



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

Created on: Nov 19, 2024

Copyright

Copyright © 2004-2024 Protegity Corporation. All rights reserved.

Protegity products are protected by and subject to patent protections;

Patent: <https://support.protegity.com/patents/>.

The Protegity logo is the trademark of Protegity Corporation.

NOTICE TO ALL PERSONS RECEIVING THIS DOCUMENT

Some of the product names mentioned herein are used for identification purposes only and may be trademarks and/or registered trademarks of their respective owners.

Windows, Azure, MS-SQL Server, Internet Explorer and Internet Explorer logo, Active Directory, and Hyper-V are registered trademarks of Microsoft Corporation in the United States and/or other countries.

Linux is a registered trademark of Linus Torvalds in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SCO and SCO UnixWare are registered trademarks of The SCO Group.

Sun, Oracle, Java, and Solaris are the registered trademarks of Oracle Corporation and/or its affiliates in the United States and other countries.

Teradata and the Teradata logo are the trademarks or registered trademarks of Teradata Corporation or its affiliates in the United States and other countries.

Hadoop or Apache Hadoop, Hadoop elephant logo, Hive, and Pig are trademarks of Apache Software Foundation.

Cloudera and the Cloudera logo are trademarks of Cloudera and its suppliers or licensors.

Hortonworks and the Hortonworks logo are the trademarks of Hortonworks, Inc. in the United States and other countries.

Greenplum Database is the registered trademark of VMware Corporation in the U.S. and other countries.

Pivotal HD is the registered trademark of Pivotal, Inc. in the U.S. and other countries.

PostgreSQL or Postgres is the copyright of The PostgreSQL Global Development Group and The Regents of the University of California.

AIX, DB2, IBM and the IBM logo, and z/OS are registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

Utimaco Safeware AG is a member of the Sophos Group.

Jaspersoft, the Jaspersoft logo, and JasperServer products are trademarks and/or registered trademarks of Jaspersoft Corporation in the United States and in jurisdictions throughout the world.

Xen, XenServer, and Xen Source are trademarks or registered trademarks of Citrix Systems, Inc. and/or one or more of its subsidiaries, and may be registered in the United States Patent and Trademark Office and in other countries.

VMware, the VMware “boxes” logo and design, Virtual SMP and VMotion are registered trademarks or trademarks of VMware, Inc. in the United States and/or other jurisdictions.

Amazon Web Services (AWS) and AWS Marks are the registered trademarks of Amazon.com, Inc. in the United States and other countries.

HP is a registered trademark of the Hewlett-Packard Company.

HPE Ezmeral Data Fabric is the trademark of Hewlett Packard Enterprise in the United States and other countries.

Dell is a registered trademark of Dell Inc.

Novell is a registered trademark of Novell, Inc. in the United States and other countries.

POSIX is a registered trademark of the Institute of Electrical and Electronics Engineers, Inc.

Mozilla and Firefox are registered trademarks of Mozilla foundation.

Chrome and Google Cloud Platform (GCP) are registered trademarks of Google Inc.

Swagger Specification and all public tools under the swagger-api GitHub account are trademarks of Apache Software Foundation and licensed under the Apache 2.0 License.

Table of Contents

| | |
|---|------------|
| Copyright..... | 2 |
| | |
| Chapter 1 Protegity Security Operations Cluster using Kubernetes..... | 6 |
| 1.1 Business Problem..... | 6 |
| 1.2 Protegity Solution..... | 6 |
| 1.3 Architecture and Components..... | 7 |
| | |
| Chapter 2 Immutable Application Protector (IAP) Deployment Model..... | 9 |
| 2.1 Verifying the Prerequisites..... | 9 |
| 2.2 Downloading the Installation Package..... | 10 |
| 2.3 Initializing the Linux Instance..... | 11 |
| 2.4 Creating Certificates and Keys for TLS Authentication..... | 14 |
| 2.5 Uploading the Images to the Container Repository..... | 15 |
| 2.6 Creating the Cloud Runtime Environment..... | 16 |
| 2.6.1 Creating the Azure Runtime Environment..... | 16 |
| 2.6.1.1 Logging in to the Azure Environment..... | 16 |
| 2.6.1.2 Registering an Application..... | 19 |
| 2.6.1.3 Creating an Azure Key..... | 23 |
| 2.6.1.4 Creating an Azure Storage Container..... | 32 |
| 2.6.1.5 Creating an Azure File Share..... | 37 |
| 2.6.1.6 Creating a Kubernetes Cluster..... | 41 |
| 2.7 Deploying the Containers with Azure Policy and Role-Based Access Control (RBAC)..... | 43 |
| 2.7.1 Applying an Azure Policy..... | 43 |
| 2.7.2 Applying RBAC..... | 50 |
| 2.7.3 Retrieving the Policy from the ESA..... | 51 |
| 2.7.4 Setting up the Environment for Deploying the Sample Application Container..... | 54 |
| 2.7.5 Deploying the Sample Application Container on the Kubernetes Cluster..... | 58 |
| 2.7.6 Upgrading the Helm Charts..... | 64 |
| 2.7.7 Upgrading the IAP Java from v7.1 MR4 to v9.1.0.5..... | 70 |
| 2.8 Deploying the Containers without an Azure Policy and RBAC..... | 76 |
| 2.8.1 Retrieving the Policy from the ESA..... | 76 |
| 2.8.2 Deploying the Sample Application Container on the Kubernetes Cluster..... | 78 |
| 2.8.3 Upgrading the Helm Charts..... | 88 |
| 2.8.4 Upgrading the IAP Java from v7.1 MR4 to v9.1.0.5..... | 94 |
| | |
| Chapter 3 Accessing Logs Using Splunk..... | 100 |
| 3.1 Understanding the Logging Architecture..... | 100 |
| 3.2 Setting Up Splunk..... | 101 |
| 3.3 Configuring Splunk..... | 102 |
| 3.4 Deploying the Splunk Connect for Kubernetes Helm Chart on the Kubernetes Cluster..... | 107 |
| | |
| Chapter 4 Protecting Data Using the Sample Application Container Deployment..... | 109 |
| 4.1 Running Security Operations..... | 109 |
| 4.2 Viewing Logs in Splunk..... | 111 |
| 4.3 Autoscaling the Protegity Reference Deployment..... | 113 |
| | |
| Chapter 5 Appendix A: Sample Application Helm Chart Details..... | 115 |
| 5.1 Chart.yaml..... | 115 |
| 5.2 Values.yaml..... | 115 |
| 5.2.1 Image Pull Secrets..... | 115 |
| 5.2.2 Name Override..... | 116 |
| 5.2.3 Container Image Repository..... | 116 |
| 5.2.4 Resources..... | 116 |



| | |
|--|-----|
| 5.2.5 Liveliness and Readiness Probe Settings..... | 117 |
| 5.2.6 Pod Security Context..... | 117 |
| 5.2.7 Container Security Context..... | 117 |
| 5.2.8 Persistent Volume Claim..... | 118 |
| 5.2.9 Private Key Source..... | 118 |
| 5.2.10 PBE Secrets..... | 118 |
| 5.2.11 Private Key Secret..... | 119 |
| 5.2.12 Azure Access Secrets..... | 119 |
| 5.2.13 Deployment..... | 119 |
| 5.2.14 Autoscaling..... | 120 |
| 5.2.15 Service Configuration..... | 120 |
| 5.2.16 Ingress Configuration..... | 120 |
| 5.3 az_credentials.json..... | 123 |
| 5.4 pty-rbac.yaml..... | 123 |

| | |
|--|------------|
| Chapter 6 Appendix B: Applying the NSS Wrapper to the Customer Application Container Image..... | 125 |
|--|------------|

| | |
|---|------------|
| Chapter 7 Appendix C: Using the Dockerfile to Build Custom Images..... | 127 |
| 7.1 Creating Custom Images Using Dockerfile..... | 127 |

Chapter 1

Protegility Security Operations Cluster using Kubernetes

[1.1 Business Problem](#)

[1.2 Protegility Solution](#)

[1.3 Architecture and Components](#)

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

1.1 Business Problem

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

- Protegility 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 Protegility 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 Protegility data security components that can leverage the autoscaling capabilities of Kubernetes to scale.

1.2 Protegility Solution

The Protegility solution leverages cloud native capabilities to deliver a Protegility 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 Protegility 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 Protegility 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 Protegity Pods. These will increase when the CPU threshold is passed, and they will be shrunk 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 Protegity Immutable Application Protector (IAP) solution.

The IAP Gen2 deployment consists of the following two stages:

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

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

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

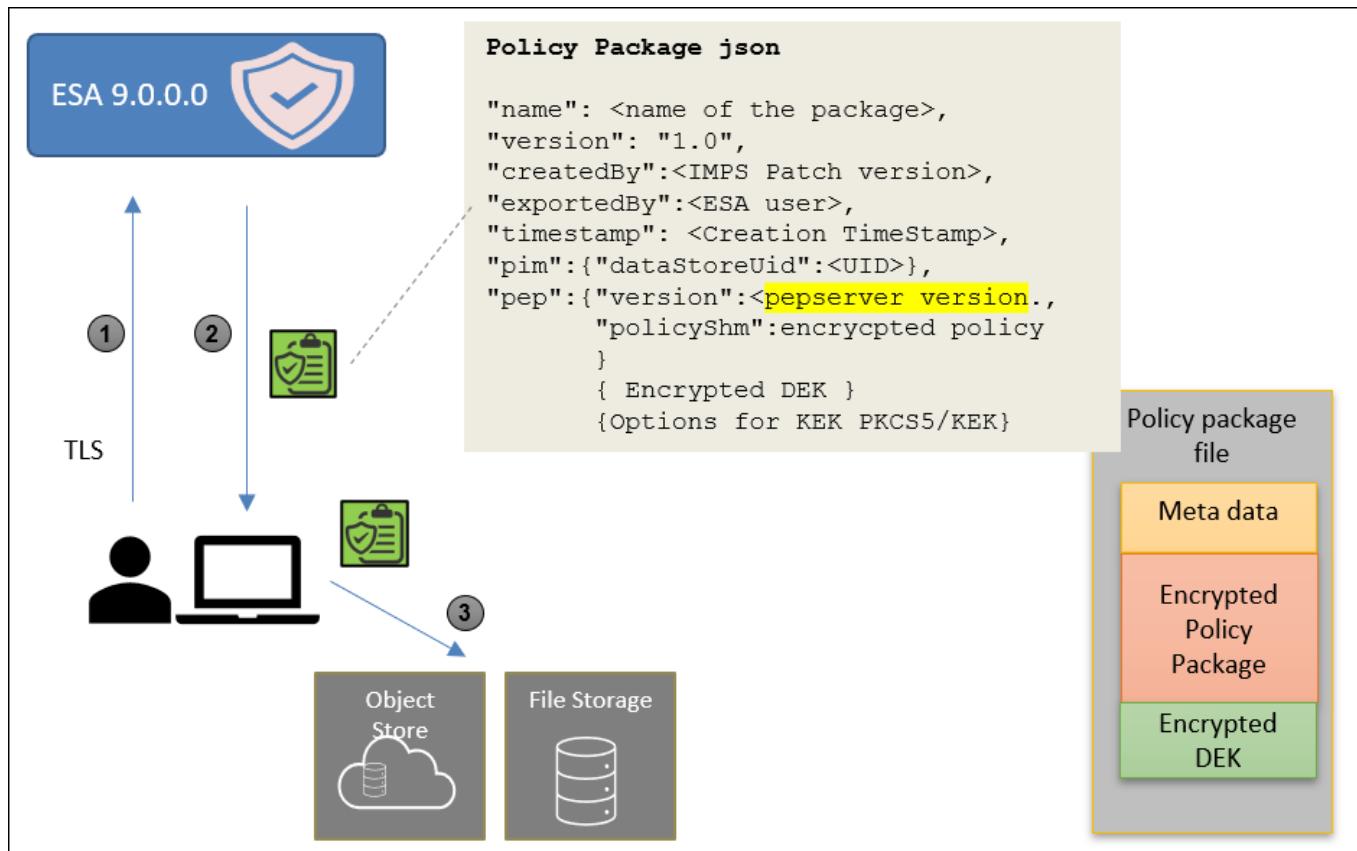


Figure 1-1: Stage 1: Fetching the ESA Policy and Creating the Policy Package

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

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

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

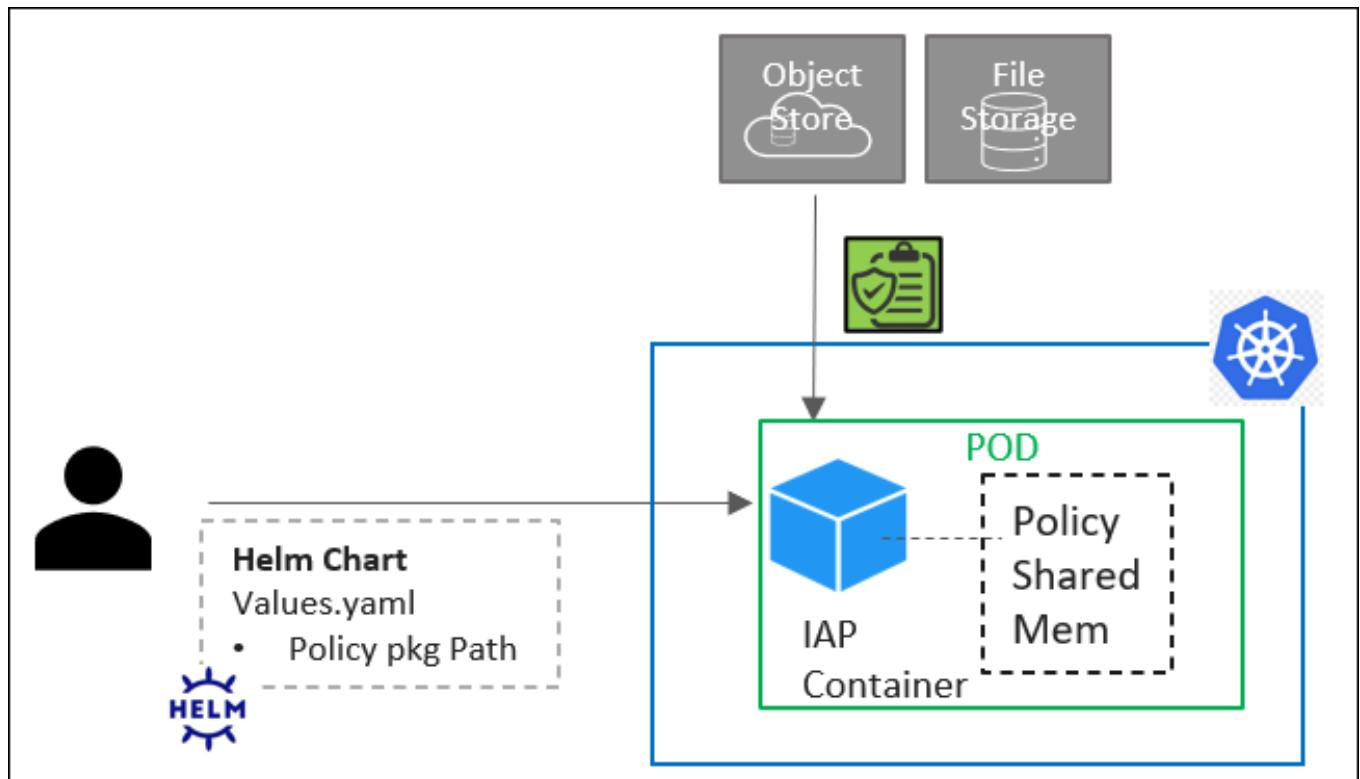


Figure 1-2: Stage 2: Deploying the Sample Application Container

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

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

Note: The Sample Application has been used for demonstration purposes. You can choose to create a custom image by integrating your custom application with the Application Protector Java libraries.
2. The Sample Container reads the immutable policy snapshot from the Azure storage container or a persistent volume, decrypts it using the Kubernetes secret configured for Passphrase-Based Encryption or KMS-related secrets, and stores the policy in the shared memory of the Pod.
3. The Client Application sends a request to the Sample application over REST for protecting, unprotecting, and reprotecting data.
4. The Sample Application internally sends the request to the Application Protector Java, and then returns the protected, unprotected, or reprotected value to the client application.

Chapter 2

Immutable Application Protector (IAP) Deployment Model

- [2.1 Verifying the Prerequisites](#)
- [2.2 Downloading the Installation Package](#)
- [2.3 Initializing the Linux Instance](#)
- [2.4 Creating Certificates and Keys for TLS Authentication](#)
- [2.5 Uploading the Images to the Container Repository](#)
- [2.6 Creating the Cloud Runtime Environment](#)
- [2.7 Deploying the Containers with Azure Policy and Role-Based Access Control \(RBAC\)](#)
- [2.8 Deploying the Containers without an Azure Policy and RBAC](#)

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

2.1 Verifying the Prerequisites

Ensure that the following prerequisites are met:

Reference Solution Deployment

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

For more information about building a custom image using the Dockerfile, refer to the section [Creating Custom Images Using Dockerfile](#).

- *PolicyUtil_Linux_x64_<version>.tgz* - Binary file containing the scripts used to retrieve the policy package from the ESA.
- *HOW-TO-BUILD* - Text file specifying how to build the Sample Application and Docker images.
- *manifest.json* - File specifying the name of the product and its components, the version number of the protector, and the build number of the 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 9.1.0.5.

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

- Trusted Application - Create a Trusted Application with the name as `com.protegity.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 B: Applying the NSS Wrapper to the Customer Application Container Image](#).

For more information about the `nss_wrapper` library, refer to the section [The nss_wrapper library](#).

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

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

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

Additional Prerequisites

The following additional prerequisites are required:

- Linux instance* - This instance can be used to communicate with the Kubernetes cluster. This instance can be on-premise or on 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.
 - Service principal with *Read* permissions on the Azure Storage Account and *Decrypt* permissions 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 Sample Application image

2.2 Downloading the Installation Package

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

► To download the installation package:

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

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

The following files are extracted:

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

2.3 Initializing the Linux Instance

This section describes how you can initialize the Linux instance.

► To initialize the Linux instance:

1. Install the 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
```

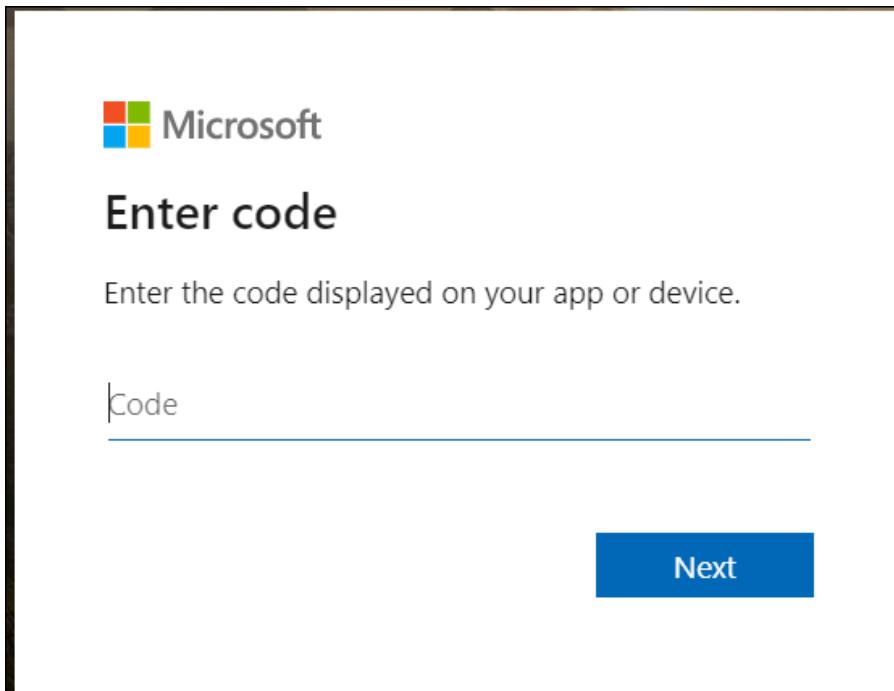
If the CLI is able to open your default browser, then the Azure sign-in page appears.

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.
```

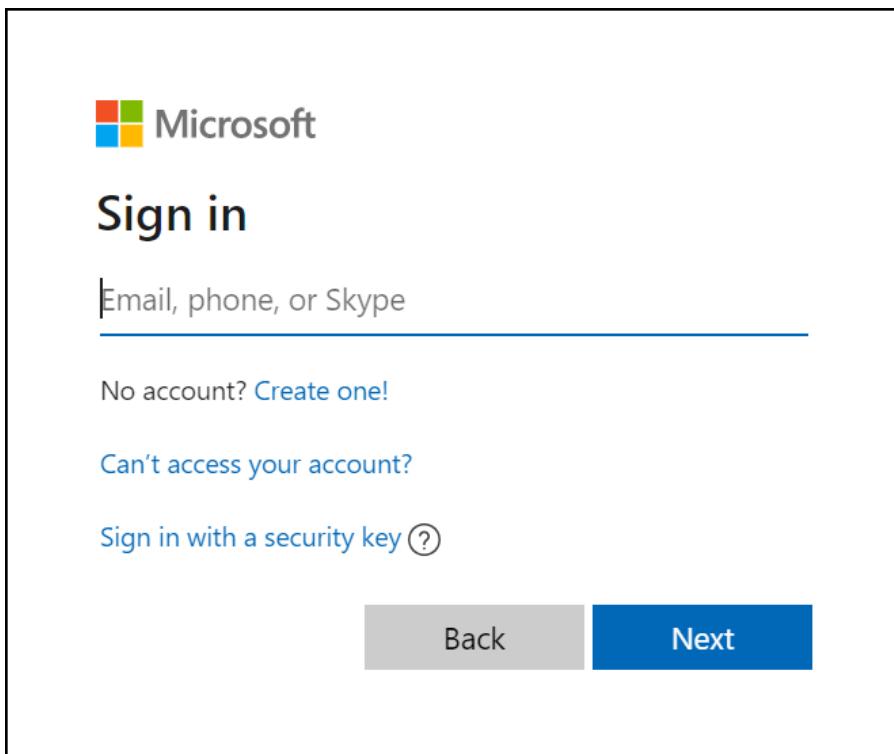
3. Copy the URL and paste it in a browser, and then press **Enter**.

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.

```
[  
 {  
   "cloudName": "AzureCloud",  
   "id": [REDACTED],  
   "isDefault": true,  
   "name": "Azure Cloud Platform",  
   "state": "Enabled",  
   "tenantId": [REDACTED],  
   "user": {  
     "name": [REDACTED],  
     "type": "user"  
   }  
 }  
 ]
```

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 command to add the repository from where the Helm charts need to be fetched.

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

9. 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
```

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
- `az_credentials.json` – Credentials file for the service principal 2, which is required for decrypting the policy packages that are uploaded on Azure. By default, the `az_credentials.json` file contains placeholders, which need to be replaced with the actual values applicable for the service principal 2.
- `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.

- 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*.

- 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
```

- Copy all the certificates to a common directory.

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

2.5 Uploading the Images to the Container Repository

This section describes how you can upload the Sample Application images to the Container Registry.

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 Customer Application container has 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 B: Applying the NSS Wrapper to the Customer Application Container Image](#).

- To upload the images to the Container Repository:

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

For more information about creating a custom image, refer to the section [Creating Custom Images Using Dockerfile](#).

4. Perform the following steps to upload the Sample Application Container image to the Azure Container Registry.

- a. Run the following command to list the Sample Application container image.

```
docker images
```

- b. 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 spring-app:AZURE <Container Registry Name>.azurecr.io/spring-app:AZURE
```

For more information about tagging an image, refer to the [Microsoft Azure](#) documentation.

- c. 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/spring-app:AZURE
```

5. Navigate to the directory where you have extracted the Helm charts packages for the Sample Application containers.

6. In the *values.yaml* file, update the required path for the *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:

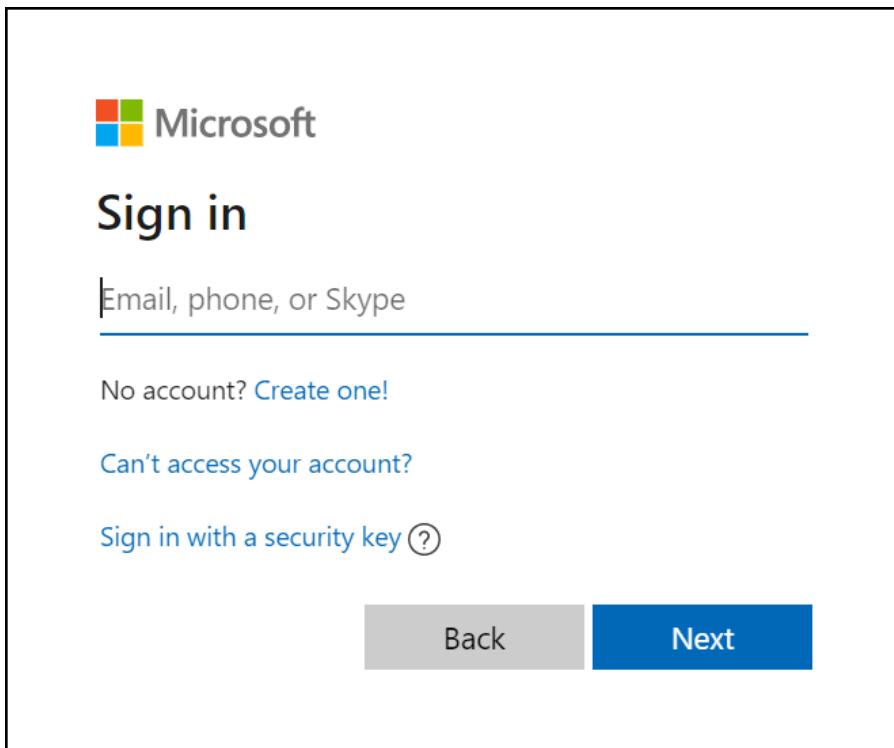
1. Access Azure using the following URL:

<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.

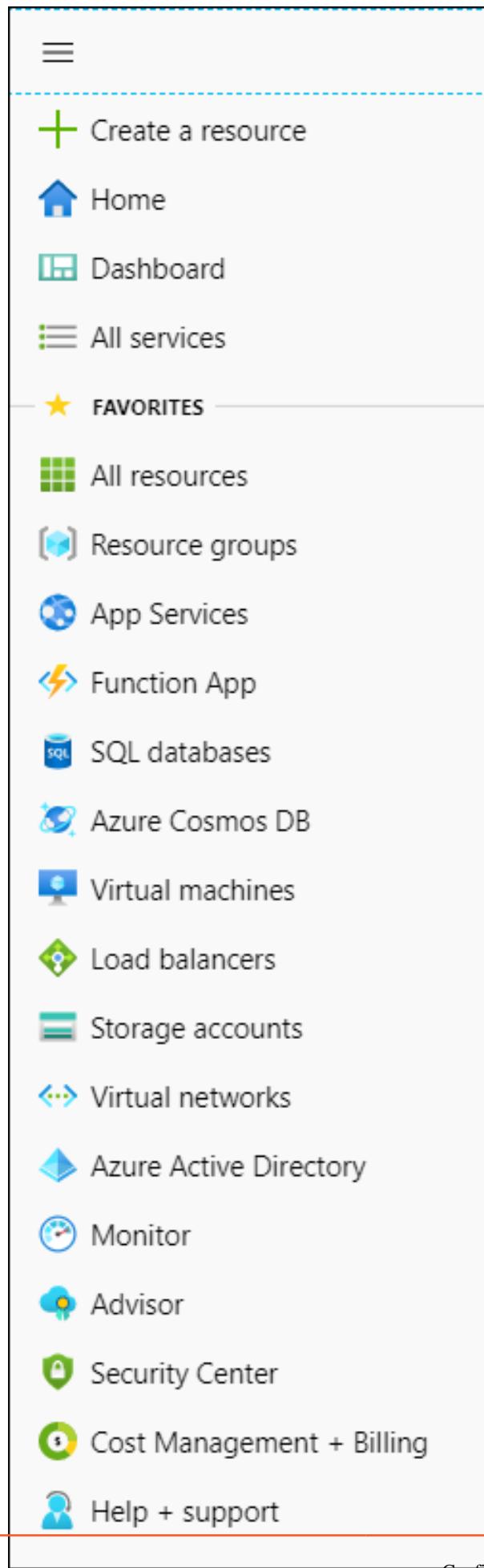
The Azure Portal home screen appears. By default, the home screen displays a list of frequently used Azure services.



Figure 2-1: Azure Portal Home Screen

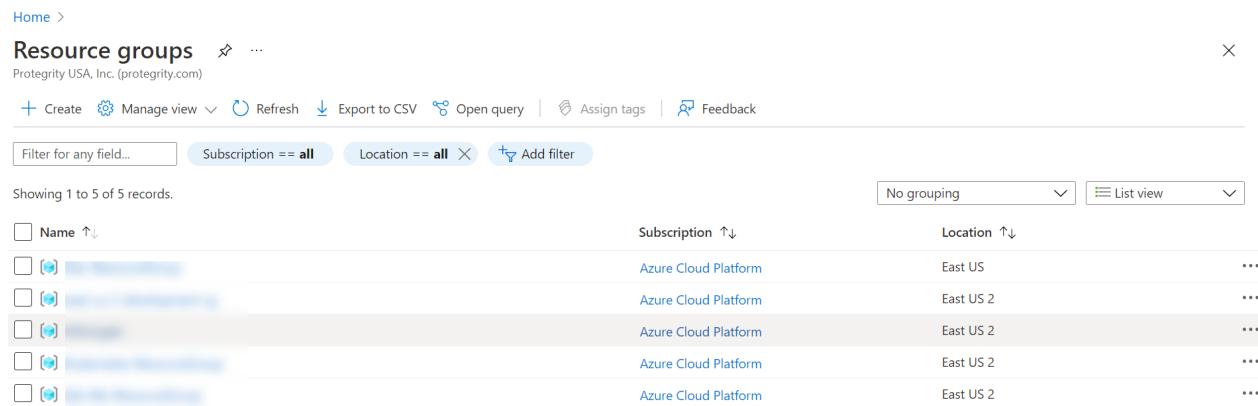
6. Click the **Portal** menu icon (≡).

The **Portal** menu appears.



7. Click **Resource groups**.

The **Resource groups** screen appears.



The screenshot shows the 'Resource groups' page in the Azure portal. At the top, there are navigation links ('Home >'), a title bar ('Resource groups'), and various action buttons ('Create', 'Manage view', 'Refresh', 'Export to CSV', 'Open query', 'Assign tags', 'Feedback'). Below the header are filter options ('Filter for any field...', 'Subscription == all', 'Location == all', 'Add filter') and grouping dropdowns ('No grouping', 'List view'). The main table lists five resource groups, each with a checkbox, a blue gear icon, and a blurred name. The columns are 'Name' (sorted by name), 'Subscription' (sorted by subscription), and 'Location' (sorted by location). Each row has a three-dot menu icon on the right.

| Name | Subscription | Location |
|-----------|----------------------|-----------|
| [Blurred] | Azure Cloud Platform | East US |
| [Blurred] | Azure Cloud Platform | East US 2 |
| [Blurred] | Azure Cloud Platform | East US 2 |
| [Blurred] | Azure Cloud Platform | East US 2 |
| [Blurred] | Azure Cloud Platform | East US 2 |

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.

The screenshot shows the Azure Active Directory Overview screen for the tenant 'Protegity USA, Inc.'. The left sidebar lists management options like Users, Groups, External Identities, etc. The main area displays basic information: Name (Protegity USA, Inc.), Tenant ID (redacted), Primary domain (protegity.com), License (redacted), and Alerts. It also shows user statistics: 589 Users, 1,278 Groups, 319 Applications, and 763 Devices.

Figure 2-2: Azure Active Directory Overview Screen

4. Navigate to App registrations.

The App Registrations screen appears.

The screenshot shows the App registrations screen for the tenant 'Protegity USA, Inc.'. The left sidebar lists management options like Users, Groups, External Identities, etc. The main area shows a list of owned applications. One application is listed: 'service-principal' (Application (client) ID starts with 'se').

Figure 2-3: App Registrations Screen

5. Click New registration.

The Register an application screen appears.

All services > Protegity USA, Inc. - App registrations > Register an application

Register an application

*** Name**
The user-facing display name for this application (this can be changed later).

Supported account types
Who can use this application or access this API?

Accounts in this organizational directory only (Protegity USA, Inc. only - Single tenant)

Accounts in any organizational directory (Any Azure AD directory - Multitenant)

Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)

[Help me choose...](#)

Redirect URI (optional)
We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Web e.g. <https://myapp.com/auth>

By proceeding, you agree to the Microsoft Platform Policies [↗](#)

Register

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.

All services > Protegity USA, Inc. - App registrations >

Display name : [REDACTED]
Application (client) ID : [REDACTED]
Directory (tenant) ID : [REDACTED]
Object ID : [REDACTED]

Supported account types : My organization only
Redirect URIs : Add a Redirect URI
Application ID URI : Add an Application ID URI
Managed application in ... : service-principal

Got a second? We would love your feedback on Microsoft identity platform (previously Azure AD for developer). →

Welcome to the new and improved App registrations. Looking to learn how it's changed from App registrations (Legacy)? [Learn more](#)

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 `az_credentials.json` file, refer to the section [az_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.

| | |
|--|---|
| Description | <input type="text"/> |
| Expires | <input checked="" type="radio"/> In 1 year <input type="radio"/> In 2 years <input type="radio"/> Never |
| <input type="button" value="Add"/> <input type="button" value="Cancel"/> | |

Figure 2-6: Add a Client Secret Screen

- 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.

Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

| Description | Expires | Value |
|------------------------------|------------|-----------------|
| secret for service principal | 11/19/2020 | [Blurred Value] |

Copy to clipboard

- 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 *az_credentials.json* file for the corresponding service principal.

For more information about the *az_credentials.json* file, refer to the section [az_credentials.json](#).

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.

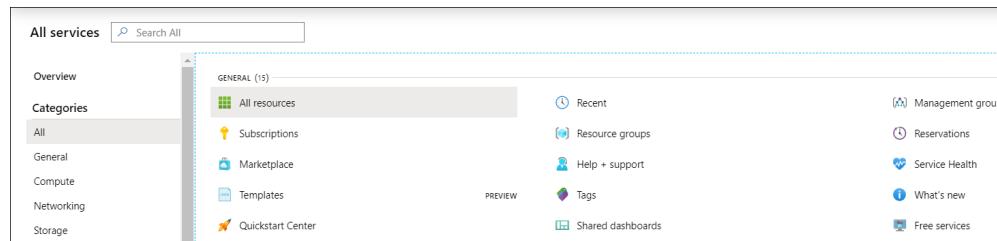


Figure 2-7: All Services Screen

4. Select **Key vaults** under the **SECURITY** section.

The **Key vaults** screen appears.

| Name | Type | Resource group | Location | Subscription |
|-----------|-----------|------------------|----------------------|--------------|
| Key vault | Key vault | East US | Azure Cloud Platform | |
| Key vault | Key vault | East US 2 | Azure Cloud Platform | |
| Key vault | Key vault | East US | Azure Cloud Platform | |
| Key vault | Key vault | East US | Azure Cloud Platform | |
| Key vault | Key vault | North Central US | Azure Cloud Platform | |
| Key vault | Key vault | East US | Azure Cloud Platform | |
| Key vault | Key vault | East US | Azure Cloud Platform | |
| Key vault | Key vault | East US | Azure Cloud Platform | |

Figure 2-8: Key Vaults Screen

5. Click Add.

The **Create key vault** screen appears.

Azure Key Vault is a cloud service used to manage keys, secrets, and certificates. Key Vault eliminates the need for developers to store security information in their code. It allows you to centralize the storage of your application secrets which greatly reduces the chances that secrets may be leaked. Key Vault also allows you to securely store secrets and keys backed by Hardware Security Modules or HSMs. The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated. In addition, key vault provides logs of all access and usage attempts of your secrets so you have a complete audit trail for compliance. [Learn more](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * Azure Cloud Platform

Resource group * Select existing... Create new

Instance details

Key vault name * Enter the name

Region * North Central US

Pricing tier * Standard

Review + create < Previous Next : Access policy >

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

| Field | Description |
|----------------|---|
| Subscription | By default, <i>Azure Cloud Platform</i> 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, <i>Standard</i> or <i>Premium</i> . |

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

The **Access policy** tab appears.



The screenshot shows the 'Access policy' tab of the 'Create key vault' wizard. At the top, there are tabs for 'Basics', 'Access policy' (which is selected), 'Virtual network', 'Tags', and 'Review + create'. Below the tabs, there's a section titled 'Enable Access to:' with three checkboxes: 'Azure Virtual Machines for deployment', 'Azure Resource Manager for template deployment', and 'Azure Disk Encryption for volume encryption'. A link '+ Add Access Policy' is present. The main area is titled 'Current Access Policies' with a table:

| Name | Category | Email | Key Permissions | Secret Permissions | Certificate Permissions | Action |
|-------------|----------|----------------------------|-----------------|--------------------|-------------------------|-------------------------|
| USER | | | 9 selected | 7 selected | 15 selected | <button>Delete</button> |
| Rohan Desai | USER | rohan.desai@protegrity.com | | | | |

At the bottom, there are buttons for 'Review + create' and navigation links '< Previous' and 'Next : Virtual network >'.

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.

The screenshot shows the 'Add access policy' screen. At the top, there's a breadcrumb trail: 'All services > Key vaults > Create key vault > Add access policy'. The title is 'Add access policy' with a sub-instruction 'Add access policy'. The form has several sections:

- Configure from template (optional)**: A dropdown menu.
- Key permissions**: A dropdown menu showing '0 selected'.
- Secret permissions**: A dropdown menu showing '0 selected'.
- Certificate permissions**: A dropdown menu showing '0 selected'.
- Select principal**: A section with a red asterisk (*) and a note 'None selected'. There's a right-pointing arrow icon next to it.
- Authorized application**: A section with a lock icon and a note 'None selected'.

At the bottom left is a blue 'Add' button.

Figure 2-10: Add Access Policy Screen

- b. In the **Key permissions** list, select the **Decrypt** check box under the **Key Management Operations** section.
You need to assign the *Decrypt* permissions to the service principal so that it can get the key from the Azure Key Vault for decrypting the policy package, if you have used the Azure Key Vault for encrypting the package.

Add access policy

Add access policy

Configure from template (optional)

Key permissions

0 selected

Select all

Key Management Operations

Get

List

Update

Create

Import

Delete

Recover

Backup

Restore

Cryptographic Operations

Decrypt

Encrypt

Unwrap Key

Wrap Key

Verify

Select principal

Authorized application ⓘ

Add

The screenshot shows the 'Add access policy' dialog. In the 'Key permissions' section, the 'Select all' checkbox is checked. Below it, two sections of operations are listed: 'Key Management Operations' (Get, List, Update, Create, Import, Delete, Recover, Backup, Restore) and 'Cryptographic Operations' (Decrypt, Encrypt, Unwrap Key, Wrap Key, Verify). The 'Select principal' and 'Authorized application' fields are also visible, along with an 'Add' button.

Figure 2-11: Key Permissions List

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

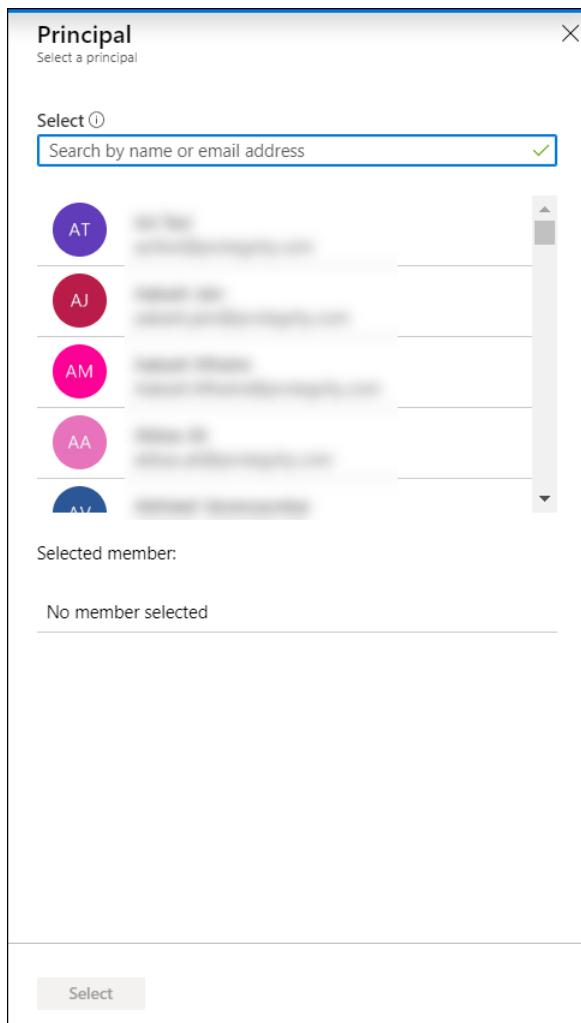


Figure 2-12: Principal Dialog Box

- d. In the **Select** field, search for and then select the required service principal.
For example, search for and select the service principal 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 permissions to the selected service principal.
9. Click **Next** to navigate to the **Virtual network** tab.

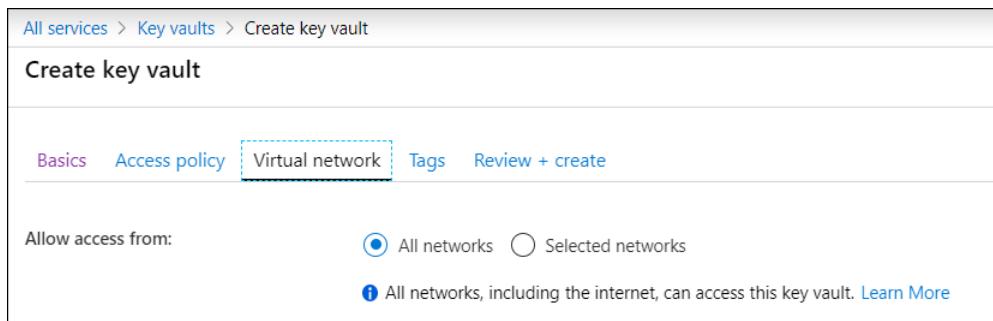


Figure 2-13: Virtual Network Tab

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

| Name | Value | Resource |
|------|-------|-----------|
| | | Key Vault |

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.

| Basics | |
|--|----------------------|
| Subscription | Azure Cloud Platform |
| Resource group | [REDACTED] |
| Key vault name | [REDACTED] |
| Region | North Central US |
| Pricing tier | Standard |
| Access policy | |
| Azure Virtual Machines for deployment | Not Enabled |
| Azure Resource Manager for template deployment | Not Enabled |
| Azure Disk Encryption for volume encryption | Not Enabled |
| Access policies | 2 |
| Virtual network | |
| Allow access from: | All networks |

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.

Your deployment is complete

Deployment name: [REDACTED]
Subscription: Azure Cloud Platform
Resource group: [REDACTED]

Start time: 11/19/2019, 3:33:47 PM
Correlation ID: f43521bf-11b1-4b66-9fc1-1be79a1ec52e

Deployment details (Download)

Next steps

Go to resource

15. Click **Go to resource**.

The **Overview** screen appears displaying information about the key vault that you have created.

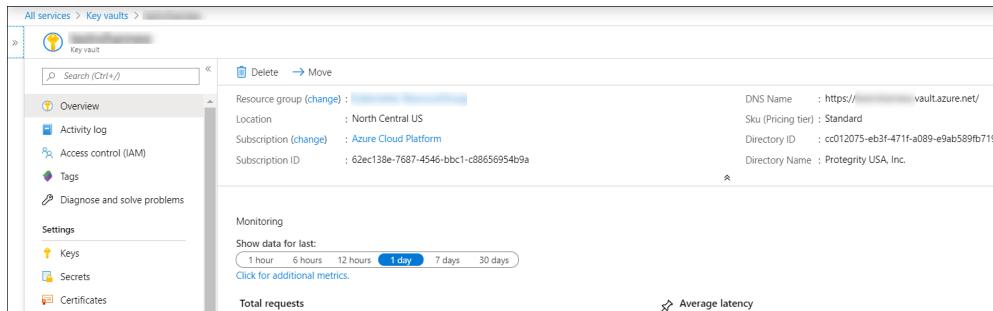


Figure 2-16: Key Vaults Overview Screen

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

The **Keys** screen appears.

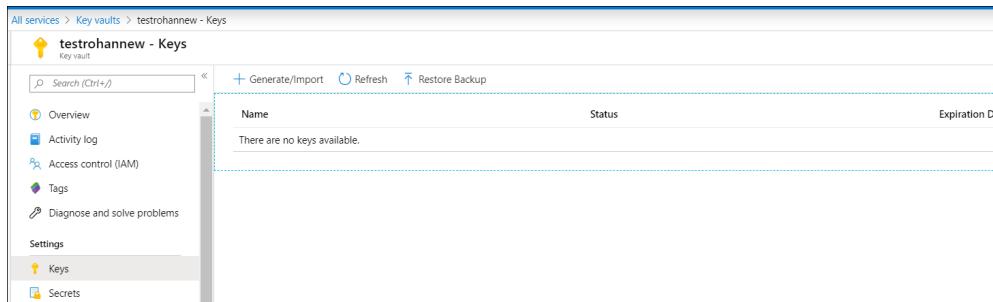


Figure 2-17: Key Screen

17. Click **Generate/Import** to create a key.

The **Create a key** screen appears.

All services > Key vaults > testrohannew - Keys > Create a key

Create a key

Options

Generate

Name * ⓘ

Key Type ⓘ

RSA EC

RSA Key Size

2048 3072 4096

Set activation date? ⓘ

Set expiration date? ⓘ

Enabled?

Yes No

Create

Figure 2-18: Create a Key Screen

- 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 <i>RSA</i> . |
| RSA Key Size | Specify the key size for RSA. Select the value as <i>3072</i> . |
| Enabled | Select <i>Yes</i> to enable the key. |

- Click **Create**.

The key is created and the following message appears.

+ Generate/Import ⏪ Refresh ⏪ Restore Backup

ⓘ The key '...' has been successfully created.

| Name | Status | Expiration Date |
|------------|-----------|-----------------|
| [redacted] | ✓ Enabled | |

- Click the key name.

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

| Version | Status | Activation Date | Expiration Date |
|----------------------------------|-----------|-----------------|-----------------|
| CURRENT VERSION | | | |
| Seb4c4630c9149aeb8f49491068e1cfa | ✓ Enabled | | |

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 `Keyid` parameter while creating the public key for encrypting the policy.

For more information about creating public key for encrypting the policy, refer to the section [Retrieving the Policy from the ESA](#).

You also need to specify this value while creating a secret for decrypting the policy package.

For more information about creating a secret for decrypting the policy on a container with an Azure Policy and RBAC, refer to [step 5](#) in section [Setting up the Environment for Deploying the Sample Application Container](#).

For more information about creating a secret for decrypting the policy on a container without an Azure Policy and RBAC, refer to [step 5](#) 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.

| All services > Storage accounts | | | | | |
|---------------------------------|-----------------|-----------|-------------------|-------------|--------------------------|
| Storage accounts | | | | | |
| Showing 1 to 53 of 53 records. | | | | | |
| Name ↑↓ | Type ↑↓ | Kind ↑↓ | Resource group ↑↓ | Location ↑↓ | Subscription ↑↓ |
| [checkbox] | Storage account | StorageV2 | [redacted] | East US | Azure Cloud Platform ... |
| [checkbox] | Storage account | Storage | [redacted] | East US | Azure Cloud Platform ... |
| [checkbox] | Storage account | Storage | [redacted] | East US | Azure Cloud Platform ... |
| [checkbox] | Storage account | Storage | [redacted] | East US | Azure Cloud Platform ... |
| [checkbox] | Storage account | StorageV2 | [redacted] | East US 2 | Azure Cloud Platform ... |
| [checkbox] | Storage account | StorageV2 | [redacted] | East US 2 | Azure Cloud Platform ... |
| [checkbox] | Storage account | Storage | [redacted] | East US | Azure Cloud Platform ... |

Figure 2-22: Storage Accounts Screen

4. Click **Add**.

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

All services > Storage accounts > Create storage account

Create storage account

Basics Networking Advanced Tags Review + create

Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below. [Learn more](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * Azure Cloud Platform

Resource group * Select existing... Create new

Instance details

The default deployment model is Resource Manager, which supports the latest Azure features. You may choose to deploy using the classic deployment model instead. [Choose classic deployment model](#)

Storage account name * (input field)

Location * (US) East US 2

Performance Standard Premium

Account kind StorageV2 (general purpose v2)

Replication (input field)

Review + create < Previous Next : Networking >

Figure 2-23: Create Storage Account Screen

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

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

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

- Click **Review + create** to validate the entered information.

Azure validates the information that you have entered.

The screenshot shows the 'Create storage account' page in the Azure portal. At the top, a green bar indicates 'Validation passed'. Below it, tabs for 'Basics', 'Networking', 'Advanced', 'Tags', and 'Review + create' are visible, with 'Review + create' being the active tab. The 'Basics' section contains the following configuration:

| | |
|-----------------------|--|
| Subscription | Azure Cloud Platform |
| Resource group | [REDACTED] |
| Location | (US) East US 2 |
| Storage account name | [REDACTED] |
| Deployment model | Resource manager |
| Account kind | StorageV2 (general purpose v2) |
| Replication | Read-access geo-redundant storage (RA-GRS) |
| Performance | Standard |
| Access tier (default) | Hot |

The 'Networking' section shows 'Connectivity method' as 'Public endpoint (all networks)'. The 'Advanced' section shows 'Secure transfer required' as 'Enabled', 'Hierarchical namespace' as 'Disabled', and 'Blob soft delete' as 'Disabled'. At the bottom, there are buttons for 'Create', '< Previous' and 'Next >', and 'Download a template for automation'.

7. Click **Create** to create the storage account.

The storage account is created and the following screen appears.

The screenshot shows the 'Microsoft.StorageAccount-20191119163820 - Overview' page. The left sidebar has 'Overview' selected, along with 'Inputs', 'Outputs', and 'Template'. The main area displays a message: 'Your deployment is complete'. It shows deployment details: 'Deployment name: Microsoft.StorageAccount-20191119163820', 'Subscription: Azure Cloud Platform', 'Resource group: [REDACTED]', 'Start time: 11/19/2019, 4:55:48 PM', and 'Correlation ID: 11535a6a-5a33-465d-9f3c-f51ab1f507bc'. There are also links for 'Deployment details (Download)', 'Next steps', and 'Go to resource'.

Figure 2-24: Storage Account Overview Screen

8. Click **Go to resource**.

The **Overview** screen appears, displaying the details of your storage account.

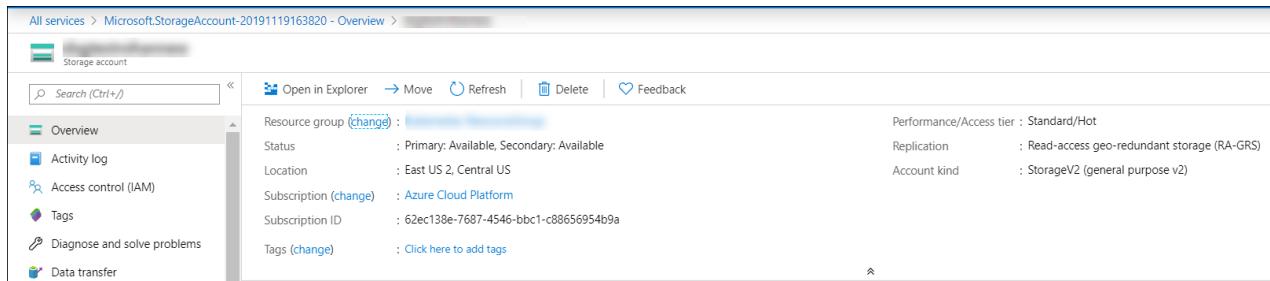


Figure 2-25: Storage Account Overview Screen

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)**.

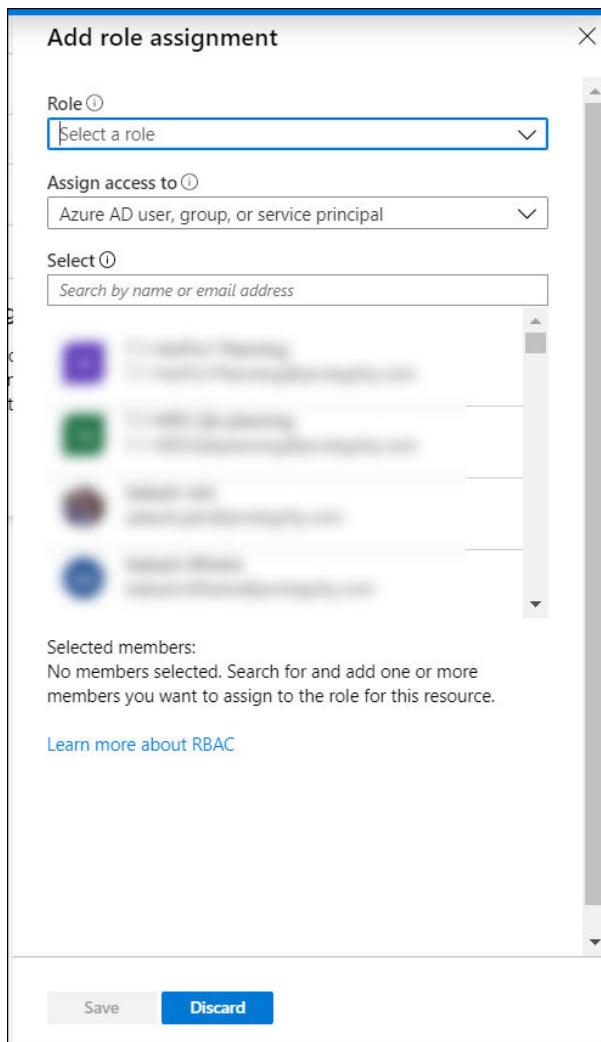
The **Access control (IAM)** screen appears.

The screenshot shows the 'Access control (IAM)' screen for a storage account. The left sidebar includes options like Overview, Activity log, and Access control (IAM). The main area has tabs for Check access, Role assignments, Deny assignments, Classic administrators, and Roles. Under 'Check access', there's a search bar for 'Find' (set to 'Azure AD user, group, or service principal') and a 'Search by name or email address' input field. To the right, there are three main sections: 'Add a role assignment' (with a 'Add' button), 'View role assignments' (with a 'View' button), and 'View deny assignments' (with a 'View' button).

- 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.

New container

Name *

Public access level ⓘ

Private (no anonymous access)

OK **Cancel**

- 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.

| Name ↑↓ | Type ↑↓ | Kind ↑↓ | Resource group ↑↓ | Location ↑↓ | Subscription ↑↓ |
|--------------------------------|-----------------|-----------|-------------------|-------------|----------------------|
| Showing 1 to 53 of 53 records. | | | | | |
| [checkbox] | Storage account | StorageV2 | [redacted] | East US | Azure Cloud Platform |
| [checkbox] | Storage account | Storage | [redacted] | East US | Azure Cloud Platform |
| [checkbox] | Storage account | Storage | [redacted] | East US | Azure Cloud Platform |
| [checkbox] | Storage account | Storage | [redacted] | East US | Azure Cloud Platform |
| [checkbox] | Storage account | StorageV2 | [redacted] | East US 2 | Azure Cloud Platform |
| [checkbox] | Storage account | StorageV2 | [redacted] | East US 2 | Azure Cloud Platform |
| [checkbox] | Storage account | Storage | [redacted] | East US | Azure Cloud Platform |

Figure 2-26: Storage Accounts Screen

4. Click **Add**.

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

All services > Storage accounts > Create storage account

Create storage account

Basics Networking Advanced Tags Review + create

Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below. [Learn more](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * Azure Cloud Platform

Resource group * Select existing... [Create new](#)

Instance details

The default deployment model is Resource Manager, which supports the latest Azure features. You may choose to deploy using the classic deployment model instead. [Choose classic deployment model](#)

Storage account name * ⓘ

Location * (US) East US 2

Performance ⓘ Standard Premium

Account kind ⓘ StorageV2 (general purpose v2)

Replication ⓘ

Review + create < Previous Next : Networking >

Figure 2-27: Create Storage Account Screen

- 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 <i>(US) US East 2</i> . |

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

- Click **Review + create** to validate the entered information.

Azure validates the information that you have entered.

The screenshot shows the 'Create storage account' page in the Azure portal. At the top, a green bar indicates 'Validation passed'. Below it, tabs for 'Basics', 'Networking', 'Advanced', 'Tags', and 'Review + create' are visible, with 'Review + create' being the active tab. The 'Basics' section contains the following configuration:

| | |
|-----------------------|--|
| Subscription | Azure Cloud Platform |
| Resource group | [REDACTED] |
| Location | (US) East US 2 |
| Storage account name | [REDACTED] |
| Deployment model | Resource manager |
| Account kind | StorageV2 (general purpose v2) |
| Replication | Read-access geo-redundant storage (RA-GRS) |
| Performance | Standard |
| Access tier (default) | Hot |

The 'Networking' section shows 'Connectivity method' as 'Public endpoint (all networks)'. The 'Advanced' section shows 'Secure transfer required' as 'Enabled', 'Hierarchical namespace' as 'Disabled', and 'Blob soft delete' as 'Disabled'. At the bottom, there are 'Create', '< Previous', 'Next >', and 'Download a template for automation' buttons.

7. Click **Create** to create the storage account.

The storage account is created and the following screen appears.

The screenshot shows the 'Microsoft.StorageAccount-20191119163820 - Overview' page. The left sidebar has 'Overview' selected, along with 'Inputs', 'Outputs', and 'Template'. The main area displays a message: 'Your deployment is complete'. It shows deployment details: Deployment name: Microsoft.StorageAccount-20191119163820, Subscription: Azure Cloud Platform, Resource group: [REDACTED]. It also shows deployment metadata: Start time: 11/19/2019, 4:55:48 PM and Correlation ID: 11535a6a-5a33-465d-9f3c-f51ab1f507bc. There are links for 'Deployment details (Download)', 'Next steps', and 'Go to resource'.

Figure 2-28: Storage Account Overview Screen

8. Click **Go to resource**.

The **Overview** screen appears, displaying the details of your storage account.

The screenshot shows the 'Overview' section of a Microsoft Storage Account. Key details include:

- Resource group:** (change) [dropdown]
- Status:** Primary: Available, Secondary: Available
- Location:** East US 2, Central US
- Subscription:** Azure Cloud Platform
- Subscription ID:** 62ec138e-7687-4546-bbc1-c88656954b9a
- Tags:** Click here to add tags
- Performance/Access tier:** Standard/Hot
- Replication:** Read-access geo-redundant storage (RA-GRS)
- Account kind:** StorageV2 (general purpose v2)

Figure 2-29: Storage Account Overview Screen

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

The **Access Keys** screen appears.

The screenshot shows the 'Access keys' section of a Microsoft Storage Account. It displays two sets of keys:

- key1** (Key): A redacted connection string.
- key2** (Key): A redacted connection string.

A note at the top states: "When you regenerate your access keys, you must update any Azure resources and applications that access this storage account to use the new keys. This action will not interrupt access to disks from your virtual machines." Below the keys, there are 'Key' and 'Connection string' fields for each key.

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.

The screenshot shows the 'File shares' section of a Microsoft Storage Account. It lists three file shares:

| Name | Modified | Quota | ... |
|-----------------|------------------------|-------|-----|
| [redacted] | 5/14/2020, 12:37:35 PM | 1 GiB | ... |
| test-new-share | 5/14/2020, 12:34:14 PM | 2 GiB | ... |
| test-protection | 5/14/2020, 12:55:44 PM | 1 GiB | ... |

- b. Click **File share**.

The **New file share** screen appears.



The dialog box is titled "New file share". It contains two input fields: "Name *" and "Quota ⓘ". The "Name" field is empty. The "Quota" field contains "GiB". At the bottom of the dialog are two buttons: "Create" and "Discard".

- 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.

| Name ↑↓ | Type ↑↓ | Resource group ↑↓ | Kubernetes version ↑↓ | Location ↑↓ | Subscription ↑↓ | ... |
|------------|--------------------|-------------------|-----------------------|-------------|----------------------|-----|
| [redacted] | Kubernetes service | [redacted] | 1.14.8 | East US 2 | Azure Cloud Platform | ... |
| [redacted] | Kubernetes service | [redacted] | 1.14.7 | East US 2 | Azure Cloud Platform | ... |
| [redacted] | Kubernetes service | [redacted] | 1.14.7 | East US | Azure Cloud Platform | ... |
| [redacted] | Kubernetes service | [redacted] | 1.14.8 | East US | Azure Cloud Platform | ... |
| [redacted] | Kubernetes service | [redacted] | 1.13.11 | East US | Azure Cloud Platform | ... |
| [redacted] | Kubernetes service | [redacted] | 1.13.12 | East US 2 | Azure Cloud Platform | ... |

4. Click **Add**.

The **Create Kubernetes cluster** screen appears.

Create Kubernetes cluster

Basics [Scale](#) [Authentication](#) [Networking](#) [Monitoring](#) [Tags](#) [Review + create](#)

Azure Kubernetes Service (AKS) manages your hosted Kubernetes environment, making it quick and easy to deploy and manage containerized applications without container orchestration expertise. It also eliminates the burden of ongoing operations and maintenance by provisioning, upgrading, and scaling resources on demand, without taking your applications offline. [Learn more about Azure Kubernetes Service](#)

Project details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * [①](#)

Resource group * [①](#) [Create new](#)

Cluster details

Kubernetes cluster name * [①](#)

Region * [①](#)

Kubernetes version * [①](#)

DNS name prefix * [①](#)

Primary node pool

The number and size of nodes in the primary node pool in your cluster. For production workloads, at least 3 nodes are required.

[Review + create](#) [< Previous](#) [Next : Scale >](#)

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 Sample Application container 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 a Kubernetes service account that will install the Helm charts for deploying the Sample Application container.

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.

| Name | Type | Resource group | Kubernetes version | Location | Subscription |
|------------|--------------------|----------------|--------------------|-----------|----------------------|
| [REDACTED] | Kubernetes service | [REDACTED] | 1.14.8 | East US 2 | Azure Cloud Platform |
| [REDACTED] | Kubernetes service | [REDACTED] | 1.14.7 | East US 2 | Azure Cloud Platform |
| [REDACTED] | Kubernetes service | [REDACTED] | 1.14.7 | East US | Azure Cloud Platform |
| [REDACTED] | Kubernetes service | [REDACTED] | 1.14.8 | East US | Azure Cloud Platform |
| [REDACTED] | Kubernetes service | [REDACTED] | 1.13.11 | East US | Azure Cloud Platform |
| [REDACTED] | Kubernetes service | [REDACTED] | 1.13.12 | East US 2 | Azure Cloud Platform |

- Select the Kubernetes cluster that you have created in the section [Creating a Kubernetes Cluster](#).
- Navigate to **Policies**.

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

- Click **Enable add-on**.

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

After you enable the policy, 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.

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

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

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

- 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.

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

The **Policy** screen appears.

The screenshot shows the Azure Policy Overview screen. The left pane includes links for Home, Getting started, Compliance, Remediation, Authoring (Assignments, Definitions, Exemptions), and Related Services (Blueprints (preview), Resource Graph, User privacy). The main area displays the following metrics:

- Overall resource compliance: 33% (1 out of 3)
- Resources by compliance state:
 - 1 - Compliant (Green)
 - 0 - Exempt (Yellow)
 - 2 - Non-compliant (Red)
 - 0 - Conflicting (Orange)
- Non-compliant initiatives: 1 (out of 2)
- Non-compliant policies: 14 (out of 158)

A banner at the top right says "Announcing Policy Exemption! Learn more https://aka.ms/AzPolicyExemption". On the far right, there's a "LEARN MORE" section with links to "Learn about Policy" and "Onboarding tutorial".

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

The **Definitions** screen appears.

The screenshot shows the Azure Policy | Definitions screen. The left pane is identical to the previous screenshot. The main area shows a table of policy definitions:

| Name | Definition location | Policies | Type | Definition type | Category |
|---------------------------------|----------------------|----------|----------|-----------------|----------------------|
| PtyAKSInitiative | Azure Cloud Platform | 7 | Custom | Initiative | Kubernetes |
| [Preview]: NIST SP 800-171 R2 | | 78 | Built-in | Initiative | Regulatory Compliant |
| Audit machines with insecu... | | 9 | Built-in | Initiative | Guest Configuration |
| IRS1075 September 2016 | | 62 | Built-in | Initiative | Regulatory Compliant |
| [Preview]: Deploy prerequisi... | | 4 | Built-in | Initiative | Guest Configuration |
| CIS Microsoft Azure Founda... | | 87 | Built-in | Initiative | Regulatory Compliant |
| Enable Monitoring in Azure ... | | 151 | Built-in | Initiative | Security Center |
| [Preview]: Australian Govern... | | 62 | Built-in | Initiative | Regulatory Compliant |

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

The policy definition details screen appears.

The screenshot shows the 'Do not allow privileged containers in Kubernetes cluster' policy definition in the Azure portal. The 'Essentials' tab is selected, displaying the policy's name, description, available effects (audit, deny, disabled), category (Kubernetes), definition location, definition ID, type (Built-in), and mode (Microsoft.Kubernetes.Data). The 'Definition' tab is active, showing the JSON code for the policy:

```
1  {
2   "properties": {
3     "displayName": "Do not allow privileged containers in Kubernetes cluster",
4     "policyType": "BuiltIn",
5     "mode": "Microsoft.Kubernetes.Data",
6     "description": "This policy does not allow privileged containers creation in a Kubernetes cluster. This policy is generally available for Kubernetes clusters.",
7     "metadata": {
8       "version": "1.0",
9       "category": "Kubernetes"
10    },
11  }
```

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.

All services > Policy > Do not allow privileged containers in Kubernetes cluster >

Do not allow privileged containers in Kubernetes cluster

Assign policy

Basics Parameters Remediation Review + create

Scope

Scope [Learn more about setting the scope *](#)

 ...

Exclusions

Optionally select resources to exclude from the policy assignment.

...

Basics

Policy definition

Do not allow privileged containers in Kubernetes cluster

Assignment name * (i)

Do not allow privileged containers in Kubernetes cluster

Review + create

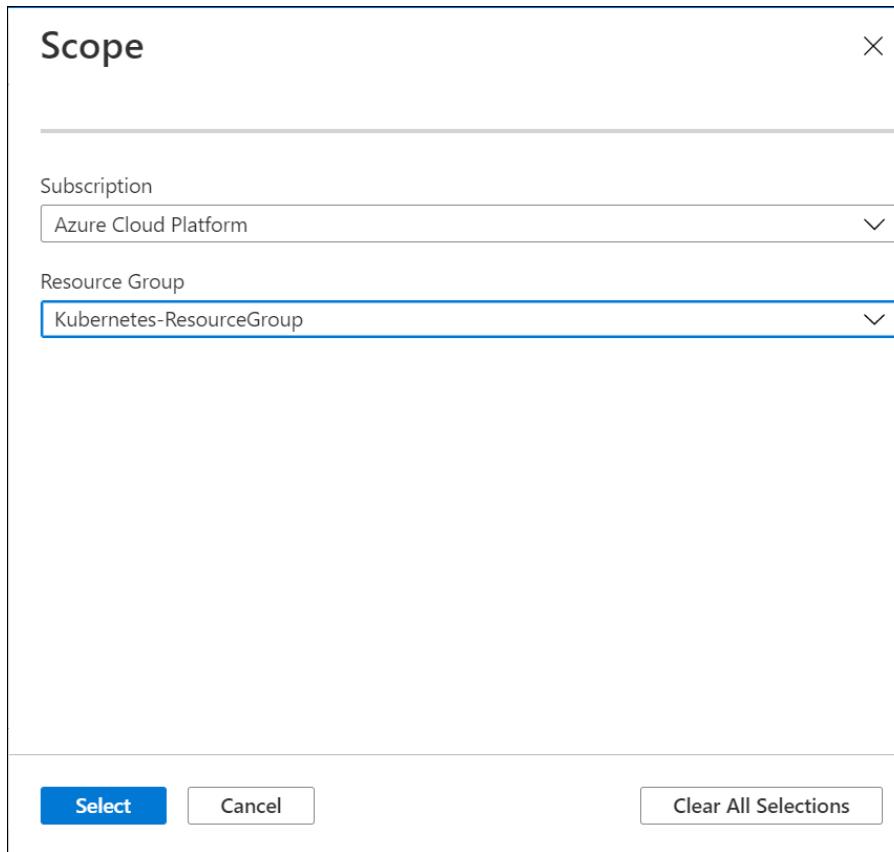
Cancel

Previous

Next

- 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.

All services > Policy > Do not allow privileged containers in Kubernetes cluster >

Do not allow privileged containers in Kubernetes cluster

Assign policy

Basics **Parameters** Remediation Review + create

Specify parameters for this policy assignment.

Effect * ⓘ

deny



Namespace exclusions * ⓘ

["kube-system", "gatekeeper-system", "azure-arc"]



Review + create

Cancel

Previous

Next

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 Sample Application containers.

Important:

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

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

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

For example:

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

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

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

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

For example:

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

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

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

For example:

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

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

You can obtain the context by running the following command:

```
kubectl config current-context
```

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

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

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

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

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

For example:

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

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

2.7.3 Retrieving the Policy from the ESA

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

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

► To retrieve the policy package from the ESA:

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

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

The command returns the following output.

```
{
  "version" : "1.1.0",
  "buildVersion" : "1.1.0+7.g307f.1.1",
  "pepVersions" : [ "1.1.0+85", "1.2.0+16" ]
}
```

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

For more information about the installation package, refer to the section [Downloading the Installation Package](#).

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

```
curl -v -k -u "<ESA user>:<ESA password>" https://<ESA IP>/pty/v1/imp/export -H
"Content-Type: application/json" -o policy_package_key.tgz --data
'{"name":"policypackage","pepVersion":"1.2.0+16","dataStoreName": "<Value>","emptyString": "<Value>","caseSensitive": "<Value>","kek": {"pkcs5": {"password": "<Password>","salt": "<Salt>"}}}'
```

For example:

```
curl -v -k -u "IMPUUser:<Password>" https://10.49.1.218/pty/v1/imp/export -H
"Content-Type: application/json" -o policy_package_key.json --data
'{"name":"policypackage","pepVersion":"1.2.0+16","dataStoreName": "<Value>","emptyString": "NULL","caseSensitive": "YES","kek": {"pkcs5": {"password": "<Password>","salt": "<Salt>"}}}'
```

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

The encrypted policy package is downloaded to your machine.

Important: Ensure that the user is assigned the **Export IMP** permissions through roles.

For more information about assigning permissions to user roles, refer to the section [Managing Roles](#) in the [Appliances Overview 9.1.0.5](#) guide.

For more information about managing users, refer to the section [Managing Users](#) in the [Appliances Overview 9.1.0.5](#) guide.

For more information about the Export API, refer to the section [Sample Immutable Service API - Exporting Policy from a PEP Server Version](#) in the [Protegity APIs, UDFs, Commands Reference Guide 9.1.0.5](#).

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

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

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

```
curl -v -k -u "<ESA user>:<ESA password>" https://<ESA IP>/pty/v1/imp/export -H
"Content-Type: application/json" -o <Policy_package_name>.tgz --data
'{"name":"policypackage","pepVersion":"1.2.0+16","dataStoreName": "<value>","emptySt
```

```
ring": "<value>","caseSensitive": "<value>","kek": {"publicKey": {"value": "Public Key of the Azure Key Vault used to encrypt the policy", "algorithm": "Supported public key algorithm"}}}'
```

For example:

```
curl -v -k -u "IMPUUser:<password>" https://10.49.1.218/pty/v1/imp/export -H "Content-Type: application/json" -o policy_package_key.tgz --data '{"name": "policypackage", "pepVersion": "1.2.0+16", "dataStoreName": "<value>", "emptyString": "<value>","caseSensitive": "<value>","kek": {"publicKey": {"value": "-----BEGIN PUBLIC KEY-----\nMIIBBojANBgkqhkiG9w0BAQEFAAOCAgEAMIIBigKCAYEAjwxqqrrNQ1kV84GEFYLR\\nVK/SyOkGTs7kxEImEYMGtugSal2G9m7hFdV1WhHH/OvLDHwdOYe6GLXHVwycfbM/\nS6pzQPS5ujzyJEaLWYAfcRgAHi9g8GvazDiv34Z+VO5FRbJzP9u9/23J4hExc+e1\\nTezKsoBj6Ypx/zBkE4dGqdamfHJUF3ptB82nhJT/Pth15ejL/SVrD/q8sORU8380\\negcIfQYmd5zizJgbWLVzFaM9QZXH9hD/0JxAhqfHP3Fnhm5MDVdg7D0SQkufIjf\\nu7djMy2I3ZRMgnkGxjq8PuBUFaH2s6DvAa4e6K0ppGjFoByVV e2tumrUVEvJg2EM\\ndD/P115kzLe1bDKjxrI8T1D8cWuGh43wf1xC+uRzsfoMSgwDWdpBmaOPF+q2k0d4\\nrQWKigZw71mxYADPKBAU UwMgD/aREuvTDlpvp10Vk2Y5i/4Xx/fK7GV5DCdy9TFM\\nZgzAA/O9z5tuVzqcgodMu06+iqtXdtUoOafA+BTvSIXLAGMBAAE=\\n-----END PUBLIC KEY-----", "algorithm": "RSA_OAEP_SHA256"}}}'
```

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

Note:

You can obtain the public key by running the following command.

```
az keyvault key download --id <keyid> -e PEM -f <keyFilePath>
```

The policy package is downloaded to your machine.

Important: The Immutable Service API supports the following algorithms:

- RSA_OAEP_SHA1
- RSA_OAEP_SHA256
- RSA_OAEP_SHA512
- RSA_PKCS1_v1_5

However, the Azure Key Vault currently supports only three out of the four algorithms.

Ensure that you specify one of the following algorithms that is supported by the Azure Key Vault:

- RSA_OAEP_SHA1
- RSA_OAEP_SHA256
- RSA_PKCS1_v1_5

For more information about the algorithms supported by the Azure Key Vault, refer to the [Azure product documentation](#).

4. Copy the policy package file to an Azure Storage Container or Azure File Share, as required.



2.7.4 Setting up the Environment for Deploying the Sample Application Container

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

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

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

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

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

For example:

```
kubectl create namespace iap
```

2. Perform the following steps if you want to use NGINX as the Ingress Controller.

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/nginx-ingress --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/nginx-ingress --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 regcred --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.

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

kubectl apply -f pv.yaml

A persistent volume resource is created.

- 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.

2.7.5 Deploying the Sample Application Container on the Kubernetes Cluster

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

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

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, 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](#).

In the *spring-apjava* directory, 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

...
.

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

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

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

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

## specify k8s secret name of azure cloud credentials to access blob storage or key vault
azAccessSecrets:

## start with blue deployment first
deploymentInUse: blue

## Policy metadata for Blue Deployments
blueDeployment:
  enabled: true
  policy:
    # absolute path in S3 from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/pathToPolicy"
    # absolute path of policy dump file in Azure Blob store.
    # <Azure blob> "https://<account name>.blob.core.windows.net/<container name>/<pathToPolicy>"
    # absolute path in PV mount from where the policy dump file will be fetched.
    # <VOLUME> "file://var/data/ptypolicy/pathInVolumeToPolicy"
    # relative path in PV mount from where the policy dump file will be fetched.
    # it will prepend the '/var/data/ptypolicy' to below path
    # <VOLUME> "file:///pathInVolumeToPolicy"
    # <VOLUME> "pathInVolumeToPolicy" # without any leading '/'
    filePath: "POLICY_PATH"

## Policy metadata for Green Deployments
greenDeployment:
  enabled: false
  policy:
    # absolute path in S3 from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/pathToPolicy"
    # absolute path of policy dump file in Azure Blob store.
    # <Azure blob> "https://<account name>.blob.core.windows.net/<container name>/<pathToPolicy>"
    # absolute path in PV mount from where the policy dump file will be fetched.
    # <VOLUME> "file://var/data/ptypolicy/pathInVolumeToPolicy"
    # relative path in PV mount from where the policy dump file will be fetched.
    # it will prepend the '/var/data/ptypolicy' to below path
    # <VOLUME> "file:///pathInVolumeToPolicy"
    # <VOLUME> "pathInVolumeToPolicy" # without any leading '/'
    filePath: "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:

  # 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:
    ##AWS
    #service.beta.kubernetes.io/aws-load-balancer-internal: "true"
    ##AZURE
    #service.beta.kubernetes.io/azure-load-balancer-internal: "true"
    ##GCP
    #networking.gke.io/load-balancer-type: "Internal"

  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

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

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

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

```



```
#   hosts:
#     - prod.example.com
```

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

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

For more information about Helm charts and their default values, refer to the section [Appendix A: Sample Application Helm Chart Details](#).

| Field | Description |
|--------------------|---|
| podSecurityContext | <p>Specify the privileges and access control settings for the pod.</p> <p>The default values are set as follows:</p> <ul style="list-style-type: none"> • <i>runAsUser</i> - 1000 • <i>runAsGroup</i> - 1000 • <i>fsGroup</i> - 1000 • <i>supplementalGroups</i> - 1000 <p>Note: Ensure that at any given point of time, the value of at least one of the four parameters must be <i>1000</i>. This ensures that the pod has the required permissions to access the <i>pty</i> directories in the containers.</p> <p>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.</p> <p>For more information about the parameters, refer to the section PodSecurityContext v1 Core in the Kubernetes API documentation.</p> |
| pvcName | <p>Specify the name of the Persistent Volume Claim where you want to store the policy package.</p> <p>If you want to use the Azure Storage Container for storing the policy package, then leave the value of this field blank.</p> <p>Important: This field is required if you want to store the immutable policy package in the Azure File Share.</p> |
| privateKeySource | <p>Specify one of the following methods to encrypt the policy package:</p> <ul style="list-style-type: none"> • <i>PKCS5</i> - Use Passphrase-Based Encryption for encrypting the policy package • <i>AZ_VAULT</i> - Use Azure Key Vault for encrypting the policy package |
| pbeSecrets | <p>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 <i>pbeSecrets</i> parameter to decrypt the policy.</p> <p>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</p> |



| Field | Description |
|--------------------------------|---|
| | <p>characters). The passphrase must also have a minimum value of entropybits as <i>30</i> and a minimum strength of <i>0.66</i>. Entropybits defines the randomness of the password and the strength defines the complexity of the password.</p> <p>For more information about the password strength, refer to the section Password Strength.</p> |
| privateKeySecret | <p>Specify the Kubernetes secret created that contains the private key of the Azure Key Vault, which encrypted the policy. The Sample Application container uses the value specified in the <i>privateKeySecret</i> parameter to decrypt the policy.</p> <p>This parameter is applicable if you have specified <i>AZ_VAULT</i> as the value of the <i>privateKeySource</i> parameter.</p> |
| azAccessSecrets | <p>Specify the value of the secret that you created in <i>step 4b</i> of the section Setting up the Environment for Deploying the Sample Application Container. This is used by the Sample Application container to access the Azure credentials required to download the policy package from the Azure storage container to the shared memory and decrypt the policy package if you have encrypted it using the Azure Key Vault.</p> |
| deploymentInUse | <p>Specify the value as <i>blue</i>. This indicates that the <i>blue</i> deployment strategy is in use.</p> |
| blueDeployment/enabled | <p>Specify the value as <i>true</i> to enable the <i>blue</i> deployment strategy.</p> |
| blueDeployment/policy/filepath | <p>Specify the path in the persistent volume or the object store where you have stored the policy package.</p> <p>For example, if you are using Azure File Share as the persistent volume, then you can specify the file path as <i>file:///var/data/ptypolicy/pathInVolumeToPolicy</i>.</p> <p>If you have stored the policy package in an Object Store, such as an Azure Storage Container, then you can specify the storage container name as the first name in the <i>filepath</i> field.</p> <p>For example, you can specify the value of the <i>filepath</i> field, as <i>https://<accountName>.blob.core.windows.net/<container name>/<pathToPolicy></i>.</p> <p>Important:</p> <p>Use only uppercase and lowercase letters, numbers, dot, and underscore in the file path. Do not use special characters in the file path.</p> |
| | <p>Note:</p> <p>This value is case-sensitive.</p> |
| greenDeployment/enabled | <p>Specify the value as <i>false</i>. While deploying Release 1, the <i>green</i> deployment strategy is always disabled.</p> |



| Field | Description |
|--|--|
| 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 <i>true</i> to enable the ingress resource in Kubernetes. The ingress resource acts as a load balancer. It enables the simultaneous deployment of <i>blue</i> and <i>green</i> 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, <i>nginx-iap</i> , if you want to use the NGINX Ingress Controller. |
| nginx.ingress.kubernetes.io/auth-tls-verify-client | Set the value of this field to <i>on</i> , if you want to enable TLS mutual authentication between the REST API client and the Azure load balancer. By default, this field is <i>disabled</i> . |
| 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 <i>disabled</i> . 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 <i>/</i> . 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 <i>/</i> . 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. |

5. 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
```

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

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

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

For example:

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

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

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

For example:

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

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

2.7.6 Upgrading the Helm Charts

This section describes the steps for upgrading the Helm charts.

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

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

► To upgrade the Helm charts:

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

For more information about the extracted Helm charts, refer to the [step 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

...
.

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

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

```

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

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

## specify k8s secret name of azure cloud credentials to access blog storage or key vault
azAccessSecrets:

## start with blue deployment first
deploymentInUse: blue

## Policy metadata for Blue Deployments
blueDeployment:
  enabled: true
  policy:
    # absolute path in S3 from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/pathToPolicy"
    # absolute path of policy dump file in Azure Blob store.
    # <Azure blob> "https://<account name>.blob.core.windows.net/<container name>/
<pathToPolicy>"
    # absolute path in PV mount from where the policy dump file will be fetched.
    # <VOLUME> "file://var/data/ptypolicy/pathInVolumeToPolicy"
    # relative path in PV mount from where the policy dump file will be fetched.
    # it will prepend the '/var/data/ptypolicy' to below path
    # <VOLUME> "file:///pathInVolumeToPolicy"
    # <VOLUME> "pathInVolumeToPolicy" # without any leading '/'
    filePath: "POLICY_PATH"

## Policy metadata for Green Deployments
greenDeployment:
  enabled: false
  policy:
    # absolute path in S3 from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/pathToPolicy"
    # absolute path of policy dump file in Azure Blob store.
    # <Azure blob> "https://<account name>.blob.core.windows.net/<container name>/
<pathToPolicy>"
    # absolute path in PV mount from where the policy dump file will be fetched.
    # <VOLUME> "file://var/data/ptypolicy/pathInVolumeToPolicy"
    # relative path in PV mount from where the policy dump file will be fetched.
    # it will prepend the '/var/data/ptypolicy' to below path
    # <VOLUME> "file:///pathInVolumeToPolicy"
    # <VOLUME> "pathInVolumeToPolicy" # without any leading '/'
    filePath: "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:

  # 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:
    ##AWS
    #service.beta.kubernetes.io/aws-load-balancer-internal: "true"
    ##AZURE
    #service.beta.kubernetes.io/azure-load-balancer-internal: "true"
    ##GCP

```

```

#networking.gke.io/load-balancer-type: "Internal"

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

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

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

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

```

In 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 A: Sample Application Helm Chart Details](#).

| Field | Description |
|--------------------|---|
| podSecurityContext | Specify the privileges and access control settings for the pod. |

| Field | Description |
|------------------|--|
| | <p>The default values are set as follows:</p> <ul style="list-style-type: none"> • <i>runAsUser</i> - 1000 • <i>runAsGroup</i> - 1000 • <i>fsGroup</i> - 1000 • <i>supplementalGroups</i> - 1000 <p>Note: Ensure that at any given point of time, the value of at least one of the four parameters must be <i>1000</i>. This ensures that the pod has the required permissions to access the <i>pty</i> directories in the containers.</p> <p>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.</p> <p>For more information about the parameters, refer to the section PodSecurityContext v1 Core in the Kubernetes API documentation.</p> |
| pvcName | <p>Specify the name of the Persistent Volume Claim where you want to store the policy package.</p> <p>If you want to use the Azure Storage Container for storing the policy package, then leave the value of this field blank.</p> <p>Important: This field is required if you want to store the immutable policy package in the Azure File Share.</p> |
| privateKeySource | <p>Specify one of the following methods to encrypt the policy package:</p> <ul style="list-style-type: none"> • <i>PKCS5</i> - Use Passphrase-Based Encryption for encrypting the policy package • <i>AZ_VAULT</i> - Use Azure Key Vault for encrypting the policy package |
| pbeSecrets | <p>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 <i>pbeSecrets</i> parameter to decrypt the policy.</p> <p>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 <i>30</i> and a minimum strength of <i>0.66</i>. Entropybits defines the randomness of the password and the strength defines the complexity of the password.</p> <p>For more information about the password strength, refer to the section Password Strength.</p> |
| privateKeySecret | <p>Specify the Kubernetes secret created that contains the private key of the Azure Key Vault, which encrypted the policy. The</p> |



| Field | Description |
|---------------------------------|--|
| | <p>Sample Application container uses the value specified in the <i>privateKeySecret</i> parameter to decrypt the policy.</p> <p>This parameter is applicable if you have specified <i>AZ_VAULT</i> as the value of the <i>privateKeySource</i> parameter.</p> |
| azAccessSecrets | <p>Specify the value of the secret that you have created using the <i>az_credentials.json</i> file. This is used by the Sample Application container to access the Azure credentials required to download the policy package from the Azure storage container to the shared memory and decrypt the policy package if you have encrypted it using the Azure Key Vault.</p> |
| deploymentInUse | <p>Specify the value as <i>blue</i>. This indicates that the <i>blue</i> deployment strategy is in use.</p> |
| greenDeployment/enabled | <p>Specify the value as <i>true</i>. While upgrading the release to Release 2, the <i>green</i> deployment strategy is enabled.</p> |
| greenDeployment/policy/filepath | <p>Specify the path in the persistent volume or the object store where you have stored the policy package.</p> <p>For example, if you are using Azure File Share as the persistent volume, then you can specify the file path as <i>file:///var/data/ptypolicy/pathInVolumeToPolicy</i>.</p> <p>If you have stored the policy package in an Object Store, such as an Azure Storage Container, then you can specify the storage container name as the first name in the <i>filepath</i> field.</p> <p>For example, you can specify the value of the <i>filepath</i> field, as <i>https://<accountName>.blob.core.windows.net/<container name>/<pathToPolicy></i>.</p> <div data-bbox="882 1184 1529 1368" style="background-color: #f0e6e6; padding: 10px;"> <p>Important:</p> <p>Use only uppercase and lowercase letters, numbers, dot, and underscore in the file path. Do not use special characters in the file path.</p> </div> <div data-bbox="882 1396 1529 1537" style="background-color: #e0f2e0; padding: 10px;"> <p>Note:</p> <p>This value is case-sensitive.</p> </div> |
| springappService/name | <p>Specify a name for the tunnel to distinguish between ports.</p> |
| springappService/port | <p>Specify the port number on which you want to expose the Kubernetes service externally.</p> |
| springappService/targetPort | <p>Specify the port on which the Sample application is running inside the Docker container.</p> |
| ingress/enabled | <p>Specify this value as <i>true</i> to enable the ingress resource in Kubernetes. The ingress resource acts as a load balancer. It enables the simultaneous deployment of <i>blue</i> and <i>green</i> strategy by exposing both the deployments on the same external IP address, but on different URL paths.</p> |

| Field | Description |
|--|--|
| ingress/ingressClassName | Specify the value of the external ingress controller, if any. For example, specify, <i>nginx-iap</i> , 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 <i>disabled</i> . |
| 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 <i>disabled</i> . 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. |
| 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 shut down 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.

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

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

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

Before you begin

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

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

For more information about invoking the Immutable Service APIs, refer to the section [Retrieving the Policy from the ESA](#).

- If you have used Azure Key Vault to encrypt the policy package, then ensure that you have created the private key secret, which is used by the Sample Application container to decrypt the policy.
- If you have used Passphrase-Based Encryption to encrypt the policy package, then ensure that you have created the passphrase secret, which is used by the Sample Application container to decrypt the policy.
- Ensure that you have created a custom image for the Sample Application container or the Custom container using the Dockerfile provided in the v9.1.0.5 installation package.



For more information about building a custom image, refer to the section [Creating Custom Images Using Dockerfile](#).

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

For more information about the extracted Helm charts, refer to the [step 9](#) of the section [Initializing the Linux Instance](#).

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

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

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

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

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

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

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

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

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

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

## specify k8s secret name of azure cloud credentials to access blob storage or key vault
azAccessSecrets:

## start with blue deployment first
deploymentInUse: blue

## Policy metadata for Blue Deployments
blueDeployment:
    enabled: true
    policy:
        # absolute path in S3 from where the policy dump file will be fetched.
        # <S3> "s3://bucketName/pathToPolicy"
        # absolute path of policy dump file in Azure Blob store.
```



```

# <Azure blob> "https://<account name>.blob.core.windows.net/<container name>/
<pathToPolicy>
    # absolute path in PV mount from where the policy dump file will be fetched.
    # <VOLUME> "file://var/data/ptypolicy/pathInVolumeToPolicy"
    # relative path in PV mount from where the policy dump file will be fetched.
    # it will prepend the '/var/data/ptypolicy' to below path
    # <VOLUME> "file:///pathInVolumeToPolicy"
    # <VOLUME> "pathInVolumeToPolicy" # without any leading '/'
    filePath: "POLICY_PATH"

## Policy metadata for Green Deployments
greenDeployment:
    enabled: false
    policy:
        # absolute path in S3 from where the policy dump file will be fetched.
        # <S3> "s3://bucketName/pathToPolicy"
        # absolute path of policy dump file in Azure Blob store.
        # <Azure blob> "https://<account name>.blob.core.windows.net/<container name>/
        <pathToPolicy>
            # absolute path in PV mount from where the policy dump file will be fetched.
            # <VOLUME> "file://var/data/ptypolicy/pathInVolumeToPolicy"
            # relative path in PV mount from where the policy dump file will be fetched.
            # it will prepend the '/var/data/ptypolicy' to below path
            # <VOLUME> "file:///pathInVolumeToPolicy"
            # <VOLUME> "pathInVolumeToPolicy" # without any leading '/'
            filePath: "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:

    # 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:
        ##AWS
        #service.beta.kubernetes.io/aws-load-balancer-internal: "true"
        ##AZURE
        #service.beta.kubernetes.io/azure-load-balancer-internal: "true"
        ##GCP
        #networking.gke.io/load-balancer-type: "Internal"

    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

```

```

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

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

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

```

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

For more information about Helm charts and their default values, refer to the section [Appendix A: Sample Application Helm Chart Details](#).

| Field | Description |
|------------------|---|
| pvcName | Specify the name of the Persistent Volume Claim where you want to store the policy snapshot. Leave the value of this field blank if you want to use Azure Storage Container for storing the policy snapshot. |
| privateKeySource | Specify one of the following methods to encrypt the policy package: <ul style="list-style-type: none"> • <i>PKCS5</i> - Use Passphrase-Based Encryption for encrypting the policy package • <i>AZ_VAULT</i> - Use Azure Key Vault for encrypting the policy package |
| pbeSecrets | Specify the Kubernetes secret that contains the passphrase and the salt that were used to generate key, which encrypted the policy. The Sample Application container uses the value specified in the <i>pbeSecrets</i> parameter to decrypt the policy. <div style="background-color: #fce4ec; padding: 10px; margin-top: 10px;"> Important: Ensure that the password complexity meets the following requirements: <ul style="list-style-type: none"> • The password must contain a minimum of 10 characters and a maximum of 128 characters </div> |

| Field | Description |
|--|--|
| | <ul style="list-style-type: none"> The user should be able to specify Unicode characters, such as Emoji, in the password The user should avoid commonly used passwords, such as, <i>123456789, 1111111, password, and qwertyuiop</i> |
| <code>privateKeySecret</code> | <p>Specify the Kubernetes secret created that contains the private key of the Azure Key Vault, which encrypted the policy. The Sample Application container uses the value specified in the <code>privateKeySecret</code> parameter to decrypt the policy.</p> <p>This parameter is applicable if you have specified <code>AZ_VAULT</code> as the value of the <code>privateKeySource</code> parameter.</p> |
| <code>azAccessSecrets</code> | <p>Specify the value of the secret that you have created using the <code>az_credentials.json</code> file. This is used by the Sample Application container to access the Azure credentials required to download the policy package from the Azure storage container to the shared memory and decrypt the policy package if you have encrypted it using the Azure Key Vault.</p> |
| <code>deploymentInUse</code> | <p>Specify the value as <i>blue</i>. This indicates that the blue deployment strategy is in use.</p> |
| <code>greenDeployment\enabled</code> | <p>Specify the value as <i>true</i>. While upgrading the release to Release 2, the <i>green</i> deployment strategy is enabled.</p> |
| <code>greenDeployment/policy/filepath</code> | <p>Specify the path in the persistent volume or the object store where you have stored the policy package.</p> <p>For example, if you are using Azure File Share as the persistent volume, then you can specify the file path as <code>file:///var/data/ptypolicy/pathInVolumeToPolicy</code>.</p> <p>If you have stored the policy package in an Object Store, such as an Azure Storage Container, then you can specify the storage container name as the first name in the <code>filepath</code> field.</p> <p>For example, you can specify the value of the <code>filepath</code> field, as <code>https://<accountName>.blob.core.windows.net/<container name>/<pathToPolicy></code>.</p> <p>Important: Use only uppercase and lowercase letters, numbers, dot, and underscore in the file path. Do not use special characters in the file path.</p> |
| | <p>Note: This value is case-sensitive.</p> |
| <code>springappService/name</code> | <p>Specify a name for the tunnel to distinguish between ports.</p> |
| <code>springappService/port</code> | <p>Specify the port number on which you want to expose the Kubernetes service externally.</p> |

| Field | Description |
|---|--|
| springappService/targetPort | Specify the port on which the Sample application is running inside the Docker container. |
| ingress/enabled | Specify this value as <i>true</i> to enable the ingress resource in Kubernetes. The ingress resource acts as a load balancer. It enables the simultaneous deployment of <i>blue and green</i> strategy by exposing both the deployments on the same external IP address, but on different URL paths. |
| ingress/annotations/kubernetes.io/ingress.class | <p>Specify the value of the external ingress controller, if any. For example, specify, <i>nginx</i> if you want to use the NGINX Ingress Controller.</p> <p>Important: In the v7.1 MR4 release, if you have deployed the NGINX Ingress Controller without specifying this parameter, then ensure that you comment out this parameter in the <i>values.yaml</i> file. Otherwise, the requests that are sent from the client application to the NGINX Ingress Controller will not be forwarded to the pods in the Sample Application container.</p> |
| ingress/hosts/host | Specify the hostname of the ingress resource. If you are using the external IP address of the ingress resource for performing the security operations, then leave this field blank. |
| ingress/hosts/httpPaths/port | Specify the port on which the Sample application is running inside the Docker container. |
| ingress/hosts/httpPaths/prod | If you are using the NGINX Ingress Controller as an SSL passthrough, then you must specify the value of this field as /. This ensures that the data traffic from the REST API client is redirected to the required production or staging URL based on the hostname value. |
| ingress/hosts/httpPaths/staging | If you are using the NGINX Ingress Controller as an SSL passthrough, then you must specify the value of this field as /. This ensures that the data traffic from the REST API client is redirected to the required production or staging URL based on the hostname value. |

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

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

For example:

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

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

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

2.8 Deploying the Containers without an Azure Policy and RBAC

This section describes the steps that you need to perform for deploying the Sample Application container 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 Retrieving the Policy from the ESA

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

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

► To retrieve the policy package from the ESA:

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

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

The command returns the following output.

```
{
  "version" : "1.1.0",
  "buildVersion" : "1.1.0+7.g307f.1.1",
  "pepVersions" : [ "1.1.0+85", "1.2.0+16" ]
}
```

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

For more information about the installation package, refer to the section [Downloading the Installation Package](#).

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

```
curl -v -k -u "<ESA user>:<ESA password>" https://<ESA IP>/pty/v1/imp/export -H
"Content-Type: application/json" -o policy_package_key.tgz --data
'{"name":"policypackage","pepVersion":"1.2.0+16","dataStoreName":"<Value>","emptyString":"<Value>","caseSensitive":"<Value>","kek":{"pkcs5":
{"password":"<Password>","salt":"<Salt>"}}}'
```

For example:

```
curl -v -k -u "IMPUUser:<Password>" https://10.49.1.218/pty/v1/imp/export -H
"Content-Type: application/json" -o policy_package_key.json --data
'{"name":"policypackage","pepVersion":"1.2.0+16","dataStoreName":"<Value>","emptyString":"NULL","caseSensitive":"YES","kek":{"pkcs5":
{"password":"<Password>","salt":"<Salt>"}}}'
```

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

The encrypted policy package is downloaded to your machine.

Important: Ensure that the user is assigned the **Export IMP** permissions through roles.

For more information about assigning permissions to user roles, refer to the section *Managing Roles* in the [Appliances Overview 9.1.0.5](#) guide.

For more information about managing users, refer to the section *Managing Users* in the [Appliances Overview 9.1.0.5](#) guide.

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

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

- The password must contain a minimum of 10 characters and a maximum of 128 characters
- The user should be able to specify Unicode characters, such as Emoji, in the password
- The user should avoid commonly used passwords, such as, *123456789, 1111111, password, and qwertyuiop*

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

```
curl -v -k -u "<ESA user>:<ESA password>" https://<ESA IP>/pty/v1/imp/export -H "Content-Type: application/json" -o <Policy_package_name>.tgz --data '{"name":"policypackage","pepVersion":"1.2.0+16","dataStoreName":<value>,"emptyString":<value>,"caseSensitive":<value>,"kek": {"publicKey": {"value": "Public Key of the Azure Key Vault used to encrypt the policy", "algorithm": "Supported public key algorithm"}}}'
```

For example:

```
curl -v -k -u "IMPUUser:<password>" https://10.49.1.218/pty/v1/imp/export -H "Content-Type: application/json" -o policy_package_key.tgz --data '{"name":"policypackage","pepVersion":"1.2.0+16","dataStoreName":<value>,"emptyString":<value>,"caseSensitive":<value>,"kek": {"publicKey": {"value": "-----BEGIN PUBLIC KEY-----\nMIIBBojANBgkqhkiG9w0BAQEFAAOCAy8AMIIBigKCAYEAjwxqqrrNQ1kV84GEFYLR\\nVK/\nSyOkGTs7kxEImEYMGtugSal2G9m7hFdVlWhHH/OvLDHwdOYe6GLXHVwyfcfBm/\nS6pzQPS5ujzyJEaLWYAfCrgAHi9g8GvazDiv34Z+VO5FRbJzP9u9/23J4hExc+e1\\nTezKsoBj6Ypx/\nzBkE4dGqdamfHJUF3ptB82nhJT/Pth15ejL/SVrD/\nq8sORU8380\\negcIfQYmd5ziZZgbWLVzFaM9QZXH9hD/\n0JxAhqfHP3Fnmc5MDVdg7D0SQkufIJf\\nu7djMy2I3ZRMgnkGxjq8PuBUFaH2s6DvAa4e6K0ppGjFoByVV\ne2tumrUVEvJg2EM\\ndD/\nP115kzLe1bDKjxrI8T1D8cWuGh43wf1xC+uRZsfoMSgwDWdpBmaOFP+q2k0d4\\nrQWKIgZw71mxYADPKBAU\nUwMgD/aREuvTD1pvpl0Vk2Y5i/4Xx/fK7GV5DCdy9TFM\\nZgzAA/\nO9z5tuVzqcgodMu06+iqtXdtUoOafA+BTvSIXLAGMBAAE=\\n-----END PUBLIC\nKEY-----","algorithm": "RSA_OAEP_SHA256"}}}'
```

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

Note:

You can obtain the public key by running the following command.

```
az keyvault key download --id <keyid> -e PEM -f <keyFilePath>
```

The policy package is downloaded to your machine.

Important: The Immutable Service API supports the following algorithms:

- RSA_OAEP_SHA1
- RSA_OAEP_SHA256
- RSA_OAEP_SHA512
- RSA_PKCS1_v1_5

However, the Azure Key Vault currently supports only three out of the four algorithms.

Ensure that you specify one of the following algorithms that is supported by the Azure Key Vault:

- RSA_OAEP_SHA1
- RSA_OAEP_SHA256
- RSA_PKCS1_v1_5

For more information about the algorithms supported by the Azure Key Vault, refer to the [Azure product documentation](#).

4. Copy the policy package file to an Azure Storage Container or Azure File Share, as required.

2.8.2 Deploying the Sample Application Container on the Kubernetes Cluster

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

► To deploy a release on the Kubernetes cluster:

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

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

For example:

```
kubectl create namespace iap
```

2. Perform the following steps if you want to use NGINX as the Ingress Controller.

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-ingress --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.

- 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
```

- 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.

- Run the following command to login to Docker.

```
docker login
```

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

- 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 regcrt --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](#).

- 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.

- Modify the *az_credentials.json* file to update the credentials for the service principal.

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

```
{
  "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 *az_credentials.json* file, refer to the section [az_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/  
az_credentials.json --namespace <NAMESPACE>
```

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

For more information about the *az_credentials.json* file, refer to the section [az_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 [9](#).

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.

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 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

...
...

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

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

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

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

```

## specify k8s secret name of azure cloud credentials to access blog storage or key vault
azAccessSecrets:

## start with blue deployment first
deploymentInUse: blue

## Policy metadata for Blue Deployments
blueDeployment:
  enabled: true
  policy:
    # absolute path in S3 from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/pathToPolicy"
    # absolute path of policy dump file in Azure Blob store.
    # <Azure blob> "https://<account name>.blob.core.windows.net/<container name>/<pathToPolicy>"
    # absolute path in PV mount from where the policy dump file will be fetched.
    # <VOLUME> "file://var/data/ptypolicy/pathInVolumeToPolicy"
    # relative path in PV mount from where the policy dump file will be fetched.
    # it will prepend the '/var/data/ptypolicy' to below path
    # <VOLUME> "file:///pathInVolumeToPolicy"
    # <VOLUME> "pathInVolumeToPolicy" # without any leading '/'
    filePath: "POLICY_PATH"

## Policy metadata for Green Deployments
greenDeployment:
  enabled: false
  policy:
    # absolute path in S3 from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/pathToPolicy"
    # absolute path of policy dump file in Azure Blob store.
    # <Azure blob> "https://<account name>.blob.core.windows.net/<container name>/<pathToPolicy>"
    # absolute path in PV mount from where the policy dump file will be fetched.
    # <VOLUME> "file://var/data/ptypolicy/pathInVolumeToPolicy"
    # relative path in PV mount from where the policy dump file will be fetched.
    # it will prepend the '/var/data/ptypolicy' to below path
    # <VOLUME> "file:///pathInVolumeToPolicy"
    # <VOLUME> "pathInVolumeToPolicy" # without any leading '/'
    filePath: "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:

  # 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:
    ##AWS
    #service.beta.kubernetes.io/aws-load-balancer-internal: "true"
    ##AZURE
    #service.beta.kubernetes.io/azure-load-balancer-internal: "true"
    ##GCP
    #networking.gke.io/load-balancer-type: "Internal"

  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

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

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

  ## specify the host names and paths for Ingress rules
  ## prod and staging http paths can be used as optional fields.
  hosts:
    - host: "prod.example.com"
      httpPaths:
        - port: 8080
          prod: "/"
    ## Add host section with the hostname used as CN while creating server certificates.
    ## For accessing the staging end-points also, we need hostname as that of CN in
    ## server certs.
    ## While creating the certificates you can use *.example.com as CN for the below
    ## example
      # - host: "prod.example.com"
      #   httpPaths:
      #     - port: 8080
      #       prod: "/"
      # - host: "stage.example.com"
      #   httpPaths:
      #     - port: 8080
      #       staging: "/"
      #

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

```

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

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

For more information about Helm charts and their default values, refer to the section [Appendix A: Sample Application Helm Chart Details](#).

| Field | Description |
|--------------------|--|
| podSecurityContext | <p>Specify the privileges and access control settings for the pod.</p> <p>The default values are set as follows:</p> <ul style="list-style-type: none"> • <i>runAsUser</i> - 1000 • <i>runAsGroup</i> - 1000 |

| Field | Description |
|------------------|--|
| | <ul style="list-style-type: none"> • <i>fsGroup</i> - 1000 • <i>supplementalGroups</i> - 1000 <p>Note: Ensure that at any given point of time, the value of at least one of the four parameters must be <i>1000</i>. This ensures that the pod has the required permissions to access the <i>pty</i> directories in the containers.</p> <p>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.</p> <p>For more information about the parameters, refer to the section PodSecurityContext v1 Core in the Kubernetes API documentation.</p> |
| pvcName | <p>Specify the name of the Persistent Volume Claim where you want to store the policy package.</p> <p>If you want to use the Azure Storage Container for storing the policy package, then leave the value of this field blank.</p> <p>Important: This field is required if you want to store the immutable policy package in the Azure File Share.</p> |
| privateKeySource | <p>Specify one of the following methods to encrypt the policy package:</p> <ul style="list-style-type: none"> • <i>PKCS5</i> - Use Passphrase-Based Encryption for encrypting the policy package • <i>AZ_VAULT</i> - Use Azure Key Vault for encrypting the policy package |
| pbeSecrets | <p>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 <i>pbeSecrets</i> parameter to decrypt the policy.</p> <p>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 <i>30</i> and a minimum strength of <i>0.66</i>.</p> <p>Entropybits defines the randomness of the password and the strength defines the complexity of the password.</p> <p>For more information about the password strength, refer to the section Password Strength.</p> |
| privateKeySecret | <p>Specify the Kubernetes secret created that contains the private key of the Azure Key Vault, which encrypted the policy. The Sample Application container uses the value specified in the <i>privateKeySecret</i> parameter to decrypt the policy.</p> |

| Field | Description |
|--------------------------------|--|
| | This parameter is applicable if you have specified <code>AZ_VAULT</code> as the value of the <code>privateKeySource</code> parameter. |
| azAccessSecrets | Specify the value of the secret that you created in step 4b . This is used by the Sample Application container to access the Azure credentials required to download the policy package from the Azure storage container to the shared memory and decrypt the policy package if you have encrypted it using the Azure Key Vault. |
| deploymentInUse | Specify the value as <code>blue</code> . This indicates that the <code>blue</code> deployment strategy is in use. |
| blueDeployment/enabled | Specify the value as <code>true</code> to enable the <code>blue</code> deployment strategy. |
| blueDeployment/policy/filepath | <p>Specify the path in the persistent volume or the object store where you have stored the policy package.</p> <p>For example, if you are using Azure File Share as the persistent volume, then you can specify the file path as <code>file:///var/data/ptypolicy/pathInVolumeToPolicy</code>.</p> <p>If you have stored the policy package in an Object Store, such as an Azure Storage Container, then you can specify the storage container name as the first name in the <code>filepath</code> field.</p> <p>For example, you can specify the value of the <code>filepath</code> field, as <code>https://<accountName>.blob.core.windows.net/<container name>/<pathToPolicy></code>.</p> <p>Important: Use only uppercase and lowercase letters, numbers, dot, and underscore in the file path. Do not use special characters in the file path.</p> <p>Note: This value is case-sensitive.</p> |
| greenDeployment/enabled | Specify the value as <code>false</code> . While deploying Release 1, the <code>green</code> 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. |
| ingress/enabled | Specify this value as <code>true</code> to enable the ingress resource in Kubernetes. The ingress resource acts as a load balancer. It enables the simultaneous deployment of <code>blue</code> and <code>green</code> 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, <code>nginx-iap</code> , if you want to use the NGINX Ingress Controller. |



| Field | Description |
|--|---|
| nginx.ingress.kubernetes.io/auth-tls-verify-client | <p>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.</p> <p>By default, this field is <i>disabled</i>.</p> |
| nginx.ingress.kubernetes.io/auth-tls-secret | <p>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.</p> <p>By default, this field is <i>disabled</i>.</p> <p>For more information about the value of this field, refer to the section Ingress Configuration.</p> |
| ingress/hosts/host | <p>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.</p> |
| ingress/hosts/httpPaths/port | <p>Specify the port on which the Sample application is running inside the Docker container.</p> |
| ingress/hosts/httpPaths/prod | <p>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.</p> |
| ingress/hosts/httpPaths/staging | <p>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.</p> |

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.

2.8.3 Upgrading the Helm Charts

This section describes the steps for upgrading the Helm charts.

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

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

► To upgrade the Helm charts:

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

For more information about the extracted Helm charts, refer to the [step 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

...
...

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

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

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

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

## specify k8s secret name of azure cloud credentials to access blob storage or key vault
azAccessSecrets:

## start with blue deployment first
deploymentInUse: blue

## Policy metadata for Blue Deployments
blueDeployment:
    enabled: true
    policy:
        # absolute path in S3 from where the policy dump file will be fetched.
        # <S3> "s3://bucketName/pathToPolicy"
        # absolute path of policy dump file in Azure Blob store.
```

```

# <Azure blob> "https://<account name>.blob.core.windows.net/<container name>/
<pathToPolicy>
    # absolute path in PV mount from where the policy dump file will be fetched.
    # <VOLUME> "file://var/data/ptypolicy/pathInVolumeToPolicy"
    # relative path in PV mount from where the policy dump file will be fetched.
    # it will prepend the '/var/data/ptypolicy' to below path
    # <VOLUME> "file:///pathInVolumeToPolicy"
    # <VOLUME> "pathInVolumeToPolicy" # without any leading '/'
    filePath: "POLICY_PATH"

## Policy metadata for Green Deployments
greenDeployment:
    enabled: false
    policy:
        # absolute path in S3 from where the policy dump file will be fetched.
        # <S3> "s3://bucketName/pathToPolicy"
        # absolute path of policy dump file in Azure Blob store.
        # <Azure blob> "https://<account name>.blob.core.windows.net/<container name>/
        <pathToPolicy>
            # absolute path in PV mount from where the policy dump file will be fetched.
            # <VOLUME> "file://var/data/ptypolicy/pathInVolumeToPolicy"
            # relative path in PV mount from where the policy dump file will be fetched.
            # it will prepend the '/var/data/ptypolicy' to below path
            # <VOLUME> "file:///pathInVolumeToPolicy"
            # <VOLUME> "pathInVolumeToPolicy" # without any leading '/'
            filePath: "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:

    # 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:
        ##AWS
        #service.beta.kubernetes.io/aws-load-balancer-internal: "true"
        ##AZURE
        #service.beta.kubernetes.io/azure-load-balancer-internal: "true"
        ##GCP
        #networking.gke.io/load-balancer-type: "Internal"

    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

```

```

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

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

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

```

In 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 A: Sample Application Helm Chart Details](#).

| Field | Description |
|--------------------|--|
| podSecurityContext | <p>Specify the privileges and access control settings for the pod.</p> <p>The default values are set as follows:</p> <ul style="list-style-type: none"> • <i>runAsUser</i> - 1000 • <i>runAsGroup</i> - 1000 • <i>fsGroup</i> - 1000 • <i>supplementalGroups</i> - 1000 <p>Note: Ensure that at any given point of time, the value of at least one of the four parameters must be <i>1000</i>. This ensures that the pod has the required permissions to access the <i>pty</i> directories in the containers.</p> |

| Field | Description |
|------------------|---|
| | <p>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.</p> <p>For more information about the parameters, refer to the section PodSecurityContext v1 Core in the Kubernetes API documentation.</p> |
| pvcName | <p>Specify the name of the Persistent Volume Claim where you want to store the policy package.</p> <p>If you want to use the Azure Storage Container for storing the policy package, then leave the value of this field blank.</p> <p>Important: This field is required if you want to store the immutable policy package in the Azure File Share.</p> |
| privateKeySource | <p>Specify one of the following methods to encrypt the policy package:</p> <ul style="list-style-type: none"> • <i>PKCS5</i> - Use Passphrase-Based Encryption for encrypting the policy package • <i>AZ_VAULT</i> - Use Azure Key Vault for encrypting the policy package |
| pbeSecrets | <p>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 <i>pbeSecrets</i> parameter to decrypt the policy.</p> <p>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 <i>30</i> and a minimum strength of <i>0.66</i>. Entropybits defines the randomness of the password and the strength defines the complexity of the password.</p> <p>For more information about the password strength, refer to the section Password Strength.</p> |
| privateKeySecret | <p>Specify the Kubernetes secret created that contains the private key of the Azure Key Vault, which encrypted the policy. The Sample Application container uses the value specified in the <i>privateKeySecret</i> parameter to decrypt the policy.</p> <p>This parameter is applicable if you have specified <i>AZ_VAULT</i> as the value of the <i>privateKeySource</i> parameter.</p> |
| azAccessSecrets | <p>Specify the value of the secret that you have created using the <i>az_credentials.json</i> file. This is used by the Sample Application container to access the Azure credentials required to download the policy package from the Azure storage container to the shared memory and decrypt the policy package if you have encrypted it using the Azure Key Vault.</p> |
| deploymentInUse | <p>Specify the value as <i>blue</i>. This indicates that the <i>blue</i> deployment strategy is in use.</p> |



| Field | Description |
|--|---|
| greenDeployment/enabled | Specify the value as <i>true</i> . While upgrading the release to Release 2, the <i>green</i> deployment strategy is enabled. |
| greenDeployment/policy/filepath | <p>Specify the path in the persistent volume or the object store where you have stored the policy package.</p> <p>For example, if you are using Azure File Share as the persistent volume, then you can specify the file path as <i>file://var/data/ptypolicy/pathInVolumeToPolicy</i>.</p> <p>If you have stored the policy package in an Object Store, such as an Azure Storage Container, then you can specify the storage container name as the first name in the <i>filepath</i> field.</p> <p>For example, you can specify the value of the <i>filepath</i> field, as <i>https://<accountName>.blob.core.windows.net/<container name>/<pathToPolicy></i>.</p> |
| | <p>Important: Use only uppercase and lowercase letters, numbers, dot, and underscore in the file path. Do not use special characters in the file path.</p> |
| | <p>Note: This value is case-sensitive.</p> |
| 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 <i>true</i> to enable the ingress resource in Kubernetes. The ingress resource acts as a load balancer. It enables the simultaneous deployment of <i>blue</i> and <i>green</i> 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, <i>nginx-iap</i> , if you want to use the NGINX Ingress Controller. |
| nginx.ingress.kubernetes.io/auth-tls-verify-client | <p>Set the value of this field to <i>on</i>, if you want to enable TLS mutual authentication between the REST API client and the Azure load balancer.</p> <p>By default, this field is <i>disabled</i>.</p> |
| 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. |



| Field | Description |
|---------------------------------|--|
| | By default, this field is <i>disabled</i> . 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. |
| 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 shut down 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.

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

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

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

Before you begin

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

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

For more information about invoking the Immutable Service APIs, refer to the section [Retrieving the Policy from the ESA](#).

- If you have used Azure Key Vault to encrypt the policy package, then ensure that you have created the private key secret, which is used by the Sample Application container to decrypt the policy.
- If you have used Passphrase-Based Encryption to encrypt the policy package, then ensure that you have created the passphrase secret, which is used by the Sample Application container to decrypt the policy.
- Ensure that you have created a custom image for the Sample Application container or the Custom container using the Dockerfile provided in the v9.1.0.5 installation package.

For more information about building a custom image, refer to the section [Creating Custom Images Using Dockerfile](#).

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

For more information about the extracted Helm charts, refer to the [step 9](#) of the section [Initializing the Linux Instance](#).

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

```
...
...
## Configure the delays for Liveness Probe here
```

```

livenessProbe:
  initialDelaySeconds: 15
  periodSeconds: 20

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

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

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

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

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

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

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

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

## specify k8s secret name of azure cloud credentials to access blob storage or key vault
azAccessSecrets:

## start with blue deployment first
deploymentInUse: blue

## Policy metadata for Blue Deployments
blueDeployment:
  enabled: true
  policy:
    # absolute path in S3 from where the policy dump file will be fetched.
    # <S3> "s3://bucketName/pathToPolicy"
    # absolute path of policy dump file in Azure Blob store.
    # <Azure blob> "https://<account name>.blob.core.windows.net/<container name>/<pathToPolicy>"
    # absolute path in PV mount from where the policy dump file will be fetched.
    # <VOLUME> "file:/var/data/ptypolicy/pathInVolumeToPolicy"
    # relative path in PV mount from where the policy dump file will be fetched.
    # it will prepend the '/var/data/ptypolicy' to below path
    # <VOLUME> "file:///pathInVolumeToPolicy"
    # <VOLUME> "pathInVolumeToPolicy" # without any leading '/'
    filePath: "POLICY_PATH"

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

```

```

# <Azure blob> "https://<account name>.blob.core.windows.net/<container name>/
<pathToPolicy>"  

# absolute path in PV mount from where the policy dump file will be fetched.  

# <VOLUME> "file:///var/data/ptypolicy/pathInVolumeToPolicy"  

# relative path in PV mount from where the policy dump file will be fetched.  

# it will prepend the '/var/data/ptypolicy' to below path  

# <VOLUME> "file:///pathInVolumeToPolicy"  

# <VOLUME> "pathInVolumeToPolicy" # without any leading '/'  

filePath: "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:

# 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:
  ##AWS
  #service.beta.kubernetes.io/aws-load-balancer-internal: "true"
  ##AZURE
  #service.beta.kubernetes.io/azure-load-balancer-internal: "true"
  ##GCP
  #networking.gke.io/load-balancer-type: "Internal"

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

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

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

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

```



```

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

```

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

For more information about Helm charts and their default values, refer to the section [Appendix A: Sample Application Helm Chart Details](#).

| Field | Description |
|------------------|---|
| pvcName | Specify the name of the Persistent Volume Claim where you want to store the policy snapshot. Leave the value of this field blank if you want to use Azure Storage Container for storing the policy snapshot. |
| privateKeySource | Specify one of the following methods to encrypt the policy package: <ul style="list-style-type: none"> • <i>PKCS5</i> - Use Passphrase-Based Encryption for encrypting the policy package • <i>AZ_VAULT</i> - Use Azure Key Vault for encrypting the policy package |
| pbeSecrets | Specify the Kubernetes secret that contains the passphrase and the salt that were used to generate key, which encrypted the policy. The Sample Application container uses the value specified in the <i>pbeSecrets</i> parameter to decrypt the policy. <div style="background-color: #fce4ec; padding: 10px; margin-top: 10px;"> Important: Ensure that the password complexity meets the following requirements: <ul style="list-style-type: none"> • 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, <i>123456789</i>, <i>11111111</i>, <i>password</i>, and <i>qwertyuiop</i> </div> |
| privateKeySecret | Specify the Kubernetes secret created that contains the private key of the Azure Key Vault, which encrypted the policy. The Sample Application container uses the value specified in the <i>privateKeySecret</i> parameter to decrypt the policy. <p>This parameter is applicable if you have specified <i>AZ_VAULT</i> as the value of the <i>privateKeySource</i> parameter.</p> |
| azAccessSecrets | Specify the value of the secret that you have created using the <i>az_credentials.json</i> file. This is used by the Sample Application |



| Field | Description |
|---|--|
| | container to access the Azure credentials required to download the policy package from the Azure storage container to the shared memory and decrypt the policy package if you have encrypted it using the Azure Key Vault. |
| deploymentInUse | Specify the value as <i>blue</i> . This indicates that the blue deployment strategy is in use. |
| greenDeployment\enabled | Specify the value as <i>true</i> . While upgrading the release to Release 2, the <i>green</i> deployment strategy is enabled. |
| greenDeployment/policy/filepath | <p>Specify the path in the persistent volume or the object store where you have stored the policy package.</p> <p>For example, if you are using Azure File Share as the persistent volume, then you can specify the file path as <i>file://var/data/ptypolicy/pathInVolumeToPolicy</i>.</p> <p>If you have stored the policy package in an Object Store, such as an Azure Storage Container, then you can specify the storage container name as the first name in the <i>filepath</i> field.</p> <p>For example, you can specify the value of the <i>filepath</i> field, as <i>https://<accountName>.blob.core.windows.net/<container name>/<pathToPolicy></i>.</p> <div data-bbox="894 967 1019 998" style="background-color: #f0e6e6; padding: 5px;">Important:</div> <p>Use only uppercase and lowercase letters, numbers, dot, and underscore in the file path. Do not use special characters in the file path.</p> <div data-bbox="894 1193 959 1222" style="background-color: #e0f2e0; padding: 5px;">Note:</div> <p>This value is case-sensitive.</p> |
| 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 <i>true</i> to enable the ingress resource in Kubernetes. The ingress resource acts as a load balancer. It enables the simultaneous deployment of <i>blue</i> and <i>green</i> strategy by exposing both the deployments on the same external IP address, but on different URL paths. |
| ingress/annotations/kubernetes.io/ingress.class | Specify the value of the external ingress controller, if any. For example, specify, <i>nginx</i> if you want to use the NGINX Ingress Controller. |

Important: In the v7.1 MR4 release, if you have deployed the NGINX Ingress Controller without specifying this parameter,

| Field | Description |
|---------------------------------|---|
| | then ensure that you comment out this parameter in the <code>values.yaml</code> file. Otherwise, the requests that are sent from the client application to the NGINX Ingress Controller will not be forwarded to the pods in the Sample Application container. |
| ingress/hosts/host | Specify the hostname of the ingress resource. If you are using the external IP address of the ingress resource for performing the security operations, then leave this field blank. |
| ingress/hosts/httpPaths/port | Specify the port on which the Sample application is running inside the Docker container. |
| ingress/hosts/httpPaths/prod | If you are using the NGINX Ingress Controller as an SSL passthrough, then you must specify the value of this field as <code>/</code> . This ensures that the data traffic from the REST API client is redirected to the required production or staging URL based on the hostname value. |
| ingress/hosts/httpPaths/staging | If you are using the NGINX Ingress Controller as an SSL passthrough, then you must specify the value of this field as <code>/</code> . This ensures that the data traffic from the REST API client is redirected to the required production or staging URL based on the hostname value. |

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

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

For example:

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

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

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

Chapter 3

Accessing Logs Using Splunk

[3.1 Understanding the Logging Architecture](#)

[3.2 Setting Up Splunk](#)

[3.3 Configuring Splunk](#)

[3.4 Deploying the Splunk Connect for Kubernetes Helm Chart on the Kubernetes Cluster](#)

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

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

3.1 Understanding the Logging Architecture

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

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

For more information about Splunk, refer to the [Splunk documentation](#) website.

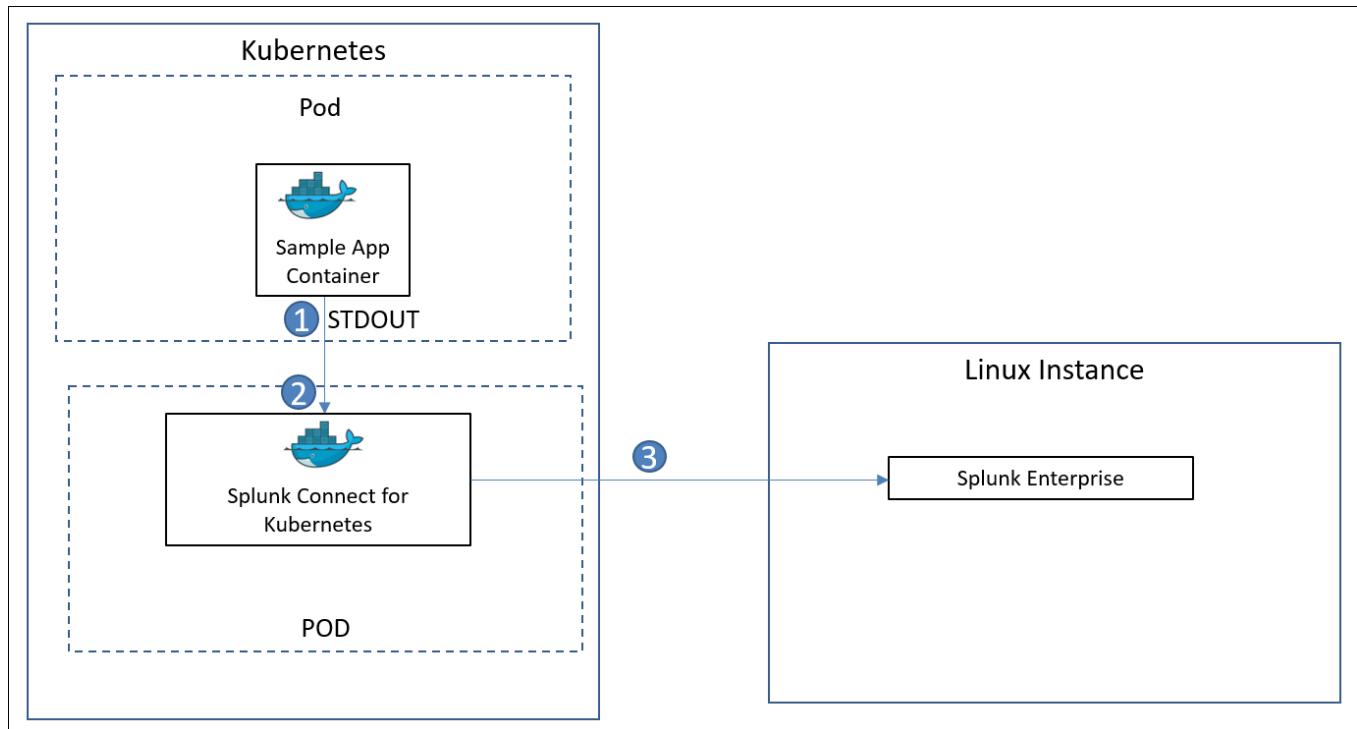


Figure 3-1: Sample Logging Workflow

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

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

For more information about Splunk Connect for Kubernetes, refer to the [Splunk Connect for Kubernetes](#) page on Github.

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

3.2 Setting Up Splunk

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

► To set up Splunk:

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

For more information about installing Splunk Enterprise on a Linux instance, refer to the section [Install Splunk Enterprise](#) in the Splunk documentation.

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

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

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

For more information about starting Splunk Enterprise on Linux, refer to the section [Start Splunk Enterprise on Linux](#) in the Splunk documentation.

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

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

5. Type a password for the user that you have created in [step 4](#), and then press **Enter**.

The following message appears on the console.

```
Waiting for web server at http://127.0.0.1:8000 to be available..... Done  
  
If you get stuck, we're here to help.  
Look for answers here: http://docs.splunk.com  
  
The Splunk web interface is at http://[REDACTED]:8000
```

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

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

3.3 Configuring Splunk

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

► To configure Splunk Enterprise:

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

The Splunk Enterprise screen appears.

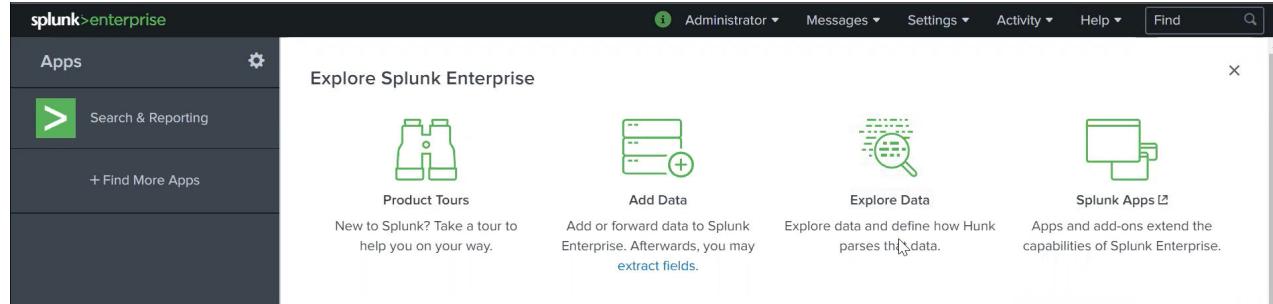


Figure 3-2: Splunk Enterprise Web UI

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.

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

Figure 3-3: Indexes Screen

- b. Click **New Index**.

The **New Index** dialog box appears.

New Index

General Settings

Index Name

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

Events
 Metrics

Home Path

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

Cold Path

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

Thawed Path

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

Figure 3-4: New Index Dialog Box

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

- d. Click **Save**.

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

For more information about indexes used in Splunk Enterprise, refer to the section [About managing indexes](#) in the Splunk documentation.

For more information about creating an index in Splunk Enterprise, refer to the section [Create custom indexes](#) in the Splunk documentation.

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

For more information about HEC, refer to the section [Set up and use HTTP Event Collector in Splunk Web](#) in the Splunk documentation.

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

The **Data inputs** screen appears.

The screenshot shows the 'Data inputs' screen in Splunk Web. At the top, there's a header bar with the Splunk logo, user info ('Administrator'), and navigation links ('Messages', 'Settings', 'Activity', 'Help'). Below the header is a search bar. The main area has a title 'Data inputs' with a subtitle: 'Set up data inputs from files and directories, network ports, and scripted inputs. If you want to set up forwarding and receiving between two Splunk instances, go to [Forwarding and receiving](#)'. A sub-section titled 'Local inputs' contains a table:

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

Figure 3-5: Data inputs Screen

- b. Click **HTTP Event Collector**.

The **HTTP Event Collector** screen appears.

The screenshot shows the 'HTTP Event Collector' screen in Splunk Web. The top navigation bar includes 'Data Inputs > HTTP Event Collector'. On the right, there are buttons for 'Global Settings' and a prominent green 'New Token' button. The main area displays a table of tokens:

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

Figure 3-6: HTTP Event Collector Screen

- c. Click **New Token**.

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

Files & Directories
Upload a file, index a local file, or monitor an entire directory.

HTTP Event Collector
Configure tokens that clients can use to send data over HTTP or HTTPS.

TCP / UDP
Configure the Splunk platform to listen on a network port.

Scripts
Get data from any API, service, or database with a script.

Configure a new token for receiving data over HTTP. [Learn More](#)

Name

Source name override?

Description?

Output Group (optional)

Enable indexer acknowledgement

Figure 3-7: Select Source Screen

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

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

For more information about HEC tokens, refer to the section [About Event Collector](#) tokens in the [Splunk documentation](#).

- e. Click **Next**.

The **Input Settings** screen appears.

type definitions. The Splunk platform loads all app contexts based on precedence rules. [Learn More](#)

Index

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

Select Allowed Indexes

| Available item(s) | Selected item(s) |
|--|--|
| <input checked="" type="checkbox"/> events | <input checked="" type="checkbox"/> events |
| <input type="checkbox"/> history | |
| <input type="checkbox"/> main | |
| <input type="checkbox"/> metrics | |
| <input type="checkbox"/> summaries | |

Default Index

Select indexes that clients will be able to select from.

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.

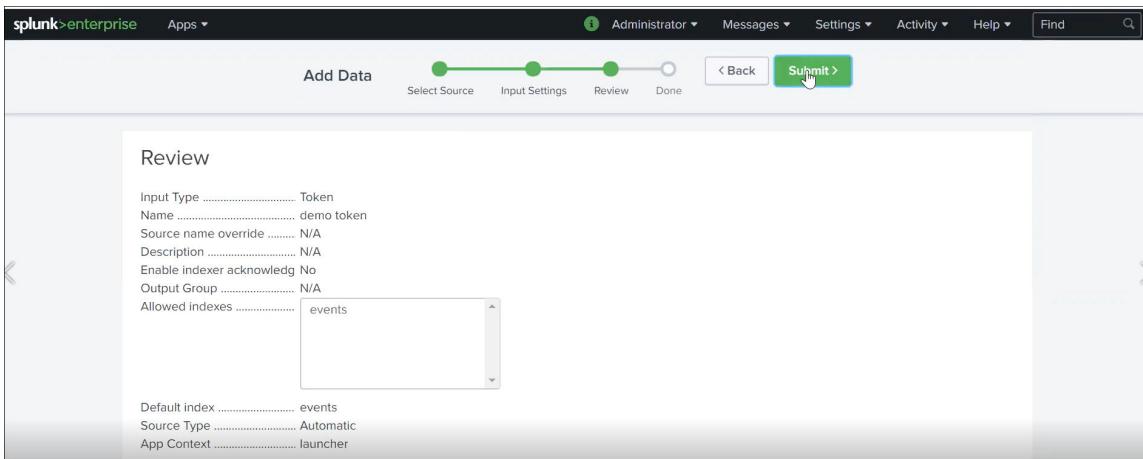


Figure 3-9: Review Screen

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.

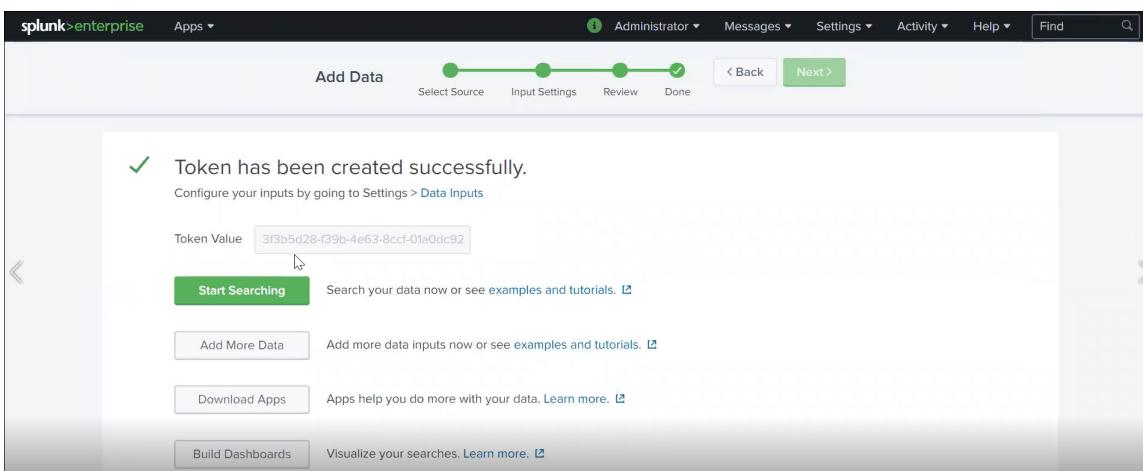


Figure 3-10: Done Screen

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.

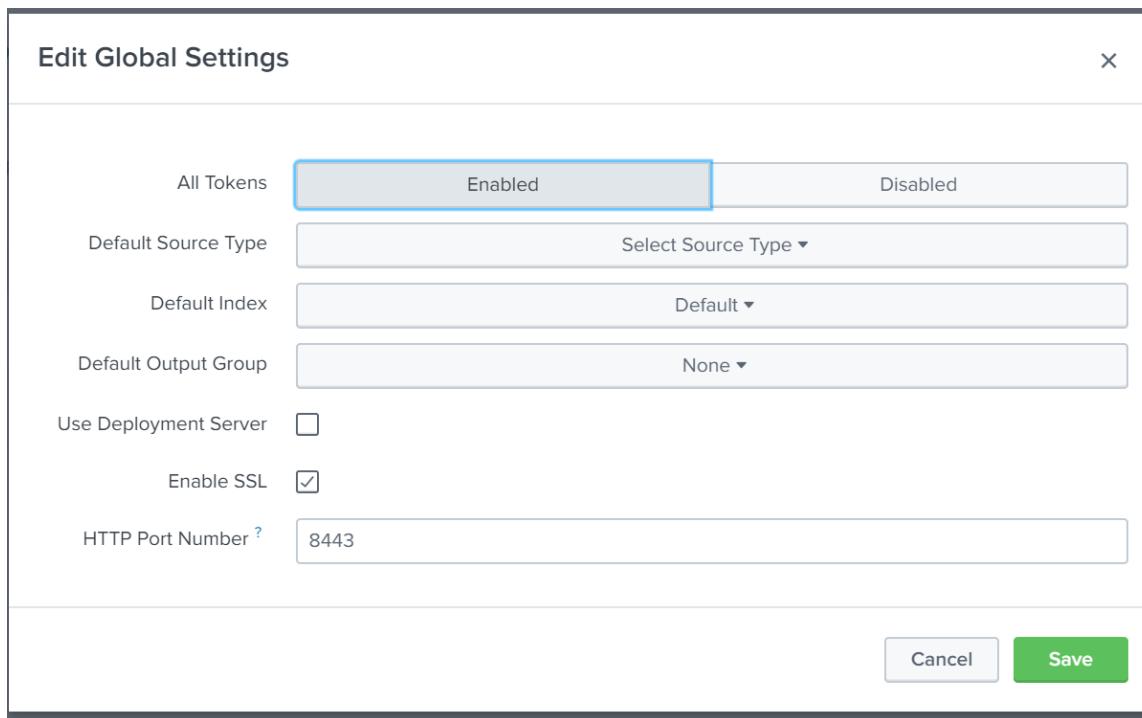


Figure 3-11: Edit Global Settings Dialog Box

- 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 <i>https</i> . |
| 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 <i>true</i> . |
| 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 4i](#) 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.

```
kubectl get pods -n kube-system
```

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

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

Chapter 4

Protecting Data Using the Sample Application Container Deployment

[4.1 Running Security Operations](#)

[4.2 Viewing Logs in Splunk](#)

[4.3 Autoscaling the Protegity Reference 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" ] }'
```

```
"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 creating 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.

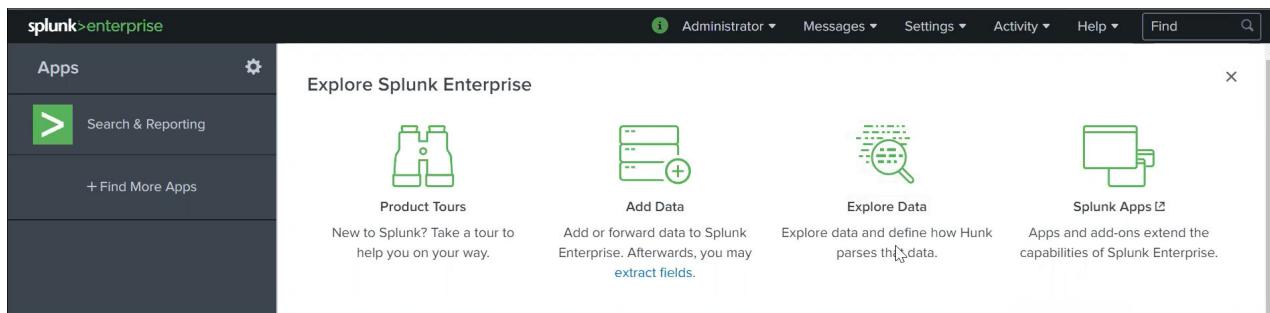


Figure 4-1: Splunk Enterprise Web UI

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.

The screenshot shows the 'Search' screen under the 'Search & Reporting' app. At the top, there's a search bar with placeholder text 'enter search here...' and a dropdown for 'Last 24 hours'. Below the search bar, there's a 'How to Search' section with a link to 'Documentation' and a 'Tutorial' button. To the right, there's a 'What to Search' section with a 'Data Summary' button. At the bottom left, there's a 'Search History' section with a link to 'Search History'.

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.

Events (31) Patterns Statistics Visualization

Format Timeline ▾ List ▾ Format 20 Per Page ▾

< Hide Fields i All Fields

SELECTED FIELDS
a host 1
a source 1
a sourcetype 1

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

> 1/6/22 8:42:09.798 AM {"additional_info": {"description": "Data protect operation was successful."}, "cnt": 1, "logtype": "Protection", "level": "SUCCESS", "origin": {"hostname": "████████", "ip": "████████", "time_utc": 1641458529}, "protection": {"dataelement": "Alpha", "operation": "Protect", "audit_code": 6, "datastore": "default", "policy_user": "ptyitusr"}, "process": {"id": "1", "user": "ptyituser"}, "client": {"ip": "0.0.0.0"}, "protector": {"family": "sdk", "version": "1.0.0", "vendor": "Java", "pcc_version": "1.0.0", "core_version": "1.0.0"}, "signature": {"key_id": "1d3fdccc-6b8e-4d5e-bf7d-6508b7980181", "checksum": "E19E1D5994755586FAC144535BAE5EC01ACCC1A2A2685B87BF83233E54708EF7"}}

Show syntax highlighted

host = source = /var/log/containers/spring-spring-apjava-blue-649464b598-sqlmp_... sourcetype = kube:container:spring-apjava

> 1/6/22 8:42:07.809 AM {"additional_info": {"description": "Data protect operation was successful."}, "cnt": 1, "logtype": "Protection", "level": "SUCCESS", "origin": {"hostname": "████████", "ip": "████████", "time_utc": 1641458527}, "protection": {"dataelement": "Alpha", "operation": "Protect", "audit_code": 6, "datastore": "default", "policy_user": "ptyitusr"}, "process": {"id": "1", "user": "ptyituser"}, "client": {"ip": "0.0.0.0"}, "protector": {"family": "sdk", "version": "1.0.0", "vendor": "Java", "pcc_version": "1.0.0", "core_version": "1.0.0"}, "signature": {"key_id": "1d3fdccc-6b8e-4d5e-bf7d-6508b7980181", "checksum": "E19E1D5994755586FAC144535BAE5EC01ACCC1A2A2685B87BF83233E54708EF7"}}

- 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"
```

- Press **Enter**.

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

Events (4) Patterns Statistics Visualization

Format Timeline ▾ List ▾ Format 20 Per Page ▾

< Hide Fields i All Fields

SELECTED FIELDS
a host 1
a source 1
a sourcetype 1

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

> 1/6/22 8:42:09.798 AM {"additional_info": {"description": "Data protect operation was successful."}, "cnt": 1, "logtype": "Protection", "level": "SUCCESS", "origin": {"hostname": "████████", "ip": "████████", "time_utc": 1641458529}, "protection": {"dataelement": "Alpha", "operation": "Protect", "audit_code": 6, "datastore": "default", "policy_user": "ptyitusr"}, "process": {"id": "1", "user": "ptyituser"}, "client": {"ip": "0.0.0.0"}, "protector": {"family": "sdk", "version": "1.0.0", "vendor": "Java", "pcc_version": "1.0.0", "core_version": "1.0.0"}, "signature": {"key_id": "1d3fdccc-6b8e-4d5e-bf7d-6508b7980181", "checksum": "E19E1D5994755586FAC144535BAE5EC01ACCC1A2A2685B87BF83233E54708EF7"}}

Show syntax highlighted

host = source = /var/log/containers/spring-spring-apjava-blue-649464b598-sqlmp_... sourcetype = kube:container:spring-apjava

> 1/6/22 8:42:07.809 AM {"additional_info": {"description": "Data protect operation was successful."}, "cnt": 1, "logtype": "Protection", "level": "SUCCESS", "origin": {"hostname": "████████", "ip": "████████", "time_utc": 1641458527}, "protection": {"dataelement": "Alpha", "operation": "Protect", "audit_code": 6, "datastore": "default", "policy_user": "ptyitusr"}, "process": {"id": "1", "user": "ptyituser"}, "client": {"ip": "0.0.0.0"}, "protector": {"family": "sdk", "version": "1.0.0", "vendor": "Java", "pcc_version": "1.0.0", "core_version": "1.0.0"}, "signature": {"key_id": "1d3fdccc-6b8e-4d5e-bf7d-6508b7980181", "checksum": "E19E1D5994755586FAC144535BAE5EC01ACCC1A2A2685B87BF83233E54708EF7"}}

4.3 Autoscaling the Protegity 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.

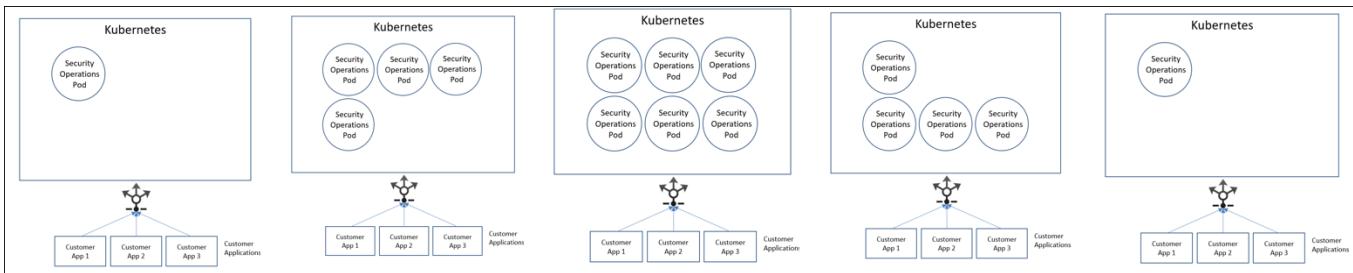


Figure 4-3: Autoscaling Kubernetes Cluster

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 Protegility Security Operation pods to meet business needs. If the workloads reduce, then Kubernetes will shrink the cluster.

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

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

Chapter 5

Appendix A: Sample Application Helm Chart Details

- [5.1 Chart.yaml](#)
- [5.2 Values.yaml](#)
- [5.3 az_credentials.json](#)
- [5.4 pty-rbac.yaml](#)

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

5.1 Chart.yaml

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

```
apiVersion: v1
description: A Spring boot application chart for Kubernetes
name: spring-apjava
version: 1.0.0
```

5.2 Values.yaml

The *Values.yaml* file contains important fields that need to be edited by the user as per the specifics of each environment.

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

5.2.1 Image Pull Secrets

This section specifies the credentials that are required to access the image repository.

```
## create image pull secrets and specify the name here.
## remove the [] after 'imagePullSecrets:' once you specify the secrets
imagePullSecrets: []
# - name: regcred
```

The user is required to create a Kubernetes secret for accessing the image registry. Kubernetes secrets can be created using existing Docker credentials.

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

For example:

```
kubectl create secret generic regcred --from-file=.dockerconfigjson=<PATH_TO_DOCKER_CONFIG>/config.json --type=kubernetes.io/dockerconfigjson --namespace <NAMESPACE>
```

5.2.2 Name Override

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

Note: Protegrity recommends that these values should not be changed by the user.

If the `fullnameoverride` parameter value is specified, then the deployment names will use that value.

If the `nameOverride` parameter value is specified, then the deployment names will be a combination of the `nameOverride` parameter value and the Helm chart release name.

If both the values are kept empty, then the Helm chart release name will be used.

```
nameOverride: ""
fullnameOverride: ""
```

5.2.3 Container Image Repository

This section provides the path for the images in the fields that the users are required to enter after loading the images into their registry.

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

The following list provides an overview of the `springappImage` parameter, which refers to the details for the Sample container image:

- `repository` – Refers to the name of the registry.

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

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

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

5.2.4 Resources

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

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

```
springContainerResources:
  limits:
    cpu: 500m
    memory: 512Mi
  requests:
    cpu: 200m
    memory: 200Mi
```

You can specify the following parameters:

- *limits* - Specify the resource limits for the containers in a pod. These limits ensure that the running container does not use additional resources than the limit you set.
- *requests* - Specify the resource request for the containers in a pod. This information determines the nodes for deploying the pods.

For more information about the resource parameters, refer to the section [Managing Resources for Containers](#) in the Kubernetes documentation.

5.2.5 Liveliness and Readiness Probe Settings

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

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

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

5.2.6 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.

5.2.7 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.

5.2.8 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.

5.2.9 Private Key Source

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

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

Specify one of the following methods to encrypt the policy package:

- *PKCS5*- Use Passphrase-Based Encryption for encrypting the policy package
- *AZ_VAULT*- Use the Azure Key Vault for encrypting the policy package

5.2.10 PBE Secrets

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

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

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

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

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

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

5.2.11 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.

5.2.12 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
```

5.2.13 Deployment

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

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

```
blueDeployment:
  enabled: true

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

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

5.2.14 Autoscaling

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

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

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

The following list provides a description for the parameters:

- *replicaCount* - Indicates the number of pods that get instantiated at the start of the deployment.
- *minReplicas* - Indicates the minimum number of pods that will keep running in the deployment for a cluster.
- *maxReplicas* - Indicates the maximum number of pods that will keep running in the deployment for a cluster.
- *targetCPU* - Horizontal Pod Autoscaler (HPA) will increase and decrease the number of replicas (through the deployment) to maintain an average CPU utilization across all Pods based on the % value specified in the *targetCPU* setting.

5.2.15 Service Configuration

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

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

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

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

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

You can specify the value of the *springappService/type* parameter as *LoadBalancer* or *ClusterIP*. This parameter is only applicable if you have set the value of the *ingress/enabled* parameter to *false*.

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

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

5.2.16 Ingress Configuration

This section explains the Ingress configuration for deployment.

```
## Ingress enables testing of Green (V2/Staging) Deployments alongside Blue ones
## simultaneously.
## This is realized by exposing both deployments on the same External IP but on different URL
## Paths.
## If you don't want expose both the deployments at the same time, keep 'ingress.enabled' as
## false.
```

```

## This means that only Blue deployment gets exposed on the Load Balancer's External IP via
prod service.
ingress:
  enabled: true

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

  ## 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
```

5.3 az_credentials.json

The *az_credentials.json* file contains information about the credentials for service principal 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 credentials.

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

5.4 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: [""]
  resources: ["services"]
  verbs: ["create", "get", "list", "patch", "watch", "delete"]
- apiGroups: [""]
  resources: ["persistentvolumeclaims"]
  verbs: ["create", "get", "list", "watch"]
- apiGroups: [""]
  resources: ["pods", "pods/log"]
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  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: 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: iap-deployer
roleRef:
  kind: Role
  name: iap-secrets-manager
  apiGroup: rbac.authorization.k8s.io
```



Chapter 6

Appendix B: 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 7

Appendix C: Using the Dockerfile to Build Custom Images

7.1 Creating Custom Images Using Dockerfile

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

7.1 Creating Custom Images Using Dockerfile

This section describes the steps for creating custom images for the Sample Application container, 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 Sample Application container.

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

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

The following files are extracted:

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

- b. Edit the *IAP_RHUBI_SAMPLE-APP_DOCKERFILE* 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-SAMPLE-APP_SRC_<version_number>.tgz* file.

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

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 Sample Application container image.

```
docker images
```

- e. 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](#).