Red Hat OpenShift Service on AWS 4

Tutorials

Red Hat OpenShift Service on AWS tutorials

Last Updated: 2026-01-15

Red Hat OpenShift Service on AWS 4 Tutorials

Red Hat OpenShift Service on AWS tutorials

Legal Notice

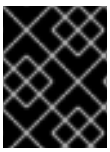
Abstract

Tutorials on creating your first Red Hat OpenShift Service on AWS (ROSA) cluster.

Table of Contents

CHAPTER 1. TUTORIALS OVERVIEW

Use the step-by-step tutorials from Red Hat experts to get the most out of your Managed OpenShift cluster.

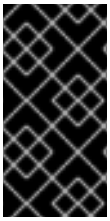


IMPORTANT

This content is authored by Red Hat experts but has not yet been tested on every supported configuration.

CHAPTER 2. TUTORIAL: RED HAT OPENSHIFT SERVICE ON AWS ACTIVATION AND ACCOUNT LINKING

This tutorial describes the process for activating Red Hat OpenShift Service on AWS and linking to an AWS account, before deploying the first cluster.



IMPORTANT

If you have received a private offer for the product, make sure to proceed according to the instructions provided with the private offer before following this tutorial. The private offer is designed either for a case when the product is already activated, which replaces an active subscription, or for first time activations.

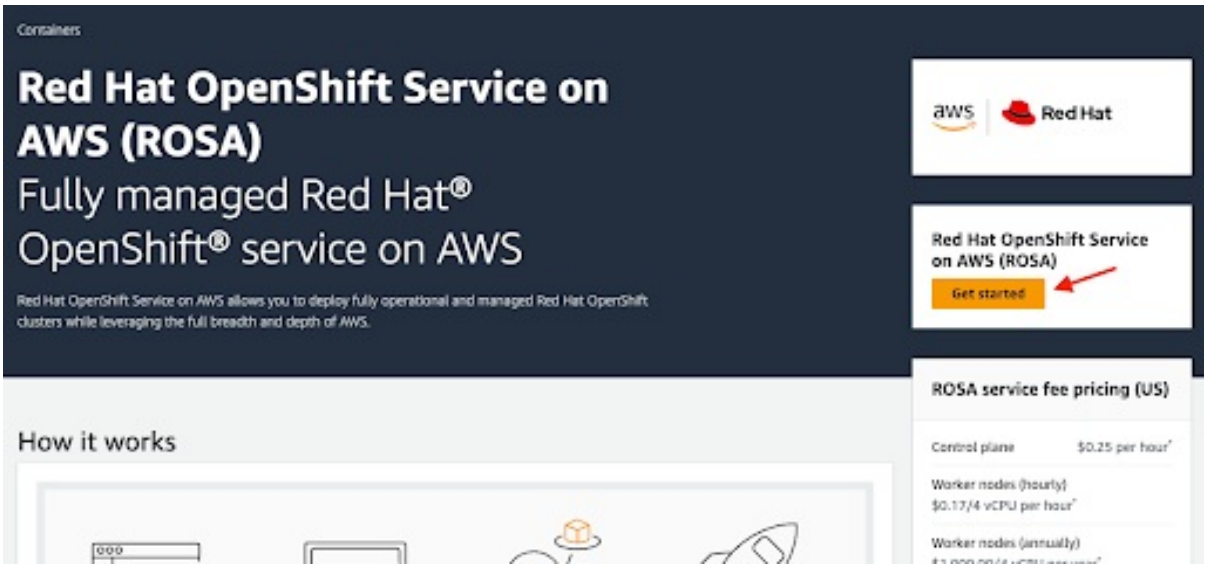
2.1. PREREQUISITES

- Log in to the Red Hat account that you want to associate with the AWS account that will activate the Red Hat OpenShift Service on AWS product subscription.
- The AWS account used for service billing can only be associated with a single Red Hat account. Typically an AWS payer account is the one that is used to subscribe to Red Hat OpenShift Service on AWS and used for account linking and billing.
- All team members belonging to the same Red Hat organization can use the linked AWS account for service billing while creating Red Hat OpenShift Service on AWS clusters.

2.2. SUBSCRIPTION ENABLEMENT AND AWS ACCOUNT SETUP

1. Activate the Red Hat OpenShift Service on AWS product at the AWS console page by clicking the **Get started** button:

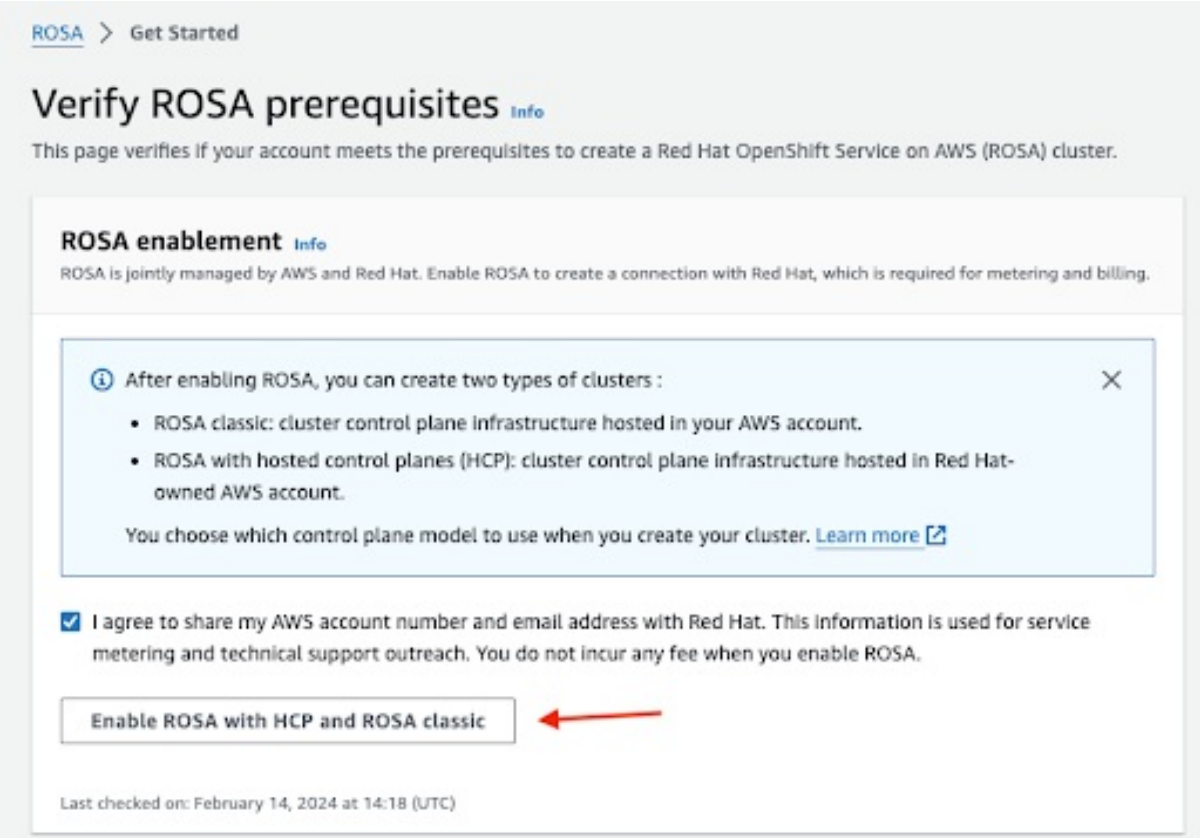
   Figure 2.1. Get started

   If you have activated Red Hat OpenShift Service on AWS before but did not complete the process, you can click the button and complete the account linking as described in the following steps.

2. Confirm that you want your contact information to be shared with Red Hat and enable the service:

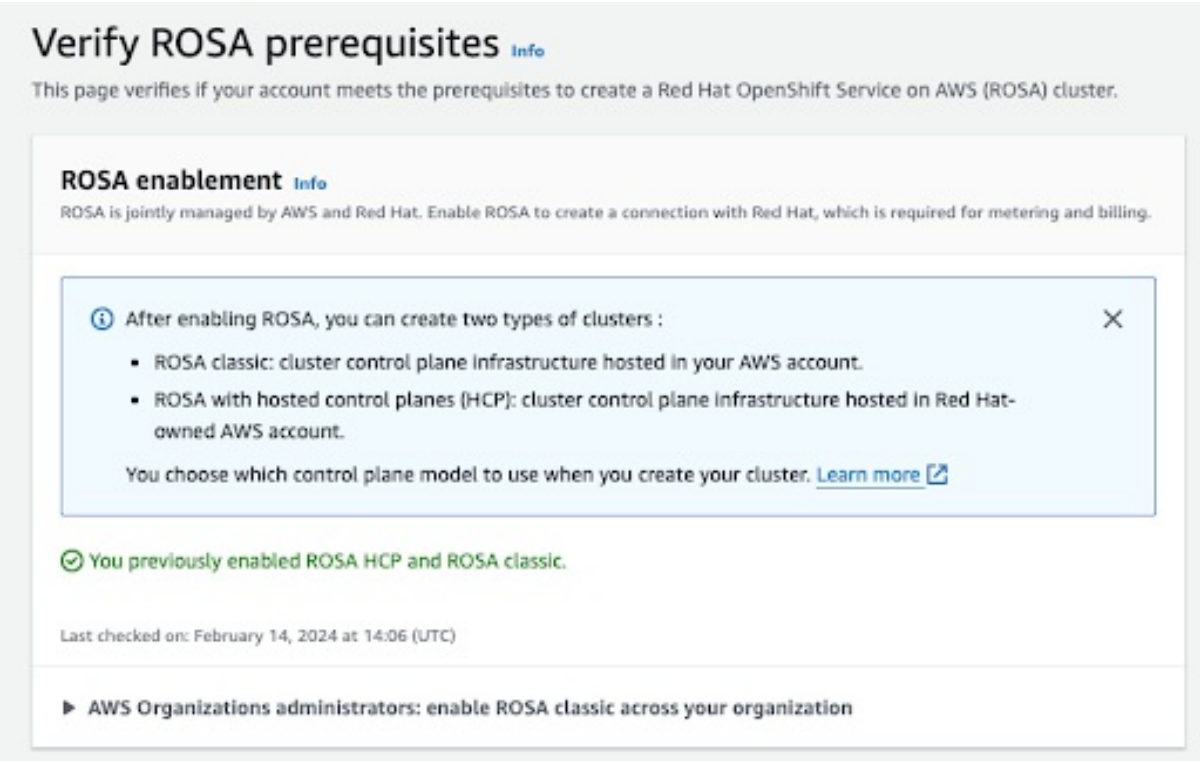
Figure 2.2. Enable Red Hat OpenShift Service on AWS



- You will not be charged by enabling the service in this step. The connection is made for billing and metering that will take place only after you deploy your first cluster. This could take a few minutes.

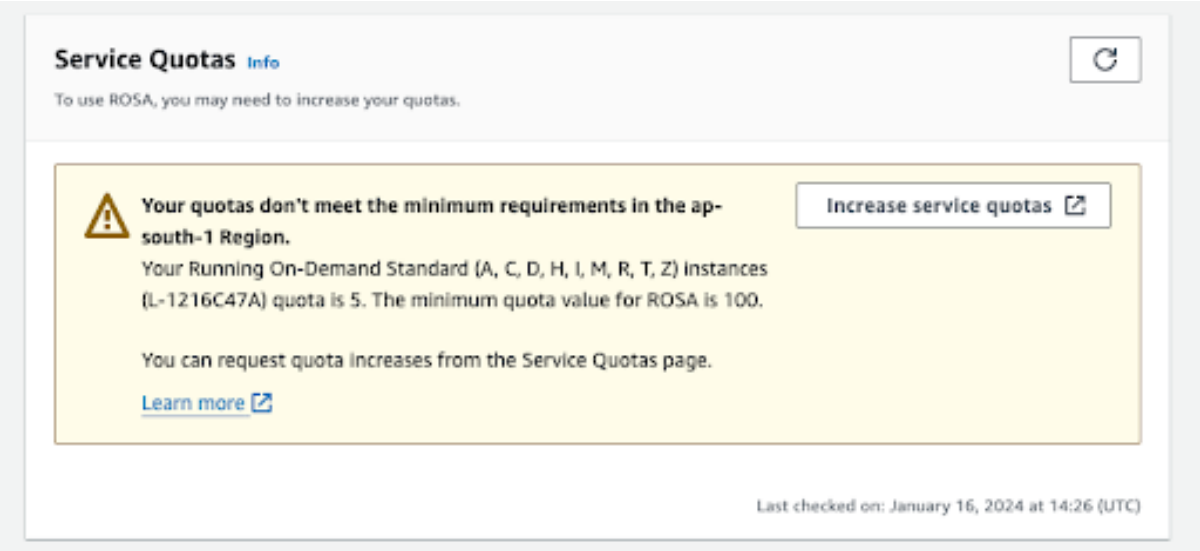
3. After the process is completed, you will see a confirmation:

Figure 2.3. Red Hat OpenShift Service on AWS enablement confirmation



4. Other sections on this verification page show the status of additional prerequisites. In case any of these prerequisites are not met, a corresponding message is shown. Here is an example of insufficient quotas in the selected region:

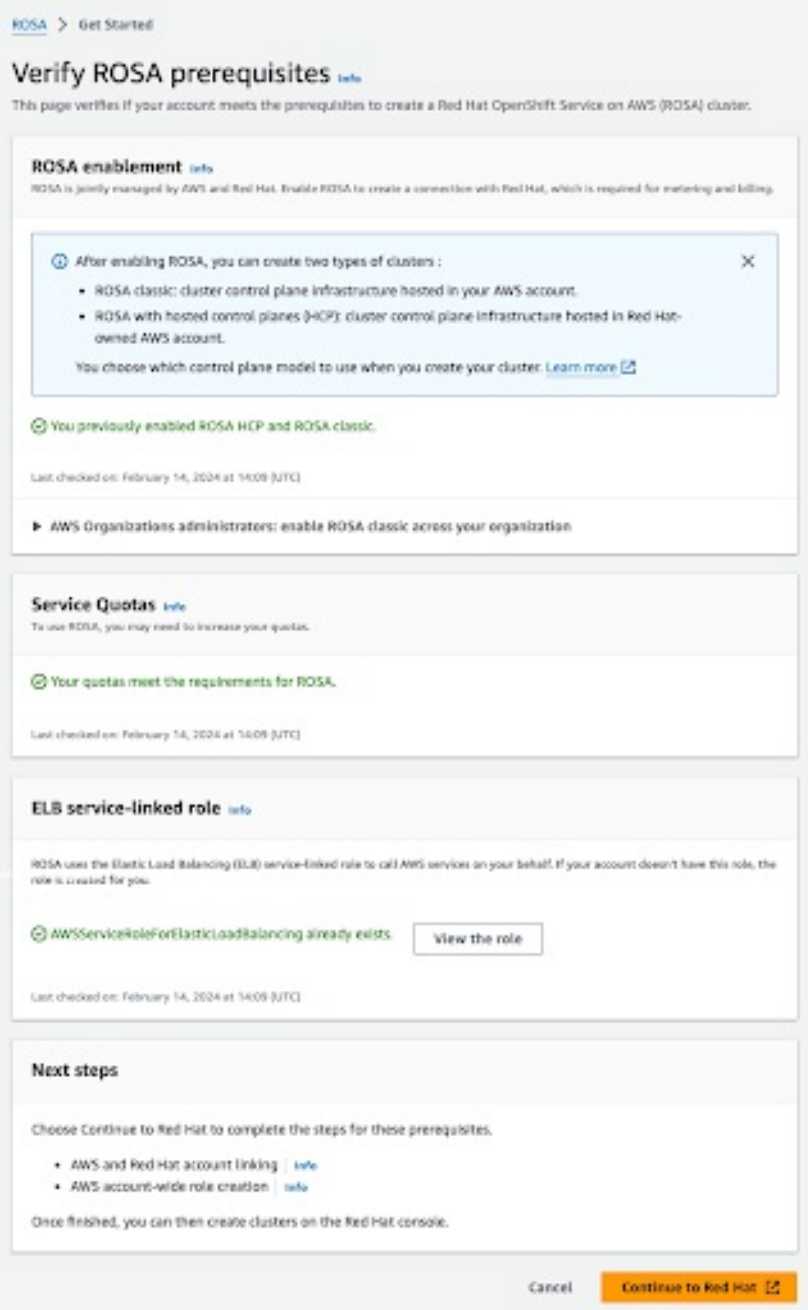
Figure 2.4. Service quotas



   a. Click the **Increase service quotas** button or use the **Learn more** link to get more information about the about how to manage service quotas. In the case of insufficient quotas, note that quotas are region-specific. You can use the region switcher in the upper right corner of the web console to re-run the quota check for any region you are interested in and then submit service quota increase requests as needed.

5. If all the prerequisites are met, the page will look like this:

Figure 2.5. Verify Red Hat OpenShift Service on AWS prerequisites

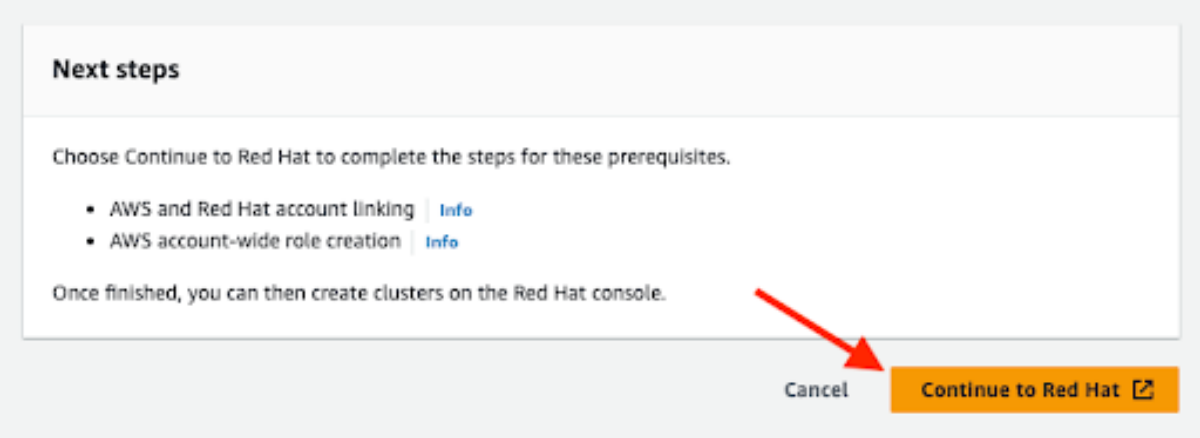


The ELB service-linked role is created for you automatically. You can click any of the small **Info** blue links to get contextual help and resources.

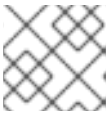
2.3. AWS AND RED HAT ACCOUNT AND SUBSCRIPTION LINKING

1. Click the orange **Continue to Red Hat** button to proceed with account linking:

Figure 2.6. Continue to Red Hat



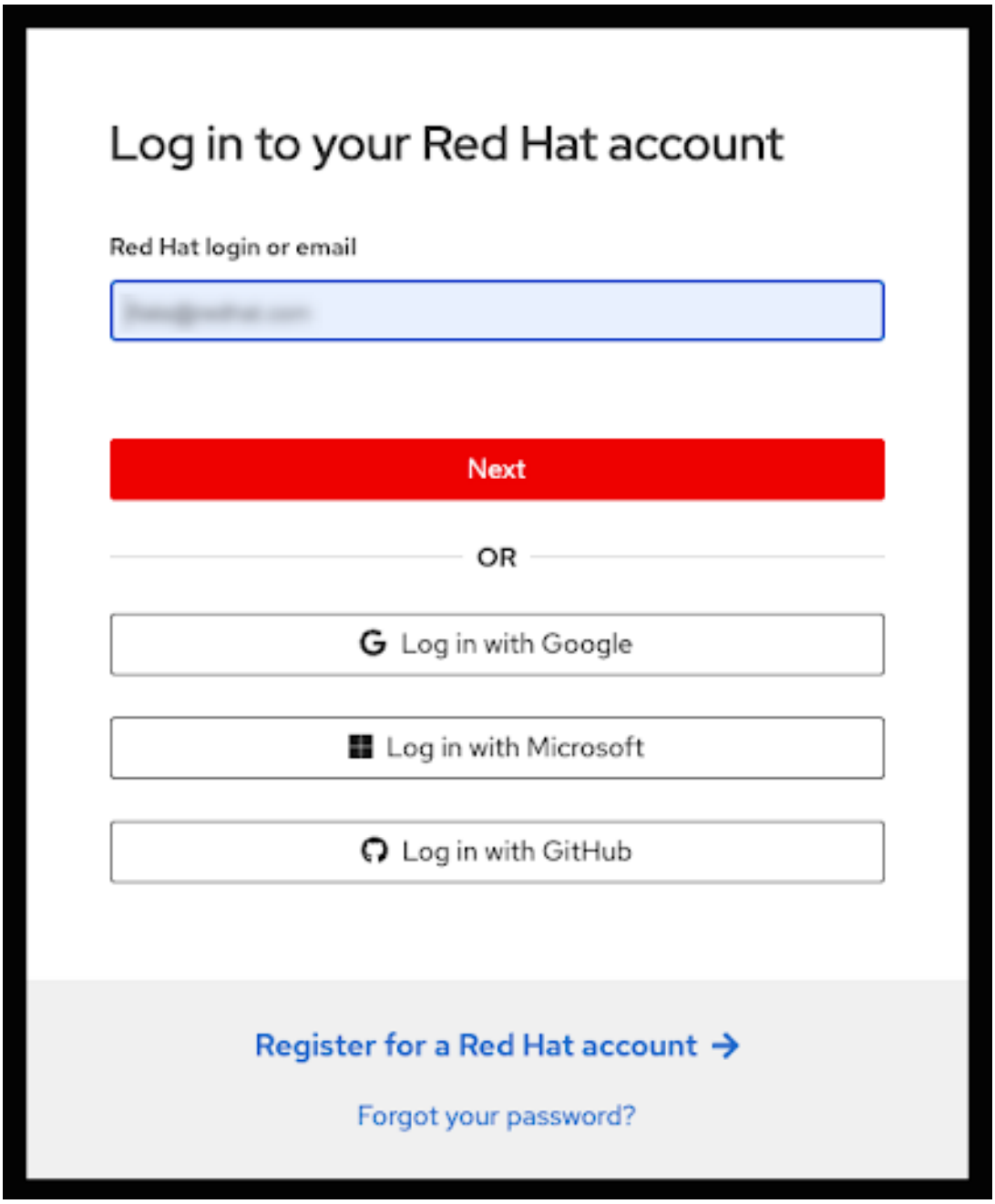
2. If you are not already logged in to your Red Hat account in your current browser's session, you will be asked to log in to your account:



NOTE

Your AWS account must be linked to a single Red Hat organization.

Figure 2.7. Log in to your Red Hat account



- You can also register for a new Red Hat account or reset your password on this page.
- Log in to the Red Hat account that you want to associate with the AWS account that has activated the Red Hat OpenShift Service on AWS product subscription.
- The AWS account used for service billing can only be associated with a single Red Hat account. Typically an AWS payer account is the one that is used to subscribe to Red Hat OpenShift Service on AWS and used for account linking and billing.
- All team members belonging to the same Red Hat organization can use the linked AWS account for service billing while creating Red Hat OpenShift Service on AWS clusters.

3. Complete the Red Hat account linking after reviewing the terms and conditions:

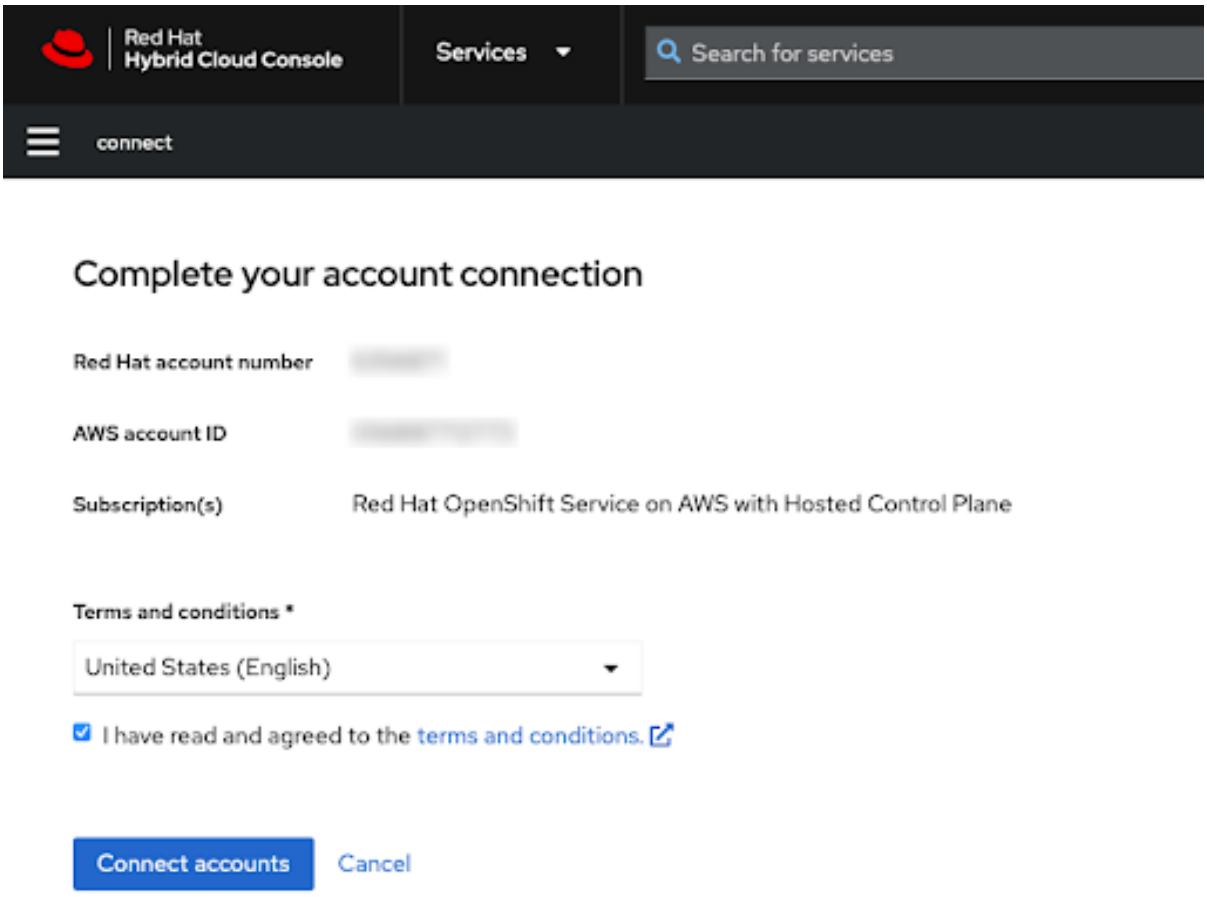
NOTE

This step is available only if the AWS account was not linked to any Red Hat account before.

This step is skipped if the AWS account is already linked to the user's logged in Red Hat account.

If the AWS account is linked to a different Red Hat account, an error will be displayed. See Correcting Billing Account Information for HCP clusters for troubleshooting.

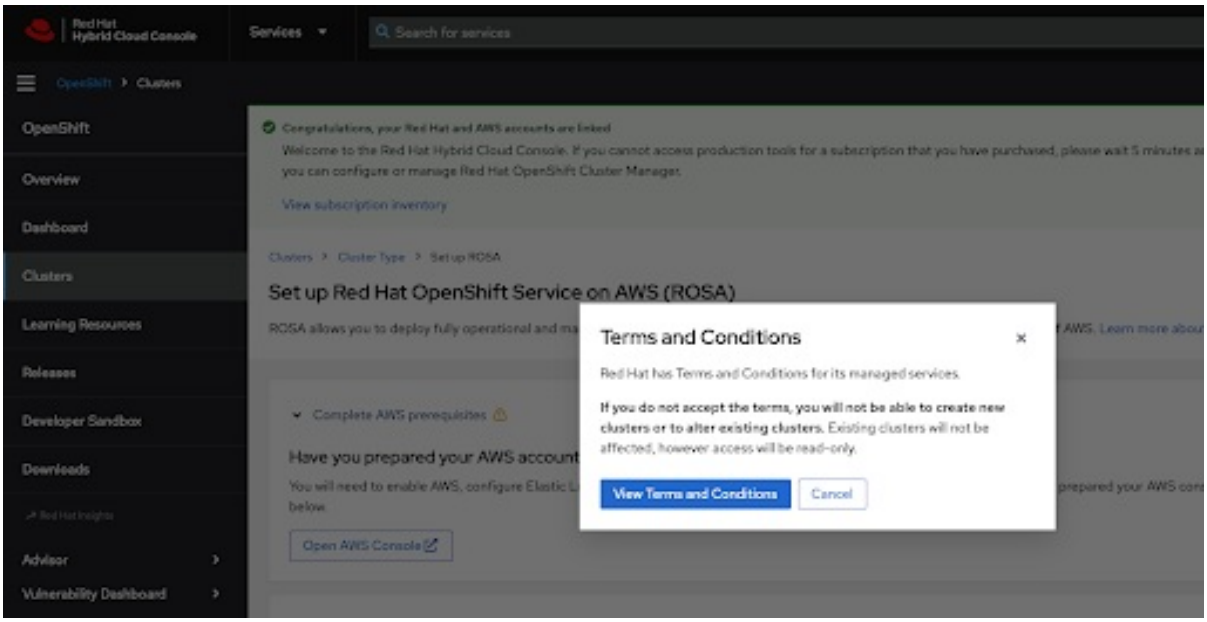
Figure 2.8. Complete your account connection



Both the Red Hat and AWS account numbers are shown on this screen.

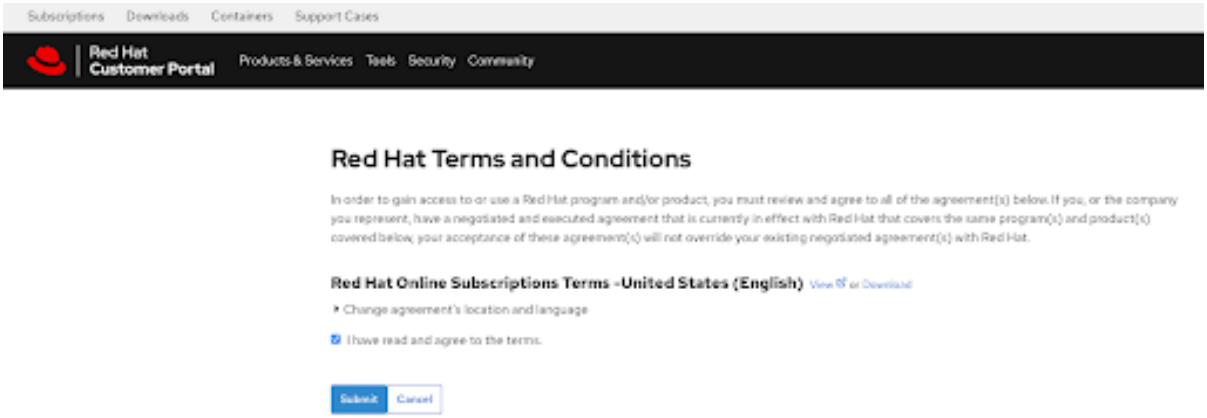
4. Click the **Connect accounts** button if you agree with the service terms.
If this is the first time you are using the Red Hat Hybrid Cloud Console, you will be asked to agree with the general managed services terms and conditions before being able to create the first cluster:

Figure 2.9. Terms and conditions



Additional terms that need to be reviewed and accepted are shown after clicking the **View Terms and Conditions** button:

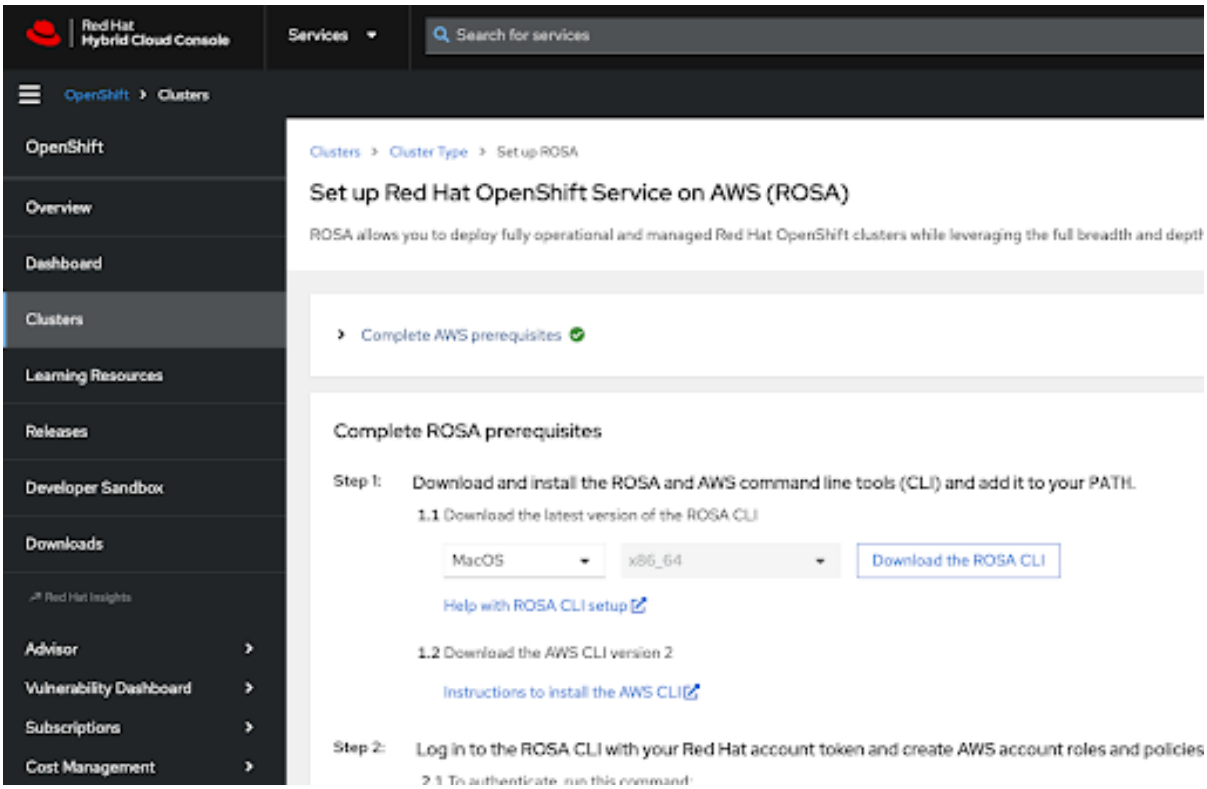
Figure 2.10. Red Hat terms and conditions



Submit your agreement once you have reviewed any additional terms when prompted at this time.

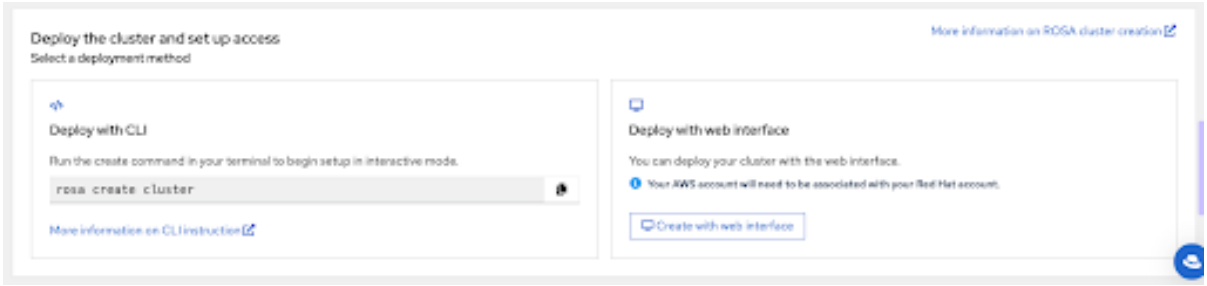
5. The Hybrid Cloud Console provides a confirmation that AWS account setup was completed and lists the prerequisites for cluster deployment:

Figure 2.11. Complete Red Hat OpenShift Service on AWS prerequisites



The last section of this page shows cluster deployment options, either using the **rosa** CLI or through the web console:

Figure 2.12. Deploy the cluster and set up access



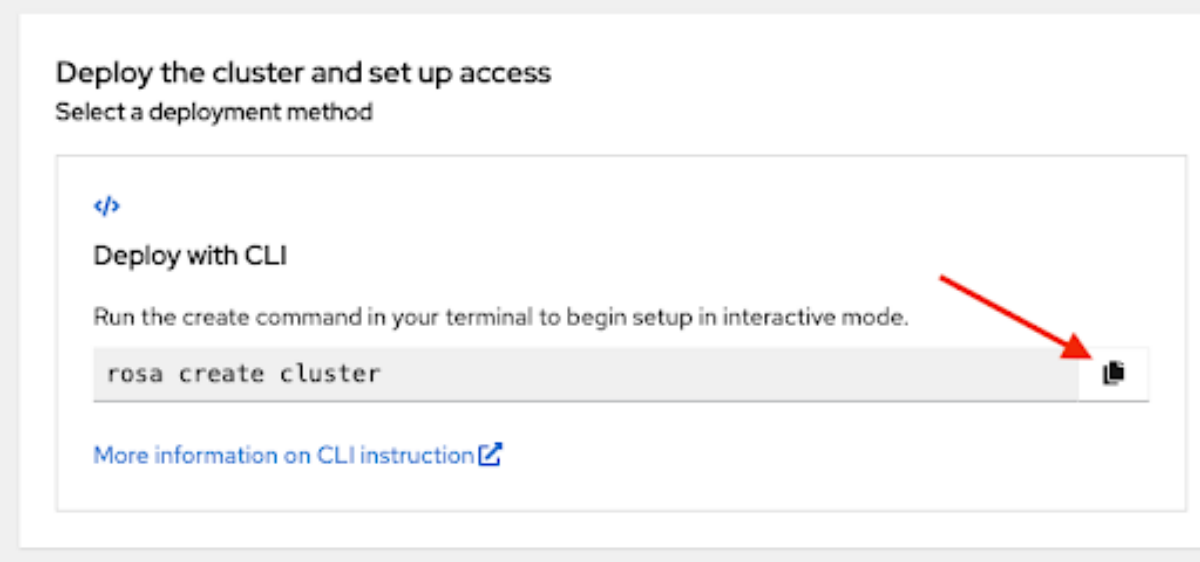
2.4. SELECTING THE AWS BILLING ACCOUNT FOR RED HAT OPENSHIFT SERVICE ON AWS DURING CLUSTER DEPLOYMENT USING THE CLI

IMPORTANT

Make sure that you have the most recent ROSA command-line interface (CLI) and AWS CLI installed and have completed the Red Hat OpenShift Service on AWS prerequisites covered in the previous section. See Help with ROSA CLI setup and Instructions to install the AWS CLI for more information.

1. Initiate the cluster deployment using the **rosa create cluster** command. You can click the **copy** button on the Set up Red Hat OpenShift Service on AWS (ROSA) console page and paste the command in your terminal. This launches the cluster creation process in interactive mode:

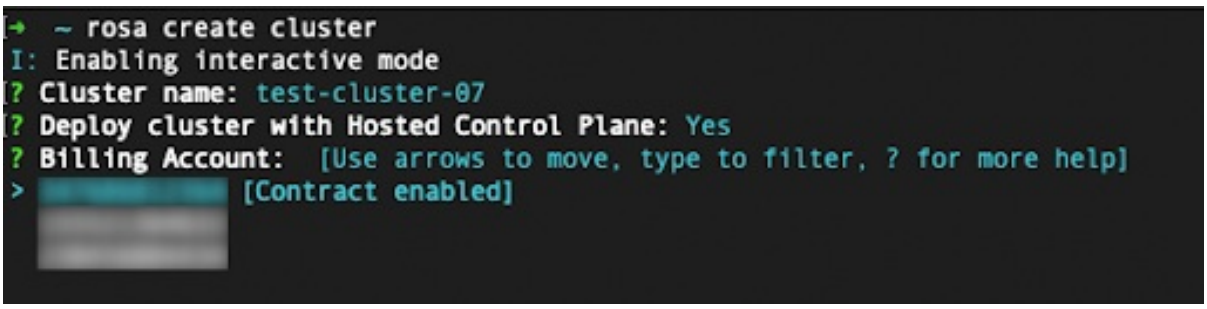
Figure 2.13. Deploy the cluster and set up access



2. To use a custom AWS profile, one of the non-default profiles specified in your ~/**.aws/credentials**, you can add the **–profile <profile_name>** selector to the rosa create cluster command so that the command looks like rosa create cluster **–profile stage**. If no AWS CLI profile is specified using this option, the default AWS CLI profile will determine the AWS infrastructure profile into which the cluster is deployed. The billing AWS profile is selected in one of the following steps.

3. When deploying a Red Hat OpenShift Service on AWS cluster, the billing AWS account needs to be specified:

Figure 2.14. Specify the Billing Account



- Only AWS accounts that are linked to the user's logged in Red Hat account are shown.
- The specified AWS account is charged for using the Red Hat OpenShift Service on AWS service.
- An indicator shows if the Red Hat OpenShift Service on AWS contract is enabled or not enabled for a given AWS billing account.
  - If you select an AWS billing account that shows the *Contract enabled* label, on-demand consumption rates are charged only after the capacity of your pre-paid contract is consumed.
  - AWS accounts without the *Contract enabled* label are charged the applicable on-demand consumption rates.

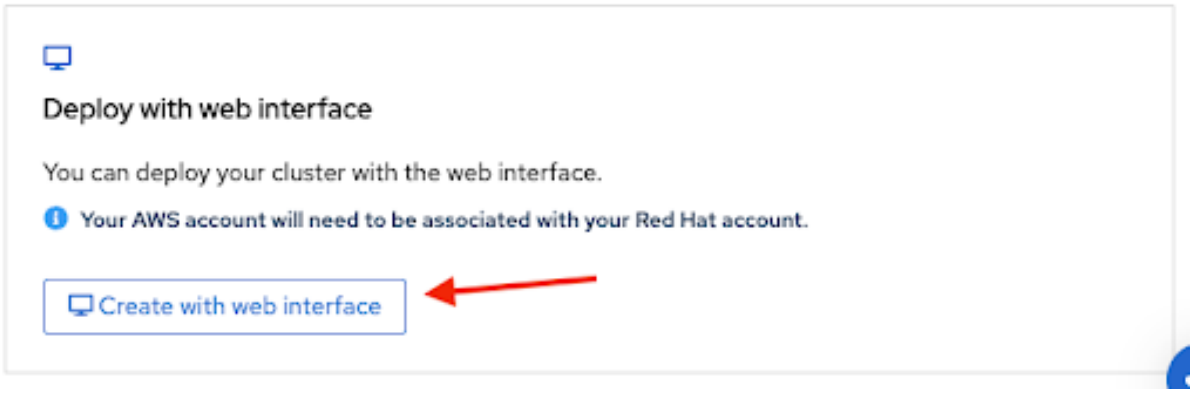
Additional resources

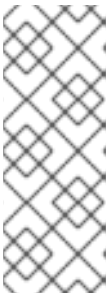
- The detailed cluster deployment steps are beyond the scope of this tutorial. See Creating Red Hat OpenShift Service on AWS clusters using the default options for more details about how to complete the Red Hat OpenShift Service on AWS cluster deployment using the CLI.

2.5. SELECTING THE AWS BILLING ACCOUNT FOR RED HAT OPENSHIFT SERVICE ON AWS DURING CLUSTER DEPLOYMENT USING THE WEB CONSOLE

1. A cluster can be created using the web console by selecting the second option in the bottom section of the introductory **Set up Red Hat OpenShift Service on AWS**page:

   Figure 2.15. Deploy with web interface





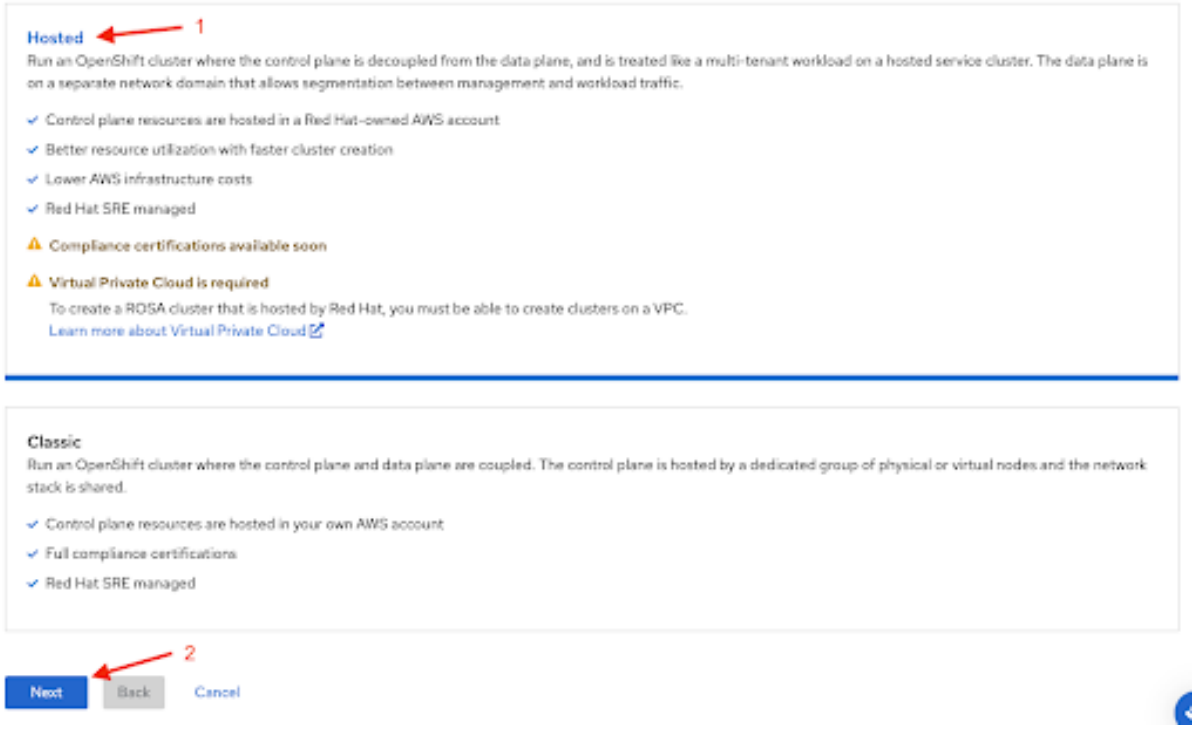
NOTE

Complete the prerequisites before starting the web console deployment process.

The **rosa** CLI is required for certain tasks, such as creating the account roles. If you are deploying Red Hat OpenShift Service on AWS for the first time, follow this the CLI steps until running the **rosa whoami** command, before starting the web console deployment steps.

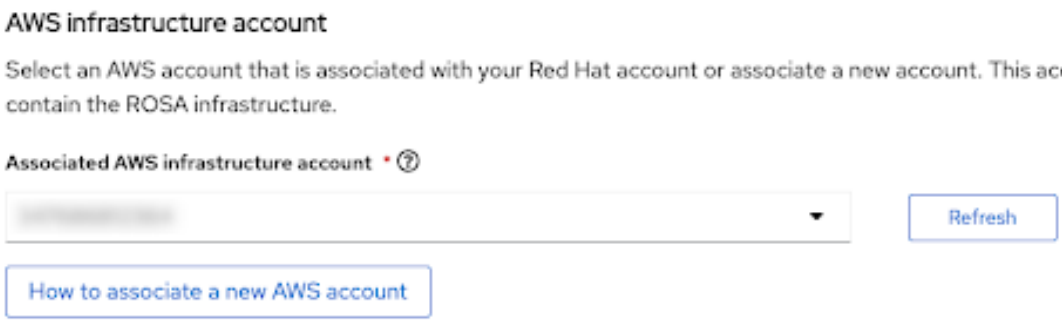
2. The first step when creating a Red Hat OpenShift Service on AWS cluster using the web console is the control plane selection. Make sure the **Hosted** option is selected before clicking the **Next** button:

Figure 2.16. Select hosted option



3. The next step **Accounts and roles** allows you specifying the infrastructure AWS account, into which the Red Hat OpenShift Service on AWS cluster is deployed and where the resources are consumed and managed:

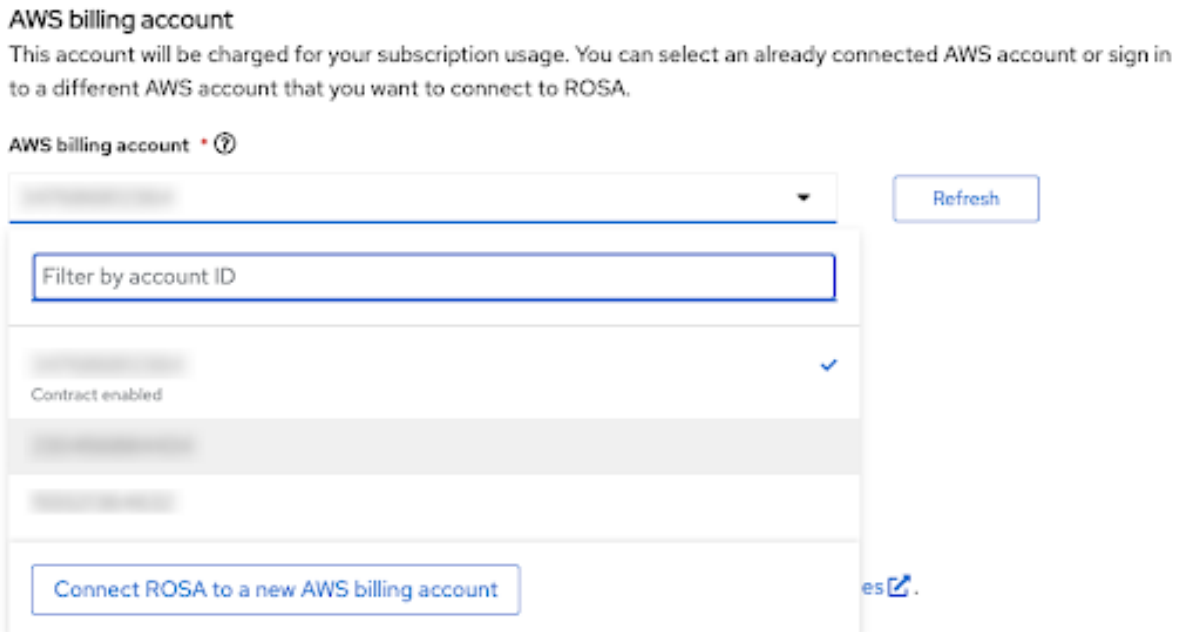
Figure 2.17. AWS infrastructure account



- Click the **How to associate a new AWS account** if you don not see the account into which you want to deploy the Red Hat OpenShift Service on AWS cluster for detailed information on how to create or link account roles for this association.
- The **rosa** CLI is used for this.
- If you are using multiple AWS accounts and have their profiles configured for the AWS CLI, you can use the **--profile** selector to specify the AWS profile when working with the **rosa** CLI commands.

4. The billing AWS account is selected in the immediately following section:

Figure 2.18. AWS billing account



- Only AWS accounts that are linked to the user's logged in Red Hat account are shown.
- The specified AWS account is charged for using the Red Hat OpenShift Service on AWS service.
- An indicator shows if the Red Hat OpenShift Service on AWS contract is enabled or not enabled for a given AWS billing account.
    - If you select an AWS billing account that shows the *Contract enabled* label, on-demand consumption rates are charged only after the capacity of your pre-paid contract is consumed.
    - AWS accounts without the *Contract enabled* label are charged the applicable on-demand consumption rates.

The following steps past the billing AWS account selection are beyond the scope of this tutorial.

Additional resources

- For information on using the CLI to create a cluster, see Creating a Red Hat OpenShift Service on AWS cluster using the CLI.
- See this learning path for more details on how to complete cluster deployment using the web console.

CHAPTER 3. TUTORIAL: RED HAT OPENSHIFT SERVICE ON AWS PRIVATE OFFER ACCEPTANCE AND SHARING

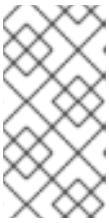
This guide describes how to accept a private offer for Red Hat OpenShift Service on AWS and how to ensure that all team members can use the private offer for the clusters they provision.

Red Hat OpenShift Service on AWS costs are composed of the AWS infrastructure costs and the Red Hat OpenShift Service on AWS service costs. AWS infrastructure costs, such as the EC2 instances that are running the needed workloads, are charged to the AWS account where the infrastructure is deployed. Red Hat OpenShift Service on AWS service costs are charged to the AWS account specified as the "AWS billing account" when deploying a cluster.

The cost components can be billed to different AWS accounts. Detailed description of how the Red Hat OpenShift Service on AWS service cost and AWS infrastructure costs are calculated can be found on the Red Hat OpenShift Service on AWS Pricing page .

3.1. ACCEPTING A PRIVATE OFFER

1. When you get a private offer for Red Hat OpenShift Service on AWS, you are provided with a unique URL that is accessible only by a specific AWS account ID that was specified by the seller.

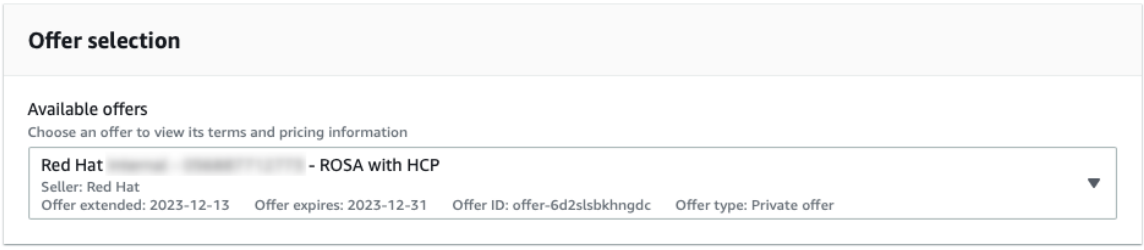


NOTE

Verify that you are logged in using the AWS account that was specified as the buyer. Attempting to access the offer using another AWS account produces a "page not found" error message as shown in Figure 11 in the troubleshooting section below.

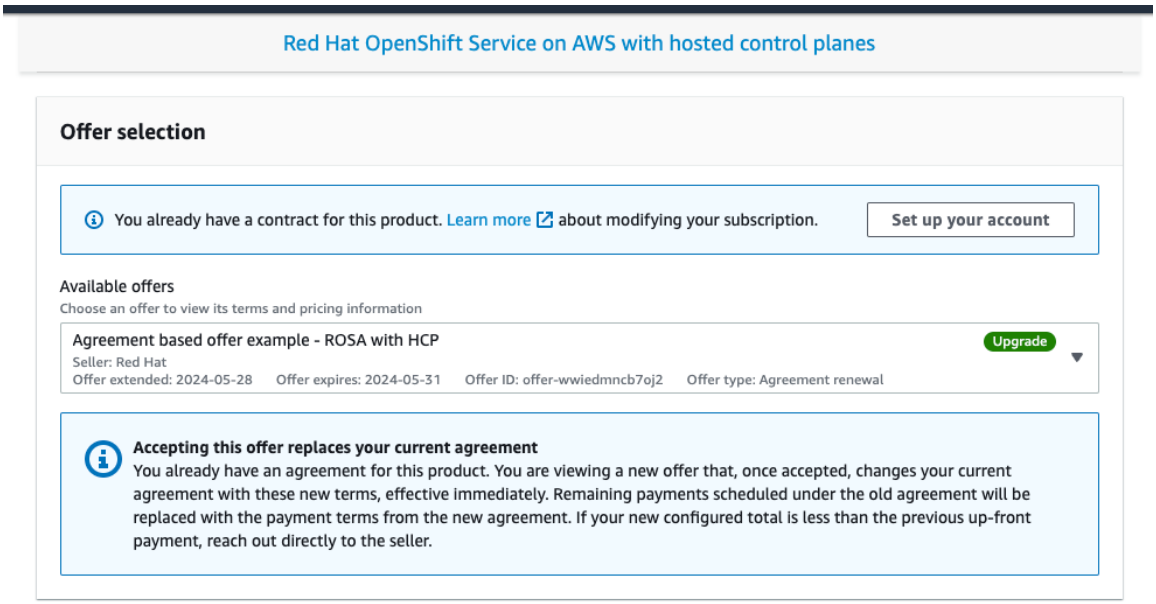
   a. You can see the offer selection drop down menu with a regular private offer pre-selected in Figure 1. This type of offer can be accepted only if the Red Hat OpenShift Service on AWS was not activated before using the public offer or another private offer.

Figure 3.1. Regular private offer



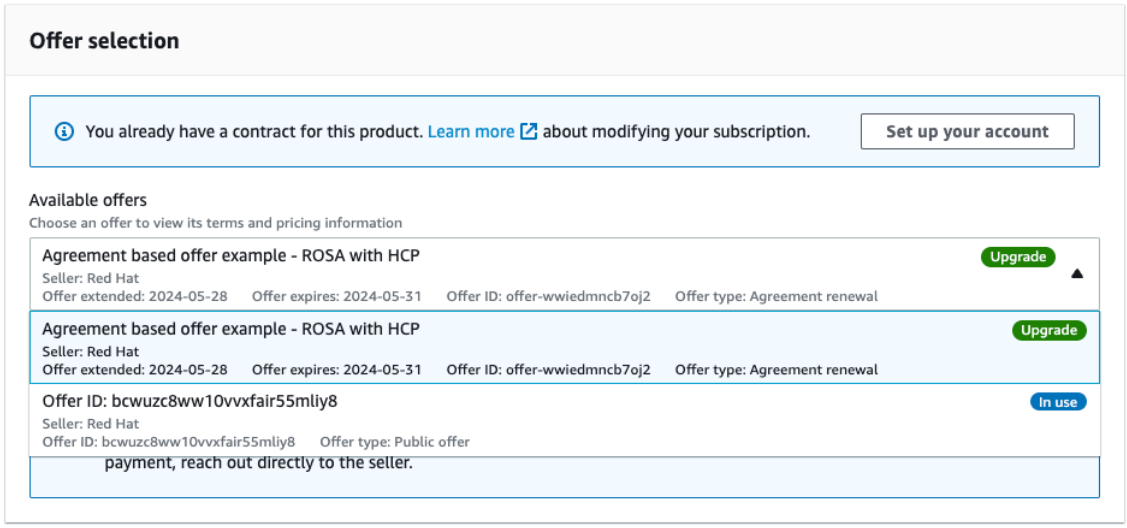
   b. You can see a private offer that was created for an AWS account that previously activated Red Hat OpenShift Service on AWS using the public offer, showing the product name and the selected private offer labeled as "Upgrade", that replaces the currently running contract for Red Hat OpenShift Service on AWS in Figure 2.

Figure 3.2. Private offer selection selection screen



c. The drop down menu allows selecting between multiple offers, if available. The previously activated public offer is shown together with the newly provided agreement based offer that is labeled as "Upgrade" in Figure 3.

Figure 3.3. Private offer selection dropdown

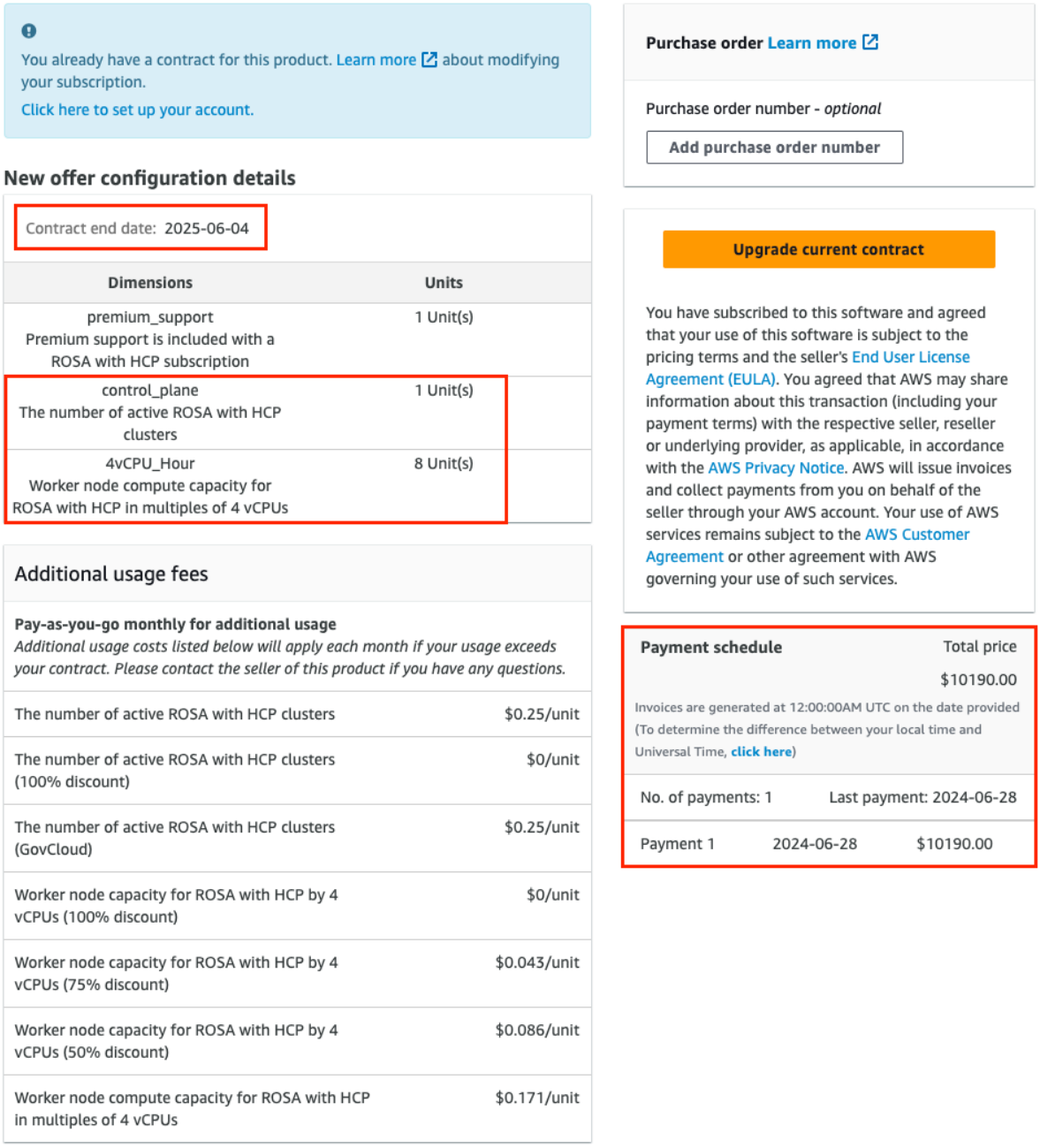


2. Verify that your offer configuration is selected. Figure 4 shows the bottom part of the offer page with the offer details.

NOTE

The contract end date, the number of units included with the offer, and the payment schedule. In this example, 1 cluster and up to 3 nodes utilizing 4 vCPUs are included.

Figure 3.4. Private offer details



3. Optional: you can add your own purchase order (PO) number to the subscription that is being purchased, so it is included on your subsequent AWS invoices. Also, check the "Additional usage fees" that are charged for any usage above the scope of the "New offer configuration details".

NOTE

Private offers have several available configurations.

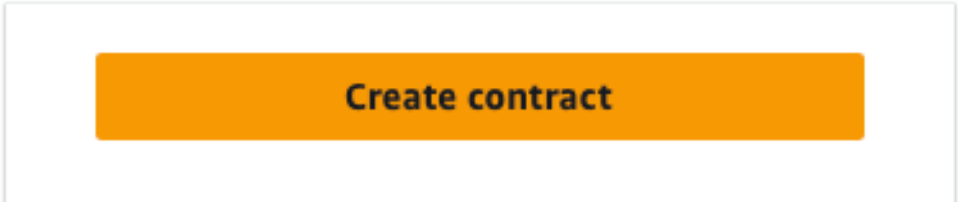
- It is possible that the private offer you are accepting is set up with a fixed future start date.
- If you do not have another active Red Hat OpenShift Service on AWS subscription at the time of accepting the private offer, a public offer or an older private offer entitlement, accept the private offer itself and continue with the account linking and cluster deployment steps after the specified service start date.

You must have an active Red Hat OpenShift Service on AWS entitlement to complete these steps. Service start dates are always reported in the UTC time zone

4. Create or upgrade your contract.

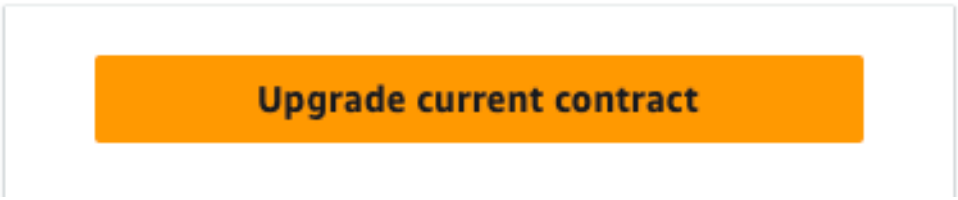
   a. For private offers accepted by an AWS account that does not have Red Hat OpenShift Service on AWS activated yet and is creating the first contract for this service, click the **Create contract button**

   Figure 3.5. Create contract button

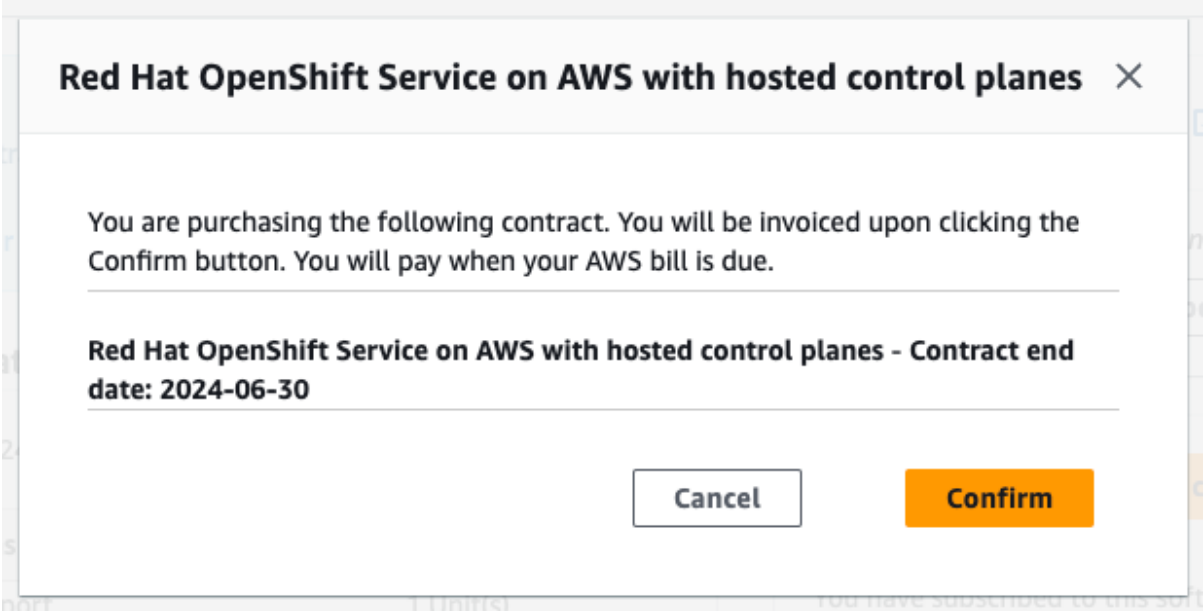
   b. For agreement-based offers, click the **Upgrade current contract** button shown in Figures 4 and 6.

   Figure 3.6. Upgrade contract button

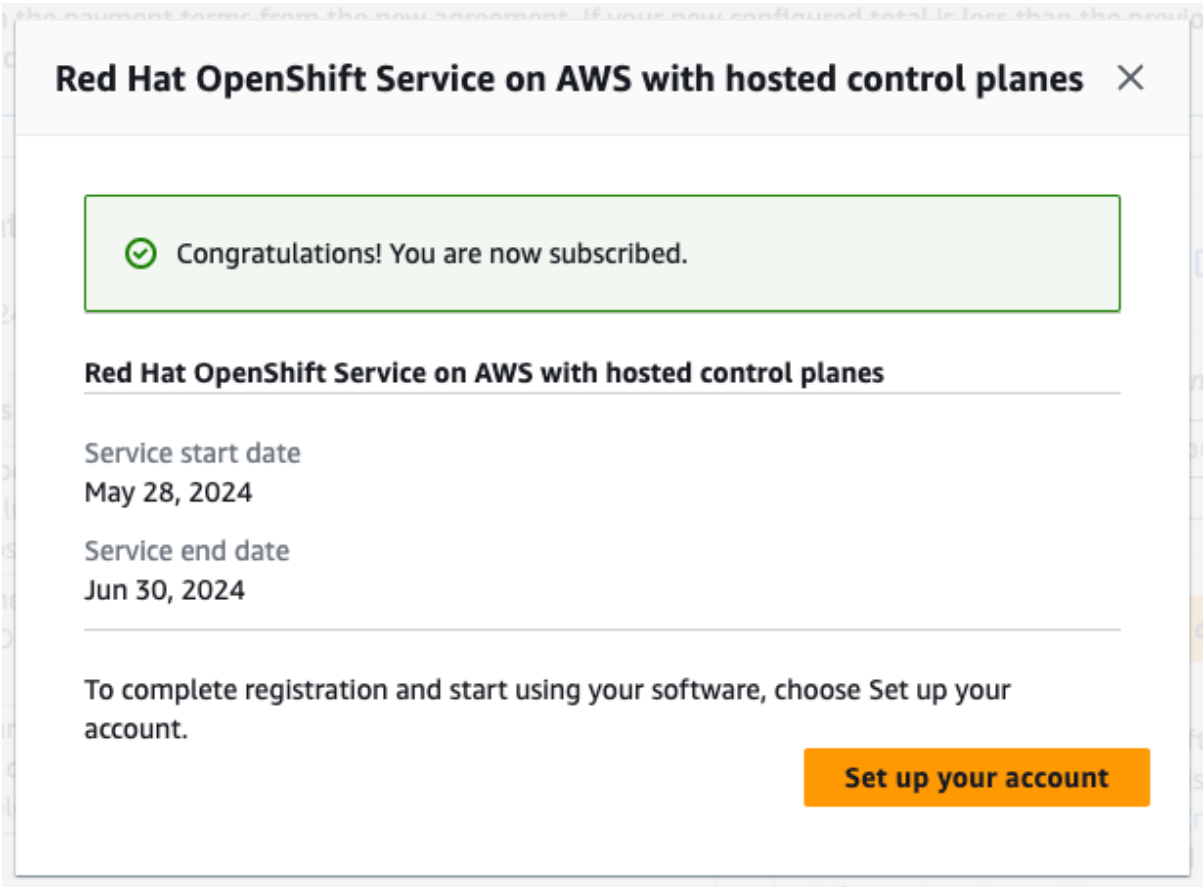
5. Click **Confirm**.

Figure 3.7. Private offer acceptance confirmation window



6. If the accepted private offer service start date is set to be immediately following the offer acceptance, click the **Set up your account** button in the confirmation modal window.

Figure 3.8. Subscription confirmation



7. If the accepted private offer has a future start date specified, return to the private offer page after the service start date, and click the **Setup your account** button to proceed with the Red Hat and AWS account linking.

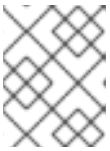
NOTE

With no agreement active, the account linking described below is not triggered, the "Account setup" process can be done only after the "Service start date".

These are always in UTC time zone.

3.2. SHARING A PRIVATE OFFER

1. Clicking the **Set up your account**button in the previous step takes you to the AWS and Red Hat account linking step. At this time, you are already logged in with the AWS account that accepted the offer. If you are not logged in with a Red Hat account, you will be prompted to do so.
   Red Hat OpenShift Service on AWS entitlement is shared with other team members through your Red Hat organization account. All existing users in the same Red Hat organization are able to select the billing AWS account that accepted the private offer by following the above described steps. You can manage users in your Red Hat organization , when logged in as the Red Hat organization administrator, and invite or create new users.



NOTE

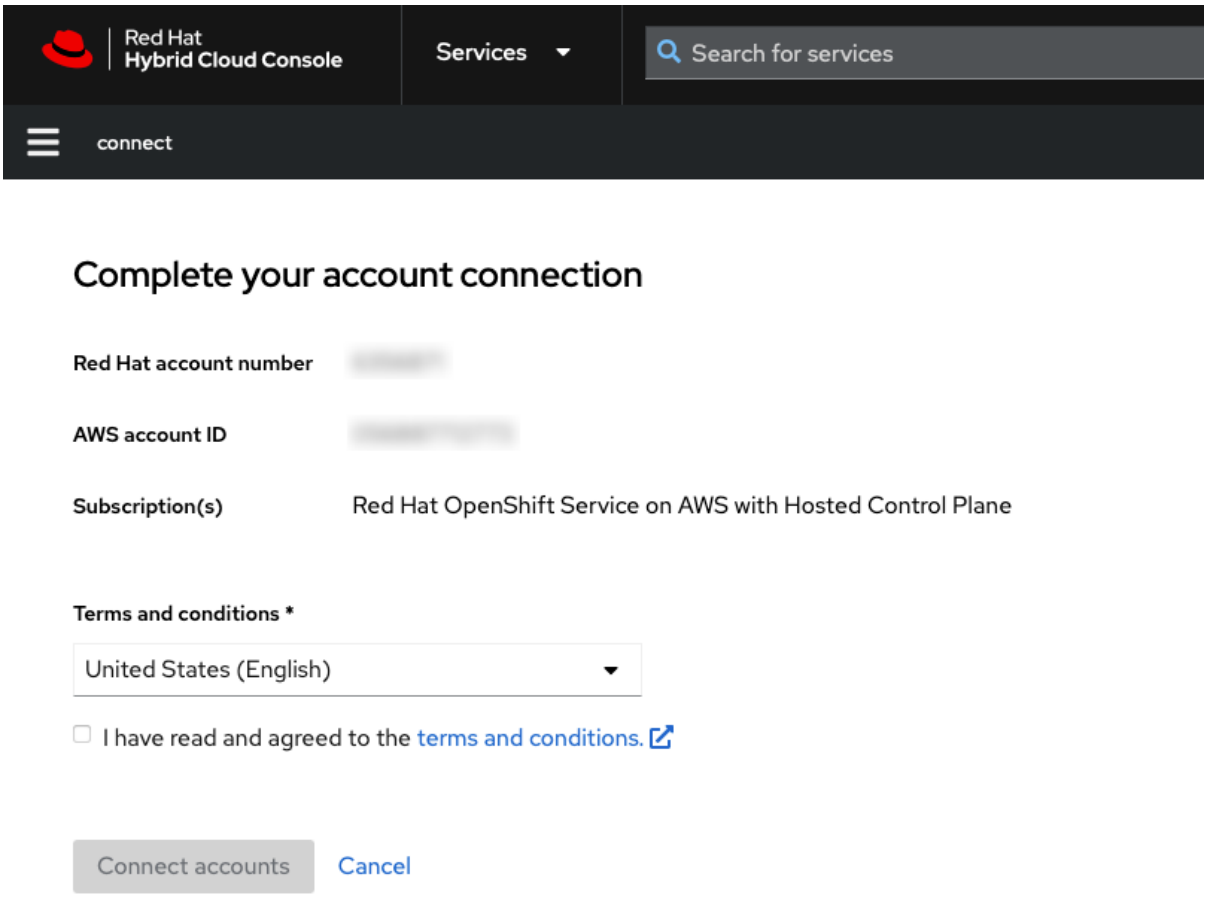
Red Hat OpenShift Service on AWS private offer cannot be shared with AWS linked accounts through the AWS License Manager.

2. Add any users that you want to deploy Red Hat OpenShift Service on AWS clusters. Check this user management FAQ for more details about Red Hat account user management tasks.

3. Verify that the already logged in Red Hat account includes all users that are meant to be Red Hat OpenShift Service on AWS cluster deployers benefiting from the accepted private offer.

4. Verify that the Red Hat account number and the AWS account ID are the desired accounts that are to be linked. This linking is unique and a Red Hat account can be connected only with a single AWS (billing) account.

Figure 3.9. AWS and Red Hat accounts connection



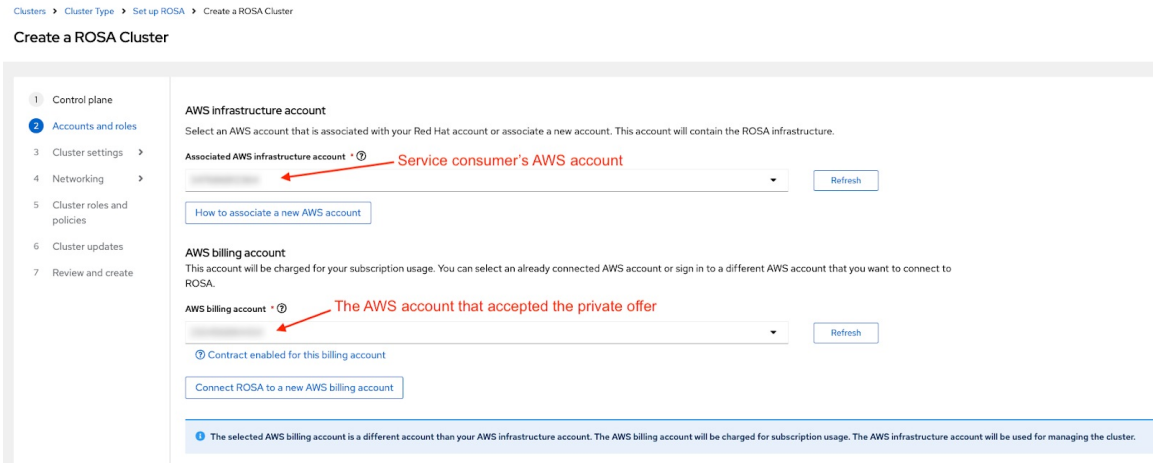
5. If you want to link the AWS account with another Red Hat account than is shown on this page in Figure 9, log out from the Red Hat Hybrid Cloud Console before connecting the accounts and repeat the step of setting the account by returning to the private offer URL that is already accepted.
   An AWS account can be connected with a single Red Hat account only. Once Red Hat and AWS accounts are connected, this cannot be changed by the user. If a change is needed, the user must create a support ticket.

6. Agree to the terms and conditions and then click **Connect accounts**.

3.3. AWS BILLING ACCOUNT SELECTION

- When deploying Red Hat OpenShift Service on AWS clusters, verify that end users select the AWS billing account that accepted the private offer.
- When using the web interface for deploying Red Hat OpenShift Service on AWS, the Associated AWS infrastructure account" is typically set to the AWS account ID used by the administrator of the cluster that is being created.
  - This can be the same AWS account as the billing AWS account.
  - AWS resources are deployed into this account and all the billing associated with those resources are processed accordingly.

Figure 3.10. Infrastructure and billing AWS account selection during Red Hat OpenShift Service on AWS cluster deployment



- The drop-down for the AWS billing account on the screenshot above should be set to the AWS account that accepted the private offer, providing the purchased quota is intended to be used by the cluster that is being created. If different AWS accounts are selected in the infrastructure and billing "roles", the blue informative note visible in Figure 10 is shown.

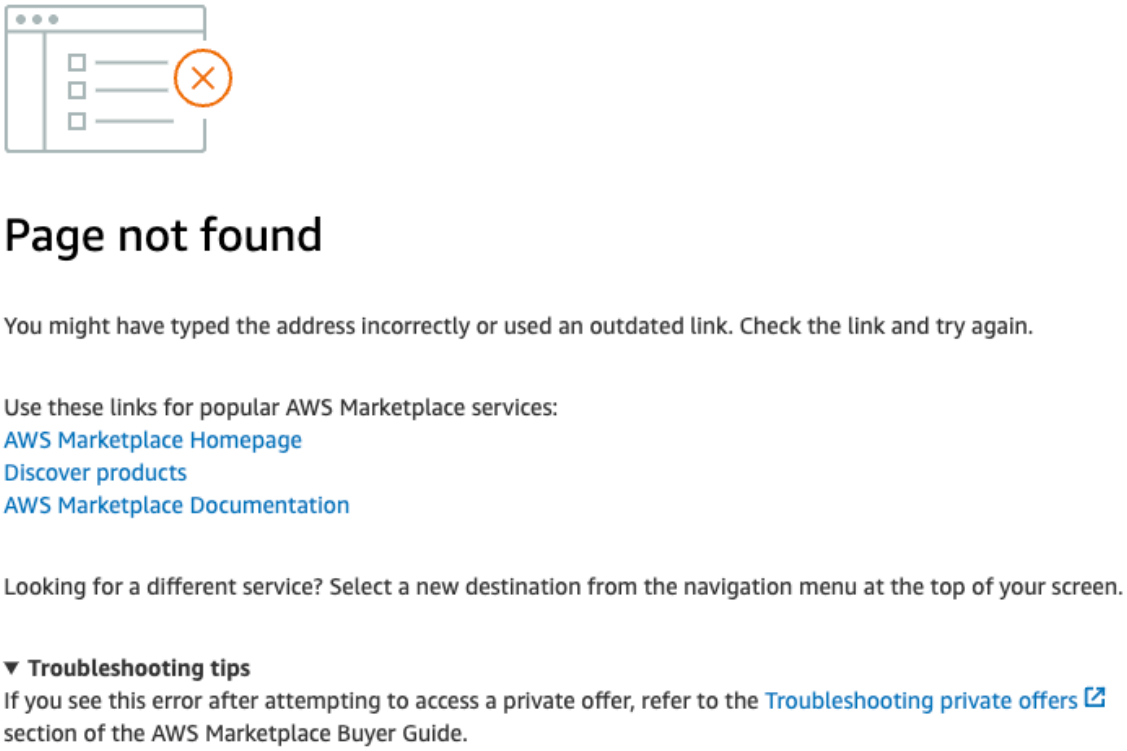
3.4. TROUBLESHOOTING

The most frequent issues associated with private offer acceptance and Red Hat account linking.

3.4.1. Accessing a private offer using a different AWS account

- If you try accessing the private offer when logged in under AWS account ID that is not defined in the offer, and see the message shown in Figure 11, then verify that you are logged in as the desired AWS billing account.

Figure 3.11. HTTP 404 error when using the private offer URL

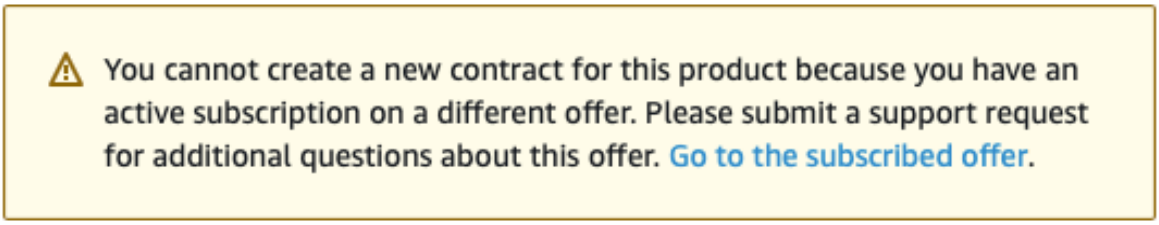


- Contact the seller if you need the private offer to be extended to another AWS account.

3.4.2. The private offer cannot be accepted because of active subscription

- If you try accessing a private offer that was created for the first time Red Hat OpenShift Service on AWS activation, while you already have Red Hat OpenShift Service on AWS activated using another public or private offer, and see the following notice, then contact the seller who provided you with the offer.
  The seller can provide you with a new offer that will seamlessly replace your current agreement, without a need to cancel your previous subscription.

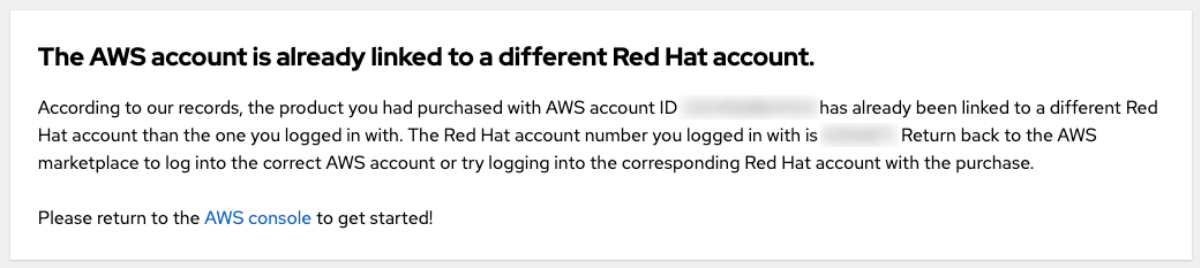
Figure 3.12. Existing subscription preventing private offer acceptance



3.4.3. The AWS account is already linked to a different Red Hat account

- If you see the error message "AWS account is already linked to a different Red Hat account" when you try to connect the AWS account that accepted the private offer with a presently logged-in Red Hat user, then the AWS account is already connected to another Red Hat user.

Figure 3.13. AWS account is already linked to a different Red Hat account



- You can either log in using another Red Hat account or another AWS account.
    - However, since this guide pertains to private offers, the assumption is that you are logged in with the AWS account that was specified as the buyer and already accepted the private offer so it is intended to be used as the billing account. Logging in as another AWS account is not expected after a private offer was accepted.
- You can still log in with another Red Hat user which is already connected to the AWS account that accepted the private offer. Other Red Hat users belonging to the same Red Hat organization are able to use the linked AWS account as the Red Hat OpenShift Service on AWS AWS billing account when creating clusters as seen in Figure 10.
- If you believe that the existing account linking might not be correct, see the "My team members belong to different Red Hat organizations" question below for tips on how you can proceed.

3.4.4. My team members belong to different Red Hat organizations

- An AWS account can be connected to a single Red Hat account only. Any user that wants to create a cluster and benefit from the private offer granted to this AWS account needs to be in the same Red Hat account. This can be achieved by inviting the user to the same Red Hat account and creating a new Red Hat user.

3.4.5. Incorrect AWS billing account was selected when creating a cluster

- If the user selected an incorrect AWS billing account, the fastest way to fix this is to delete the cluster and create a new one, while selecting the correct AWS billing account.
- If this is a production cluster that cannot be easily deleted, please contact Red Hat support to change the billing account for an existing cluster. Expect some turnaround time for this to be resolved.

CHAPTER 4. TUTORIAL: DEPLOYING RED HAT OPENSHIFT SERVICE ON AWS WITH A CUSTOM DNS RESOLVER

A custom DHCP option set enables you to customize your VPC with your own DNS server, domain name, and more. Red Hat OpenShift Service on AWS clusters support using custom DHCP option sets. By default, Red Hat OpenShift Service on AWS clusters require setting the "domain name servers" option to **AmazonProvidedDNS** to ensure successful cluster creation and operation. Customers who want to use custom DNS servers for DNS resolution must do additional configuration to ensure successful Red Hat OpenShift Service on AWS cluster creation and operation.

In this tutorial, we will configure our DNS server to forward DNS lookups for specific DNS zones (further detailed below) to an Amazon Route 53 Inbound Resolver .

NOTE

This tutorial uses the open-source BIND DNS server (**named**) to demonstrate the configuration necessary to forward DNS lookups to an Amazon Route 53 Inbound Resolver located in the VPC you plan to deploy a Red Hat OpenShift Service on AWS cluster into. Refer to the documentation of your preferred DNS server for how to configure zone forwarding.

4.1. PREREQUISITES

- ROSA CLI (**rosa**)
- AWS CLI (**aws**)
- A manually created AWS VPC
- A DHCP option set configured to point to a custom DNS server and set as the default for your VPC

4.2. SETTING UP YOUR ENVIRONMENT

1. Configure the following environment variables:

```
$ export VPC_ID=<vpc_ID> 1
$ export REGION=<region> 2
$ export VPC_CIDR=<vpc_CIDR> 3
```

1. Replace **<vpc_ID>** with the ID of the VPC you want to install your cluster into.
2. Replace **<region>** with the AWS region you want to install your cluster into.
3. Replace **<vpc_CIDR>** with the CIDR range of your VPC.

2. Ensure all fields output correctly before moving to the next section:

```
$ echo "VPC ID: ${VPC_ID}, VPC CIDR Range: ${VPC_CIDR}, Region: ${REGION}"
```

4.3. CREATE AN AMAZON ROUTE 53 INBOUND RESOLVER

Use the following procedure to deploy an Amazon Route 53 Inbound Resolver in the VPC we plan to deploy the cluster into.

WARNING

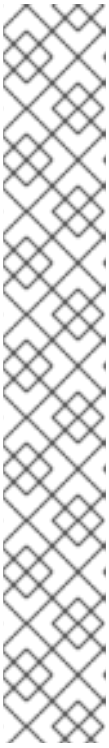
In this example, we deploy the Amazon Route 53 Inbound Resolver into the same VPC the cluster will use. If you want to deploy it into a separate VPC, you must manually associate the private hosted zone(s) detailed below **once cluster creation is started**. You cannot associate the zone before the cluster creation process begins. Failure to associate the private hosted zone during the cluster creation process will result in cluster creation failures.

1. Create a security group and allow access to ports **53/tcp** and **53/udp** from the VPC:

```
$ SG_ID=$(aws ec2 create-security-group --group-name rosa-inbound-resolver --description "Security group for ROSA inbound resolver" --vpc-id ${VPC_ID} --region ${REGION} --output text)
$ aws ec2 authorize-security-group-ingress --group-id ${SG_ID} --protocol tcp --port 53 --cidr ${VPC_CIDR} --region ${REGION}
$ aws ec2 authorize-security-group-ingress --group-id ${SG_ID} --protocol udp --port 53 --cidr ${VPC_CIDR} --region ${REGION}
```

2. Create an Amazon Route 53 Inbound Resolver in your VPC:

```
$ RESOLVER_ID=$(aws route53resolver create-resolver-endpoint \
  --name rosa-inbound-resolver \
  --creator-request-id rosa-$(date '+%Y-%m-%d') \
  --security-group-ids ${SG_ID} \
  --direction INBOUND \
  --ip-addresses $(aws ec2 describe-subnets --filter Name=vpc-id,Values=${VPC_ID} --region ${REGION} | jq -jr '.Subnets | map("SubnetId=\(.SubnetId) ") | .[]') \
  --region ${REGION} \
  --output text \
  --query 'ResolverEndpoint.Id')
```



NOTE

The above command attaches Amazon Route 53 Inbound Resolver endpoints to *all subnets* in the provided VPC using dynamically allocated IP addresses. If you prefer to manually specify the subnets and/or IP addresses, run the following command instead:

```
$ RESOLVER_ID=$(aws route53resolver create-resolver-endpoint \
  --name rosa-inbound-resolver \
  --creator-request-id rosa-$(date '+%Y-%m-%d') \
  --security-group-ids ${SG_ID} \
  --direction INBOUND \
  --ip-addresses SubnetId=<subnet_ID>,Ip=<endpoint_IP> SubnetId=
<subnet_ID>,Ip=<endpoint_IP> \ 1
  --region ${REGION} \
  --output text \
  --query 'ResolverEndpoint.Id')
```

1 Replace **<subnet_ID>** with the subnet IDs and **<endpoint_IP>** with the static IP addresses you want inbound resolver endpoints added to.

3. Get the IP addresses of your inbound resolver endpoints to configure in your DNS server configuration:

```
$ aws route53resolver list-resolver-endpoint-ip-addresses \
  --resolver-endpoint-id ${RESOLVER_ID} \
  --region=${REGION} \
  --query 'IpAddresses[*].Ip'
```

Example output

```
[
    "10.0.45.253",
    "10.0.23.131",
    "10.0.148.159"
]
```

4.4. CONFIGURE YOUR DNS SERVER

Use the following procedure to configure your DNS server to forward the necessary private hosted zones to your Amazon Route 53 Inbound Resolver.

4.4.1. Red Hat OpenShift Service on AWS

Red Hat OpenShift Service on AWS clusters require you to configure DNS forwarding for two private hosted zones:

- **<cluster-name>.hypershift.local**
- **rosa.<domain-prefix>.<unique-ID>.p3.openshiftapps.com**

These Amazon Route 53 private hosted zones are created during cluster creation. The **cluster-name** and **domain-prefix** are customer-specified values, but the **unique-ID** is randomly generated during

cluster creation and cannot be preselected. As such, you must wait for the cluster creation process to begin before configuring forwarding for the **p3.openshiftapps.com** private hosted zone.

1. Before the cluster is created, configure your DNS server to forward all DNS requests for **<cluster-name>.hypershift.local** to your Amazon Route 53 Inbound Resolver endpoints. For BIND DNS servers, edit your **/etc/named.conf** file in your favorite text editor and add a new zone using the below example:

   Example

   ```
   zone "<cluster-name>.hypershift.local" { 1
     type forward;
     forward only;
     forwarders { 2
       10.0.45.253;
       10.0.23.131;
       10.0.148.159;
     };
   };
   ```

   1 Replace **<cluster-name>** with your Red Hat OpenShift Service on AWS cluster name.

   2 Replace with the IP addresses of your inbound resolver endpoints collected above, ensuring that following each IP address there is a **;**.

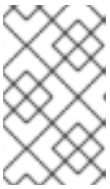
2. Create your cluster.

3. Once your cluster has begun the creation process, locate the newly created private hosted zone:

   ```
   $ aws route53 list-hosted-zones-by-vpc \
     --vpc-id ${VPC_ID} \
     --vpc-region ${REGION} \
     --query 'HostedZoneSummaries[*].Name' \
     --output table
   ```

   Example output

   ```
   ----------------------------------------------------
   |              ListHostedZonesByVPC                |
   +--------------------------------------------------+
   |  rosa.domain-prefix.lkmb.p3.openshiftapps.com.   |
   |  cluster-name.hypershift.local.                  |
   +--------------------------------------------------+
   ```

   NOTE

   It may take a few minutes for the cluster creation process to create the private hosted zones in Route 53. If you do not see an **p3.openshiftapps.com** domain, wait a few minutes and run the command again.

4. Once you know the unique ID of the cluster domain, configure your DNS server to forward all

DNS requests for **rosa.<domain-prefix>.<unique-ID>.p3.openshiftapps.com** to your Amazon Route 53 Inbound Resolver endpoints. For BIND DNS servers, edit your **/etc/named.conf** file in your favorite text editor and add a new zone using the below example:

Example

```
zone "rosa.<domain-prefix>.<unique-ID>.p3.openshiftapps.com" { 1
  type forward;
  forward only;
  forwarders { 2
    10.0.45.253;
    10.0.23.131;
    10.0.148.159;
  };
};
```

1. Replace **<domain-prefix>** with your cluster domain prefix and **<unique-ID>** with your unique ID collected above.

2. Replace with the IP addresses of your inbound resolver endpoints collected above, ensuring that following each IP address there is a **;**.

CHAPTER 5. TUTORIAL: USING AWS WAF AND AMAZON CLOUDFRONT TO PROTECT RED HAT OPENSHIFT SERVICE ON AWS WORKLOADS

AWS WAF is a web application firewall that lets you monitor the HTTP and HTTPS requests that are forwarded to your protected web application resources.

You can use an Amazon CloudFront to add a Web Application Firewall (WAF) to your Red Hat OpenShift Service on AWS workloads. Using an external solution protects Red Hat OpenShift Service on AWS resources from experiencing denial of service due to handling the WAF.

NOTE

WAFv1, WAF classic, is no longer supported. Use WAFv2.

5.1. PREREQUISITES

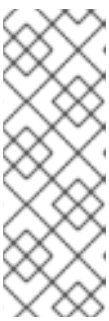
- A Red Hat OpenShift Service on AWS cluster.
- You have access to the OpenShift CLI (**oc**).
- You have access to the AWS CLI (**aws**).

5.1.1. Environment setup

- Prepare the environment variables:

```
$ export DOMAIN=apps.example.com 1
$ export AWS_PAGER=""
$ export CLUSTER_NAME=$(oc get infrastructure cluster -o=jsonpath="
{.status.infrastructureName}"  | sed 's/-[a-z0-9]\{5\}$//')
$ export REGION=$(oc get infrastructure cluster -o=jsonpath="
{.status.platformStatus.aws.region}")
$ export AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
$ export SCRATCH="/tmp/${CLUSTER}/cloudfront-waf"
$ mkdir -p ${SCRATCH}
$ echo "Cluster: ${CLUSTER}, Region: ${REGION}, AWS Account ID:
${AWS_ACCOUNT_ID}"
```

1 Replace with the custom domain you want to use for the **IngressController**.



NOTE

The "Cluster" output from the previous command might be the name of your cluster, the internal ID of your cluster, or the cluster's domain prefix. If you prefer to use another identifier, you can manually set this value by running the following command:

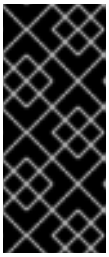
```
$ export CLUSTER=my-custom-value
```

5.2. SETTING UP THE SECONDARY INGRESS CONTROLLER

It is necessary to configure a secondary ingress controller to segment your external WAF-protected traffic from your standard (and default) cluster ingress controller.

Prerequisites

- A publicly trusted SAN or wildcard certificate for your custom domain, such as **CN=*.apps.example.com**



IMPORTANT

Amazon CloudFront uses HTTPS to communicate with your cluster's secondary ingress controller. As explained in the Amazon CloudFront documentation, you cannot use a self-signed certificate for HTTPS communication between CloudFront and your cluster. Amazon CloudFront verifies that the certificate was issued by a trusted certificate authority.

Procedure

1. Create a new TLS secret from a private key and a public certificate, where **fullchain.pem** is your full wildcard certificate chain (including any intermediaries) and **privkey.pem** is your wildcard certificate's private key.

   Example

   ```
   $ oc -n openshift-ingress create secret tls waf-tls --cert=fullchain.pem --key=privkey.pem
   ```

2. Create a new **IngressController** resource:

   Example waf-ingress-controller.yaml

   ```
   apiVersion: operator.openshift.io/v1
   kind: IngressController
   metadata:
     name: cloudfront-waf
     namespace: openshift-ingress-operator
   spec:
     domain: apps.example.com (1)
     defaultCertificate:
       name: waf-tls
     endpointPublishingStrategy:
       loadBalancer:
         dnsManagementPolicy: Unmanaged
         providerParameters:
           aws:
             type: NLB
           type: AWS
         scope: External
       type: LoadBalancerService
     routeSelector: (2)
       matchLabels:
        route: waf
   ```

1. Replace with the custom domain you want to use for the **IngressController**.
2. Filters the set of routes serviced by the Ingress Controller. In this tutorial, we will use the **waf** route selector, but if no value was to be provided, no filtering would occur.

3. Apply the **IngressController**:

Example

```
$ oc apply -f waf-ingress-controller.yaml
```

4. Verify that your IngressController has successfully created an external load balancer:

```
$ oc -n openshift-ingress get service/router-cloudfront-waf
```

Example output

```
NAME                 TYPE          CLUSTER-IP      EXTERNAL-IP
PORT(S)                   AGE
router-cloudfront-waf   LoadBalancer   172.30.16.141
a68a838a7f26440bf8647809b61c4bc8-4225395f488830bd.elb.us-east-1.amazonaws.com
80:30606/TCP,443:31065/TCP   2m19s
```

5.2.1. Configure the AWS WAF

The AWS WAF service is a web application firewall that lets you monitor, protect, and control the HTTP and HTTPS requests that are forwarded to your protected web application resources, like Red Hat OpenShift Service on AWS.

1. Create a AWS WAF rules file to apply to our web ACL:

```
$ cat << EOF > ${SCRATCH}/waf-rules.json
[
  {
    "Name": "AWS-AWSManagedRulesCommonRuleSet",
    "Priority": 0,
    "Statement": {
      "ManagedRuleGroupStatement": {
        "VendorName": "AWS",
        "Name": "AWSManagedRulesCommonRuleSet"
      }
    },
    "OverrideAction": {
      "None": {}
    },
    "VisibilityConfig": {
      "SampledRequestsEnabled": true,
      "CloudWatchMetricsEnabled": true,
      "MetricName": "AWS-AWSManagedRulesCommonRuleSet"
    }
  },
  {
    "Name": "AWS-AWSManagedRulesSQLiRuleSet",
```

```
    "Priority": 1,
    "Statement": {
      "ManagedRuleGroupStatement": {
        "VendorName": "AWS",
        "Name": "AWSManagedRulesSQLiRuleSet"
      }
    },
    "OverrideAction": {
      "None": {}
    },
    "VisibilityConfig": {
      "SampledRequestsEnabled": true,
      "CloudWatchMetricsEnabled": true,
      "MetricName": "AWS-AWSManagedRulesSQLiRuleSet"
    }
  }
]
EOF
```

This will enable the Core (Common) and SQL AWS Managed Rule Sets.

2. Create an AWS WAF Web ACL using the rules we specified above:

```
$ WAF_WACL=$(aws wafv2 create-web-acl \
  --name cloudfront-waf \
  --region ${REGION} \
  --default-action Allow={} \
  --scope CLOUDFRONT \
  --visibility-config
SampledRequestsEnabled=true,CloudWatchMetricsEnabled=true,MetricName=${CLUSTER}-
waf-metrics \
  --rules file://${SCRATCH}/waf-rules.json \
  --query 'Summary.Name' \
  --output text)
```

5.3. CONFIGURE AMAZON CLOUDFRONT

1. Retrieve the newly created custom ingress controller's NLB hostname:

```
$ NLB=$(oc -n openshift-ingress get service router-cloudfront-waf \
  -o jsonpath='{.status.loadBalancer.ingress[0].hostname}')
```

2. Import your certificate into Amazon Certificate Manager, where **cert.pem** is your wildcard certificate, **fullchain.pem** is your wildcard certificate's chain and **privkey.pem** is your wildcard certificate's private key.

NOTE

Regardless of what region your cluster is deployed, you must import this certificate to **us-east-1** as Amazon CloudFront is a global AWS service.

Example

```
$ aws acm import-certificate --certificate file://cert.pem \
  --certificate-chain file://fullchain.pem \
  --private-key file://privkey.pem \
  --region us-east-1
```

3. Log into the AWS console to create a CloudFront distribution.
4. Configure the CloudFront distribution by using the following information:

NOTE

If an option is not specified in the table below, leave them the default (which may be blank).

Option	Value
Origin domain	Output from the previous command [1]
Name	rosa-waf-ingress [2]
Viewer protocol policy	Redirect HTTP to HTTPS
Allowed HTTP methods	GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE
Cache policy	CachingDisabled
Origin request policy	AllViewer
Web Application Firewall (WAF)	Enable security protections
Use existing WAF configuration	true
Choose a web ACL	**cloudfront-waf**
Alternate domain name (CNAME)	*.apps.example.com [3]
Custom SSL certificate	Select the certificate you imported from the step above [4]

1. Run **echo ${NLB}** to get the origin domain.
2. If you have multiple clusters, ensure the origin name is unique.
3. This should match the wildcard domain you used to create the custom ingress controller.
4. This should match the alternate domain name entered above.

5. Retrieve the Amazon CloudFront Distribution endpoint:

```
$ aws cloudfront list-distributions --query "DistributionList.Items[?Origins.Items[?
DomainName=='${NLB}']].DomainName" --output text
```

6. Update the DNS of your custom wildcard domain with a CNAME to the Amazon CloudFront Distribution endpoint from the step above.

Example

```
*.apps.example.com CNAME d1b2c3d4e5f6g7.cloudfront.net
```

5.4. DEPLOY A SAMPLE APPLICATION

1. Create a new project for your sample application by running the following command:

```
$ oc new-project hello-world
```

2. Deploy a hello world application:

```
$ oc -n hello-world new-app --image=docker.io/openshift/hello-openshift
```

3. Create a route for the application specifying your custom domain name:

Example

```
$ oc -n hello-world create route edge --service=hello-openshift hello-openshift-tls \
--hostname hello-openshift.${DOMAIN}
```

4. Label the route to admit it to your custom ingress controller:

```
$ oc -n hello-world label route.route.openshift.io/hello-openshift-tls route=waf
```

5.5. TEST THE WAF

1. Test that the app is accessible behind Amazon CloudFront:

Example

```
$ curl "https://hello-openshift.${DOMAIN}"
```

Example output

```
Hello OpenShift!
```

2. Test that the WAF denies a bad request:

Example

```
$ curl -X POST "https://hello-openshift.${DOMAIN}" \
  -F "user='<script><alert>Hello></alert></script>'"
```

Example output

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<HTML><HEAD><META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-
8859-1">
<TITLE>ERROR: The request could not be satisfied</TITLE>
</HEAD><BODY>
<H1>403 ERROR</H1>
<H2>The request could not be satisfied.</H2>
<HR noshade size="1px">
Request blocked.
We can't connect to the server for this app or website at this time. There might be too much
traffic or a configuration error. Try again later, or contact the app or website owner.
<BR clear="all">
If you provide content to customers through CloudFront, you can find steps to troubleshoot
and help prevent this error by reviewing the CloudFront documentation.
<BR clear="all">
<HR noshade size="1px">
<PRE>
Generated by cloudfront (CloudFront)
Request ID: nFk9q2yB8jddI6FZOTjdliexzx-FwZtr8xUQUNT75HThPlrALDxbag==
</PRE>
<ADDRESS>
</ADDRESS>
</BODY></HTML>
```

The expected result is a **403 ERROR**, which means the AWS WAF is protecting your application.

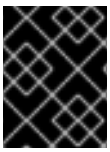
5.6. ADDITIONAL RESOURCES

- Adding Extra Security with AWS WAF, CloudFront and ROSA | Amazon Web Services on YouTube

CHAPTER 6. TUTORIAL: USING AWS WAF AND AWS ALBS TO PROTECT RED HAT OPENSHIFT SERVICE ON AWS WORKLOADS

AWS WAF is a web application firewall that lets you monitor the HTTP and HTTPS requests that are forwarded to your protected web application resources.

You can use an AWS Application Load Balancer (ALB) to add a Web Application Firewall (WAF) to your Red Hat OpenShift Service on AWS workloads. Using an external solution protects Red Hat OpenShift Service on AWS resources from experiencing denial of service due to handling the WAF.



IMPORTANT

It is recommended that you use the more flexible CloudFront method unless you absolutely must use an ALB based solution.

6.1. PREREQUISITES

- Multiple availability zone (AZ) Red Hat OpenShift Service on AWS cluster.

NOTE

AWS ALBs require at least two *public* subnets across AZs, per the AWS documentation. For this reason, only multiple AZ Red Hat OpenShift Service on AWS clusters can be used with ALBs.

- You have access to the OpenShift CLI (**oc**).
- You have access to the AWS CLI (**aws**).

6.1.1. Environment setup

- Prepare the environment variables:

```
$ export AWS_PAGER=""
$ export CLUSTER=$(oc get infrastructure cluster -o=jsonpath="
{.status.infrastructureName}")
$ export REGION=$(oc get infrastructure cluster -o=jsonpath="
{.status.platformStatus.aws.region}")
$ export OIDC_ENDPOINT=$(oc get authentication.config.openshift.io cluster -o
jsonpath='{.spec.serviceAccountIssuer}' | sed  's|^https://||')
$ export AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
$ export SCRATCH="/tmp/${CLUSTER}/alb-waf"
$ mkdir -p ${SCRATCH}
$ echo "Cluster: $(echo ${CLUSTER} | sed 's/-[a-z0-9]\{5\}$//'), Region: ${REGION}, OIDC
Endpoint: ${OIDC_ENDPOINT}, AWS Account ID: ${AWS_ACCOUNT_ID}"
```

6.1.2. AWS VPC and subnets

NOTE

This section only applies to clusters that were deployed into existing VPCs. If you did not deploy your cluster into an existing VPC, skip this section and proceed to the installation section below.

1. Set the below variables to the proper values for your Red Hat OpenShift Service on AWS deployment:

```
$ export VPC_ID=<vpc-id> 1
$ export PUBLIC_SUBNET_IDS=(<space-separated-list-of-ids>) 2
$ export PRIVATE_SUBNET_IDS=(<space-separated-list-of-ids>) 3
```

1 Replace with the VPC ID of the cluster, for example: **export VPC_ID=vpc-04c429b7dbc4680ba**.

2 Replace with a space-separated list of the private subnet IDs of the cluster, making sure to preserve the **()**. For example: **export PUBLIC_SUBNET_IDS=(subnet-056fd6861ad332ba2 subnet-08ce3b4ec753fe74c subnet-071aa28228664972f)**.

3 Replace with a space-separated list of the private subnet IDs of the cluster, making sure to preserve the **()**. For example: **export PRIVATE_SUBNET_IDS=(subnet-0b933d72a8d72c36a subnet-0817eb72070f1d3c2 subnet-0806e64159b66665a)**.

2. Add a tag to your cluster's VPC with the cluster identifier:

```
$ aws ec2 create-tags --resources ${VPC_ID} \
  --tags Key=kubernetes.io/cluster/${CLUSTER},Value=shared --region ${REGION}
```

3. Add a tag to your public subnets:

```
$ aws ec2 create-tags \
  --resources ${PUBLIC_SUBNET_IDS} \
  --tags Key=kubernetes.io/role/elb,Value='1' \
        Key=kubernetes.io/cluster/${CLUSTER},Value=shared \
  --region ${REGION}
```

4. Add a tag to your private subnets:

```
$ aws ec2 create-tags \
  --resources ${PRIVATE_SUBNET_IDS} \
  --tags Key=kubernetes.io/role/internal-elb,Value='1' \
        Key=kubernetes.io/cluster/${CLUSTER},Value=shared \
  --region ${REGION}
```

6.2. DEPLOY THE AWS LOAD BALANCER OPERATOR

The AWS Load Balancer Operator is used to used to install, manage and configure an instance of **aws-load-balancer-controller** in a Red Hat OpenShift Service on AWS cluster. To deploy ALBs in Red Hat OpenShift Service on AWS, we need to first deploy the AWS Load Balancer Operator.

1. Create a new project to deploy the AWS Load Balancer Operator into by running the following command:

```
$ oc new-project aws-load-balancer-operator
```

2. Create an AWS IAM policy for the AWS Load Balancer Controller if one does not already exist by running the following command:

NOTE

The policy is sourced from the upstream AWS Load Balancer Controller policy . This is required by the operator to function.

```
$ POLICY_ARN=$(aws iam list-policies --query \
    "Policies[?PolicyName=='aws-load-balancer-operator-policy'].{ARN:Arn}" \
    --output text)
```

```
$ if [[ -z "${POLICY_ARN}" ]]; then
   wget -O "${SCRATCH}/load-balancer-operator-policy.json" \
     https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/main/docs/install/iam_policy.json
    POLICY_ARN=$(aws --region "$REGION" --query Policy.Arn \
    --output text iam create-policy \
    --policy-name aws-load-balancer-operator-policy \
    --policy-document "file://${SCRATCH}/load-balancer-operator-policy.json")
fi
```

3. Create an AWS IAM trust policy for AWS Load Balancer Operator:

```
$ cat <<EOF > "${SCRATCH}/trust-policy.json"
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Condition": {
   "StringEquals" : {
    "${OIDC_ENDPOINT}:sub": ["system:serviceaccount:aws-load-balancer-operator:aws-
load-balancer-operator-controller-manager", "system:serviceaccount:aws-load-balancer-
operator:aws-load-balancer-controller-cluster"]
   }
 },
 "Principal": {
   "Federated": "arn:aws:iam::$AWS_ACCOUNT_ID:oidc-provider/${OIDC_ENDPOINT}"
 },
 "Action": "sts:AssumeRoleWithWebIdentity"
 }
 ]
}
EOF
```

4. Create an AWS IAM role for the AWS Load Balancer Operator:

```
$ ROLE_ARN=$(aws iam create-role --role-name "${CLUSTER}-alb-operator" \
```

```
    --assume-role-policy-document "file://${SCRATCH}/trust-policy.json" \
    --query Role.Arn --output text)
```

5. Attach the AWS Load Balancer Operator policy to the IAM role we created previously by running the following command:

```
$ aws iam attach-role-policy --role-name "${CLUSTER}-alb-operator" \
     --policy-arn ${POLICY_ARN}
```

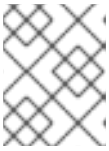
6. Create a secret for the AWS Load Balancer Operator to assume our newly created AWS IAM role:

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: aws-load-balancer-operator
  namespace: aws-load-balancer-operator
stringData:
  credentials: |
    [default]
    role_arn = ${ROLE_ARN}
    web_identity_token_file = /var/run/secrets/openshift/serviceaccount/token
EOF
```

7. Install the AWS Load Balancer Operator:

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: aws-load-balancer-operator
  namespace: aws-load-balancer-operator
spec:
  upgradeStrategy: Default
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: aws-load-balancer-operator
  namespace: aws-load-balancer-operator
spec:
  channel: stable-v1.0
  installPlanApproval: Automatic
  name: aws-load-balancer-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: aws-load-balancer-operator.v1.0.0
EOF
```

8. Deploy an instance of the AWS Load Balancer Controller using the operator:



NOTE

If you get an error here wait a minute and try again, it means the Operator has not completed installing yet.

```
$ cat << EOF | oc apply -f -
apiVersion: networking.olm.openshift.io/v1
kind: AWSLoadBalancerController
metadata:
  name: cluster
spec:
  credentials:
    name: aws-load-balancer-operator
  enabledAddons:
    - AWSWAFv2
EOF
```

9. Check the that the operator and controller pods are both running:

```
$ oc -n aws-load-balancer-operator get pods
```

You should see the following, if not wait a moment and retry:

```
NAME                                                READY   STATUS    RESTARTS   AGE
aws-load-balancer-controller-cluster-6ddf658785-pdp5d           1/1     Running   0          99s
aws-load-balancer-operator-controller-manager-577d9ffcb9-w6zqn   2/2     Running   0
2m4s
```

6.3. DEPLOY A SAMPLE APPLICATION

1. Create a new project for our sample application:

```
$ oc new-project hello-world
```

2. Deploy a hello world application:

```
$ oc new-app -n hello-world --image=docker.io/openshift/hello-openshift
```

3. Convert the pre-created service resource to a NodePort service type:

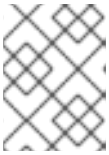
```
$ oc -n hello-world patch service hello-openshift -p '{"spec":{"type":"NodePort"}}'
```

4. Deploy an AWS ALB using the AWS Load Balancer Operator:

```
$ cat << EOF | oc apply -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hello-openshift-alb
  namespace: hello-world
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
```

```
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: hello-openshift
                port:
                  number: 8080
EOF
```

5. Curl the AWS ALB Ingress endpoint to verify the hello world application is accessible:



NOTE

AWS ALB provisioning takes a few minutes. If you receive an error that says **curl: (6) Could not resolve host**, please wait and try again.

```
$ INGRESS=$(oc -n hello-world get ingress hello-openshift-alb -o
jsonpath='{.status.loadBalancer.ingress[0].hostname}')
$ curl "http://${INGRESS}"
```

Example output

```
Hello OpenShift!
```

6.3.1. Configure the AWS WAF

The AWS WAF service is a web application firewall that lets you monitor, protect, and control the HTTP and HTTPS requests that are forwarded to your protected web application resources, like Red Hat OpenShift Service on AWS.

1. Create a AWS WAF rules file to apply to our web ACL:

```
$ cat << EOF > ${SCRATCH}/waf-rules.json
[
  {
    "Name": "AWS-AWSManagedRulesCommonRuleSet",
    "Priority": 0,
    "Statement": {
      "ManagedRuleGroupStatement": {
        "VendorName": "AWS",
        "Name": "AWSManagedRulesCommonRuleSet"
      }
    },
    "OverrideAction": {
      "None": {}
    },
    "VisibilityConfig": {
      "SampledRequestsEnabled": true,
```

```
        "CloudWatchMetricsEnabled": true,
        "MetricName": "AWS-AWSManagedRulesCommonRuleSet"
      }
    },
    {
      "Name": "AWS-AWSManagedRulesSQLiRuleSet",
      "Priority": 1,
      "Statement": {
        "ManagedRuleGroupStatement": {
          "VendorName": "AWS",
          "Name": "AWSManagedRulesSQLiRuleSet"
        }
      },
      "OverrideAction": {
        "None": {}
      },
      "VisibilityConfig": {
        "SampledRequestsEnabled": true,
        "CloudWatchMetricsEnabled": true,
        "MetricName": "AWS-AWSManagedRulesSQLiRuleSet"
      }
    }
]
EOF
```

This will enable the Core (Common) and SQL AWS Managed Rule Sets.

2. Create an AWS WAF Web ACL using the rules we specified above:

```
$ WAF_ARN=$(aws wafv2 create-web-acl \
  --name ${CLUSTER}-waf \
  --region ${REGION} \
  --default-action Allow={} \
  --scope REGIONAL \
  --visibility-config
SampledRequestsEnabled=true,CloudWatchMetricsEnabled=true,MetricName=${CLUSTER}-
waf-metrics \
  --rules file://${SCRATCH}/waf-rules.json \
  --query 'Summary.ARN' \
  --output text)
```

3. Annotate the Ingress resource with the AWS WAF Web ACL ARN:

```
$ oc annotate -n hello-world ingress.networking.k8s.io/hello-openshift-alb \
  alb.ingress.kubernetes.io/wafv2-acl-arn=${WAF_ARN}
```

4. Wait for 10 seconds for the rules to propagate and test that the app still works:

```
$ curl "http://${INGRESS}"
```

Example output

```
Hello OpenShift!
```

5. Test that the WAF denies a bad request:

```
$ curl -X POST "http://${INGRESS}" \
  -F "user='<script><alert>Hello></alert></script>'"
```

Example output

```
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
</body>
</html
```

NOTE

Activation of the AWS WAF integration can sometimes take several minutes. If you do not receive a **403 Forbidden** error, please wait a few seconds and try again.

The expected result is a **403 Forbidden** error, which means the AWS WAF is protecting your application.

6.4. ADDITIONAL RESOURCES

- Adding Extra Security with AWS WAF, CloudFront and ROSA | Amazon Web Services on YouTube

CHAPTER 7. TUTORIAL: DEPLOYING OPENSHIFT API FOR DATA PROTECTION ON A RED HAT OPENSHIFT SERVICE ON AWS CLUSTER

IMPORTANT

This content is authored by Red Hat experts, but has not yet been tested on every supported configuration.

Prerequisites

- A Red Hat OpenShift Service on AWS cluster

Environment

- Prepare the environment variables:

NOTE

Change the cluster name to match your Red Hat OpenShift Service on AWS cluster and ensure you are logged into the cluster as an Administrator. Ensure all fields are outputted correctly before moving on.

```
$ export CLUSTER_NAME=$(oc get infrastructure cluster -o=jsonpath="
{.status.infrastructureName}"  | sed 's/-[a-z0-9]\{5\}$//')
$ export ROSA_CLUSTER_ID=$(rosa describe cluster -c ${CLUSTER_NAME} --output json
| jq -r .id)
$ export REGION=$(rosa describe cluster -c ${CLUSTER_NAME} --output json | jq -r
.region.id)
$ export OIDC_ENDPOINT=$(oc get authentication.config.openshift.io cluster -o
jsonpath='{.spec.serviceAccountIssuer}' | sed  's|^https://||')
$ export AWS_ACCOUNT_ID=`aws sts get-caller-identity --query Account --output text`
$ export CLUSTER_VERSION=`rosa describe cluster -c ${CLUSTER_NAME} -o json | jq -r
.version.raw_id | cut -f -2 -d '.'`
$ export ROLE_NAME="${CLUSTER_NAME}-openshift-oadp-aws-cloud-credentials"
$ export AWS_PAGER=""
$ export SCRATCH="/tmp/${CLUSTER_NAME}/oadp"
$ mkdir -p ${SCRATCH}
$ echo "Cluster ID: ${ROSA_CLUSTER_ID}, Region: ${REGION}, OIDC Endpoint:
${OIDC_ENDPOINT}, AWS Account ID: ${AWS_ACCOUNT_ID}"
```

7.1. PREPARE AWS ACCOUNT

1. Create an IAM Policy to allow for S3 Access:

```
$ POLICY_ARN=$(aws iam list-policies --query "Policies[?PolicyName=='RosaOadpVer1'].
{ARN:Arn}" --output text)
if [[ -z "${POLICY_ARN}" ]]; then
$ cat << EOF > ${SCRATCH}/policy.json
{
"Version": "2012-10-17",
"Statement": [
```

```
 {
   "Effect": "Allow",
   "Action": [
     "s3:CreateBucket",
     "s3:DeleteBucket",
     "s3:PutBucketTagging",
     "s3:GetBucketTagging",
     "s3:PutEncryptionConfiguration",
     "s3:GetEncryptionConfiguration",
     "s3:PutLifecycleConfiguration",
     "s3:GetLifecycleConfiguration",
     "s3:GetBucketLocation",
     "s3:ListBucket",
     "s3:GetObject",
     "s3:PutObject",
     "s3:DeleteObject",
     "s3:ListBucketMultipartUploads",
     "s3:AbortMultipartUpload",
     "s3:ListMultipartUploadParts",
     "ec2:DescribeSnapshots",
     "ec2:DescribeVolumes",
     "ec2:DescribeVolumeAttribute",
     "ec2:DescribeVolumesModifications",
     "ec2:DescribeVolumeStatus",
     "ec2:CreateTags",
     "ec2:CreateVolume",
     "ec2:CreateSnapshot",
     "ec2:DeleteSnapshot"
   ],
   "Resource": "*"
 }
]}
EOF
$ POLICY_ARN=$(aws iam create-policy --policy-name "RosaOadpVer1" \
--policy-document file:///${SCRATCH}/policy.json --query Policy.Arn \
--tags Key=rosa_openshift_version,Value=${CLUSTER_VERSION}
Key=rosa_role_prefix,Value=ManagedOpenShift
Key=operator_namespace,Value=openshift-oadp Key=operator_name,Value=openshift-oadp
\
--output text)
fi
$ echo ${POLICY_ARN}
```

2. Create an IAM Role trust policy for the cluster:

```
$ cat <<EOF > ${SCRATCH}/trust-policy.json
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Federated": "arn:aws:iam::${AWS_ACCOUNT_ID}:oidc-provider/${OIDC_ENDPOINT}"
    },
    "Action": "sts:AssumeRoleWithWebIdentity",
    "Condition": {
      "StringEquals": {
```

```
          "${OIDC_ENDPOINT}:sub": [
            "system:serviceaccount:openshift-adp:openshift-adp-controller-manager",
            "system:serviceaccount:openshift-adp:velero"]
        }
      }
    }]
}
EOF
$ ROLE_ARN=$(aws iam create-role --role-name \
 "${ROLE_NAME}" \
  --assume-role-policy-document file://${SCRATCH}/trust-policy.json \
  --tags Key=rosa_cluster_id,Value=${ROSA_CLUSTER_ID}
Key=rosa_openshift_version,Value=${CLUSTER_VERSION}
Key=rosa_role_prefix,Value=ManagedOpenShift
Key=operator_namespace,Value=openshift-adp Key=operator_name,Value=openshift-oadp \
  --query Role.Arn --output text)

$ echo ${ROLE_ARN}
```

3. Attach the IAM Policy to the IAM Role:

```
$ aws iam attach-role-policy --role-name "${ROLE_NAME}" \
 --policy-arn ${POLICY_ARN}
```

7.2. DEPLOY OADP ON THE CLUSTER

1. Create a namespace for OADP:

```
$ oc create namespace openshift-adp
```

2. Create a credentials secret:

```
$ cat <<EOF > ${SCRATCH}/credentials
[default]
role_arn = ${ROLE_ARN}
web_identity_token_file = /var/run/secrets/openshift/serviceaccount/token
region=<aws_region> (1)
EOF
$ oc -n openshift-adp create secret generic cloud-credentials \
 --from-file=${SCRATCH}/credentials
```

(1) Replace **<aws_region>** with the AWS region to use for the Security Token Service (STS) endpoint.

3. Deploy the OADP Operator:

NOTE

There is currently an issue with version 1.1 of the Operator with backups that have a **PartiallyFailed** status. This does not seem to affect the backup and restore process, but it should be noted as there are issues with it.

```
$ cat << EOF | oc create -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
 generateName: openshift-adp-
 namespace: openshift-adp
 name: oadp
spec:
 targetNamespaces:
 - openshift-adp
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
 name: redhat-oadp-operator
 namespace: openshift-adp
spec:
 channel: stable-1.2
 installPlanApproval: Automatic
 name: redhat-oadp-operator
 source: redhat-operators
 sourceNamespace: openshift-marketplace
EOF
```

4. Wait for the Operator to be ready:

```
$ watch oc -n openshift-adp get pods
```

Example output

```
NAME                                        READY   STATUS    RESTARTS   AGE
openshift-adp-controller-manager-546684844f-qqjhn   1/1     Running   0          22s
```

5. Create Cloud Storage:

```
$ cat << EOF | oc create -f -
apiVersion: oadp.openshift.io/v1alpha1
kind: CloudStorage
metadata:
 name: ${CLUSTER_NAME}-oadp
 namespace: openshift-adp
spec:
 creationSecret:
   key: credentials
   name: cloud-credentials
 enableSharedConfig: true
 name: ${CLUSTER_NAME}-oadp
 provider: aws
 region: $REGION
EOF
```

6. Check your application's storage default storage class:

```
$ oc get pvc -n <namespace> 1
```

–

1 Enter your application's namespace.

Example output

```
NAME     STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
applog   Bound    pvc-351791ae-b6ab-4e8b-88a4-30f73caf5ef8   1Gi        RWO            gp3-
csi        4d19h
mysql    Bound    pvc-16b8e009-a20a-4379-accc-bc81fedd0621   1Gi        RWO            gp3-
csi        4d19h
```

```
$ oc get storageclass
```

Example output

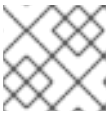
```
NAME              PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION   AGE
gp2               kubernetes.io/aws-ebs   Delete          WaitForFirstConsumer   true
4d21h
gp2-csi           ebs.csi.aws.com         Delete          WaitForFirstConsumer   true
4d21h
gp3               ebs.csi.aws.com         Delete          WaitForFirstConsumer   true
4d21h
gp3-csi (default)   ebs.csi.aws.com         Delete          WaitForFirstConsumer   true
4d21h
```

Using either gp3-csi, gp2-csi, gp3 or gp2 will work. If the application(s) that are being backed up are all using PV's with CSI, include the CSI plugin in the OADP DPA configuration.

7. CSI only: Deploy a Data Protection Application:

```
$ cat << EOF | oc create -f -
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
 name: ${CLUSTER_NAME}-dpa
 namespace: openshift-adp
spec:
 backupImages: true
 features:
   dataMover:
     enable: false
 backupLocations:
 - bucket:
     cloudStorageRef:
       name: ${CLUSTER_NAME}-oadp
     credential:
       key: credentials
       name: cloud-credentials
     prefix: velero
     default: true
     config:
       region: ${REGION}
```

```
 configuration:
   velero:
     defaultPlugins:
     - openshift
     - aws
     - csi
   restic:
     enable: false
EOF
```

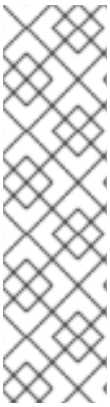


NOTE

If you run this command for CSI volumes, you can skip the next step.

8. Non-CSI volumes: Deploy a Data Protection Application:

```
$ cat << EOF | oc create -f -
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
 name: ${CLUSTER_NAME}-dpa
 namespace: openshift-adp
spec:
 backupImages: true
 features:
   dataMover:
     enable: false
 backupLocations:
 - bucket:
     cloudStorageRef:
       name: ${CLUSTER_NAME}-oadp
     credential:
       key: credentials
       name: cloud-credentials
     prefix: velero
     default: true
     config:
       region: ${REGION}
 configuration:
   velero:
     defaultPlugins:
     - openshift
     - aws
   restic:
     enable: false
 snapshotLocations:
   - velero:
       config:
         credentialsFile: /tmp/credentials/openshift-adp/cloud-credentials-credentials
         enableSharedConfig: 'true'
         profile: default
         region: ${REGION}
       provider: aws
EOF
```



NOTE

- In OADP 1.1.x Red Hat OpenShift Service on AWS STS environments, the container image backup and restore (**spec.backupImages**) value must be set to **false** as it is not supported.
- The Restic feature (**restic.enable=false**) is disabled and not supported in Red Hat OpenShift Service on AWS STS environments.
- The DataMover feature (**dataMover.enable=false**) is disabled and not supported in Red Hat OpenShift Service on AWS STS environments.

7.3. PERFORM A BACKUP

NOTE

The following sample hello-world application has no attached persistent volumes. Either DPA configuration will work.

1. Create a workload to back up:

```
$ oc create namespace hello-world
$ oc new-app -n hello-world --image=docker.io/openshift/hello-openshift
```

2. Expose the route:

```
$ oc expose service/hello-openshift -n hello-world
```

3. Check that the application is working:

```
$ curl `oc get route/hello-openshift -n hello-world -o jsonpath='{.spec.host}'`
```

Example output

```
Hello OpenShift!
```

4. Back up the workload:

```
$ cat << EOF | oc create -f -
apiVersion: velero.io/v1
kind: Backup
metadata:
 name: hello-world
 namespace: openshift-adp
spec:
 includedNamespaces:
 - hello-world
 storageLocation: ${CLUSTER_NAME}-dpa-1
 ttl: 720h0m0s
EOF
```

5. Wait until the backup is done:

```
$ watch "oc -n openshift-adp get backup hello-world -o json | jq .status"
```

Example output

```
{
 "completionTimestamp": "2022-09-07T22:20:44Z",
 "expiration": "2022-10-07T22:20:22Z",
 "formatVersion": "1.1.0",
 "phase": "Completed",
 "progress": {
   "itemsBackedUp": 58,
   "totalItems": 58
 },
 "startTimestamp": "2022-09-07T22:20:22Z",
 "version": 1
}
```

6. Delete the demo workload:

```
$ oc delete ns hello-world
```

7. Restore from the backup:

```
$ cat << EOF | oc create -f -
apiVersion: velero.io/v1
kind: Restore
metadata:
 name: hello-world
 namespace: openshift-adp
spec:
 backupName: hello-world
EOF
```

8. Wait for the Restore to finish:

```
$ watch "oc -n openshift-adp get restore hello-world -o json | jq .status"
```

Example output

```
{
 "completionTimestamp": "2022-09-07T22:25:47Z",
 "phase": "Completed",
 "progress": {
   "itemsRestored": 38,
   "totalItems": 38
 },
 "startTimestamp": "2022-09-07T22:25:28Z",
 "warnings": 9
}
```

9. Check that the workload is restored:

```
$ oc -n hello-world get pods
```

Example output

```
NAME                                READY   STATUS    RESTARTS   AGE
hello-openshift-9f885f7c6-kdjpj   1/1     Running   0          90s
```

```
$ curl `oc get route/hello-openshift -n hello-world -o jsonpath='{.spec.host}'`
```

Example output

```
Hello OpenShift!
```

10. For troubleshooting tips please refer to the OADP team's troubleshooting documentation
11. Additional sample applications can be found in the OADP team's sample applications directory

7.4. CLEANUP

1. Delete the workload:

```
$ oc delete ns hello-world
```

2. Remove the backup and restore resources from the cluster if they are no longer required:

```
$ oc delete backups.velero.io hello-world
$ oc delete restores.velero.io hello-world
```

3. To delete the backup/restore and remote objects in s3:

```
$ velero backup delete hello-world
$ velero restore delete hello-world
```

4. Delete the Data Protection Application:

```
$ oc -n openshift-adp delete dpa ${CLUSTER_NAME}-dpa
```

5. Delete the Cloud Storage:

```
$ oc -n openshift-adp delete cloudstorage ${CLUSTER_NAME}-oadp
```

WARNING

If this command hangs, you might need to delete the finalizer:

```
$ oc -n openshift-adp patch cloudstorage ${CLUSTER_NAME}-oadp -p
'{"metadata":{"finalizers":null}}' --type=merge
```

6. Remove the Operator if it is no longer required:

```
$ oc -n openshift-adp delete subscription oadp-operator
```

7. Remove the namespace for the Operator:

```
$ oc delete ns redhat-openshift-adp
```

8. Remove the Custom Resource Definitions from the cluster if you no longer wish to have them:

```
$ for CRD in `oc get crds | grep velero | awk '{print $1}'`; do oc delete crd $CRD; done
$ for CRD in `oc get crds | grep -i oadp | awk '{print $1}'`; do oc delete crd $CRD; done
```

9. Delete the AWS S3 Bucket:

```
$ aws s3 rm s3://${CLUSTER_NAME}-oadp --recursive
$ aws s3api delete-bucket --bucket ${CLUSTER_NAME}-oadp
```

10. Detach the Policy from the role:

```
$ aws iam detach-role-policy --role-name "${ROLE_NAME}" \
 --policy-arn "${POLICY_ARN}"
```

11. Delete the role:

```
$ aws iam delete-role --role-name "${ROLE_NAME}"
```

CHAPTER 8. TUTORIAL: AWS LOAD BALANCER OPERATOR ON RED HAT OPENSHIFT SERVICE ON AWS

IMPORTANT

This content is authored by Red Hat experts, but has not yet been tested on every supported configuration.

TIP

Load Balancers created by the AWS Load Balancer Operator cannot be used for OpenShift Routes, and should only be used for individual services or ingress resources that do not need the full layer 7 capabilities of an OpenShift Route.

The AWS Load Balancer Controller manages AWS Elastic Load Balancers for a Red Hat OpenShift Service on AWS cluster. The controller provisions AWS Application Load Balancers (ALB) when you create Kubernetes Ingress resources and AWS Network Load Balancers (NLB) when implementing Kubernetes Service resources with a type of LoadBalancer.

Compared with the default AWS in-tree load balancer provider, this controller is developed with advanced annotations for both ALBs and NLBs. Some advanced use cases are:

- Using native Kubernetes Ingress objects with ALBs
- Integrate ALBs with the AWS Web Application Firewall (WAF) service

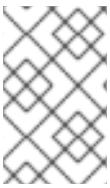
NOTE

WAFv1, WAF classic, is no longer supported. Use WAFv2.

- Specify custom NLB source IP ranges
- Specify custom NLB internal IP addresses

The AWS Load Balancer Operator is used to used to install, manage and configure an instance of **aws-load-balancer-controller** in a Red Hat OpenShift Service on AWS cluster.

8.1. PREREQUISITES



NOTE

AWS ALBs require a multi-AZ cluster, as well as three public subnets split across three AZs in the same VPC as the cluster. This makes ALBs unsuitable for many PrivateLink clusters. AWS NLBs do not have this restriction.

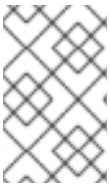
- A multi-AZ Red Hat OpenShift Service on AWS cluster
- BYO VPC cluster
- AWS CLI
- OC CLI

8.1.1. Environment

- Prepare the environment variables:

```
$ export AWS_PAGER=""
$ export ROSA_CLUSTER_NAME=$(oc get infrastructure cluster -o=jsonpath="
{.status.infrastructureName}"  | sed 's/-[a-z0-9]\{5\}$//')
$ export REGION=$(oc get infrastructure cluster -o=jsonpath="
{.status.platformStatus.aws.region}")
$ export OIDC_ENDPOINT=$(oc get authentication.config.openshift.io cluster -o
jsonpath='{.spec.serviceAccountIssuer}' | sed  's|^https://||')
$ export AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
$ export SCRATCH="/tmp/${ROSA_CLUSTER_NAME}/alb-operator"
$ mkdir -p ${SCRATCH}
$ echo "Cluster: ${ROSA_CLUSTER_NAME}, Region: ${REGION}, OIDC Endpoint:
${OIDC_ENDPOINT}, AWS Account ID: ${AWS_ACCOUNT_ID}"
```

8.1.2. AWS VPC and subnets



NOTE

This section only applies to clusters that were deployed into existing VPCs. If you did not deploy your cluster into an existing VPC, skip this section and proceed to the installation section below.

1. Set the below variables to the proper values for your cluster deployment:

```
$ export VPC_ID=<vpc-id>
$ export PUBLIC_SUBNET_IDS=<public-subnets>
$ export PRIVATE_SUBNET_IDS=<private-subnets>
$ export CLUSTER_NAME=$(oc get infrastructure cluster -o=jsonpath="
{.status.infrastructureName}")
```

2. Add a tag to your cluster's VPC with the cluster name:

```
$ aws ec2 create-tags --resources ${VPC_ID} --tags
Key=kubernetes.io/cluster/${CLUSTER_NAME},Value=owned --region ${REGION}
```

3. Add a tag to your public subnets:

```
$ aws ec2 create-tags \
    --resources ${PUBLIC_SUBNET_IDS} \
    --tags Key=kubernetes.io/role/elb,Value='' \
    --region ${REGION}
```

4. Add a tag to your private subnets:

```
$ aws ec2 create-tags \
    --resources "${PRIVATE_SUBNET_IDS}" \
    --tags Key=kubernetes.io/role/internal-elb,Value='' \
    --region ${REGION}
```

8.2. INSTALLATION

1. Create an AWS IAM policy for the AWS Load Balancer Controller:

NOTE

The policy is sourced from the upstream AWS Load Balancer Controller policy plus permission to create tags on subnets. This is required by the Operator to function.

```
$ oc new-project aws-load-balancer-operator
$ POLICY_ARN=$(aws iam list-policies --query \
    "Policies[?PolicyName=='aws-load-balancer-operator-policy'].{ARN:Arn}" \
    --output text)
$ if [[ -z "${POLICY_ARN}" ]]; then
   wget -O "${SCRATCH}/load-balancer-operator-policy.json" \
     https://raw.githubusercontent.com/rh-mobb/documentation/main/content/rosa/aws-load-balancer-operator/load-balancer-operator-policy.json
    POLICY_ARN=$(aws --region "$REGION" --query Policy.Arn \
    --output text iam create-policy \
    --policy-name aws-load-balancer-operator-policy \
    --policy-document "file://${SCRATCH}/load-balancer-operator-policy.json")
fi
$ echo $POLICY_ARN
```

2. Create an AWS IAM trust policy for AWS Load Balancer Operator:

```
$ cat <<EOF > "${SCRATCH}/trust-policy.json"
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Condition": {
   "StringEquals" : {
    "${OIDC_ENDPOINT}:sub": ["system:serviceaccount:aws-load-balancer-operator:aws-load-balancer-operator-controller-manager", "system:serviceaccount:aws-load-balancer-operator:aws-load-balancer-controller-cluster"]
   }
 },
 "Principal": {
   "Federated": "arn:aws:iam::$AWS_ACCOUNT_ID:oidc-provider/${OIDC_ENDPOINT}"
 },
 "Action": "sts:AssumeRoleWithWebIdentity"
 }
 ]
}
EOF
```

3. Create an AWS IAM role for the AWS Load Balancer Operator:

```
$ ROLE_ARN=$(aws iam create-role --role-name "${ROSA_CLUSTER_NAME}-alb-operator" \
   --assume-role-policy-document "file://${SCRATCH}/trust-policy.json" \
```

```
  --query Role.Arn --output text)
$ echo $ROLE_ARN

$ aws iam attach-role-policy --role-name "${ROSA_CLUSTER_NAME}-alb-operator" \
    --policy-arn $POLICY_ARN
```

4. Create a secret for the AWS Load Balancer Operator to assume our newly created AWS IAM role:

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: aws-load-balancer-operator
  namespace: aws-load-balancer-operator
stringData:
  credentials: |
    [default]
    role_arn = $ROLE_ARN
    web_identity_token_file = /var/run/secrets/openshift/serviceaccount/token
EOF
```

5. Install the AWS Load Balancer Operator:

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: aws-load-balancer-operator
  namespace: aws-load-balancer-operator
spec:
  upgradeStrategy: Default
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: aws-load-balancer-operator
  namespace: aws-load-balancer-operator
spec:
  channel: stable-v1.0
  installPlanApproval: Automatic
  name: aws-load-balancer-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: aws-load-balancer-operator.v1.0.0
EOF
```

6. Deploy an instance of the AWS Load Balancer Controller using the Operator:

NOTE

If you get an error here wait a minute and try again, it means the Operator has not completed installing yet.

```
$ cat << EOF | oc apply -f -
apiVersion: networking.olm.openshift.io/v1
kind: AWSLoadBalancerController
metadata:
  name: cluster
spec:
  credentials:
    name: aws-load-balancer-operator
EOF
```

7. Check the that the Operator and controller pods are both running:

```
$ oc -n aws-load-balancer-operator get pods
```

You should see the following, if not wait a moment and retry:

```
NAME                                                READY   STATUS    RESTARTS   AGE
aws-load-balancer-controller-cluster-6ddf658785-pdp5d           1/1     Running   0          99s
aws-load-balancer-operator-controller-manager-577d9ffcb9-w6zqn   2/2     Running   0
2m4s
```

8.3. VALIDATING THE DEPLOYMENT

1. Create a new project:

```
$ oc new-project hello-world
```

2. Deploy a hello world application:

```
$ oc new-app -n hello-world --image=docker.io/openshift/hello-openshift
```

3. Configure a NodePort service for the AWS ALB to connect to:

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Service
metadata:
  name: hello-openshift-nodeport
  namespace: hello-world
spec:
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
  type: NodePort
  selector:
    deployment: hello-openshift
EOF
```

4. Deploy an AWS ALB using the AWS Load Balancer Operator:

```
$ cat << EOF | oc apply -f -
apiVersion: networking.k8s.io/v1
```

```
kind: Ingress
metadata:
  name: hello-openshift-alb
  namespace: hello-world
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: hello-openshift-nodeport
                port:
                  number: 80
EOF
```

5. Curl the AWS ALB Ingress endpoint to verify the hello world application is accessible:

NOTE

AWS ALB provisioning takes a few minutes. If you receive an error that says **curl: (6) Could not resolve host**, please wait and try again.

```
$ INGRESS=$(oc -n hello-world get ingress hello-openshift-alb \
    -o jsonpath='{.status.loadBalancer.ingress[0].hostname}')
$ curl "http://${INGRESS}"
```

Example output

```
Hello OpenShift!
```

6. Deploy an AWS NLB for your hello world application:

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Service
metadata:
  name: hello-openshift-nlb
  namespace: hello-world
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: instance
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
```

```
  selector:
    deployment: hello-openshift
EOF
```

7. Test the AWS NLB endpoint:

NOTE

NLB provisioning takes a few minutes. If you receive an error that says **curl: (6) Could not resolve host**, please wait and try again.

```
$ NLB=$(oc -n hello-world get service hello-openshift-nlb \
  -o jsonpath='{.status.loadBalancer.ingress[0].hostname}')
$ curl "http://${NLB}"
```

Example output

```
Hello OpenShift!
```

8.4. CLEANING UP

1. Delete the hello world application namespace (and all the resources in the namespace):

```
$ oc delete project hello-world
```

2. Delete the AWS Load Balancer Operator and the AWS IAM roles:

```
$ oc delete subscription aws-load-balancer-operator -n aws-load-balancer-operator
$ aws iam detach-role-policy \
  --role-name "${ROSA_CLUSTER_NAME}-alb-operator" \
  --policy-arn $POLICY_ARN
$ aws iam delete-role \
  --role-name "${ROSA_CLUSTER_NAME}-alb-operator"
```

3. Delete the AWS IAM policy:

```
$ aws iam delete-policy --policy-arn $POLICY_ARN
```

CHAPTER 9. TUTORIAL: CONFIGURING MICROSOFT ENTRA ID (FORMERLY AZURE ACTIVE DIRECTORY) AS AN IDENTITY PROVIDER

You can configure Microsoft Entra ID (formerly Azure Active Directory) as the cluster identity provider in Red Hat OpenShift Service on AWS.

This tutorial guides you to complete the following tasks:

1. Register a new application in Entra ID for authentication.
2. Configure the application registration in Entra ID to include optional and group claims in tokens.
3. Configure the Red Hat OpenShift Service on AWS cluster to use Entra ID as the identity provider.
4. Grant additional permissions to individual groups.

9.1. PREREQUISITES

- You created a set of security groups and assigned users by following the Microsoft documentation.

9.2. REGISTERING A NEW APPLICATION IN ENTRA ID FOR AUTHENTICATION

To register your application in Entra ID, first create the OAuth callback URL, then register your application.

Procedure

1. Create the cluster's OAuth callback URL by changing the specified variables and running the following command:

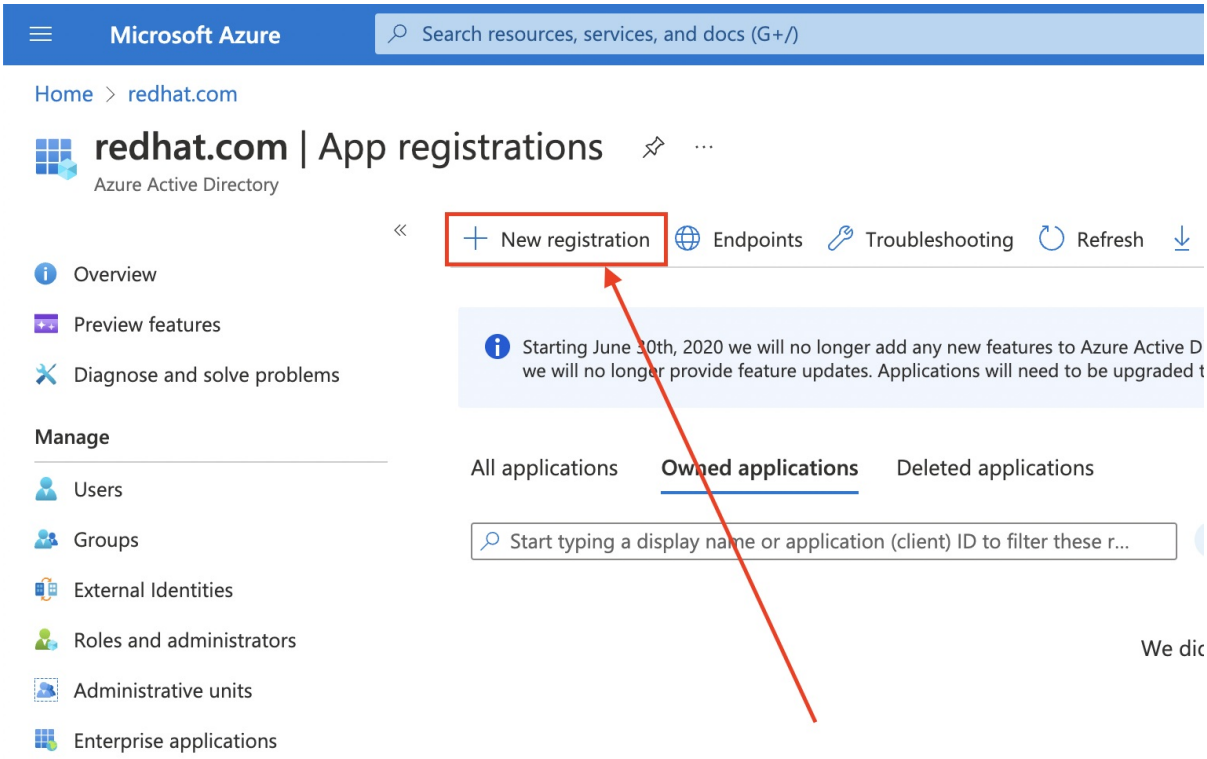
NOTE

Remember to save this callback URL; it will be required later in the process.

```
$ domain=$(rosa describe cluster -c <cluster_name> | grep "DNS" | grep -oE
'\S+.openshiftapps.com')
echo "OAuth callback URL: https://oauth.${domain}/oauth2callback/AAD"
```

The "AAD" directory at the end of the OAuth callback URL must match the OAuth identity provider name that you will set up later in this process.

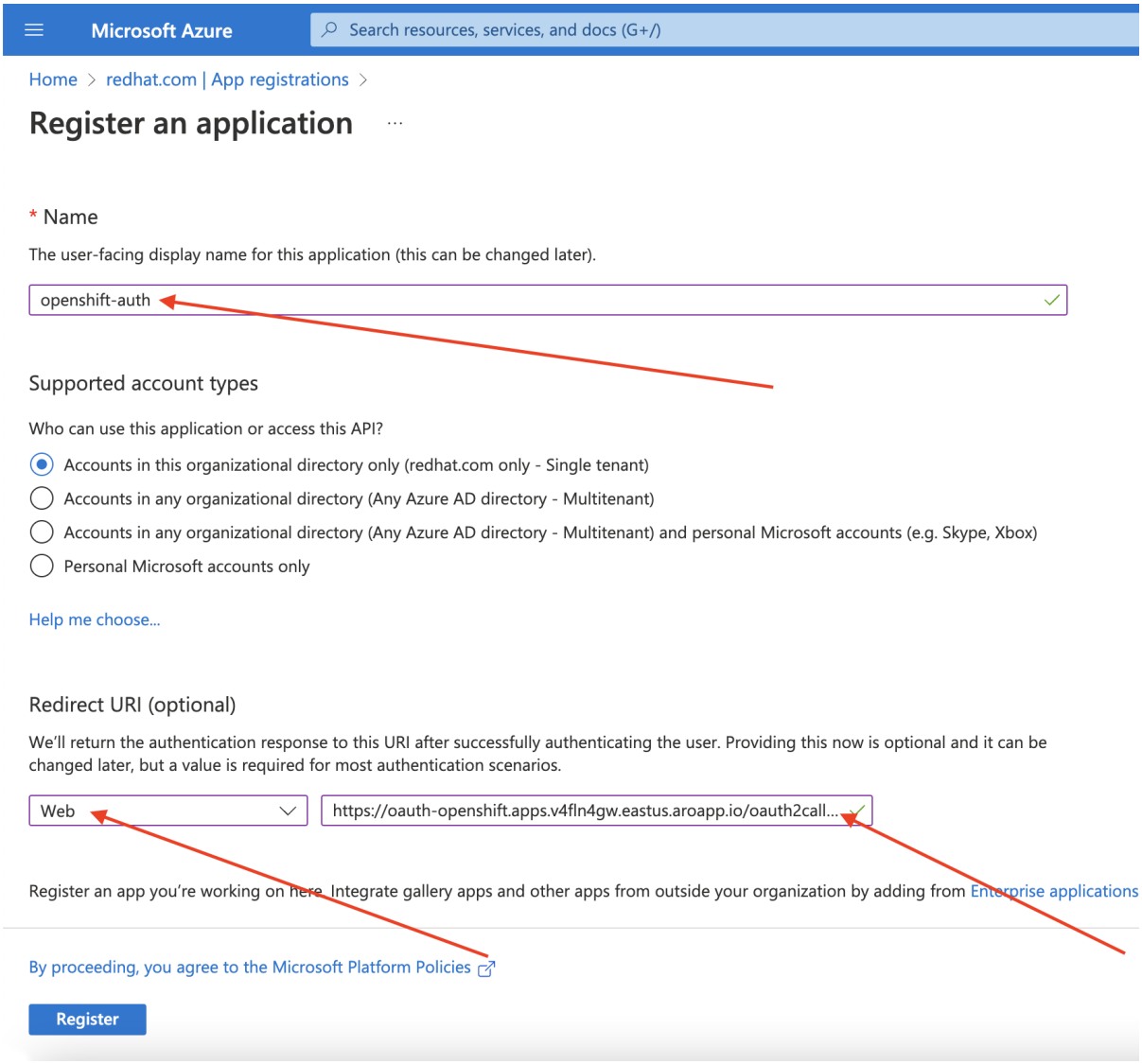
2. Create the Entra ID application by logging in to the Azure portal, and select the App registrations blade. Then, select **New registration** to create a new application.



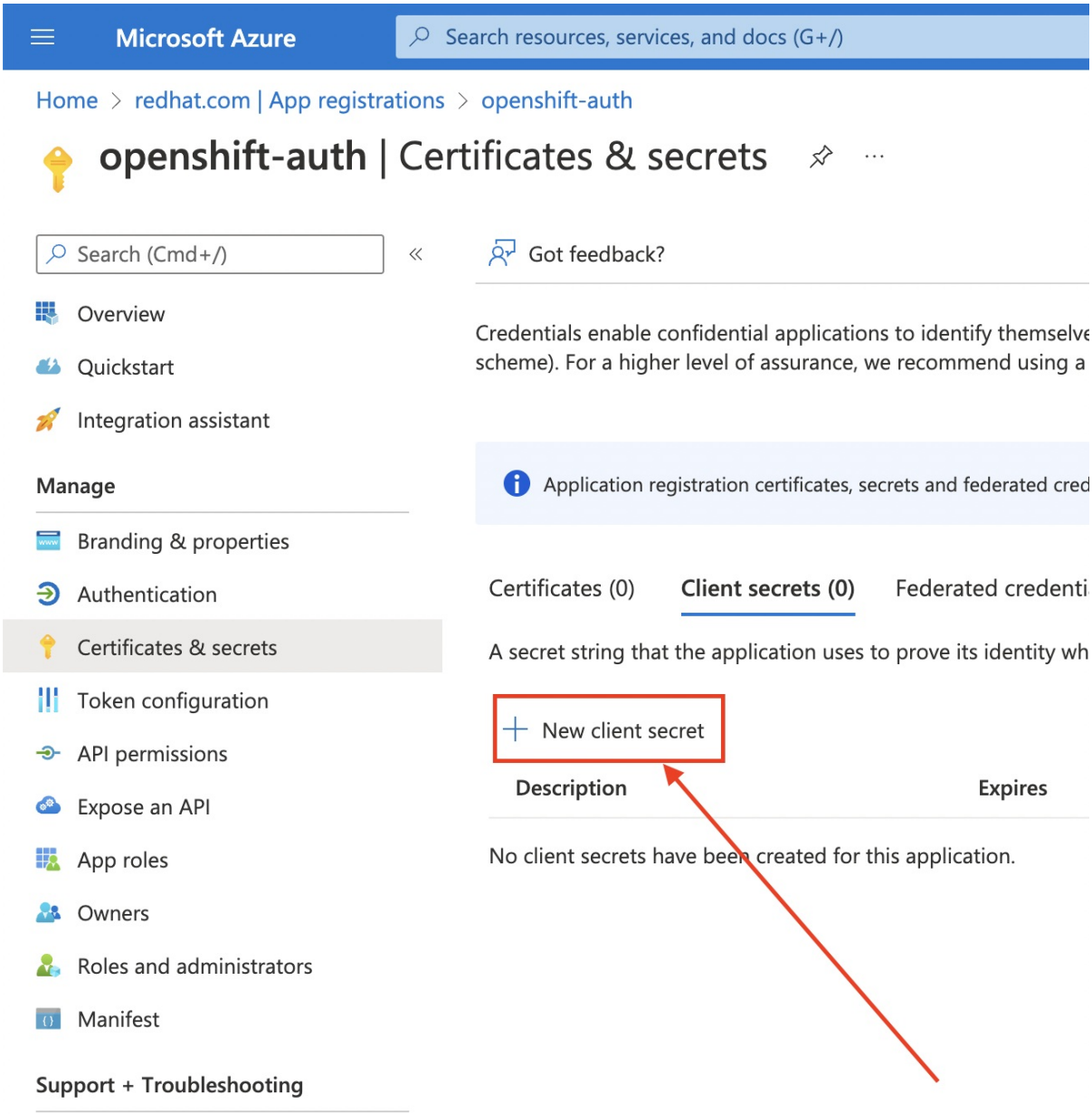
3. Name the application, for example **openshift-auth**.

4. Select **Web** from the *Redirect URI* dropdown and enter the value of the OAuth callback URL you retrieved in the previous step.

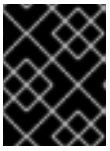
5. After providing the required information, click **Register** to create the application.



6. Select the **Certificates & secrets** sub-blade and select **New client secret**.

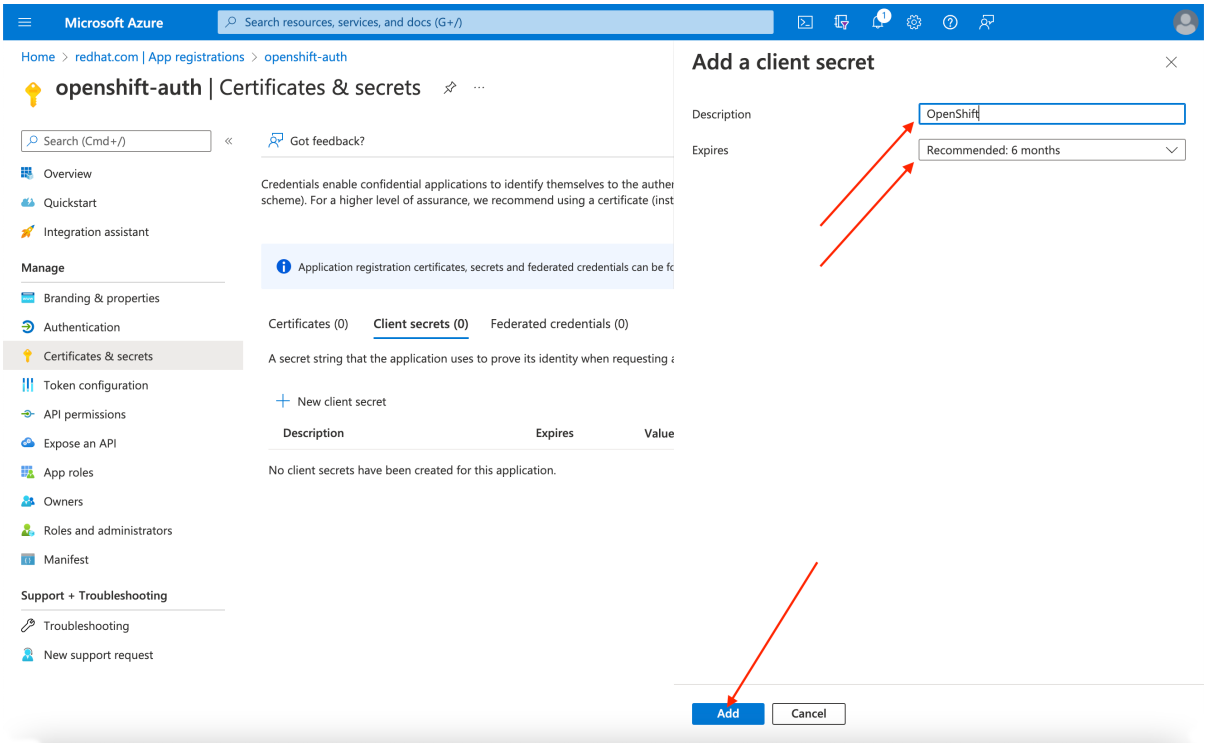


7. Complete the requested details and store the generated client secret value. This secret is required later in this process.

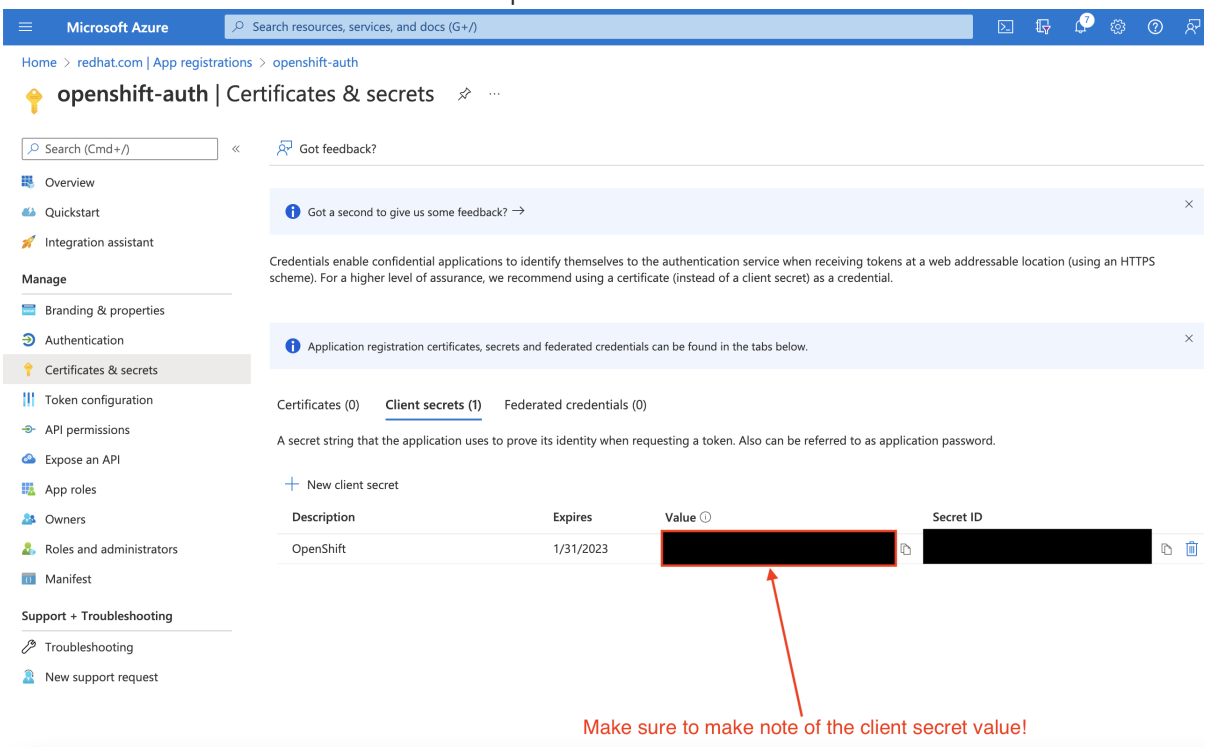


IMPORTANT

After initial setup, you cannot see the client secret. If you did not record the client secret, you must generate a new one.



8. Select the **Overview** sub-blade and note the **Application (client) ID** and **Directory (tenant) ID**. You will need these values in a future step.



9.3. CONFIGURING THE APPLICATION REGISTRATION IN ENTRA ID TO INCLUDE OPTIONAL AND GROUP CLAIMS

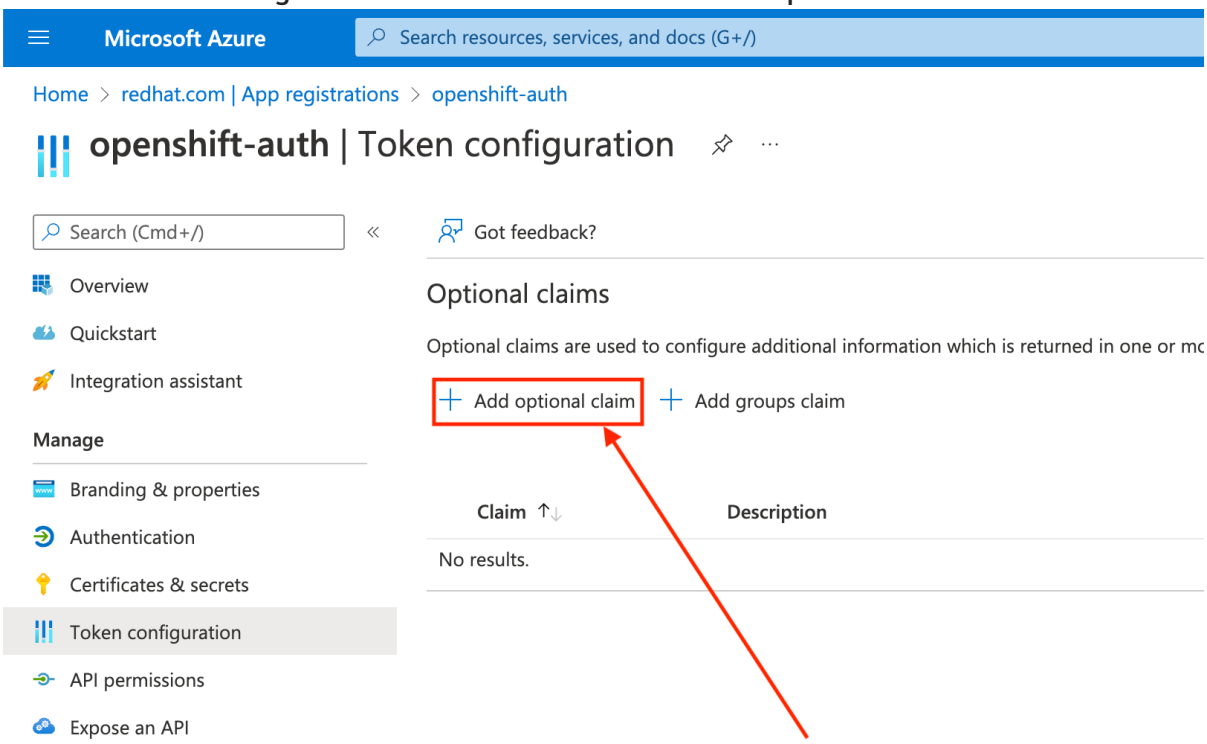
So that Red Hat OpenShift Service on AWS has enough information to create the user's account, you must configure Entra ID to give two optional claims: **email** and **preferred_username**. For more information about optional claims in Entra ID, see the Microsoft documentation.

In addition to individual user authentication, Red Hat OpenShift Service on AWS provides group claim functionality. This functionality allows an OpenID Connect (OIDC) identity provider, such as Entra ID, to offer a user's group membership for use within Red Hat OpenShift Service on AWS.

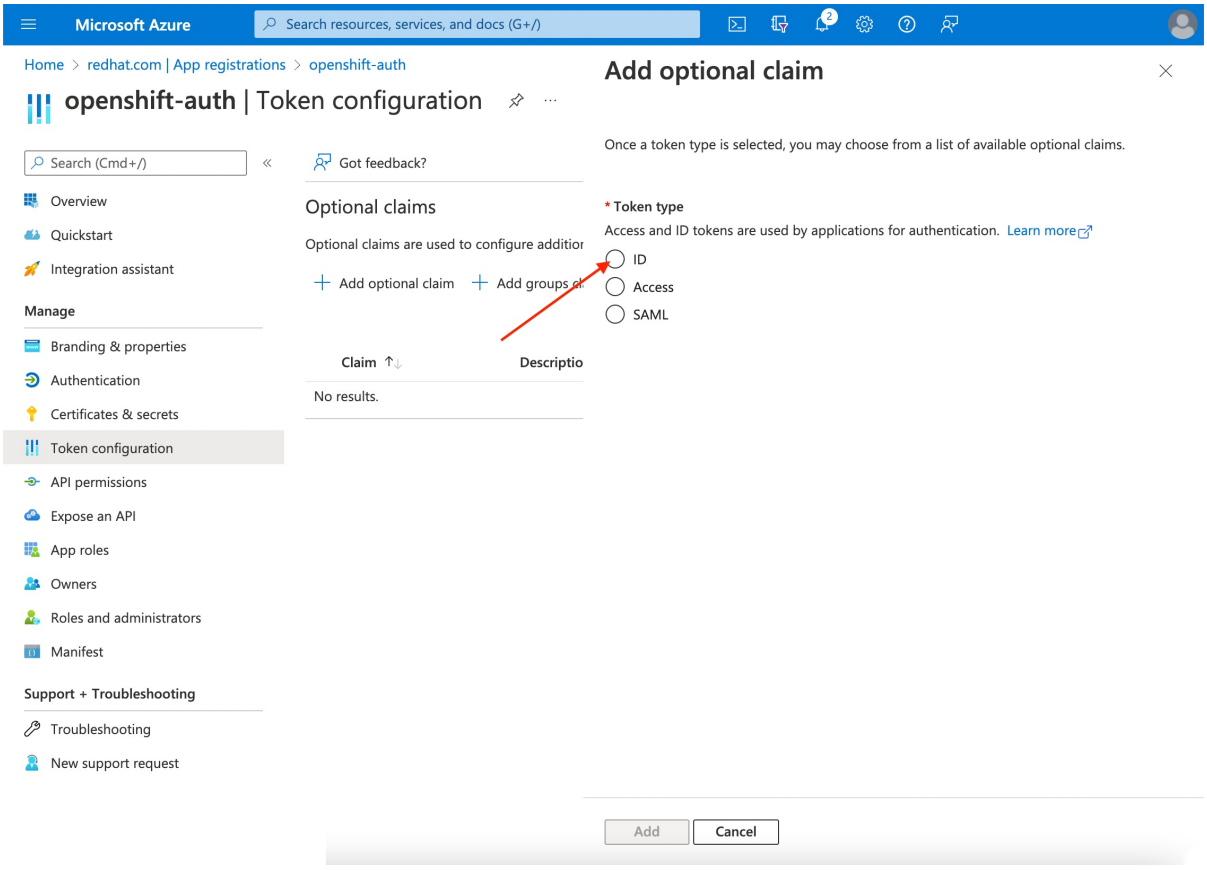
9.3.1. Configuring optional claims

You can configure the optional claims in Entra ID.

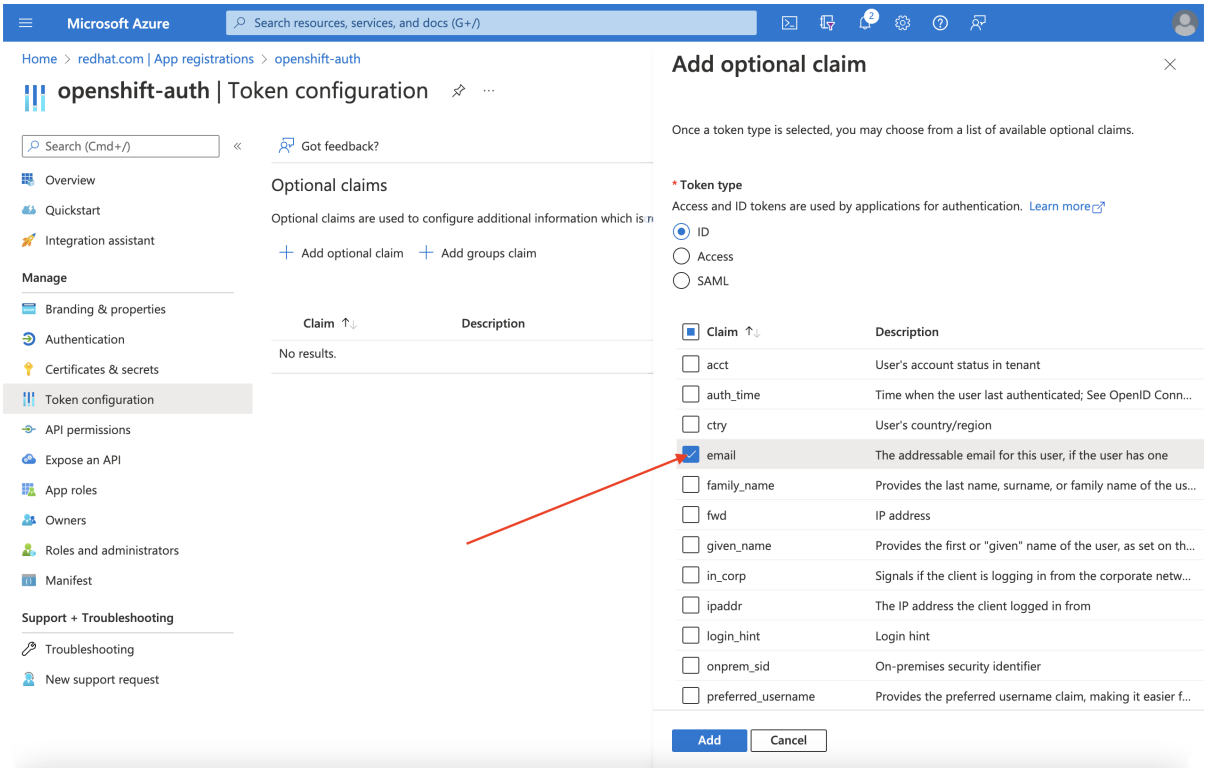
1. Click the **Token configuration** sub-blade and select the **Add optional claim** button.



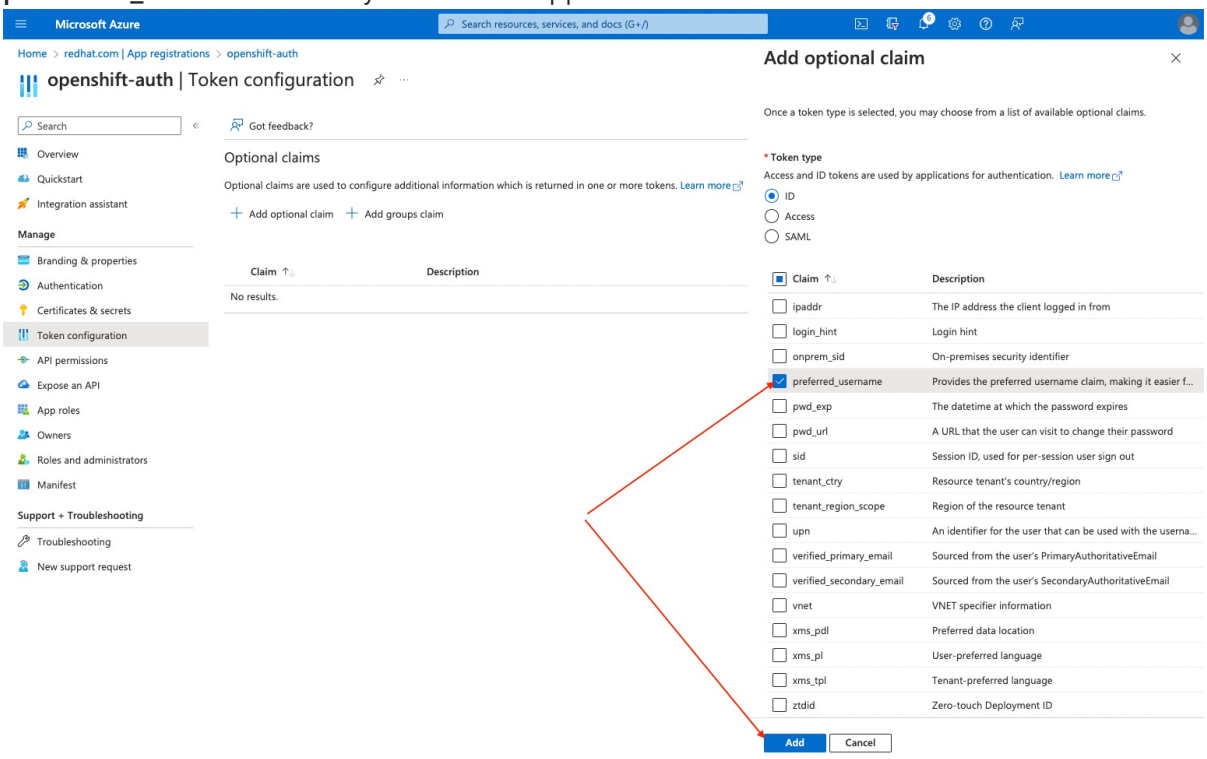
2. Select the **ID** radio button.



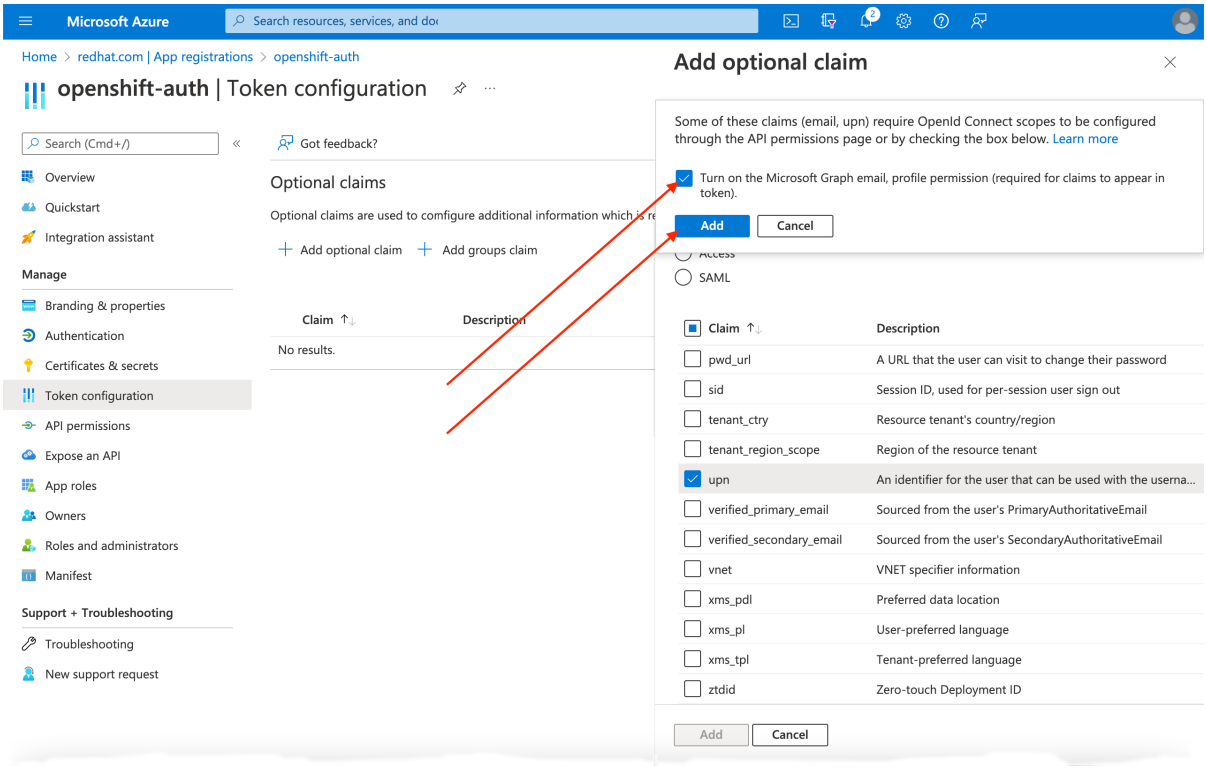
3. Select the **email** claim checkbox.



4. Select the **preferred_username** claim checkbox. Then, click **Add** to configure the **email** and **preferred_username** claims your Entra ID application.



5. A dialog box appears at the top of the page. Follow the prompt to enable the necessary Microsoft Graph permissions.

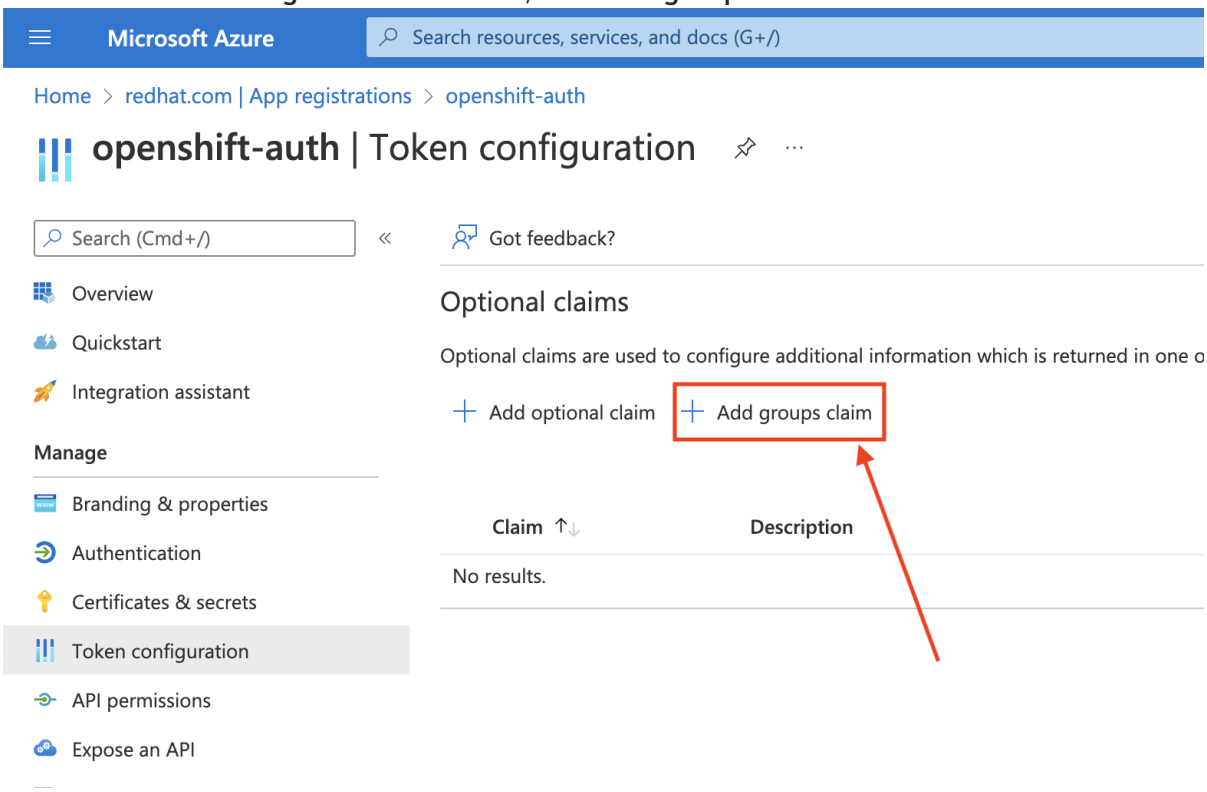


9.3.2. Configuring group claims (optional)

Configure Entra ID to offer a groups claim.

Procedure

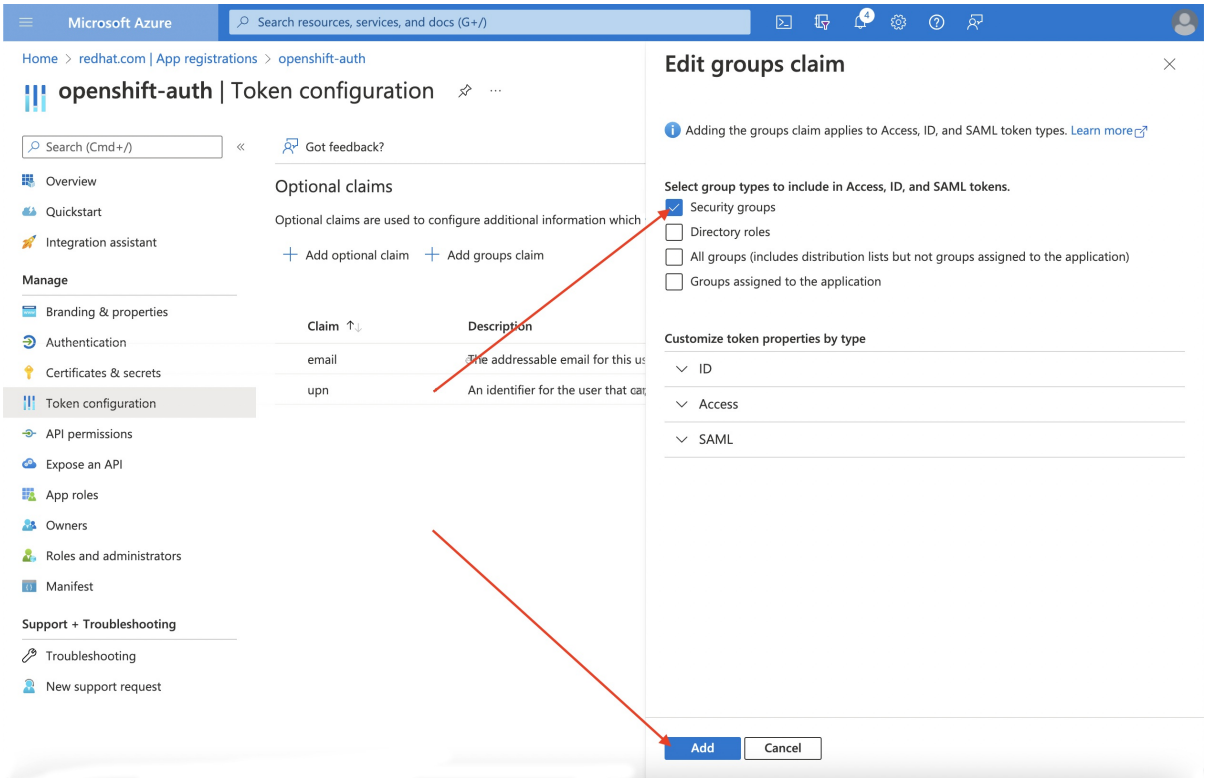
1. From the **Token configuration** sub-blade, click **Add groups claim**.



2. To configure group claims for your Entra ID application, select **Security groups** and then click the **Add**.

NOTE

In this example, the group claim includes all of the security groups that a user is a member of. In a real production environment, ensure that the groups that the group claim only includes groups that apply to Red Hat OpenShift Service on AWS.



9.4. CONFIGURING THE RED HAT OPENSHIFT SERVICE ON AWS CLUSTER TO USE ENTRA ID AS THE IDENTITY PROVIDER

You must configure Red Hat OpenShift Service on AWS to use Entra ID as its identity provider.

Although Red Hat OpenShift Service on AWS offers the ability to configure identity providers by using OpenShift Cluster Manager, use the ROSA CLI to configure the cluster's OAuth provider to use Entra ID as its identity provider. Before configuring the identity provider, set the necessary variables for the identity provider configuration.

Procedure

1. Create the variables by running the following command:

```
$ CLUSTER_NAME=example-cluster 1
$ IDP_NAME=AAD 2
$ APP_ID=yyyyyyyy-yyyy-yyyy-yyyy-yyyyyyyyyyyy 3
$ CLIENT_SECRET=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx 4
$ TENANT_ID=zzzzzzzz-zzzz-zzzz-zzzz-zzzzzzzzzzzz 5
```

1. Replace this with the name of your cluster.
2. Replace this value with the name you used in the OAuth callback URL that you generated earlier in this process.

3. Replace this with the Application (client) ID.
4. Replace this with the Client Secret.
5. Replace this with the Directory (tenant) ID.

2. Configure the cluster's OAuth provider by running the following command. If you enabled group claims, ensure that you use the **--group-claims groups** argument.

   - If you enabled group claims, run the following command:

```
$ rosa create idp \
--cluster ${CLUSTER_NAME} \
--type openid \
--name ${IDP_NAME} \
--client-id ${APP_ID} \
--client-secret ${CLIENT_SECRET} \
--issuer-url https://login.microsoftonline.com/${TENANT_ID}/v2.0 \
--email-claims email \
--name-claims name \
--username-claims preferred_username \
--extra-scopes email,profile \
--groups-claims groups
```

   - If you did not enable group claims, run the following command:

```
$ rosa create idp \
--cluster ${CLUSTER_NAME} \
--type openid \
--name ${IDP_NAME} \
--client-id ${APP_ID} \
--client-secret ${CLIENT_SECRET} \
--issuer-url https://login.microsoftonline.com/${TENANT_ID}/v2.0 \
--email-claims email \
--name-claims name \
--username-claims preferred_username \
--extra-scopes email,profile
```

After a few minutes, the cluster authentication Operator reconciles your changes, and you can log in to the cluster by using Entra ID.

9.5. GRANTING ADDITIONAL PERMISSIONS TO INDIVIDUAL USERS AND GROUPS

When your first log in, you might notice that you have very limited permissions. By default, Red Hat OpenShift Service on AWS only grants you the ability to create new projects, or namespaces, in the cluster. Other projects are restricted from view.

You must grant these additional abilities to individual users and groups.

9.5.1. Granting additional permissions to individual users

Red Hat OpenShift Service on AWS includes a significant number of preconfigured roles, including the **cluster-admin** role that grants full access and control over the cluster.

Procedure

- Grant a user access to the **cluster-admin** role by running the following command:

```
$ rosa grant user cluster-admin \
  --user=<USERNAME> 1
  --cluster=${CLUSTER_NAME}
```

1 Provide the Entra ID username that you want to have cluster admin permissions.

9.5.2. Granting additional permissions to individual groups

If you opted to enable group claims, the cluster OAuth provider automatically creates or updates the user's group memberships by using the group ID. The cluster OAuth provider does not automatically create **RoleBindings** and **ClusterRoleBindings** for the groups that are created; you are responsible for creating those bindings by using your own processes.

To grant an automatically generated group access to the **cluster-admin** role, you must create a **ClusterRoleBinding** to the group ID.

Procedure

- Create the **ClusterRoleBinding** by running the following command:

```
$ oc create clusterrolebinding cluster-admin-group \
--clusterrole=cluster-admin \
--group=<GROUP_ID> 1
```

1 Provide the Entra ID group ID that you want to have cluster admin permissions.

Now, any user in the specified group automatically receives **cluster-admin** access.

9.6. ADDITIONAL RESOURCES

For more information about how to use RBAC to define and apply permissions in Red Hat OpenShift Service on AWS, see the Red Hat OpenShift Service on AWS documentation .

CHAPTER 10. TUTORIAL: USING AWS SECRETS MANAGER CSI ON RED HAT OPENSHIFT SERVICE ON AWS WITH STS

The AWS Secrets and Configuration Provider (ASCP) provides a way to expose AWS Secrets as Kubernetes storage volumes. With the ASCP, you can store and manage your secrets in Secrets Manager and then retrieve them through your workloads running on Red Hat OpenShift Service on AWS.

10.1. PREREQUISITES

Ensure that you have the following resources and tools before starting this process:

- A Red Hat OpenShift Service on AWS cluster deployed with STS
- Helm 3
- **aws** CLI
- **oc** CLI
- **jq** CLI

10.1.1. Additional environment requirements

1. Log in to your Red Hat OpenShift Service on AWS cluster by running the following command:

```
$ oc login --token=<your-token> --server=<your-server-url>
```

   You can find your login token by accessing your cluster in pull secret from Red Hat OpenShift Cluster Manager.

2. Validate that your cluster has STS by running the following command:

```
$ oc get authentication.config.openshift.io cluster -o json \
  | jq .spec.serviceAccountIssuer
```

   Example output

```
"https://xxxxx.cloudfront.net/xxxxx"
```

   If your output is different, do not proceed. See Red Hat documentation on creating an STS cluster before continuing this process.

3. Set the **SecurityContextConstraints** permission to allow the CSI driver to run by running the following command:

```
$ oc new-project csi-secrets-store
$ oc adm policy add-scc-to-user privileged \
    system:serviceaccount:csi-secrets-store:secrets-store-csi-driver
$ oc adm policy add-scc-to-user privileged \
    system:serviceaccount:csi-secrets-store:csi-secrets-store-provider-aws
```

4. Create environment variables to use later in this process by running the following command:

```
$ export REGION=$(oc get infrastructure cluster -o=jsonpath="
{.status.platformStatus.aws.region}")
$ export OIDC_ENDPOINT=$(oc get authentication.config.openshift.io cluster \
  -o jsonpath='{.spec.serviceAccountIssuer}' | sed  's|^https://||')
$ export AWS_ACCOUNT_ID=`aws sts get-caller-identity --query Account --output text`
$ export AWS_PAGER=""
```

10.2. DEPLOYING THE AWS SECRETS AND CONFIGURATION PROVIDER

1. Use Helm to register the secrets store CSI driver by running the following command:

```
$ helm repo add secrets-store-csi-driver \
    https://kubernetes-sigs.github.io/secrets-store-csi-driver/charts
```

2. Update your Helm repositories by running the following command:

```
$ helm repo update
```

3. Install the secrets store CSI driver by running the following command:

```
$ helm upgrade --install -n csi-secrets-store \
    csi-secrets-store-driver secrets-store-csi-driver/secrets-store-csi-driver
```

4. Deploy the AWS provider by running the following command:

```
$ oc -n csi-secrets-store apply -f \
    https://raw.githubusercontent.com/rh-mobb/documentation/main/content/misc/secrets-
store-csi/aws-provider-installer.yaml
```

5. Check that both Daemonsets are running by running the following command:

```
$ oc -n csi-secrets-store get ds \
    csi-secrets-store-provider-aws \
    csi-secrets-store-driver-secrets-store-csi-driver
```

6. Label the Secrets Store CSI Driver to allow use with the restricted pod security profile by running the following command:

```
$ oc label csidriver.storage.k8s.io/secrets-store.csi.k8s.io security.openshift.io/csi-ephemeral-
volume-profile=restricted
```

10.3. CREATING A SECRET AND IAM ACCESS POLICIES

1. Create a secret in Secrets Manager by running the following command:

```
$ SECRET_ARN=$(aws --region "$REGION" secretsmanager create-secret \
    --name MySecret --secret-string \
    '{"username":"shadowman", "password":"hunter2"}' \
    --query ARN --output text); echo $SECRET_ARN
```

2. Create an IAM Access Policy document by running the following command:

```
$ cat << EOF > policy.json
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "secretsmanager:GetSecretValue",
      "secretsmanager:DescribeSecret"
    ],
    "Resource": ["$SECRET_ARN"]
    }]
}
EOF
```

3. Create an IAM Access Policy by running the following command:

```
$ POLICY_ARN=$(aws --region "$REGION" --query Policy.Arn \
--output text iam create-policy \
--policy-name openshift-access-to-mysecret-policy \
--policy-document file://policy.json); echo $POLICY_ARN
```

4. Create an IAM Role trust policy document by running the following command:

NOTE

The trust policy is locked down to the default service account of a namespace you create later in this process.

```
$ cat <<EOF > trust-policy.json
{
  "Version": "2012-10-17",
  "Statement": [
  {
  "Effect": "Allow",
  "Condition": {
    "StringEquals" : {
      "${OIDC_ENDPOINT}:sub": ["system:serviceaccount:my-application:default"]
    }
   },
   "Principal": {
      "Federated": "arn:aws:iam::$AWS_ACCOUNT_ID:oidc-provider/${OIDC_ENDPOINT}"
   },
   "Action": "sts:AssumeRoleWithWebIdentity"
   }
   ]
}
EOF
```

5. Create an IAM role by running the following command:

```
$ ROLE_ARN=$(aws iam create-role --role-name openshift-access-to-mysecret \
--assume-role-policy-document file://trust-policy.json \
--query Role.Arn --output text); echo $ROLE_ARN
```

6. Attach the role to the policy by running the following command:

```
$ aws iam attach-role-policy --role-name openshift-access-to-mysecret \
    --policy-arn $POLICY_ARN
```

10.4. CREATE AN APPLICATION TO USE THIS SECRET

1. Create an OpenShift project by running the following command:

```
$ oc new-project my-application
```

2. Annotate the default service account to use the STS Role by running the following command:

```
$ oc annotate -n my-application serviceaccount default \
    eks.amazonaws.com/role-arn=$ROLE_ARN
```

3. Create a secret provider class to access our secret by running the following command:

```
$ cat << EOF | oc apply -f -
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: my-application-aws-secrets
spec:
  provider: aws
  parameters:
    objects: |
      - objectName: "MySecret"
        objectType: "secretsmanager"
EOF
```

4. Create a deployment by using our secret in the following command:

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: my-application
  labels:
    app: my-application
spec:
  volumes:
  - name: secrets-store-inline
    csi:
      driver: secrets-store.csi.k8s.io
      readOnly: true
      volumeAttributes:
        secretProviderClass: "my-application-aws-secrets"
  containers:
```

```
  - name: my-application-deployment
    image: k8s.gcr.io/e2e-test-images/busybox:1.29
    command:
      - "/bin/sleep"
      - "10000"
    volumeMounts:
    - name: secrets-store-inline
      mountPath: "/mnt/secrets-store"
      readOnly: true
EOF
```

5. Verify the pod has the secret mounted by running the following command:

```
$ oc exec -it my-application -- cat /mnt/secrets-store/MySecret
```

10.5. CLEAN UP

1. Delete the application by running the following command:

```
$ oc delete project my-application
```

2. Delete the secrets store csi driver by running the following command:

```
$ helm delete -n csi-secrets-store csi-secrets-store-driver
```

3. Delete the security context constraints by running the following command:

```
$ oc adm policy remove-scc-from-user privileged \
    system:serviceaccount:csi-secrets-store:secrets-store-csi-driver; oc adm policy remove-
scc-from-user privileged \
    system:serviceaccount:csi-secrets-store:csi-secrets-store-provider-aws
```

4. Delete the AWS provider by running the following command:

```
$ oc -n csi-secrets-store delete -f \
https://raw.githubusercontent.com/rh-mobb/documentation/main/content/misc/secrets-store-
csi/aws-provider-installer.yaml
```

5. Delete AWS Roles and Policies by running the following command:

```
$ aws iam detach-role-policy --role-name openshift-access-to-mysecret \
    --policy-arn $POLICY_ARN; aws iam delete-role --role-name openshift-access-to-
mysecret; aws iam delete-policy --policy-arn $POLICY_ARN
```

6. Delete the Secrets Manager secret by running the following command:

```
$ aws secretsmanager --region $REGION delete-secret --secret-id $SECRET_ARN
```

CHAPTER 11. TUTORIAL: USING AWS CONTROLLERS FOR KUBERNETES ON RED HAT OPENSHIFT SERVICE ON AWS

AWS Controllers for Kubernetes (ACK) lets you define and use AWS service resources directly from Red Hat OpenShift Service on AWS. With ACK, you can take advantage of AWS-managed services for your applications without needing to define resources outside of the cluster or run services that provide supporting capabilities such as databases or message queues within the cluster.

You can install various ACK Operators directly from the software catalog. This makes it easy to get started and use the Operators with your applications. This controller is a component of the AWS Controller for Kubernetes project, which is currently in developer preview.

Use this tutorial to deploy the ACK S3 Operator. You can also adapt it for any other ACK Operator in the software catalog of your cluster.

11.1. PREREQUISITES

- A Red Hat OpenShift Service on AWS cluster
- A user account with **cluster-admin** privileges
- The OpenShift CLI (**oc**)
- The Amazon Web Services (AWS) CLI (**aws**)

11.2. SETTING UP YOUR ENVIRONMENT

1. Configure the following environment variables, changing the cluster name to suit your cluster:

```
$ export CLUSTER_NAME=$(oc get infrastructure cluster -o=jsonpath="
{.status.infrastructureName}"  | sed 's/-[a-z0-9]\{5\}$//')
$ export REGION=$(rosa describe cluster -c ${ROSA_CLUSTER_NAME} --output json | jq -r
.region.id)
$ export OIDC_ENDPOINT=$(oc get authentication.config.openshift.io cluster -o json | jq -r
.spec.serviceAccountIssuer | sed  's|^https://||')
$ export AWS_ACCOUNT_ID=`aws sts get-caller-identity --query Account --output text`
$ export ACK_SERVICE=s3
$ export ACK_SERVICE_ACCOUNT=ack-${ACK_SERVICE}-controller
$ export POLICY_ARN=arn:aws:iam::aws:policy/AmazonS3FullAccess
$ export AWS_PAGER=""
$ export SCRATCH="/tmp/${ROSA_CLUSTER_NAME}/ack"
$ mkdir -p ${SCRATCH}
```

2. Ensure all fields output correctly before moving to the next section:

```
$ echo "Cluster: ${ROSA_CLUSTER_NAME}, Region: ${REGION}, OIDC Endpoint:
${OIDC_ENDPOINT}, AWS Account ID: ${AWS_ACCOUNT_ID}"
```

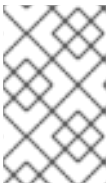
11.3. PREPARING YOUR AWS ACCOUNT

1. Create an AWS Identity Access Management (IAM) trust policy for the ACK Operator:

```
$ cat <<EOF > "${SCRATCH}/trust-policy.json"
```

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Condition": {
   "StringEquals" : {
     "${OIDC_ENDPOINT}:sub": "system:serviceaccount:ack-
system:${ACK_SERVICE_ACCOUNT}"
   }
 },
 "Principal": {
   "Federated": "arn:aws:iam::$AWS_ACCOUNT_ID:oidc-provider/${OIDC_ENDPOINT}"
 },
 "Action": "sts:AssumeRoleWithWebIdentity"
 }
 ]
}
EOF
```

2. Create an AWS IAM role for the ACK Operator to assume with the **AmazonS3FullAccess** policy attached:



NOTE

You can find the recommended policy in each project's GitHub repository, for example https://github.com/aws-controllers-k8s/s3-controller/blob/main/config/iam/recommended-policy-arn.

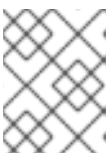
```
$ ROLE_ARN=$(aws iam create-role --role-name "ack-${ACK_SERVICE}-controller" \
  --assume-role-policy-document "file://${SCRATCH}/trust-policy.json" \
  --query Role.Arn --output text)
$ echo $ROLE_ARN

$ aws iam attach-role-policy --role-name "ack-${ACK_SERVICE}-controller" \
    --policy-arn ${POLICY_ARN}
```

11.4. INSTALLING THE ACK S3 CONTROLLER

1. Create a project to install the ACK S3 Operator into:

```
$ oc new-project ack-system
```

2. Create a file with the ACK S3 Operator configuration:



NOTE

ACK_WATCH_NAMESPACE is purposefully left blank so the controller can properly watch all namespaces in the cluster.

```
$ cat << EOF  "${SCRATCH}/config.txt"
ACK_ENABLE_DEVELOPMENT_LOGGING=true
```

```
ACK_LOG_LEVEL=debug
ACK_WATCH_NAMESPACE=
AWS_REGION=${REGION}
AWS_ENDPOINT_URL=
ACK_RESOURCE_TAGS=${CLUSTER_NAME}
ENABLE_LEADER_ELECTION=true
LEADER_ELECTION_NAMESPACE=
RECONCILE_DEFAULT_MAX_CONCURRENT_SYNCS=1
FEATURE_FLAGS=
FEATURE_GATES=
EOF
```

3. Use the file from the previous step to create a ConfigMap:

```
$ oc -n ack-system create configmap \
  --from-env-file=${SCRATCH}/config.txt ack-${ACK_SERVICE}-user-config
```

4. Install the ACK S3 Operator from the software catalog:

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ack-${ACK_SERVICE}-controller
  namespace: ack-system
spec:
  upgradeStrategy: Default
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ack-${ACK_SERVICE}-controller
  namespace: ack-system
spec:
  channel: alpha
  installPlanApproval: Automatic
  name: ack-${ACK_SERVICE}-controller
  source: community-operators
  sourceNamespace: openshift-marketplace
EOF
```

5. Annotate the ACK S3 Operator service account with the AWS IAM role to assume and restart the deployment:

```
$ oc -n ack-system annotate serviceaccount ${ACK_SERVICE_ACCOUNT} \
  eks.amazonaws.com/role-arn=${ROLE_ARN} && \
  oc -n ack-system rollout restart deployment ack-${ACK_SERVICE}-controller
```

6. Verify that the ACK S3 Operator is running:

```
$ oc -n ack-system get pods
```

Example output

```
NAME                                READY   STATUS    RESTARTS   AGE
ack-s3-controller-585f6775db-s4lfz  1/1     Running   0          51s
```

11.5. VALIDATING THE DEPLOYMENT

1. Deploy an S3 bucket resource:

```
$ cat << EOF | oc apply -f -
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: ${CLUSTER-NAME}-bucket
  namespace: ack-system
spec:
  name: ${CLUSTER-NAME}-bucket
EOF
```

2. Verify the S3 bucket was created in AWS:

```
$ aws s3 ls | grep ${CLUSTER_NAME}-bucket
```

Example output

```
2023-10-04 14:51:45 mrmc-test-maz-bucket
```

11.6. CLEANING UP

1. Delete the S3 bucket resource:

```
$ oc -n ack-system delete bucket.s3.services.k8s.aws/${CLUSTER-NAME}-bucket
```

2. Delete the ACK S3 Operator and the AWS IAM roles:

```
$ oc -n ack-system delete subscription ack-${ACK_SERVICE}-controller
$ aws iam detach-role-policy \
  --role-name "ack-${ACK_SERVICE}-controller" \
  --policy-arn ${POLICY_ARN}
$ aws iam delete-role \
  --role-name "ack-${ACK_SERVICE}-controller"
```

3. Delete the **ack-system** project:

```
$ oc delete project ack-system
```

CHAPTER 12. TUTORIAL: ASSIGNING A CONSISTENT EGRESS IP FOR EXTERNAL TRAFFIC

You can assign a consistent IP address for traffic that leaves your cluster such as security groups which require an IP-based configuration to meet security standards.

By default, Red Hat OpenShift Service on AWS uses the OVN-Kubernetes container network interface (CNI) to assign random IP addresses from a pool. This can make configuring security lockdowns unpredictable or open.

See Configuring an egress IP address for more information.

Objectives

- Learn how to configure a set of predictable IP addresses for egress cluster traffic.

Prerequisites

- A Red Hat OpenShift Service on AWS cluster deployed with OVN-Kubernetes
- The OpenShift CLI (**oc**)
- The ROSA CLI (**rosa**)
- **jq**

12.1. SETTING YOUR ENVIRONMENT VARIABLES

- Set your environment variables by running the following command:

NOTE

Replace the value of the **ROSA_MACHINE_POOL_NAME** variable to target a different machine pool.

```
$ export ROSA_CLUSTER_NAME=$(oc get infrastructure cluster -o=jsonpath="
{.status.infrastructureName}"  | sed 's/-[a-z0-9]\{5\}$//')
$ export ROSA_MACHINE_POOL_NAME=worker
```

12.2. ENSURING CAPACITY

The number of IP addresses assigned to each node is limited for each public cloud provider.

- Verify sufficient capacity by running the following command:

```
$ oc get node -o json | \
    jq '.items[] |
        {
            "name": .metadata.name,
            "ips": (.status.addresses | map(select(.type == "InternalIP") | .address)),
```

```
            "capacity": (.metadata.annotations."cloud.network.openshift.io/egress-ipconfig" |
fromjson[] | .capacity.ipv4)
        }'
```

Example output

```
---
{
  "name": "ip-10-10-145-88.ec2.internal",
  "ips": [
    "10.10.145.88"
  ],
  "capacity": 14
}
{
  "name": "ip-10-10-154-175.ec2.internal",
  "ips": [
    "10.10.154.175"
  ],
  "capacity": 14
}
---
```

12.3. CREATING THE EGRESS IP RULES

1. Before creating the egress IP rules, identify which egress IPs you will use.

NOTE

The egress IPs that you select should exist as a part of the subnets in which the worker nodes are provisioned.

2. **Optional**: Reserve the egress IPs that you requested to avoid conflicts with the AWS Virtual Private Cloud (VPC) Dynamic Host Configuration Protocol (DHCP) service.
Request explicit IP reservations on the AWS documentation for CIDR reservations page.

12.4. ASSIGNING AN EGRESS IP TO A NAMESPACE

1. Create a new project by running the following command:

```
$ oc new-project demo-egress-ns
```

2. Create the egress rule for all pods within the namespace by running the following command:

```
$ cat <<EOF | oc apply -f -
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: demo-egress-ns
spec:
  # NOTE: these egress IPs are within the subnet range(s) in which my worker nodes
  #       are deployed.
  egressIPs:
```

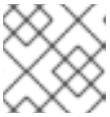
```
    - 10.10.100.253
    - 10.10.150.253
    - 10.10.200.253
  namespaceSelector:
    matchLabels:
      kubernetes.io/metadata.name: demo-egress-ns
EOF
```

12.5. ASSIGNING AN EGRESS IP TO A POD

1. Create a new project by running the following command:

```
$ oc new-project demo-egress-pod
```

2. Create the egress rule for the pod by running the following command:



NOTE

spec.namespaceSelector is a mandatory field.

```
$ cat <<EOF | oc apply -f -
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: demo-egress-pod
spec:
  # NOTE: these egress IPs are within the subnet range(s) in which my worker nodes
  #       are deployed.
  egressIPs:
    - 10.10.100.254
    - 10.10.150.254
    - 10.10.200.254
  namespaceSelector:
    matchLabels:
      kubernetes.io/metadata.name: demo-egress-pod
  podSelector:
    matchLabels:
      run: demo-egress-pod
EOF
```

12.5.1. Labeling the nodes

1. Obtain your pending egress IP assignments by running the following command:

```
$ oc get egressips
```

Example output

```
NAME              EGRESSIPS       ASSIGNED NODE   ASSIGNED EGRESSIPS
demo-egress-ns    10.10.100.253
demo-egress-pod   10.10.100.254
```

The egress IP rule that you created only applies to nodes with the **k8s.ovn.org/egress-assignable** label. Make sure that the label is only on a specific machine pool.

2. Assign the label to your machine pool using the following command:

WARNING

If you rely on node labels for your machine pool, this command will replace those labels. Be sure to input your desired labels into the **--labels** field to ensure your node labels remain.

```
$ rosa update machinepool ${ROSA_MACHINE_POOL_NAME} \
  --cluster="${ROSA_CLUSTER_NAME}" \
  --labels "k8s.ovn.org/egress-assignable="
```

12.5.2. Reviewing the egress IPs

- Review the egress IP assignments by running the following command:

```
$ oc get egressips
```

Example output

```
NAME              EGRESSIPS       ASSIGNED NODE                 ASSIGNED EGRESSIPS
demo-egress-ns    10.10.100.253   ip-10-10-156-122.ec2.internal   10.10.150.253
demo-egress-pod   10.10.100.254   ip-10-10-156-122.ec2.internal   10.10.150.254
```

12.6. VERIFICATION

12.6.1. Deploying a sample application

To test the egress IP rule, create a service that is restricted to the egress IP addresses which we have specified. This simulates an external service that is expecting a small subset of IP addresses.

1. Run the **echoserver** command to replicate a request:

```
$ oc -n default run demo-service --image=gcr.io/google_containers/echoserver:1.4
```

2. Expose the pod as a service and limit the ingress to the egress IP addresses you specified by running the following command:

```
$ cat <<EOF | oc apply -f -
apiVersion: v1
kind: Service
metadata:
  name: demo-service
  namespace: default
```

```
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-scheme: "internal"
    service.beta.kubernetes.io/aws-load-balancer-internal: "true"
spec:
  selector:
    run: demo-service
  ports:
    - port: 80
      targetPort: 8080
  type: LoadBalancer
  externalTrafficPolicy: Local
  # NOTE: this limits the source IPs that are allowed to connect to our service.  It
  #       is being used as part of this demo, restricting connectivity to our egress
  #       IP addresses only.
  # NOTE: these egress IPs are within the subnet range(s) in which my worker nodes
  #       are deployed.
  loadBalancerSourceRanges:
    - 10.10.100.254/32
    - 10.10.150.254/32
    - 10.10.200.254/32
    - 10.10.100.253/32
    - 10.10.150.253/32
    - 10.10.200.253/32
EOF
```

3. Retrieve the load balancer hostname and save it as an environment variable by running the following command:

```
$ export LOAD_BALANCER_HOSTNAME=$(oc get svc -n default demo-service -o json | jq -
r '.status.loadBalancer.ingress[].hostname')
```

12.6.2. Testing the namespace egress

1. Start an interactive shell to test the namespace egress rule:

```
$ oc run \
  demo-egress-ns \
  -it \
  --namespace=demo-egress-ns \
  --env=LOAD_BALANCER_HOSTNAME=$LOAD_BALANCER_HOSTNAME \
  --image=registry.access.redhat.com/ubi9/ubi -- \
  bash
```

2. Send a request to the load balancer and ensure that you can successfully connect:

```
$ curl -s http://$LOAD_BALANCER_HOSTNAME
```

3. Check the output for a successful connection:

NOTE

The **client_address** is the internal IP address of the load balancer not your egress IP. You can verify that you have configured the client address correctly by connecting with your service limited to **.spec.loadBalancerSourceRanges**.

Example output

```
CLIENT VALUES:
client_address=10.10.207.247
command=GET
real path=/
query=nil
request_version=1.1
request_uri=http://internal-a3e61de18bfca4a53a94a208752b7263-148284314.us-east-
1.elb.amazonaws.com:8080/

SERVER VALUES:
server_version=nginx: 1.10.0 - lua: 10001

HEADERS RECEIVED:
accept=*/*
host=internal-a3e61de18bfca4a53a94a208752b7263-148284314.us-east-
1.elb.amazonaws.com
user-agent=curl/7.76.1
BODY:
-no body in request-
```

4. Exit the pod by running the following command:

```
$ exit
```

12.6.3. Testing the pod egress

1. Start an interactive shell to test the pod egress rule:

```
$ oc run \
  demo-egress-pod \
  -it \
  --namespace=demo-egress-pod \
  --env=LOAD_BALANCER_HOSTNAME=$LOAD_BALANCER_HOSTNAME \
  --image=registry.access.redhat.com/ubi9/ubi -- \
  bash
```

2. Send a request to the load balancer by running the following command:

```
$ curl -s http://$LOAD_BALANCER_HOSTNAME
```

3. Check the output for a successful connection:

NOTE

The **client_address** is the internal IP address of the load balancer not your egress IP. You can verify that you have configured the client address correctly by connecting with your service limited to **.spec.loadBalancerSourceRanges**.

Example output

```
CLIENT VALUES:
```

```
client_address=10.10.207.247
command=GET
real path=/
query=nil
request_version=1.1
request_uri=http://internal-a3e61de18bfca4a53a94a208752b7263-148284314.us-east-
1.elb.amazonaws.com:8080/

SERVER VALUES:
server_version=nginx: 1.10.0 - lua: 10001

HEADERS RECEIVED:
accept=*/*
host=internal-a3e61de18bfca4a53a94a208752b7263-148284314.us-east-
1.elb.amazonaws.com
user-agent=curl/7.76.1
BODY:
-no body in request-
```

4. Exit the pod by running the following command:

```
$ exit
```

12.6.4. Optional: Testing blocked egress

1. **Optional:** Test that the traffic is successfully blocked when the egress rules do not apply by running the following command:

```
$ oc run \
  demo-egress-pod-fail \
  -it \
  --namespace=demo-egress-pod \
  --env=LOAD_BALANCER_HOSTNAME=$LOAD_BALANCER_HOSTNAME \
  --image=registry.access.redhat.com/ubi9/ubi -- \
  bash
```

2. Send a request to the load balancer by running the following command:

```
$ curl -s http://$LOAD_BALANCER_HOSTNAME
```

3. If the command is unsuccessful, egress is successfully blocked.

4. Exit the pod by running the following command:

```
$ exit
```

12.7. CLEANING UP YOUR CLUSTER

1. Clean up your cluster by running the following commands:

```
$ oc delete svc demo-service -n default; \
$ oc delete pod demo-service -n default; \
$ oc delete project demo-egress-ns; \
```

```
$ oc delete project demo-egress-pod; \
$ oc delete egressip demo-egress-ns; \
$ oc delete egressip demo-egress-pod
```

2. Clean up the assigned node labels by running the following command:

WARNING

If you rely on node labels for your machine pool, this command replaces those labels. Input your desired labels into the **--labels** field to ensure your node labels remain.

```
$ rosa update machinepool ${ROSA_MACHINE_POOL_NAME} \
  --cluster="${ROSA_CLUSTER_NAME}" \
  --labels ""
```