vault for encrypting the policy package, then you must comment out the *pbeSecrets* entry in the *values.yaml* file or leave the value of the *pbeSecrets* as blank.
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 want to store the policy package on a volume that is mounted on the pod. Use this option if you are using Azure File Share for storing the policy package. • *ObjectStore* - Specify this value if you want to store the policy package in the Azure Storage Container. By default, the value is set to *VolumeMount*.

Field	Description
greenDeployment/policy/filePath	Specify the path in the persistent volume or the object store from where you want to retrieve the immutable policy package. Ensure that you specify the same value that you have specified as the value of the *filePath* parameter in the *values.yaml* file for the Get ESA Policy container in *step 9* of the section *Deploying the Get ESA Policy Container on the Kubernetes Cluster*. **Important:** Use only uppercase and lowercase letters, numbers, dot, and underscore in the file path. Do not use special characters in the file path. **Note:** This value is case-sensitive.
springappService/name	Specify a name for the tunnel to distinguish between ports.
springappService/port	Specify the port number on which you want to expose the Kubernetes service externally.
springappService/targetPort	Specify the port on which the Sample application is running inside the Docker container.
ingress/enabled	Specify this value as *true* to enable the ingress resource in Kubernetes. The ingress resource acts as a load balancer. It enables the simultaneous deployment of *blue* and *green* strategy by exposing both the deployments on the same external IP address, but on different URL paths.
ingress/ingressClassName	Specify the value of the external ingress controller, if any. For example, specify, *nginx-iap*, if you want to use the NGINX Ingress Controller.
nginx.ingress.kubernetes.io/auth-tls-verify-client	Set the value of this field to on, if you want to enable TLS mutual authentication between the REST API client and the Azure load balancer. By default, this field is *disabled*.
nginx.ingress.kubernetes.io/auth-tls-secret	Specify a value for the CA secret, if you have enabled the TLS mutual authentication between the REST API client and the Azure load balancer. By default, this field is *disabled*. For more information about the value of this field, refer to the section *Ingress Configuration*.
ingress/hosts/host	Specify the hostname of the ingress resource.
ingress/hosts/httpPaths/port	Specify the port on which the Sample application is running inside the Docker container.
ingress/hosts/httpPaths/prod	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.

Field	Description
ingress/hosts/httpPaths/staging	Specify the value of this field as /. This ensures that the data traffic from the REST API client is redirected to the required production or staging URL based on the hostname value.

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

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

   For example:

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

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

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

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

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

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

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

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

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

   For example:

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

   Using this configuration ensures that the Sample application is deployed with the updated policy package 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 be shutdown the all the pods that were deployed as part of the *blue* deployment.

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

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

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

   For example:

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

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

If you want to modify any policy or subsequent releases, then you need to repeat *step 1* to *step 8*. The only difference is that you need to toggle the value of the *deploymentInUse* field from *green* to *blue* and vice versa depending on which deployment contains your modified configurations.

2.8 Deploying the Containers without an Azure Policy and RBAC

This section describes the steps that you need to perform for deploying the Get ESA Policy, Init, and Sample Application containers without an Azure Policy and RBAC.

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

2.8.1 Deploying the Get ESA Policy Container on the Kubernetes Cluster

This section describes how to deploy the Get ESA Policy container on the Kubernetes cluster by installing the Helm charts.

➤ To deploy the Get ESA Policy container on the Kubernetes cluster:

1. On the Linux instance, run the following command to create the namespace required for Helm deployment.
   ```
   kubectl create namespace <Namespace name>
   ```
   For example:
   ```
   kubectl create namespace iap
   ```
2. If you want to authenticate the communication between the Kubernetes cluster and the Azure Container Registry by using your existing Docker credentials, then perform the following steps.
   a. Run the following command to login to Docker.
      ```
      docker login
      ```
      The login process creates a *config.json* file, which contains an authorization token.

   b. Run the following command to specify a secret for pulling the Get ESA Policy container images from the Azure Container Registry to Kubernetes.
      ```
      kubectl create secret generic <Secret_name> --from-file=.dockerconfigjson=/<Path to the config.json file> --type=kubernetes.io/dockerconfigjson --namespace iap
      ```
      For example:
      ```
      kubectl create secret generic regcrd --from-file=.dockerconfigjson=/home/<user>/.docker/config.json --type=kubernetes.io/dockerconfigjson --namespace iap
      ```
      For more information about creating secrets, refer to the section *Pull an Image from a Private Registry* in the Kubernetes documentation.

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

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

3. Run the following command to create the *esa-creds* secret, which is used by the Get ESA Policy container to access the ESA using the credentials of a *non-admin ESA user*.

```
kubectl create secret generic esa-creds --from-literal=esa_user=<ESA user> --from-literal=esa_pass=<ESA user password> --namespace <NAMESPACE>

kubectl create secret generic esa-creds --from-literal=esa_user=non-admin --from-literal=esa_pass=<ESA user password> --namespace iap
```

4. Run the following command to create a passphrase secret, which is used by the Get ESA Policy container to encrypt the policy.

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

This command is used to implement the Password-Based Encryption (PBE) for encrypting the policy.

You need to provide a passphrase and a salt as a Kubernetes secret to the Get ESA Policy container. 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.

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

5. If you want to use an Azure Storage Container for storing the policy package, instead of the Azure File Share, then modify the *credentials.json* file to update the credentials for the service principal 1.

The *credentials.json* file has the following format.

```
{
  "storage_account_name" : "AZURE_STORAGE_ACCOUNT_NAME",
  "tenant_id" : "AZURE_TENANT_ID",
  "client_id" : "AZURE_CLIENT_ID",
  "client_secret": "AZURE_CLIENT_SECRET"
}
```

For more information about the values for the *tenant_id*, *client_id*, and *client_secret* keys, refer to the section *Registering an Application*.

For more information about the structure of the *credentials.json* file, refer to the section *Credentials.json*.

6. If you want to use an Azure Storage Container for storing the policy package, instead of the Azure File Share, then run the following command to create the *create-obj-access* secret, which is used by the Get ESA Policy container to upload the immutable policy package to the Azure storage container.

```
kubectl create secret generic create-obj-access --from-file=get-esa-policy/credentials.json --namespace <NAMESPACE>

kubectl create secret generic create-obj-access --from-file=get-esa-policy/credentials.json --namespace iap
```

For more information about the *credentials.json* file, refer to the section *Credentials.json*.

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

For more information about the *storageAccessSecrets* parameters, refer to the section *Storage Access Secrets*.

7. If you want to use Azure File Share for storing the policy package, instead of the Azure Storage Container, then run the following command to create the *azure-secret* secret, which is used by the Get ESA Policy container to upload the immutable policy package to the Azure File Share in the Azure Storage Account.

```
kubectl create secret generic azure-secret --from-
literal=azurestorageaccountname='<Storage Account Name>' --from-
literal=azurestorageaccountkey='<Storage Account Key>' --namespace <Namespace>

kubectl create secret generic azure-
secret --from-literal=azurestorageaccountname='test' --from-
literal=azurestorageaccountkey='<Storage Account Key>' --namespace iap
```

In the *azurestorageaccountname* parameter, specify the value of the Azure Storage Account that you have created in *step 7* of the section *Creating an Azure File Share*.

In the *azurestorageaccountkey* parameter, specify the value of any one of the storage access keys for the Azure Storage Account, as specified in *step 9* of the section *Creating an Azure File Share*.

> **Note:** Ensure that you specify the name of the secret as the value of the *azureFile/secretName* field in the *pv.yaml* file in *step 8a*.

8. Perform the following steps if you want to use an Azure File Share for storing the policy dump instead of the Azure Storage Container.
   a. Create a file named *pv.yaml* for creating a persistent volume resource.

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

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: azurefile
  labels:
    target: policy
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  storageClassName: azurefile
  azureFile:
    secretName: azure-secret
    shareName: <AZURE_SHARE_NAME>
    readOnly: false
  mountOptions:
  - dir_mode=0744
  - file_mode=0644
  - uid=1000
  - gid=1000
    #- mfsymlinks
    #- nobrl
```

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

In the *storageClassName* parameter, ensure that you specify *azurefile*, for using the Azure File Share as the persistent volume.

In the *azureFile/secretName* parameter, ensure that you specify name of the secret that you have created in *step 6*.

For example, specify the value of the *azureFile/secretName* parameter as *azure-secret*.

In the *azureFile/shareName* parameter, ensure that you specify name of the Azure File Share that you have created in *step 10* of the section *Creating an Azure File Share*.

Ensure that the value specified in the *metadata/labels/target* is also specified as the value of the *selector/matchLabels/target* parameter in the *claim.yaml* file in *step 8c*.

You can specify the access control settings for the files and directories on the Azure File Share by using the *mountOptions* parameter.

For more information about the mount option parameters, refer to the section *Use Azure Files with Linux* in the Azure documentation.

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

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

A persistent volume resource is created.

c. Create a file named *claim.yaml* for creating a claim on the persistent volume that you have created in *step 8b*.

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

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: azurefile
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: azurefile
  resources:
    requests:
      storage: 1Gi
  selector:
    matchLabels:
      target: policy
```

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

In the *storageClassName* parameter, ensure that you specify *azurefile*, for using the Azure File Share as the persistent volume.

In the *selector/matchLabels/target* parameter, ensure that you specify the value that was defined in the *metadata/labels/target* parameter in the *pv.yaml* file in *step 8a*.

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

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

For example:

```
kubectl apply -f claim.yaml -n iap
```

A persistent volume claim is created.

9. On the Linux instance, replace the value of the *azureKeyId* parameter in the *encryption-config.yaml* file with the resource ID of the Azure key that you have created in the section *Creating an Azure Key*.

   The Azure key is used to encrypt the policy package.

   The *encryption-config.yaml* file is part of the Get Policy Helm chart package.

   For more information about the extracted Helm charts, refer to the *step 10* of the section *Initializing the Linux Instance*.

   > **Note:** This step is optional, and is required only if you want to encrypt the policy package using the Azure Key Vault.

10. Run the following command to create a ConfigMap for storing the Azure encryption configuration in YAML.

    ```
    kubectl create configmap encryption-config --from-file=<Path of the Azure encryption configuration file> --namespace <Namespace name>
    ```

    For example:

    ```
    kubectl create configmap encryption-config --from-file=get-esa-policy/encryption-config.yaml --namespace iap
    ```

    The ConfigMap is used to encrypt the policy package.

    Ensure that the name of the ConfigMap matches the value of the *encryptionConfigMap* parameter in the *values.yaml* file.

    > **Note:** This step is optional, and is required only if you want to encrypt the policy package using the Azure Key Vault.

11. On the Linux instance, navigate to the location where you have extracted the Helm charts for deploying the Get ESA Policy application.

    For more information about the extracted Helm charts, refer to the *step 10* of the section *Initializing the Linux Instance*.

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

    ```
    ...
    ..
    .

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

    ....
    ....
    ....

    # k8s secret for storing ESA credentials
    esaCredSecrets: ESA_CREDENTIAL_SECRET_NAME

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

    # encryption configmap name
    # create a configmap from the file 'encryption-config.yaml' as given below:
    # kubectl create configmap encryption-config --from-file=encryption-
    ```

```
config.yaml=encryption-config.yaml --namespace <SAME_AS_OF_CHART_DEPLOYMENT>
# Enable below if you want to enable envelope encryption.
encryptionConfigMap: ENCRYPTION_CONFIGMAP

## choose your storage destination. Currently we support <VolumeMount | ObjectStore>
securityConfigDestination: VolumeMount

## If securityConfigDestination is Volume Mount,
## specify the name of persistent volume claim to be used for this deployment.
pvcName: PVC_NAME

# k8s secret for storing credentials with write access to az storage
storageAccessSecrets: WRITE_ACCESS_CREDENTIAL_SECRET_NAME

policy:
  esaIP: ESA_IP

  ## absolute path in PV or ObjectStore where the policy dump file will be stored. <eg. /
test/xyz/policy >
  ## If destination is Object Store, make sure first name in path is bucket name i.e.
test will be our existing bucket
  filePath: ABSOLUTE_POLICY_PATH

  ## time (in mins) to wait till the policies get downloaded
  ## default is 15 (mins)
  timeout: 15
```

12. 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: Get ESA Policy Helm Chart Details*.

Field	Description
podSecurityContext	Specify the privilege and access control settings for the pod. The default values are set as follows: • *runAsUser* - 1000 • *runAsGroup* - 1000 • *fsGroup* - 1000 • *supplementalGroups* - 1000 **Note:** Ensure that at any given point of time, the value of at least one of the four 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.
esaCredSecrets	Specify the value of the secret that is used to store the ESA credentials. This is used by the Get ESA Policy container to retrieve the policy from the ESA.

Field	Description
pbeSecrets	Specify the value of the secret that is used to store the passphrase and salt. The Get ESA Policy container encrypts the policy using the key generated by using the passphrase and salt. **Important:** Ensure that the passphrase has a minimum length of 12 characters, with atleast 1 uppercase letter, 2 digits, 2 special characters, and 2 non-letters (such as, digits or special characters). The passphrase must also have a minimum value of entropybits as *30* and a minimum strength of *0.66*. Entropybits defines the randomness of the password and the strength defines the complexity of the password. For more information about the password strength, refer to the section *Password Strength*.
encryptionConfigMap	Specify the name of the ConfigMap that you have created in *step 7* to encrypt the policy package using Azure Key Vault. **Note:** This field is optional, and is required only if you want to encrypt the policy package using the Azure Key Vault. **Note:** By default, the Passphrase-Based Encryption is used to encrypt the policy package. If you want to use the Azure Key Vault for encrypting the policy package, then you must comment out the *pbeSecrets* entry in the *values.yaml* file or leave the value of the *pbeSecrets* parameter as blank.
securityConfigDestination	Specify one of the following options: • *VolumeMount* - Specify this value if you want to store the policy package on a volume that is mounted on the pod. Use this option if you are using Azure File Share for storing the policy package. • *ObjectStore* - Specify this value if you want to store the policy package in the Azure Storage Container. By default, the value is set to *VolumeMount*.
storageAccessSecrets	Specify the value of the secret that is used to store the credentials with write access to the Azure storage container. This is used by the Get ESA Policy container to upload the policy package to the Azure storage container. If you want to use the Azure File Share for storing the policy package, then leave the value of this field blank. **Important:** This field is required if you want to store the immutable policy package in the Azure Storage Container.

Field	Description
pvcName	Specify the name of the Persistent Volume Claim where you want to store the policy package. If you want to use the Azure Storage Container for storing the policy package, then leave the value of this field blank. **Important:** This field is required if you want to store the immutable policy package in the Azure File Share.
policy/esaIP	Specify the IP address of the ESA from where you want to download the policy.
policy/filePath	Specify the path in the persistent volume or the object store where you want to store the immutable policy package. For example, if you are using Azure File Share as the persistent volume, then you can specify the file path as */<Mount point>/xyz/policy*. In this case, an immutable policy package with the name as *policy* will be stored in the */<Mount point>/xyz* directory in the required persistent volume. If you want to store the policy package in an Object Store, such as an Azure Storage Container, then you can specify the container name as the first name in the *filePath* field. For example, you can specify the value of the *filePath* field, as *test/LOB/policy*, where *test* is the container name, *LOB* is the directory inside the container, and *policy* is the name of the immutable policy package. In this example, the container name is specified as the first name of the file path. **Important:** Use only uppercase and lowercase letters, numbers, dot, and underscore in the file path. Do not use special characters in the file path. **Note:** This value is case-sensitive.
policy/timeout	Specify the time for which the Get ESA Policy container tries to communicate with the ESA for fetching the policies, after which the connection times out. By default, this value is set to *15*. The unit of the timeout parameter is minutes.

13. Run the following command to deploy the Get ESA Policy application on the Kubernetes cluster.

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

    For example:

    ```
    helm install get-policy-esa --namespace iap get-esa-policy
    ```

14. Run the following command to check the deployment status.

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

    After the Get ESA Policy container is up, it retrieves the policy from the ESA and uploads it to the Azure storage container specified in the *values.yaml* file. After uploading the policy package, the Kubernetes job is completed.

2.8.2 Verifying the IAP Policy Package on the Azure Storage Container

The Get ESA Policy container retrieves the policy from the ESA and uploads a snapshot of this immutable policy to the Azure storage container. This section describes how to verify the policy package in the Azure environment.

> **Important:** This procedure is optional, and is required only if you want to use the Azure Storage Container storing the policy package, instead of using the Azure File Share.

➤ To verify the policy package on the Azure Storage Container:

Perform the following tasks to verify whether the immutable policy package has been uploaded to the Azure storage container.

1. Login to the Azure environment.
   For more information about logging in to the Azure environment, refer to the section *Logging in to the Azure Environment*.
2. Click the **Portal** menu icon ( ).
   The **Portal** menu appears.
3. Click **Storage accounts**.
   The **Storage accounts** screen appears.
4. Navigate to the path that you have specified as the value of the *filePath* parameter in the *values.yaml* file in *step 10* of the section *Deploying the Get ESA Policy Container on the Kubernetes Cluster*.
   The immutable policy package appears.
5. Click the immutable policy package to view additional details about the file.

2.8.3 Verifying the IAP Policy Package on the Azure File Share

The Get ESA Policy container retrieves the policy from the ESA and uploads a snapshot of this immutable policy to the Azure file share. This section describes how to verify the policy package in the Azure environment.

➤ To verify the policy package on the Azure File Share:

Perform the following tasks to verify whether the policy package file has been uploaded to the Azure file share.

1. Login to the Azure environment.
   For more information about logging in to the Azure environment, refer to the section *Logging in to the Azure Environment*.
2. Click the **Portal** menu icon ( ).
   The **Portal** menu appears.
3. Click **Storage accounts**.
   The **Storage accounts** screen appears.
4. Navigate to **File service** > **File shares**.
   The **File shares** screen appears.

5. Navigate to the path that you have specified as the value of the *filePath* parameter in the *values.yaml* file in *step 10* of the section *Deploying the Get ESA Policy Container on the Kubernetes Cluster*.

   The immutable policy package appears.
6. Click the immutable policy package to view additional details about the file.

2.8.4 Deploying the Sample Application Container on the Kubernetes Cluster

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

➤ To deploy a release on the Kubernetes cluster:

1. On the Linux instance, run the following command to create the namespace required for Helm deployment.
   ```
   kubectl create namespace <Namespace name>
   ```
   For example:
   ```
   kubectl create namespace iap
   ```
2. Perform the following steps if you want to use NGINX as the Ingress Controller.

   > **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 <NGINX
   Namespace name> --set controller.replicaCount=2 ingress-
   nginx/ingress-nginx --set controller.publishService.enabled=true
   --set controller.ingressClassResource.name=nginx-iap --
   set controller.service.annotations."service\.beta\.kubernetes\.io/azure-load-
   balancer-internal"="true"
   ```
   For example:
   ```
   helm install nginx-ingress --namespace nginx --set controller.replicaCount=2
   ingress-nginx/ingress-nginx --set controller.publishService.enabled=true
   --set controller.ingressClassResource.name=nginx-iap --
   set controller.service.annotations."service\.beta\.kubernetes\.io/azure-load-
   balancer-internal"="true"
   ```
   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 authenticate the communication between the Kubernetes cluster and the Azure Container Registry by using your existing Docker credentials, then perform the following steps.

   a. Run the following command to login to Docker.

   ```
   docker login
   ```

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

   b. Run the following command to specify a secret for pulling the Init and Sample Application container images from the Azure Container Registry to Kubernetes.

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

   For example:

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

   For more information about creating secrets, refer to the section *Pull an Image from a Private Registry* in the Kubernetes documentation.

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

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

4. If you want to use an Azure Storage Container for storing the policy package, instead of the Azure File Share, then modify the *credentials.json* file to update the credentials for the service principal 2.

   The following snippet contains the format of the *credentials.json* file.

   ```
   {
     "storage_account_name" : "AZURE_STORAGE_ACCOUNT_NAME",
     "tenant_id" : "AZURE_TENANT_ID",
     "client_id" : "AZURE_CLIENT_ID",
     "client_secret": "AZURE_CLIENT_SECRET"
   }
   ```

   For more information about the values for the *tenant_id*, *client_id*, and *client_secret* keys, refer to the section *Registering an Application*.

   For more information about the structure of the *credentials.json* file, refer to the section *Credentials.json*.

5. If you want to use an Azure Storage Container for storing the policy package, instead of the Azure File Share, then run the following command to create the *read-obj-access* secret, which is used by the Sample Application container to download the immutable policy package from the Azure storage container to the shared memory.

   ```
   kubectl create secret generic read-obj-access --from-file=spring-apjava/
   credentials.json --namespace <NAMESPACE>
   ```

   ```
   kubectl create secret generic read-obj-access --from-file=spring-apjava/
   credentials.json --namespace iap
   ```

For more information about the *credentials.json* file, refer to the section *Credentials.json*.

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

For more information about the *storageAccessSecrets* parameters, refer to the section *Storage Access Secrets*.

6. On the Linux instance, replace the value of the *azureKeyId* parameter in the *decryption-config.yaml* file with the resource ID of the Azure key that you have created in the section *Creating an Azure Key*.

   The Azure key is used to decrypt the policy package.

   The *decryption-config.yaml* file is part of the Helm chart package.

   For more information about the extracted Helm charts, refer to the *step 11* of the section *Initializing the Linux Instance*.

   > **Note:** This step is optional, and is required only if you want to encrypt the policy package using the Azure Key Vault.

7. Run the following command to create a ConfigMap map for storing the Azure decryption configuration in YAML:

   ```
   kubectl create configmap decryption-config --from-file=<Path of the Azure decryption configuration file> --namespace <Namespace name>
   ```

   For example:

   ```
   kubectl create configmap decryption-config --from-file=spring-apjava/decryption-config.yaml --namespace dsg
   ```

   The ConfigMap is used to decrypt the policy package.

   Ensure that the name of the ConfigMap matches the value of the *decryptionConfigMap* parameter in the *values.yaml* file.

   > **Note:** This step is optional, and is required only if you want to encrypt the policy package using the Azure Key Vault.

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

   For more information about the extracted Helm charts, refer to the *step 11* of the section *Initializing the Linux Instance*.

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

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

# If Volume Mount, name of persistent volume claim to be used for this deployment.
pvcName: PVC_NAME

# k8s secret for storing credentials with read access to az storage
storageAccessSecrets: READ_ACCESS_CREDENTIAL_SECRET_NAME
```

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

# create a configmap from the file 'decryption-config.yaml' as given below:
# kubectl create configmap decryption-config --from-file=decryption-
config.yaml=decryption-config.yaml --namespace <SAME_AS_OF_CHART_DEPLOYMENT>
# applicable if policy dump was encrypted using envelope encryption.
decryptionConfigMap: DECRYPTION_CONFIGMAP

## start with blue deployment first
deploymentInUse: blue

## Policy metadata for Blue Deployments
blueDeployment:
  enabled: true
  # Choose your storage source. Currently we support <VolumeMount | ObjectStore>
  securityConfigSource: VolumeMount
  policy:
    # absolute path in PV or ObjectStore where the policy dump file will be stored. <eg. /
test/xyz/policy >
    # If Object Store make sure first name in path is bucket name i.e. /test will be our
existing bucket
    filePath: ABSOULTE_POLICY_PATH

## Policy metadata for Green Deployments
greenDeployment:
  enabled: false
  # Choose your storage source. Currently we support <VolumeMount | ObjectStore>
  securityConfigSource: VolumeMount
   policy:
    # absolute path in PV or ObjectStore where the policy dump file will be stored. <eg. /
test/xyz/policy >
    # If Object Store make sure first name in path is bucket name i.e. /test will be our
existing bucket
    filePath: ABSOULTE_POLICY_PATH

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

...
..
.


## specify the ports exposed in your springapp configurations where,
## name - distinguishes between different ports.
## port - the port on which you wan't to expose the service externally.
## targetPort - the port no. configured while creating Tunnel.
springappService:
    name: "restapi"
    port: 8080
    targetPort: 8080

## Ingress enables testing of Green (V2/Staging) Deployments alongside Blue ones
simultaneously.
## This is realized by exposing both deployments on the same External IP but on different
URL Paths.
## If you don't want expose both the deployments at the same time, keep 'ingress.enabled'
as false.
## This means that only Blue deployment gets exposed on the Load Balancer's External IP
via prod service.
ingress:
  enabled: true

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

  ## specify external Ingress Controller if any,
  ## else keep the 'ingress.class' field empty
  ## to use cloud native Ingress Controller.
  annotations:
```

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

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

  ## specify the host names and paths for Ingress rules
  ## prod and staging http paths can be used as optional fields.
  hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8080
        prod: "/"
    ## Add host section with the hostname used as CN while creating server certificates.
    ## For accessing the staging end-points also, we need hostname as that of CN in
server certs.
    ## While creating the certificates you can use *.example.com as CN for the below
example
    # - host: "prod.example.com"
    #   httpPaths:
    #   - port: 8080
    #     prod: "/"
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8080
    #     staging: "/"
    #
  ## If TLS is terminated on the Ingress Controller Load Balancer,
  ## then uncomment the tls section below and,
  ## provide K8s TLS Secret containing the certificate and key.
  # tls:
  #  - secretName: example-tls
  #    hosts:
  #      - prod.example.com
```

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 B: Sample Application Helm Chart Details*.

Field	Description
podSecurityContext	Specify the privilege and access control settings for the pod. The default values are set as follows: • *runAsUser* - 1000 • *runAsGroup* - 1000 • *fsGroup* - 1000 • *supplementalGroups* - 1000 **Note:** Ensure that at any given point of time, the value of at least one of the four parameters must be *1000*. This ensures that the pod has the required permissions to access the *pty* directories in the containers.

Field	Description
	For more information about the Pod Security Context, refer to the section *Configure a Security Context for a Pod or Container* in the Kubernetes documentation. For more information about the parameters, refer to the section *PodSecurityContext v1 Core* in the Kubernetes API documentation.
pvcName	Specify the name of the Persistent Volume Claim where you want to store the policy package. If you want to use the Azure Storage Container for storing the policy package, then leave the value of this field blank. **Important:** This field is required if you want to store the immutable policy package in the Azure File Share.
storageAccessSecrets	Specify the value of the secret that is used to store the credentials with read access to the Azure storage container. This is used by the Sample Application container to download the policy package from the Azure storage container to the shared memory. If you want to use the Azure File Share for storing the policy package, then leave the value of this field blank. **Important:** This field is required if you want to store the immutable policy package in the Azure Storage Container.
pbeSecrets	Specify the Kubernetes secret that contains the passphrase and the salt that were used to generate the key, which encrypted the policy. The Sample Application container uses the value specified in the *pbeSecrets* parameter to decrypt the policy. **Important:** Ensure that the passphrase has a minimum length of 12 characters, with atleast 1 uppercase letter, 2 digits, 2 special characters, and 2 non-letters (such as, digits or special characters). The passphrase must also have a minimum value of entropybits as *30* and a minimum strength of *0.66*. Entropybits defines the randomness of the password and the strength defines the complexity of the password. For more information about the password strength, refer to the section *Password Strength*.
decryptionConfigMap	Specify the value as *decryption-config*. This indicates the name of the ConfigMap that will be created from the *decryption-config.yaml* file using the `kubectl` command. The *decryption-config.yaml*

Field	Description
	file includes the *azureKeyId*, which contains the resource ID of the Azure key. **Note:** This field is optional, and is required only if you want to encrypt the policy package using the Azure Key Vault. **Note:** By default, the Passphrase-Based Encryption is used to encrypt the policy package. If you want to use the Azure Key Vault for encrypting the policy package, then you must comment out the *pbeSecrets* entry in the *values.yaml* file or leave the value of the *pbeSecrets* parameter as blank.
deploymentInUse	Specify the value as *blue*. This indicates that the *blue* deployment strategy is in use.
blueDeployment/enabled	Specify the value as *true* to enable the *blue* deployment strategy.
blueDeployment/securityConfigSource	Specify one of the following options: • *VolumeMount* - Specify this value if you want to store the policy package on a volume that is mounted on the pod. Use this option if you are using Azure File Share for storing the policy package. • *ObjectStore* - Specify this value if you want to store the policy package in the Azure Storage Container. By default, the value is set to *VolumeMount*.
blueDeployment/policy/filePath	Specify the path in the persistent volume or the object store from where you want to retrieve the immutable policy package. Ensure that you specify the same value that you have specified as the value of the *filePath* parameter in the *values.yaml* file for the Get ESA Policy container in *step 9* of the section *Deploying the Get ESA Policy Container on the Kubernetes Cluster*. **Important:** Use only uppercase and lowercase letters, numbers, dot, and underscore in the file path. Do not use special characters in the file path. **Note:** This value is case-sensitive.
greenDeployment/enabled	Specify the value as *false*. While deploying Release 1, the *green* deployment strategy is always disabled.
springappService/name	Specify a name for the tunnel to distinguish between ports.
springappService/port	Specify the port number on which you want to expose the Kubernetes service externally.
springappService/targetPort	Specify the port on which the Sample application is running inside the Docker container.

Field	Description
ingress/enabled	Specify this value as *true* to enable the ingress resource in Kubernetes. The ingress resource acts as a load balancer. It enables the simultaneous deployment of *blue* and *green* strategy by exposing both the deployments on the same external IP address, but on different URL paths.
ingress/ingressClassName	Specify the value of the external ingress controller, if any. For example, specify, *nginx-iap*, if you want to use the NGINX Ingress Controller.
nginx.ingress.kubernetes.io/auth-tls-verify-client	Set the value of this field to on, if you want to enable TLS mutual authentication between the REST API client and the Azure load balancer. By default, this field is *disabled*.
nginx.ingress.kubernetes.io/auth-tls-secret	Specify a value for the CA secret, if you have enabled the TLS mutual authentication between the REST API client and the Azure load balancer. By default, this field is *disabled*. For more information about the value of this field, refer to the section *Ingress Configuration*.
ingress/hosts/host	Specify the hostname of the ingress resource. If you are using the external IP address of the ingress resource for performing the security operations, then leave this field blank.
ingress/hosts/httpPaths/port	Specify the port on which the Sample application is running inside the Docker container.
ingress/hosts/httpPaths/prod	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	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 Sample Application container on the Kubernetes cluster.

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

    For example:

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

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
    ```

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

kubectl get ingress -n <namespace>

For example:

kubectl get ingress -n iap

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

2.8.5 Upgrading the Helm Charts

This section describes the steps for upgrading the Helm charts.

> **Note:** Protegrity recommends using the *Blue-Green* method of upgrading the Helm charts. In this method, you use two modes of deploying the application, production mode and staging mode. In the staging mode, you can test the changes to the application in a staging environment, without impacting the production environment. After you have tested the application in the staging environment, you can switch the environments by making the staging environment as the new production environment and the production environment as the new staging environment.
>
> In this way, you can seamless divert the customer traffic to the modified application without any production downtime. After your modified application is running on the new production environment, you can shutdown the application that is running on the staging environment. In the *Blue-Green* method of deploying applications, only one Release is active at a given point of time.

➤ To upgrade the Helm charts:

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

   For more information about the extracted Helm charts, refer to the *step 11* of the section *Initializing the Linux Instance*.

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

```
...
..
## set the Pod's security context object
## leave the field empty if not applicable
podSecurityContext:
  runAsUser: 1000
  runAsGroup: 1000
  fsGroup: 1000
  supplementalGroups: 1000
...
...
...
# If Volume Mount, name of persistent volume claim to be used for this deployment.
pvcName: PVC_NAME

# k8s secret for storing credentials with read access to az storage
storageAccessSecrets: READ_ACCESS_CREDENTIAL_SECRET_NAME

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

# create a configmap from the file 'decryption-config.yaml' as given below:
# kubectl create configmap decryption-config --from-file=decryption-
config.yaml=decryption-config.yaml --namespace <SAME_AS_OF_CHART_DEPLOYMENT>
# applicable if policy dump was encrypted using envelope encryption.
decryptionConfigMap: DECRYPTION_CONFIGMAP

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

```
## Policy metadata for Blue Deployments
blueDeployment:
  enabled: true
  # Choose your storage source. Currently we support <VolumeMount | ObjectStore>
  securityConfigSource: VolumeMount
  policy:
    # absolute path in PV or ObjectStore where the policy dump file will be stored. <eg. /
test/xyz/policy >
    # If Object Store make sure first name in path is bucket name i.e. /test will be our
existing bucket
    filePath: ABSOULTE_POLICY_PATH

## Policy metadata for Green Deployments
greenDeployment:
  enabled: false
  # Choose your storage source. Currently we support <VolumeMount | ObjectStore>
  securityConfigSource: VolumeMount
  policy:
    # absolute path in PV or ObjectStore where the policy dump file will be stored. <eg. /
test/xyz/policy >
    # If Object Store make sure first name in path is bucket name i.e. /test will be our
existing bucket
    filePath: ABSOULTE_POLICY_PATH

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

...
..
.


## specify the ports exposed in your springapp configurations where,
## name - distinguishes between different ports.
## port - the port on which you wan't to expose the service externally.
## targetPort - the port no. configured while creating Tunnel.
springappService:
    name: "restapi"
    port: 8080
    targetPort: 8080

## Ingress enables testing of Green (V2/Staging) Deployments alongside Blue ones
simultaneously.
## This is realized by exposing both deployments on the same External IP but on different
URL Paths.
## If you don't want expose both the deployments at the same time, keep 'ingress.enabled'
as false.
## This means that only Blue deployment gets exposed on the Load Balancer's External IP
via prod service.
ingress:
  enabled: true

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

  ## specify external Ingress Controller if any,
  ## else keep the 'ingress.class' field empty
  ## to use cloud native Ingress Controller.
  annotations:
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"

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

  ## specify the host names and paths for Ingress rules
  ## prod and staging http paths can be used as optional fields.
  hosts:
```

```
    - host: "prod.example.com"
      httpPaths:
      - port: 8080
        prod: "/"
    ## Add host section with the hostname used as CN while creating server certificates.
    ## For accessing the staging end-points also, we need hostname as that of CN in
server certs.
    ## While creating the certificates you can use *.example.com as CN for the below
example
    # - host: "prod.example.com"
    #   httpPaths:
    #   - port: 8080
    #     prod: "/"
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8080
    #     staging: "/"
    #
  ## If TLS is terminated on the Ingress Controller Load Balancer,
  ## then uncomment the tls section below and,
  ## provide K8s TLS Secret containing the certificate and key.
  # tls:
  #  - secretName: example-tls
  #    hosts:
  #      - prod.example.com
```

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

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

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

Field	Description
podSecurityContext	Specify the privilege and access control settings for the pod. The default values are set as follows: • *runAsUser* - 1000 • *runAsGroup* - 1000 • *fsGroup* - 1000 • *supplementalGroups* - 1000 **Note:** Ensure that at any given point of time, the value of at least one of the four parameters must be *1000*. This ensures that the pod has the required permissions to access the *pty* directories in the containers. For more information about the Pod Security Context, refer to the section *Configure a Security Context for a Pod or Container* in the Kubernetes documentation. For more information about the parameters, refer to the section *PodSecurityContext v1 Core* in the Kubernetes API documentation.
pvcName	Specify the name of the Persistent Volume Claim where you want to store the policy package.

Field	Description
	If you want to use the Azure Storage Container for storing the policy package, then leave the value of this field blank. **Important:** This field is required if you want to store the immutable policy package in the Azure File Share.
storageAccessSecrets	Specify the value of the secret that is used to store the credentials with read access to the Azure storage container. This is used by the Sample Application container to download the policy package from the Azure storage container to the shared memory. If you want to use the Azure File Share for storing the policy package, then leave the value of this field blank. **Important:** This field is required if you want to store the immutable policy package in the Azure Storage Container.
pbeSecrets	Specify the Kubernetes secret that contains the passphrase and the salt that were used to generate the key, which encrypted the policy. The Sample Application container uses the value specified in the *pbeSecrets* parameter to decrypt the policy.
decryptionConfigMap	Specify the value as *decryption-config*. This indicates the name of the ConfigMap that will be created from the *decryption-config.yaml* file using the `kubectl` command. The *decryption-config.yaml* file includes the *azureKeyId*, which contains the resource ID of the Azure key. **Note:** This field is optional, and is required only if you want to encrypt the policy package using the Azure Key Vault. **Note:** By default, the Passphrase-Based Encryption is used to encrypt the policy package. If you want to use the Azure Key Vault for encrypting the policy package, then you must comment out the *pbeSecrets* entry in the *values.yaml* file or leave the value of the *pbeSecrets* as blank.
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 want to store the policy package on a volume that is mounted on the pod. Use this option if you are using Azure File Share for storing the policy package. • *ObjectStore* - Specify this value if you want to store the policy package in the Azure Storage Container. By default, the value is set to *VolumeMount*.

Field	Description
greenDeployment/policy/filePath	Specify the path in the persistent volume or the object store from where you want to retrieve the immutable policy package. Ensure that you specify the same value that you have specified as the value of the *filePath* parameter in the *values.yaml* file for the Get ESA Policy container in *step 9* of the section *Deploying the Get ESA Policy Container on the Kubernetes Cluster*. **Important:** Use only uppercase and lowercase letters, numbers, dot, and underscore in the file path. Do not use special characters in the file path. **Note:** This value is case-sensitive.
springappService/name	Specify a name for the tunnel to distinguish between ports.
springappService/port	Specify the port number on which you want to expose the Kubernetes service externally.
springappService/targetPort	Specify the port on which the Sample application is running inside the Docker container.
ingress/enabled	Specify this value as *true* to enable the ingress resource in Kubernetes. The ingress resource acts as a load balancer. It enables the simultaneous deployment of *blue* and *green* strategy by exposing both the deployments on the same external IP address, but on different URL paths.
ingress/ingressClassName	Specify the value of the external ingress controller, if any. For example, specify, *nginx-iap*, if you want to use the NGINX Ingress Controller.
nginx.ingress.kubernetes.io/auth-tls-verify-client	Set the value of this field to on, if you want to enable TLS mutual authentication between the REST API client and the Azure load balancer. By default, this field is *disabled*.
nginx.ingress.kubernetes.io/auth-tls-secret	Specify a value for the CA secret, if you have enabled the TLS mutual authentication between the REST API client and the Azure load balancer. By default, this field is *disabled*. For more information about the value of this field, refer to the section *Ingress Configuration*.
ingress/hosts/host	Specify the hostname of the ingress resource.
ingress/hosts/httpPaths/port	Specify the port on which the Sample application is running inside the Docker container.
ingress/hosts/httpPaths/prod	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.

Field	Description
ingress/hosts/httpPaths/staging	Specify the value of this field as /. This ensures that the data traffic from the REST API client is redirected to the required production or staging URL based on the hostname value.

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

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

   For example:

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

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

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

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

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

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

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

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

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

   For example:

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

   Using this configuration ensures that the Sample application is deployed with the updated policy package 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 be shutdown the all the pods that were deployed as part of the *blue* deployment.

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

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

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

   For example:

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

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

If you want to modify any policy or subsequent releases, then you need to repeat *step 1* to *step 8*. The only difference is that you need to toggle the value of the *deploymentInUse* field from *green* to *blue* and vice versa depending on which deployment contains your modified configurations.

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.

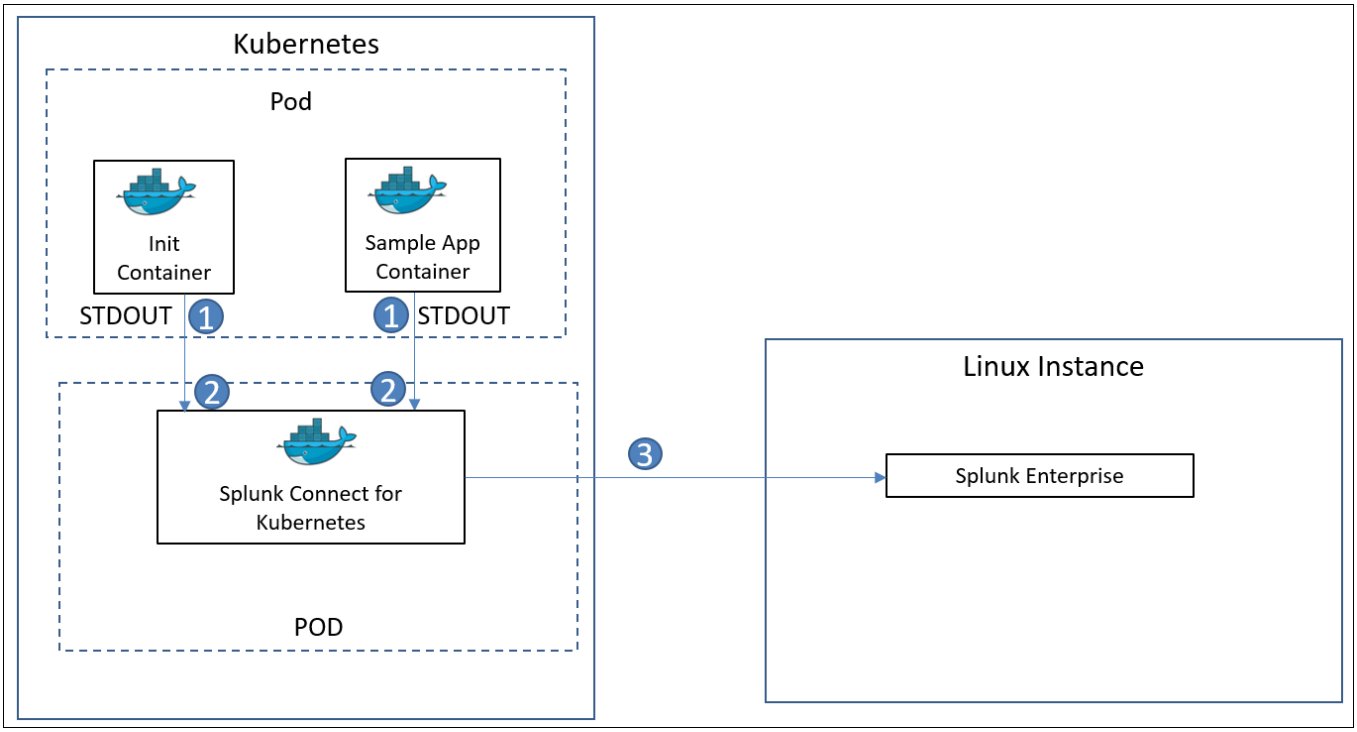


Figure 3-1: Sample Logging Workflow

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

1. The Init and Sample Application container write the logs to the Standard Output (STDOUT) stream.
2. The Splunk Connect for Kubernetes is used to collect the logs from the STDOUT stream.
   For more information about Splunk Connect for Kubernetes, refer to the *Splunk Connect for Kuberntes* page on Github.
3. The Splunk Connect for Kubernetes then forwards the logs to the Splunk Enterprise, which is used to view the logs.

3.2 Setting Up Splunk

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

➤ To set up Splunk:

1. Create a Linux instance, either in an on-premise or on a Cloud environment.
2. Install the latest version of Splunk Enterprise on the Linux instance.
   By default, Splunk is installed in the */opt/splunk* directory.

   For more information about installing Splunk Enterprise on a Linux instance, refer to the section *Install Splunk Enterprise* in the Splunk documentation.
3. Navigate to the */opt/splunk/bin* directory and start Splunk using the following command.
   ```
   ./splunk start --accept license
   ```
   You are prompted to create a username that will be used to login to Splunk Enterprise.

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

4. Type the username for the administrator account for logging into Splunk Enterprise, and then press **Enter**.
   You are prompted to create a password for the created user.

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

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

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


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

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

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

For example, if the IP address of your Linux instance is *10.xx.xx.xx*, then you can access Splunk Web at *http://10.xx.xx.xx:8000*.

3.3 Configuring Splunk

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

To configure Splunk Enterprise:

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

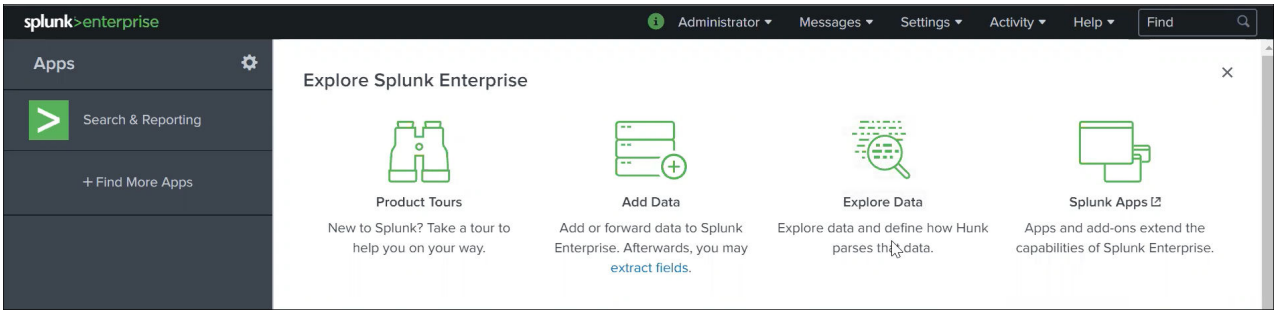


Figure 3-2: Splunk Enterprise Web UI

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.

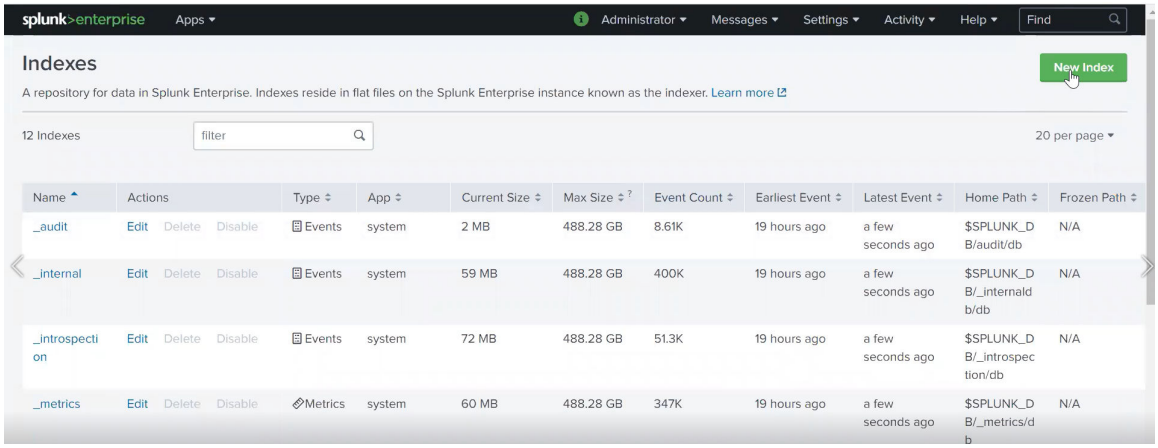


Figure 3-3: Indexes Screen

   b. Click **New Index**.
   The **New Index** dialog box appears.

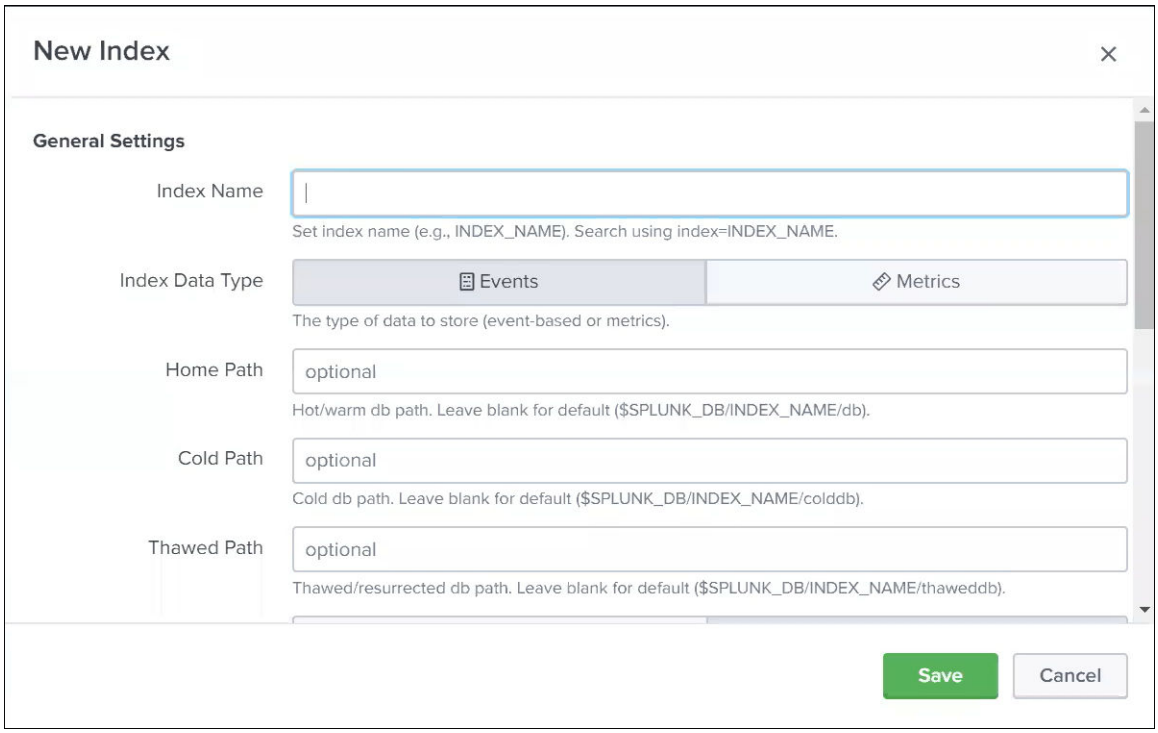


Figure 3-4: New Index Dialog Box

   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 Sample Application and Container logs.

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

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

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

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

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

   The **Data inputs** screen appears.

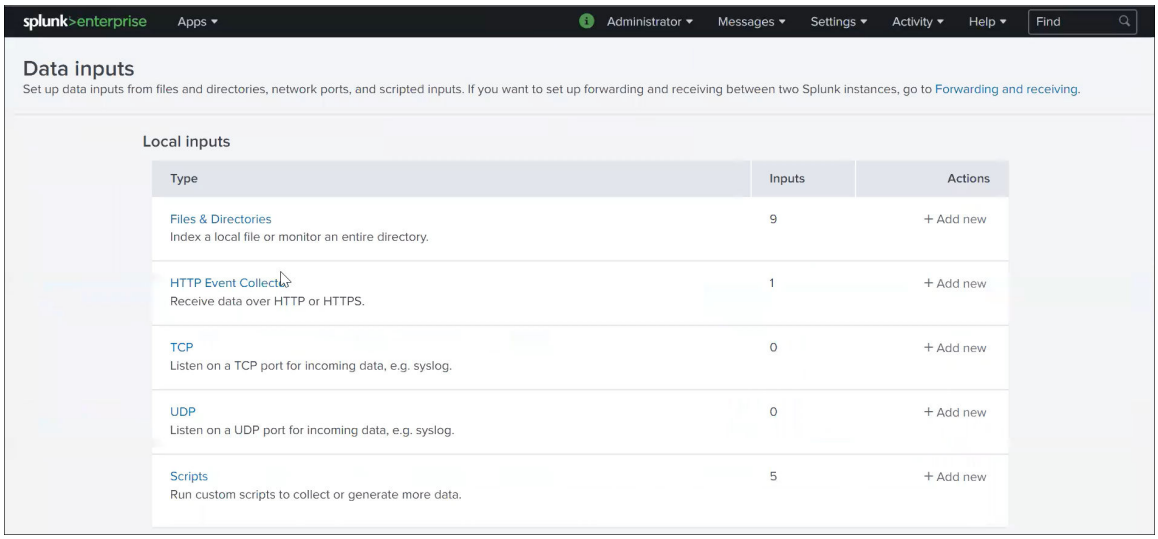


Figure 3-5: Data inputs Screen

   b. Click **HTTP Event Collector**.

   The **HTTP Event Collector** screen appears.

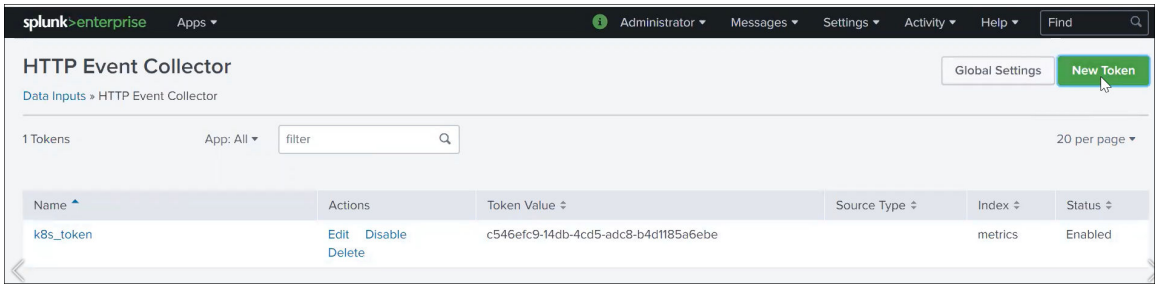


Figure 3-6: HTTP Event Collector Screen

   c. Click **New Token**.

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

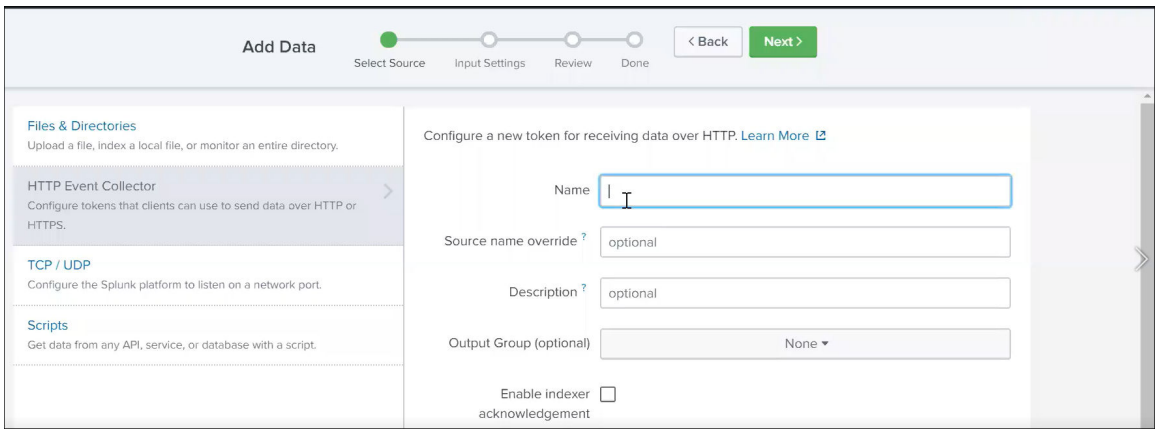


Figure 3-7: Select Source Screen

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.

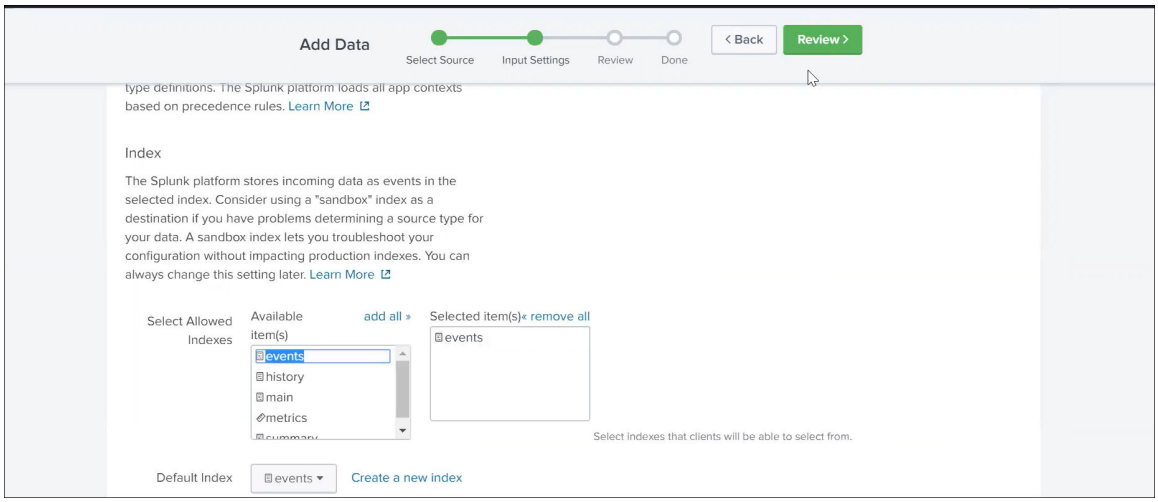


Figure 3-8: Input Settings Screen

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.

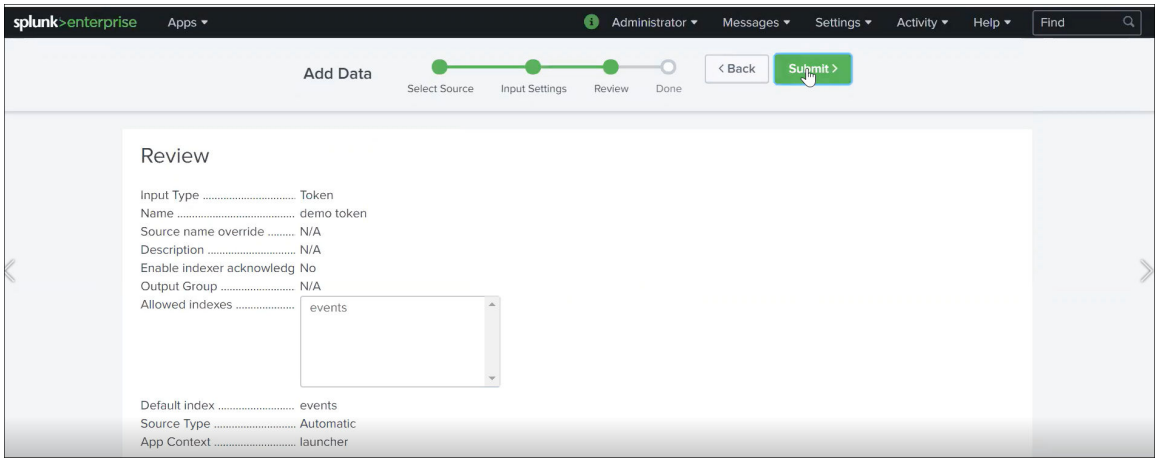


Figure 3-9: Review Screen

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.

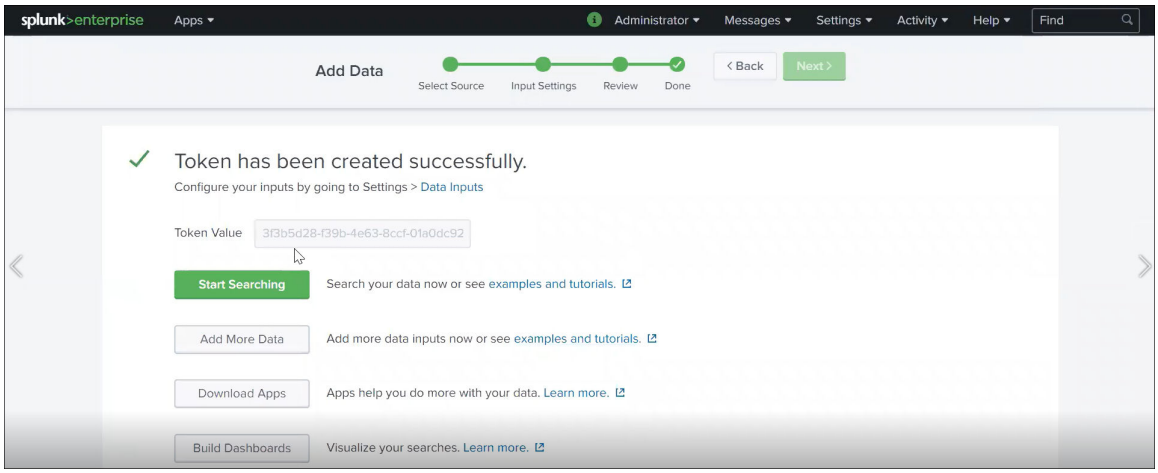


Figure 3-10: Done Screen

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.

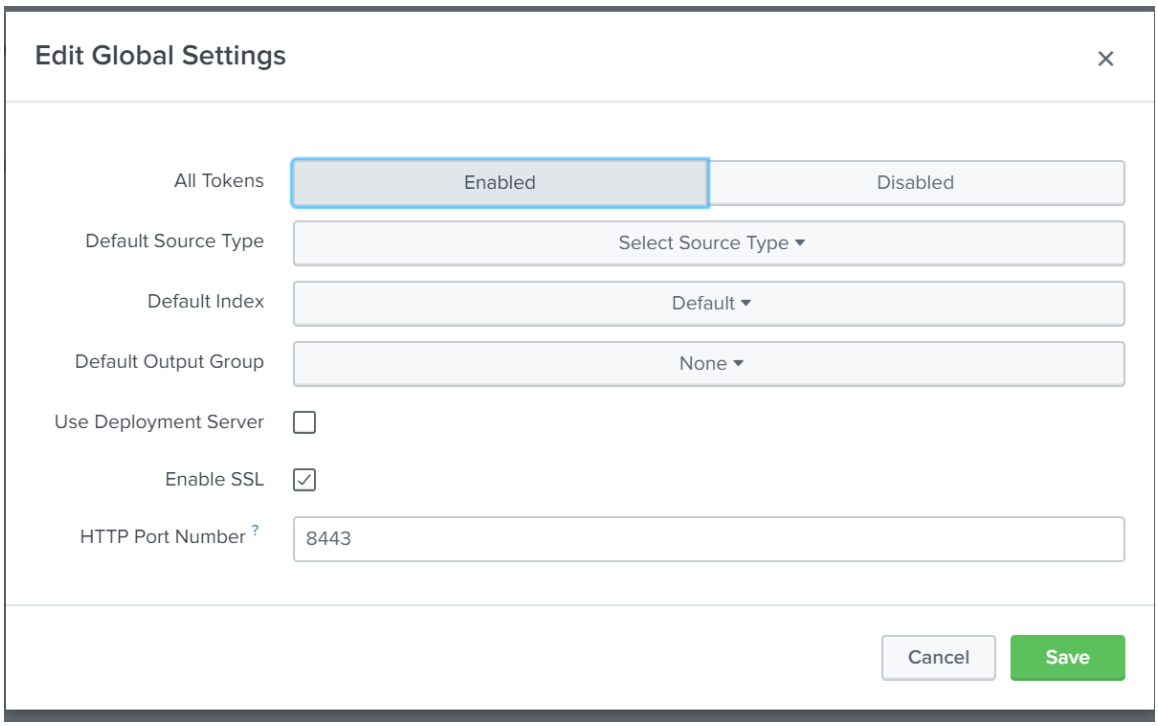


Figure 3-11: Edit Global Settings Dialog Box

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 Sample Application container.

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

➤ To deploy the Splunk Connect for Kubernetes Helm chart:

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

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

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

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

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

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

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

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

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

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

Note: In *step 4l* of the section *Applying an Azure Policy*, the *kube-system* namespace has not been assigned the Azure Policy by specifying the namespace in the **Namespace** exclusions field.

If you use a different namespace, instead of the *kube-system* namespace, then you must exclude it from the Azure Policy by specifying the new namespace in the **Namespace** exclusions field.

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 Sample Application Container Deployment

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

4.1 Running Security Operations

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

To run security operations:

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

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

> **Note:** The external IP address is the IP address of the ingress object that you obtain after you run the *kubectl get ingress* command after deploying the application on the Kubernetes cluster.
>
> For more information about the external IP address associated with the ingress object, refer to *step 9b* of the section *Deploying the Sample Application Container on the Kubernetes Cluster*.

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

If you want to protect data that contains Comma Separated Values (CSV), then you can run the following command to send a request to the */protect/csv* endpoint.

```
curl --location --request POST 'http://<External IP of Ingress>/
protect/csv' --header 'Content-Type: application/json' --header 'Host:
prod.example.com' --header 'Content-Type: application/json' --data-raw
'{ "dataElementMapping": { "1": "Alphanum", "3": "Alpha" }, "policyUser": "user1",
```

```
"input": [ "1,AP120457684,feebpeg@uta.mc", "2,AP120457685,fa@moftebjar.lv",
"3,AP120457686,irtod@okirivo.tr" ] }'
```

In this example, the *dataElementMapping* parameter is used to specify the data elements for protecting specific columns in the input CSV data. For example, the *Alphanum* data element is used to protect the first column of the CSV data and the *Alpha* data element is used to protect the third column of the CSV data. In this example, the second column of the CSV data is not being protected.

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

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

If you want to unprotect CSV data, then you can run the following command to send a request to the */protect/csv* endpoint.

```
curl --location --request POST 'http://<External IP of Ingress>/
unprotect/csv' --header 'Content-Type: application/json' --header 'Host:
prod.example.com' --header 'Content-Type: application/json' --data-raw
'{ "dataElementMapping": { "1": "Alphanum", "3": "Alpha" }, "policyUser":
"user1", "input": ["Q,AP120457684,ZMdFbIE@TIB.wI", "2,AP120457685,fx@boaNDTKBw.LO",
"M,AP120457686,ZYtcV@acIpwDr.Rz"] }'
```

In this example, the *dataElementMapping* parameter is used to specify the data elements for unprotecting specific columns in the input CSV data. For example, the *Alphanum* data element is used to unprotect the first column of the CSV data and the *Alpha* data element is used to unprotect the third column of the CSV data.

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

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

2. If you are using TLS authentication between the Sample Application instance and the REST API client, then send the following CURL request from the Linux instance.

```
curl --location --request POST 'https://prod.example.com/protect' --connect-to
"prod.example.com:443:<External IP of Ingress>:443" --header 'Content-Type:
application/json' --data-raw '{ "dataElement": "Alphanum", "policyUser": "user1",
"input": [ "SampleAppJava", "OnKubernetes" ] }' --cacert ./iap-ca.crt --cert ./iap-
client.crt --key ./iap-client.key
```

> **Note:** The external IP address is the IP address of the ingress object that you obtain after you run the *kubectl get ingress* command after deploying the application on the Kubernetes cluster.
>
> For more information about the external IP address associated with the ingress object, refer to *step 11* of the section *Deploying the Sample Application Container on the Kubernetes Cluster*.

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

If you want to protect data that contains Comma Separated Values (CSV), then you can run the following command to send a request to the */protect/csv* endpoint.

```
curl --location --request POST 'https://prod.example.com/protect/csv' --connect-
to "prod.example.com:443:<External IP of Ingress>:443" --header 'Content-Type:
application/json' --data-raw '{ "dataElementMapping": { "1": "Alphanum", "3":
"Alpha" }, "policyUser": "user1", "input": [ "1,AP120457684,feebpeg@uta.mc",
```

```
"2,AP120457685,fa@moftebjar.lv", "3,AP120457686,irtod@okirivo.tr" ] }' --cacert ./
iap-ca.crt --cert ./iap-client.crt --key ./iap-client.key
```

In this example, the *dataElementMapping* parameter is used to specify the data elements for protecting specific columns in the input CSV data. For example, the *Alphanum* data element is used to protect the first column of the CSV data and the *Alpha* data element is used to protect the third column of the CSV data. In this example, the second column of the CSV data is not being protected.

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

```
curl --location --request POST 'https://prod.example.com/unprotect' --connect-
to "prod.example.com:443:<External IP of Ingress>:443" --header 'Content-Type:
application/json' --data-raw '{ "dataElement": "Alphanum", "policyUser": "user1",
"input": [ "iaDDNBdH6EI8U", "9jB7cRSuk98B" ] }' --cacert ./iap-ca.crt --cert ./iap-
client.crt --key ./iap-client.key
```

If you want to unprotect CSV data, then you can run the following command to send a request to the */protect/csv* endpoint.

```
curl --location --request POST 'https://prod.example.com/unprotect' --connect-
to "prod.example.com:443:<External IP of Ingress>:443" --header 'Content-Type:
application/json' --data-raw '{ "dataElementMapping": { "1": "Alphanum", "3":
"Alpha" }, "policyUser": "user1", "input": ["Q,AP120457684,ZMdFbIE@TIB.wI",
"2,AP120457685,fx@boaNDTKBw.LO", "M,AP120457686,ZYtcV@acIpwDr.Rz"] }' --cacert ./
iap-ca.crt --cert ./iap-client.crt --key ./iap-client.key
```

In this example, the *dataElementMapping* parameter is used to specify the data elements for unprotecting specific columns in the input CSV data. For example, the *Alphanum* data element is used to unprotect the first column of the CSV data and the *Alpha* data element is used to unprotect the third column of the CSV data.

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

```
curl --location --request POST 'https://prod.example.com/reprotect' --connect-
to "prod.example.com:443:<External IP of Ingress>:443" --header 'Content-Type:
application/json' --data-raw '{ "dataElement": "Alphanum", "newDataElement":
"Alphanum1", "policyUser": "user1", "input": [ "iaDDNBdH6EI8U", "9jB7cRSuk98B" ] }'
--cacert ./iap-ca.crt --cert ./iap-client.crt --key ./iap-client.key
```

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

4.2 Viewing Logs in Splunk

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

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

➤ To view logs:

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

   The Splunk Enterprise screen appears.

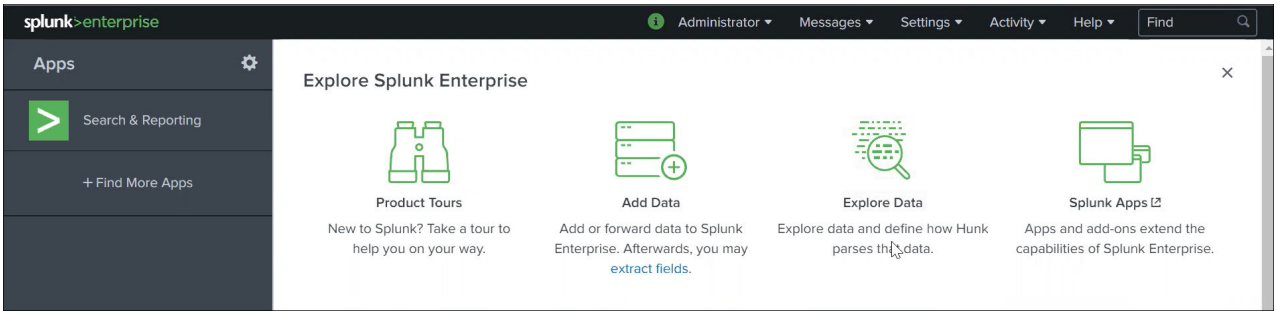


Figure 4-1: Splunk Enterprise Web UI

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.

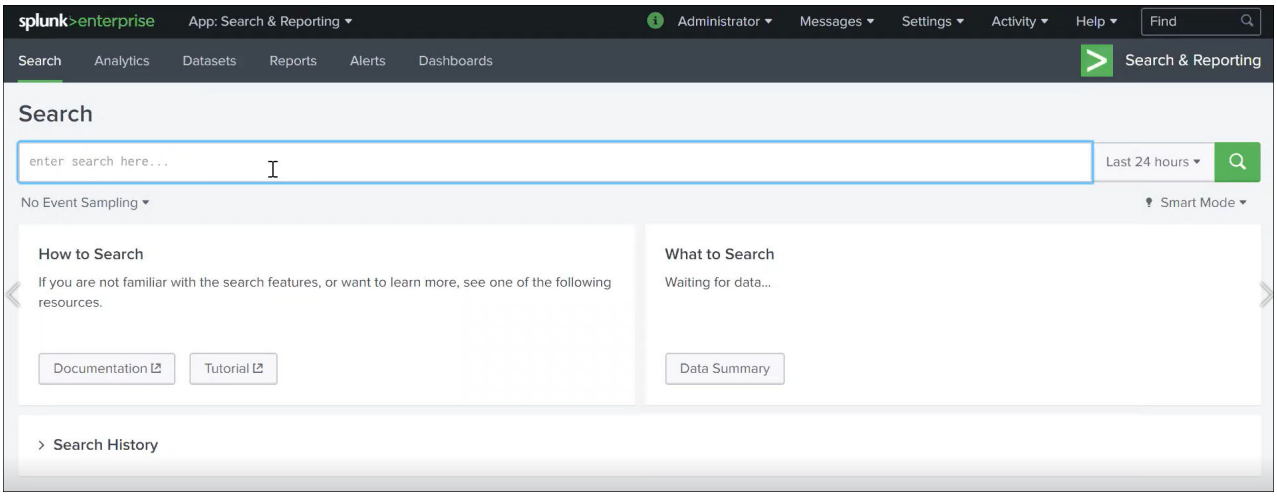


Figure 4-2: Search Screen

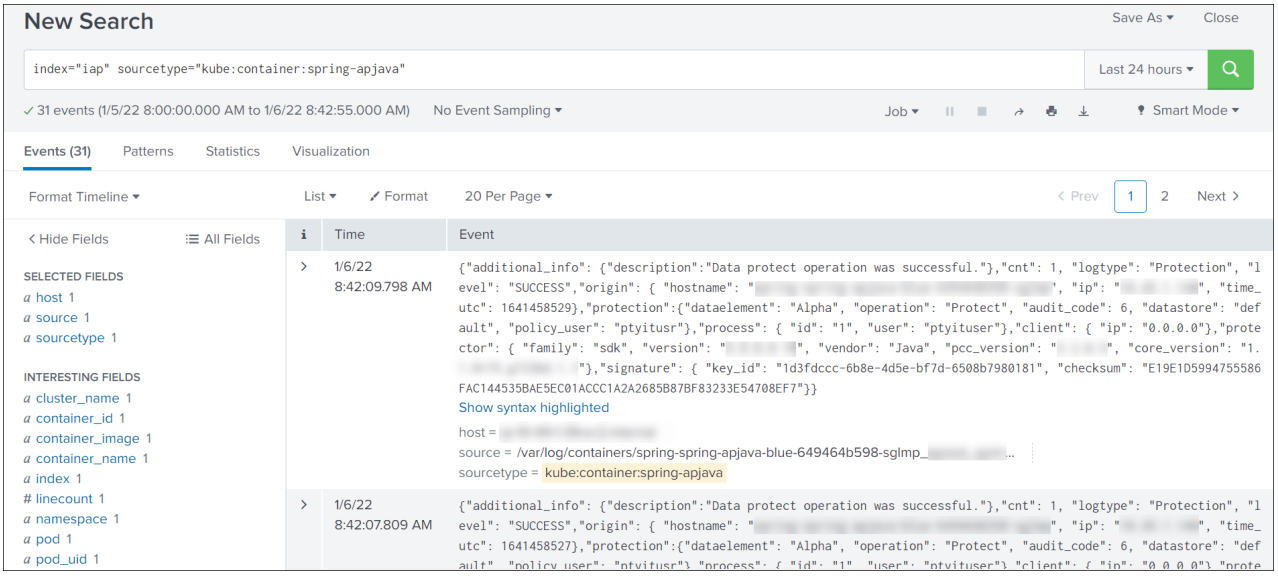
4. In the **Search** field, type the following text to filter the logs.

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

   For example:

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

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

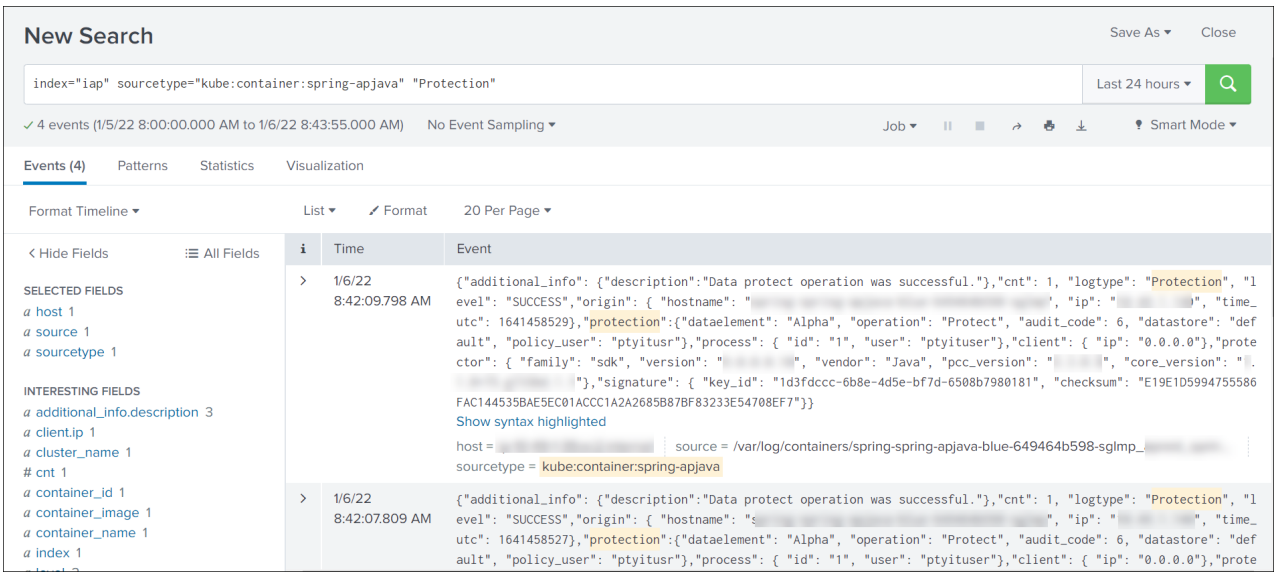


6. If you want to view only the logs that are related to the Application Protector Java, then you can modify the text in the **Search** field to filter the logs, as shown in the following snippet.

```
index="<Index_name>" sourcetype="kube:container:spring-app-java" "Protection"
```

7. Press **Enter**.

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



4.3 Autoscaling the Protegrity Reference Deployment

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

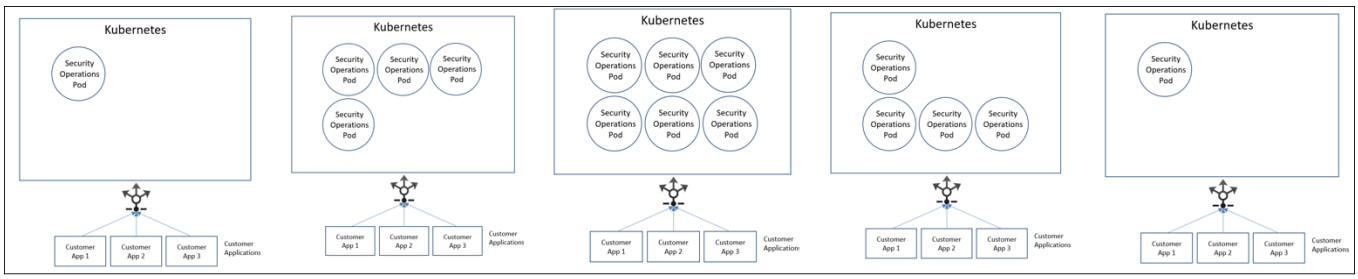


Figure 4-3: Autoscaling Kubernetes Cluster

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

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

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

Chapter 5

Appendix A: Get ESA Policy Helm Chart Details

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

5.1 Chart.yaml

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

```
apiVersion: v1
description: A chart for getting policy from ESA
name: get-esa-policy
version: 1.0.0
```

5.2 Encryption-config.yaml

The *Encryption-config.yaml* file contains information about the key used to encrypt the policy package. The content shown in the following snippet is for Azure. The users are required to edit the *azureKeyId* parameter and replace the value of the Azure Key ID with the resource ID of the Azure key that you have created in the section *Creating an Azure Key*.

```
dekProvider:
  length: 256
  algorithm: AES
  mode: ECB
  padding: PKCS5Padding
  provider: local
kekProvider:
  length: 3072
  algorithm: RSA
  mode: ECB
  padding: OAEPWithSHA-256AndMGF1Padding
  provider: AZURE
  azureKeyId: KEY_IDENTIFIER
```

5.3 Values.yaml

The *Values.yaml* file contains important fields that need to be edited by the user as per the requirements of each environment. The following sections will explain the details of each field that needs to be replaced.

5.3.1 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.

For example:

```
oc create secret generic regcred --from-
file=.dockerconfigjson=<PATH_TO_DOCKER_CONFIG>/config.json --type=kubernetes.io/
dockerconfigjson --namespace <NAMESPACE>
```

5.3.2 Name Override

These values 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: ""
```

5.3.3 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.

```
getEsaPolicyImage:
  repository: GET_ESA_POLICY_IMAGE_REPOSITORY
  tag: GET_ESA_POLICY_IMAGE_TAG
  pullPolicy: Always
  debug: false
```

The following list provides an overview of the *getEsaPolicyImage* parameter, which refers to details for the Get ESA Policy container image:

- *repository* – Refers to the name of the registry.

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

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

5.3.4 Resources

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

```
resources:
  ## We usually recommend not to specify default resources and to leave this as a conscious
  ## choice for the user. This also increases chances charts run on environments with little
  ## resources, such as Minikube. If you do want to specify resources, uncomment the following
  ## lines, adjust them as necessary, and remove the curly braces after 'resources:'.
  # limits:
  #   cpu: 500m
  #   memory: 3500Mi
  requests:
    cpu: 200m
    memory: 200Mi
```

You can specify the following parameters:

- *limits* - Specify the resource limits for the containers in a pod. These limits ensure that the running container does not use additional resources than the limit you set.
- *requests* - Specify the resource request for the containers in a pod. This information determines the nodes for deploying the pods.

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

5.3.5 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
  supplementalGroups: 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.

5.3.6 Container Security Context

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

```
## set the Init Container's security context object
## leave the field empty if not applicable
initContainerSecurityContext:
  capabilities:
    drop:
```

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

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

> **Note:** It is recommended to not modify 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.

5.3.7 ESAcred Secrets

This section provides information for the credentials that are required to access the ESA for retrieving the policy.

```
esaCredSecrets: esa-creds
```

The user is required to create a Kubernetes secret for accessing the ESA. Kubernetes secrets can be created using existing Docker credentials.

> **Important:** You require a *cluster-admin* role to create the ESAcred secrets.

For example:

```
kubectl create secret generic esa-creds --from-literal=esa_user=<ESA user> --from-
literal=esa_pass=<ESA user password> --namespace <NAMESPACE>

kubectl create secret generic esa-creds --from-literal=esa_user=admin --from-
literal=esa_pass=<ESA user password> --namespace iap
```

5.3.8 PBE Secrets

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

```
pbeSecrets: PBE_SECRET_NAME
```

The Get ESA Policy container uses this key to encrypt the policy. This is used to implement Passphrase Based Encryption (PBE).

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

> **Important:** You require a *cluster-admin* role to create the PBE secrets.

For example:

```
kubectl create secret generic <PBE_SECRET_NAME> --from-
literal='passphrase=<passphrase>' --from-literal='salt=<salt>' -n <NAMESPACE>
```

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

Important: Ensure that the passphrase has a minimum length of 12 characters, with atleast 1 uppercase letter, 2 digits, 2 special characters, and 2 non-letters (such as, digits or special characters). The passphrase must also have a minimum value of entropybits as *30* and a minimum strength of *0.66*.

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

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

5.3.9 Encryption Configuration Settings

This section gives an overview of the Encryption key configuration settings. The Encryption key configuration needs to be provided in the form of ConfigMap at the time of deployment. The user needs to create the ConfigMap using the `kubectl` command and mention the name of the configuration.

```
# encryption configmap name
# create a configmap from the file 'encryption-config.yaml' as given below:
# kubectl create configmap encryption-config --from-file=encryption-config.yaml --namespace
<SAME_AS_OF_CHART_DEPLOYMENT>
# Enable this if you want to enable envelope encryption.

#encryptionConfigMap: encryption-config
```

The *encryption-config.yaml* file is provided as a part of the Helm package. The users need to add *azureKeyId* with the Azure key resource ID information.

Note:

You need to uncomment the *encryptionConfigMap* entry to enable policy encryption using Azure KMS.

encryption-config.yaml

```
dekProvider:
  length: 256
  algorithm: AES
  mode: ECB
  padding: PKCS5Padding
  provider: local
kekProvider:
  length: 3072
  algorithm: RSA
  mode: ECB
  padding: OAEPWithSHA-256AndMGF1Padding
  provider: AZURE
  azureKeyId: KEY_IDENTIFIER
```

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

5.3.10 Security Configuration Destination

This section provides information on how you can specify the storage destination where you want to store the policy package.

```
## choose your storage destination. Currently we support <VolumeMount | ObjectStore>
securityConfigDestination: VolumeMount
```

For example, you can choose either *ObjectStore* or *VolumeMount*. By default, the value of the *securityConfigDestination* parameter is set to *VolumeMount*.

> **Important:** You can use a non-privileged role to perform this task.

5.3.11 Persistent Volume Claim

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

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

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

5.3.12 Storage Access Secrets

This section describes the credentials required to access the Azure storage container for uploading the policy package. The user is required to create a Kubernetes secret for writing to the Azure storage container. Kubernetes secrets can be created using existing Docker credentials.

```
storageAccessSecrets: WRITE_ACCESS_CREDENTIAL_SECRET_NAME
```

5.3.13 Policy

This section specifies the configurations related to the ESA policy for the current release.

```
policy:
  esaIP: ESA_IP

  ## pepserver configurations that can be changed.
    ## emptystring: Set the output behavior for empty strings
      ## null = (default) null value is returned for empty input string
      ## empty = empty value is returned for empty input string
      ## encrypt = encrypted values is returned for empty input string
    ## logLevel: Specifies the logging level for pepserver
      ## OFF (no logging)
      ## SEVERE
      ## WARNING
      ## INFO
      ## CONFIG
      ## ALL (default)
    ## caseSensitive: Specifies how policy users are checked against policy
      ## yes = (The default) policy users are treated in case sensitive manner
      ## no = policy users are treated in case insensitive manner.
  pepserverConfig:
    emptyString: "null"
    logLevel: "ALL"
    caseSensitive: "yes"

  ## absolute path in PV or ObjectStore where the policy dump file will be stored. <eg. /
test/xyz/policy >
  ## If destination is Object Store, make sure first name in path is bucket name i.e. test
will be our existing bucket
  filePath: ABSOLUTE_POLICY_PATH
```

```
  ## time (in mins) to wait till the policies get downloaded
  ## default is 15 (mins)
  timeout: 15
```

The *esaIP* field denotes the IP address of the ESA from where you want to retrieve the policy.

The *pepserverConfig* field enables you to specify the PEP server configurations.

Users are required to update the file path where you want to upload the policy package.

You can also specify the maximum time until which the Get ESA Policy container tries to establish communication with the ESA for fetching the policies, after which the connection times out. The default value is 15 minutes.

5.4 Credentials.json

The *credentials.json* file contains information about the credentials for the service principal 1 that are required to access the Azure storage account. The credentials are required to enable the Get ESA Policy container to upload the immutable policy package to the Azure storage container. The following snippet below is shown for Azure. Users are required to edit the file and replace the values for the service principal 1 credentials.

```
{
  "storage_account_name" : "AZURE_STORAGE_ACCOUNT_NAME",
  "tenant_id" : "AZURE_TENANT_ID",
  "client_id" : "AZURE_CLIENT_ID",
  "client_secret": "AZURE_CLIENT_SECRET"
}
```

5.5 pty-rbac.yaml

The *pty-rbac.yaml* file defines the roles that are assigned the Azure Policy.

> **Important:** Do not modify the values.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: iap-install-upgrade
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["create", "get" ,"list", "patch" ,"watch","delete"]
- apiGroups: ["autoscaling"] # "" indicates the core API group
  resources: ["horizontalpodautoscalers"]
  verbs: ["create", "get" ,"list", "patch" ,"watch","delete"]
- apiGroups: ["networking.k8s.io"] # "" indicates the core API group
  resources: ["ingresses"]
  verbs: ["create", "get" ,"list", "patch" ,"watch","delete"]
- apiGroups: [""] # "" indicates the core API group
  resources: ["services" ]
  verbs: ["create", "get" ,"list", "patch" , "watch","delete"]
- apiGroups: [""] # "" indicates the core API group
  resources: ["persistentvolumeclaims"]
  verbs: ["create", "get" ,"list", "watch"]
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods", "pods/log" ]
  verbs: [ "get" , "list" , "watch"]
- apiGroups: [""] # "" indicates the core API group
  resources: ["events" ]
  verbs: [ "get" , "list" , "watch"]
- apiGroups: ["metrics.k8s.io"] # Top pods permission
```

```
  resources: ["pods" ]
  verbs: [ "get" , "list" , "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: iap-install-upgrade
subjects:
- kind: ServiceAccount
  name: iap-deployer
roleRef:
  kind: Role
  name: iap-install-upgrade
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: get-esa-policy-install-delete
rules:
- apiGroups: ["batch"]
  resources: ["jobs"]
  verbs: ["create", "get","delete"]
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["create"]
- apiGroups: [""] # "" indicates the core API group
  resources: ["persistentvolumeclaims"]
  verbs: ["create", "get" ,"list", "watch"]
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods", "pods/log" ]
  verbs: [ "get" , "list" , "watch"]
- apiGroups: [""] # "" indicates the core API group
  resources: ["events" ]
  verbs: [ "get" , "list" , "watch"]
- apiGroups: ["metrics.k8s.io"] # Top pods permission
  resources: ["pods" ]
  verbs: [ "get" , "list" , "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: get-esa-policy-install-delete
subjects:
- kind: ServiceAccount
  name: get-esa-deployer
roleRef:
  kind: Role
  name: get-esa-policy-install-delete
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: iap-secrets-manager
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["create", "get" ,"list", "delete" , "update" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: iap-secrets-manager
subjects:
- kind: ServiceAccount
  name: get-esa-deployer
- kind: ServiceAccount
  name: iap-deployer
roleRef:
  kind: Role
  name: iap-secrets-manager
  apiGroup: rbac.authorization.k8s.io
```

Chapter 6

Appendix B: Sample Application Helm Chart Details

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

6.1 Chart.yaml

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

```
apiVersion: v1
description: A Spring boot application chart for Kubernetes
name: spring-apjava
version: 1.0.0
```

6.2 Decryption-config.yaml

Decryption-config.yaml – This file contains information about the key used to decrypt policy packages. The following snippet is shown for Azure. The users are required to edit the *azureKeyId* field and replace the value of the Azure key identifier.

```
dekProvider:
  length: 256
  algorithm: AES
  mode: ECB
  padding: PKCS5Padding
  provider: local
kekProvider:
  length: 3072
  algorithm: RSA
  mode: ECB
  padding: OAEPWithSHA-256AndMGF1Padding
  provider: AZURE
  azureKeyId: KEY_IDENTIFIER
```

6.3 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.

6.3.1 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>

6.3.2 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: ""
```

6.3.3 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.

```
initImage:
  repository: INIT_CONTAINER_IMAGE_REPOSITORY
  tag: INIT_CONTAINER_IMAGE_TAG
  pullPolicy: Always
  debug: false # enable this flag to get imp container debug logs on STDOUT.

springappImage:
  repository: SPRING_APJAVA_IMAGE_REPOSITORY
  tag: SPRING_APJAVA_IMAGE_TAG
  pullPolicy: Always
```

The following list provides an overview of the *initImage* parameter, which refers to the details for the Init container image:

- *repository* – Refers to the name of the registry.

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

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

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

6.3.4 Resources

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

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

springContainerResources:
  limits:
    cpu: 500m
    memory: 512Mi
  requests:
    cpu: 200m
    memory: 200Mi
```

You can specify the following parameters:

- *limits* - Specify the resource limits for the containers in a pod. These limits ensure that the running container does not use additional resources than the limit you set.
- *requests* - Specify the resource request for the containers in a pod. This information determines the nodes for deploying the pods.

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

6.3.5 Liveliness and Readiness Probe Settings

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

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

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

6.3.6 Pod Security Context

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

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

Specify the privileges and access control settings for the pod.

For more information about the Pod Security Context, refer to the section *Configure a Security Context for a Pod or Container* in the Kubernetes documentation.

For more information about the parameters, refer to the section *PodSecurityContext v1 Core* in the Kubernetes API documentation.

6.3.7 Container Security Context

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

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

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

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

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

For more information about the Container Security Context, refer to the section *Configure a Security Context for a Pod or Container* in the Kubernetes documentation.

For more information about the parameter, refer to the section *SecurityContext v1 core* in the Kubernetes API documentation.

6.3.8 Persistent Volume Claim

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

```
# If Volume Mount, 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.

6.3.9 PBE Secrets

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

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

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

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

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

6.3.10 Decryption Configuration Settings

This section provides an overview of the Decryption key configuration settings. The Decryption key configuration needs to be provided in the form of ConfigMap at the time of deployment.

```
## Decryption configmap name
## create a configmap from the file 'decryption-config.yaml' as given below:
## kubectl create configmap <CONFIG_MAP_NAME> --from-file=dsg/decryption-config.yaml --
namespace <SAME_AS_OF_CHART_DEPLOYMENT>
## specify the CONFIG_MAP_NAME below
decryptionConfigMap: decryption-config
```

The user needs to create the ConfigMap using the **`kubectl`** command and mention the name of the configuration. The *decryption-config.yaml* file is provided as a part of the Helm package.

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

6.3.11 Private Key Secret

This section provides information for the Azure Key Vault private key secret.

```
privateKeySecret: PKEY_SECRET_NAME
```

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

6.3.12 Azure Access Secrets

This section credentials required to access the Azure storage container for downloading the policy package. The user is required to create a Kubernetes secret for reading from the Azure storage container. This secret is also used to decrypt the policy package using the Azure Key Vault. Kubernetes secrets can be created using existing Docker credentials.

```
azAccessSecrets: READ_ACCESS_CREDENTIAL_SECRET_NAME
```

6.3.13 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.

6.3.14 Autoscaling

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

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

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

The following list provides a description for the parameters:

- *replicaCount* - Indicates the number of pods that get instantiated at the start of the deployment.
- *minReplicas* - Indicates the minimum number of pods that will keep running in the deployment for a cluster.
- *maxReplicas* - Indicates the maximum number of pods that will keep running in the deployment for a cluster.
- *targetCPU* - Horizontal Pod Autoscaler (HPA) will increase and decrease the number of replicas (through the deployment) to maintain an average CPU utilization across all Pods based on the % value specified in the *targetCPU* setting.

6.3.15 Service Configuration

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

```
## specify the ports exposed in your springapp configurations where,
## name - distinguishes between different ports.
## port - the port on which you wan't to expose the service externally.
## targetPort - the port no. configured while creating Tunnel.
springappService:

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

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

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

You can specify the value of the *springappService/type* parameter as *LoadBalancer* or *ClusterIP*.This parameter is only applicable if you have set the value of the *ingress/enabled* parameter to *false*.

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

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

6.3.16 Ingress Configuration

This section explains the Ingress configuration for deployment.

```
## Ingress enables testing of Green (V2/Staging) Deployments alongside Blue ones
simultaneously.
## This is realized by exposing both deployments on the same External IP but on different URL
Paths.
## If you don't want expose both the deployments at the same time, keep 'ingress.enabled' as
false.
## This means that only Blue deployment gets exposed on the Load Balancer's External IP via
prod service.
ingress:
  enabled: true

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

  ## specify external Ingress Controller if any,
  ## else keep the 'ingress.class' field empty
  ## to use cloud native Ingress Controller.
  annotations:
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"

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

  ## specify the host names and paths for Ingress rules
  ## prod and staging http paths can be used as optional fields.
  hosts:
```

```
    - host: "prod.example.com"
      httpPaths:
      - port: 8080
        prod: "/"
    ## Add host section with the hostname used as CN while creating server certificates.
    ## For accessing the staging end-points also, we need hostname as that of CN in server
certs.
    ## While creating the certificates you can use *.example.com as CN for the below example
    # - host: "prod.example.com"
    #   httpPaths:
    #   - port: 8080
    #     prod: "/"
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8080
    #     staging: "/"
    #
  ## If TLS is terminated on the Ingress Controller Load Balancer,
  ## then uncomment the tls section below and,
  ## provide K8s TLS Secret containing the certificate and key.
  # tls:
  #  - secretName: example-tls
  #    hosts:
  #      - prod.example.com
```

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

```
ingress:
  enabled: true
```

When the parameter is set to *enabled*: *true*, the ingress controller will be based on the value used in the annotation *ingress.class*. If the *enabled* parameter 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 *ingress.class* annotation is set to *nginx-iap* to indicate that the NGINX Ingress Controller is used.

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

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

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

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

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

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

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

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

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

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

```
 ## specify the host names and paths for Ingress rules
  ## prod and staging http paths can be used as optional fields.
hosts:
    - host: "prod.example.com"
      httpPaths:
      - port: 8080
        prod: "/"
    ## Add host section with the hostname used as CN while creating server certificates.
    ## For accessing the staging end-points also, we need hostname as that of CN in server
certs.
    ## While creating the certificates you can use *.example.com as CN for the below example
    # - host: "prod.example.com"
    #   httpPaths:
    #   - port: 8080
    #     prod: "/"
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8080
    #     staging: "/"
```

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

Following is an overview of the list of parameters that can be configured:

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

```
# - host: "prod.example.com"
    #   httpPaths:
    #   - port: 8080
    #     prod: "/"
    # - host: "stage.example.com"
    #   httpPaths:
    #   - port: 8080
    #     staging: "/"
```

The ingress maps the value of the provided hostname to the external IP address. You also need to configure the hostname in your DNS.

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

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

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

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

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

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

6.4 Credentials.json

The *credentials.json* file contains information about the credentials for service principal 2 that are required to access the Azure storage account. The credentials are required to enable the Sample Application container to download the immutable policy package that has been uploaded to the Azure storage container. The following snippet displays the content for Azure. Users are required to edit the file and replace the values for the service principal 2 credentials.

```
{
  "storage_account_name" : "AZURE_STORAGE_ACCOUNT_NAME",
  "tenant_id" : "AZURE_TENANT_ID",
  "client_id" : "AZURE_CLIENT_ID",
  "client_secret": "AZURE_CLIENT_SECRET"
}
```

Chapter 7

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

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

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

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

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

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

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

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

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

   This command creates file named *passwd.template*.

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

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

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

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

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

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

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

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

```
...

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

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

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

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

Chapter 8

Appendix D: Using the Dockerfile to Build Custom Images

This section explains how to use the Dockerfiles, which are provided as part of the installation package, to build custom images for the Get ESA Policy, Init, and Sample Application containers. You can use the custom image instead of the default images that are provided by Protegrity as part of the installation package. 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 custom images for the Get ESA Policy, Init, and Sample Application containers, using the Dockerfile provided with the installation package.

➤ To create custom images:

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-DOCKER-IMAGES* 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-DOCKER-IMAGES* file, refer to the section *Verifying the Prerequisites*.

2. Perform the following steps to build a Docker image for the Get ESA Policy container.

   a. Run the following command to extract the files from the *IAP-GET-ESA-POLICY_AZURE_SRC_<version_number>.tgz* file.

   ```
   tar -xvf IAP-GET-ESA-POLICY_AZURE_SRC_<version_number>.tgz
   ```

   The following files are extracted:

   - *GET-ESA-POLICY_DOCKERFILE_<version_number>*
   - *PepServerSetup_Linux_x64_<version_number>.iap.tgz*
   - *HealthCheckSetup_Linux_x64_<version_number>.tgz*
   - *PtyShmCatSetup_Linux_x64_<version_number>.tgz*

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

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

c. Run the following command in the directory where you have extracted the contents of the *IAP-GET-ESA-POLICY_AZURE_SRC_<version_number>.tgz* file.

```
docker build -t <image-name>:<image-tag> -f GET-ESA-
POLICY_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 *Microsoft Azure* documentation.

d. Run the following command to list the Get ESA Policy container image.

```
docker images
```

e. Push the Get ESA Policy container image to your preferred Container Repository.

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

3. Perform the following steps to build a Docker image for the Init container.

a. Run the following command to extract the files from the *IAP-INIT-CONTAINER_AZURE_SRC_<version_number>.tgz* file.

```
tar -xvf IAP-INIT-CONTAINER_AZURE_SRC_<version_number>.tgz
```

The following files are extracted:

- *INIT-CONTAINER_DOCKERFILE_<version_number>*
- *PtyShmPutSetup_Linux_x64_<version_number>.tgz*

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

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

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

```
docker build -t <image-name>:<image-tag> -f INIT-
CONTAINER_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 *Microsoft Azure* documentation.

d. Run the following command to list the Init container image.

```
docker images
```

e. Push the Init 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*.

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

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

```
tar -C IAP-SAMPLE-APP_AZURE_SRC_<version_number>.tgz
```

The following files are extracted:

- *IAP_RHUBI_SAMPLE-APP_DOCKERFILE_<version_number>*
- *libs/*

- *src/*
- *HealthCheckSetup_Linux_x64_<version_number>.tgz*
- *docker-entrypoint.sh*
- *passwd.template*
- *pom.xml*

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

c. Execute the following command in the directory.

```
mvn clean install
```

The *apjava-springboot-<version_number>.jar* file appears in the *./target* directory.

d. Copy the *APJavaIMSetup_<OS>_x64_<App_Protector_version>.tgz* file from the installation package to the directory where you have extracted the *IAP-SAMPLE-APP_AZURE_SRC_<version_number>.tgz* package.

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

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

f. Run the following command in the directory where you have extracted the contents of the *IAP-SAMPLE-APP_AZURE_SRC_<version_number>.tgz* file.

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

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

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

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

```
docker images
```

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

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