Red Hat OpenShift

Installing managed clusters with RHACM and SiteConfig resources

You can provision OpenShift Container Platform clusters at scale with Red Hat Advanced Cluster Management (RHACM) using the assisted service and the GitOps plugin policy generator with core-reduction technology enabled. The zero touch priovisioning (ZTP) pipeline performs the cluster installations. ZTP can be used in a disconnected environment.

GitOps ZTP and Topology Aware Lifecycle Manager

GitOps zero touch provisioning (ZTP) generates installation and configuration CRs from manifests stored in Git. These artifacts are applied to a centralized hub cluster where Red Hat Advanced Cluster Management (RHACM), the assisted service, and the Topology Aware Lifecycle Manager (TALM) use the CRs to install and configure the managed cluster. The configuration phase of the ZTP pipeline uses the TALM to orchestrate the application of the configuration CRs to the cluster. There are several key integration points between GitOps ZTP and the TALM.

Inform policies

By default, GitOps ZTP creates all policies with a remediation action of `inform`. These policies cause RHACM to report on compliance status of clusters relevant to the policies but does not apply the desired configuration. During the ZTP process, after OpenShift installation, the TALM steps through the created `inform` policies and enforces them on the target managed cluster(s). This applies the configuration to the managed cluster. Outside of the ZTP phase of the cluster lifecycle, this allows you to change policies without the risk of immediately rolling those changes out to affected managed clusters. You can control the timing and the set of remediated clusters by using TALM.

Automatic creation of ClusterGroupUpgrade CRs

To automate the initial configuration of newly deployed clusters, TALM monitors the state of all `ManagedCluster` CRs on the hub cluster. Any `ManagedCluster` CR that does not have a `ztp-done` label applied, including newly created `ManagedCluster` CRs, causes the TALM to automatically create a `ClusterGroupUpgrade` CR with the following characteristics:

- The `ClusterGroupUpgrade` CR is created and enabled in the `ztp-install` namespace.
- `ClusterGroupUpgrade` CR has the same name as the `ManagedCluster` CR.
- The cluster selector includes only the cluster associated with that `ManagedCluster` CR.
- The set of managed policies includes all policies that RHACM has bound to the cluster at the time the `ClusterGroupUpgrade` is created.
- Pre-caching is disabled.
- Timeout set to 4 hours (240 minutes).

The automatic creation of an enabled `ClusterGroupUpgrade` ensures that initial zero-touch deployment of clusters proceeds without the need for user intervention. Additionally, the automatic creation of a `ClusterGroupUpgrade` CR for any `ManagedCluster` without the `ztp-done` label allows a failed ZTP installation to be restarted by simply deleting the `ClusterGroupUpgrade` CR for the cluster.

Waves

Each policy generated from a `PolicyGenTemplate` CR includes a `ztp-deploy-wave` annotation. This annotation is based on the same annotation from each CR which is included in that policy. The wave annotation is used to order the policies in the auto-generated `ClusterGroupUpgrade` CR. The wave annotation is not used other than for the auto-generated `ClusterGroupUpgrade` CR.

All CRs in the same policy must have the same setting for the `ztp-deploy-wave` annotation. The default value of this annotation for each CR can be overridden in the `PolicyGenTemplate` . The wave annotation in the source CR is used for determining and setting the policy wave annotation. This annotation is removed from each built CR which is included in the generated policy at runtime.

The TALM applies the configuration policies in the order specified by the wave annotations. The TALM waits for each policy to be compliant before moving to the next policy. It is important to ensure that the wave annotation for each CR takes into account any prerequisites for those CRs to be applied to the cluster. For example, an Operator must be installed before or concurrently with the configuration for the Operator. Similarly, the `CatalogSource` for an Operator must be installed in a wave before or concurrently with the Operator Subscription. The default wave value for each CR takes these prerequisites into account.

Multiple CRs and policies can share the same wave number. Having fewer policies can result in faster deployments and lower CPU usage. It is a best practice to group many CRs into relatively few waves.
To check the default wave value in each source CR, run the following command against the `out/source-crs` directory that is extracted from the `ztp-site-generate` container image:

```
$ grep -r "ztp-deploy-wave" out/source-crs
```

Phase labels
The `ClusterGroupUpgrade` CR is automatically created and includes directives to annotate the `ManagedCluster` CR with labels at the start and end of the ZTP process.

When ZTP configuration post-installation commences, the `ManagedCluster` has the `ztp-running` label applied. When all policies are remediated to the cluster and are fully compliant, these directives cause the TALM to remove the `ztp-running` label and apply the `ztp-done` label.

For deployments that make use of the `informDuValidator` policy, the `ztp-done` label is applied when the cluster is fully ready for deployment of applications. This includes all reconciliation and resulting effects of the ZTP applied configuration CRs. The `ztp-done` label affects automatic `ClusterGroupUpgrade` CR creation by TALM. Do not manipulate this label after the initial ZTP installation of the cluster.

Linked CRs

The automatically created `ClusterGroupUpgrade` CR has the owner reference set as the `ManagedCluster` from which it was derived. This reference ensures that deleting the `ManagedCluster` CR causes the instance of the `ClusterGroupUpgrade` to be deleted along with any supporting resources.

Overview of deploying managed clusters with ZTP

Red Hat Advanced Cluster Management (RHACM) uses zero touch provisioning (ZTP) to deploy single-node OpenShift Container Platform clusters, three-node clusters, and standard clusters. You manage site configuration data as OpenShift Container Platform custom resources (CRs) in a Git repository. ZTP uses a declarative GitOps approach for a develop once, deploy anywhere model to deploy the managed clusters.

The deployment of the clusters includes:

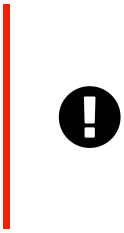
- Installing the host operating system (RHCOS) on a blank server
- Deploying OpenShift Container Platform
- Creating cluster policies and site subscriptions
- Making the necessary network configurations to the server operating system
- Deploying profile Operators and performing any needed software-related configuration, such as performance profile, PTP, and SR-IOV

Overview of the managed site installation process

After you apply the managed site custom resources (CRs) on the hub cluster, the following actions happen automatically:

- A Discovery image ISO file is generated and booted on the target host.
- When the ISO file successfully boots on the target host it reports the host hardware information to RHACM.
- After all hosts are discovered, OpenShift Container Platform is installed.
- When OpenShift Container Platform finishes installing, the hub installs the `klusterlet` service on the target cluster.
- The requested add-on services are installed on the target cluster.

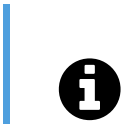
The Discovery image ISO process is complete when the `Agent` CR for the managed cluster is created on the hub cluster.



The target bare-metal host must meet the networking, firmware, and hardware requirements listed in Recommended single-node OpenShift cluster configuration for vDU application workloads.

Creating the managed bare-metal host secrets

Add the required `Secret` custom resources (CRs) for the managed bare-metal host to the hub cluster. You need a secret for the ZTP pipeline to access the Baseboard Management Controller (BMC) and a secret for the assisted installer service to pull cluster installation images from the registry.



The secrets are referenced from the `SiteConfig` CR by name. The namespace must match the `SiteConfig` namespace.

Procedure

- Create a YAML secret file containing credentials for the host Baseboard Management Controller (BMC) and a pull secret required for installing OpenShift and all add-on cluster Operators:
    - Save the following YAML as the file `example-sno-secret.yaml`:

```
apiVersion: v1
kind: Secret
metadata:
  name: example-sno-bmc-secret
  namespace: example-sno (1)
data: (2)
  password: <base64_password>
  username: <base64_username>
type: Opaque
---
apiVersion: v1
kind: Secret
metadata:
  name: pull-secret
  namespace: example-sno  (3)
data:
  .dockerconfigjson: <pull_secret> (4)
type: kubernetes.io/dockerconfigjson
```

1. Must match the namespace configured in the related `SiteConfig` CR
2. Base64-encoded values for `password` and `username`
3. Must match the namespace configured in the related `SiteConfig` CR
4. Base64-encoded pull secret

- Add the relative path to `example-sno-secret.yaml` to the `kustomization.yaml` file that you use to install the cluster.

Configuring Discovery ISO kernel arguments for installations using GitOps ZTP

The GitOps ZTP workflow uses the Discovery ISO as part of the OpenShift Container Platform installation process on managed bare-metal hosts. You can edit the `InfraEnv` resource to specify kernel arguments for the Discovery ISO. This is useful for cluster installations with specific environmental requirements. For example, configure the `rd.net.timeout.carrier` kernel argument for the Discovery ISO to facilitate static networking for the cluster or to receive a DHCP address before downloading the root file system during installation.

In OpenShift Container Platform 4.12, you can only add kernel arguments. You can not replace or delete kernel arguments.

Prerequisites

- You have installed the OpenShift CLI (oc).
- You have logged in to the hub cluster as a user with cluster-admin privileges.

Procedure

- Create the `InfraEnv` CR and edit the `spec.kernelArguments` specification to configure kernel arguments.
  - Save the following YAML in an `InfraEnv-example.yaml` file:

The `InfraEnv` CR in this example uses template syntax such as `{{ .Cluster.ClusterName }}` that is populated based on values in the `SiteConfig` CR. The `SiteConfig` CR automatically populates values for these templates during deployment. Do not edit the templates manually.

```
apiVersion: agent-install.openshift.io/v1beta1
kind: InfraEnv
metadata:
  annotations:
    argocd.argoproj.io/sync-wave: "1"
  name: "{{ .Cluster.ClusterName }}"
  namespace: "{{ .Cluster.ClusterName }}"
spec:
  clusterRef:
    name: "{{ .Cluster.ClusterName }}"
    namespace: "{{ .Cluster.ClusterName }}"
  kernelArguments:
    - operation: append (1)
      value: audit=0 (2)
    - operation: append
      value: trace=1
  sshAuthorizedKey: "{{ .Site.SshPublicKey }}"
  proxy: "{{ .Cluster.ProxySettings }}"
  pullSecretRef:
    name: "{{ .Site.PullSecretRef.Name }}"
  ignitionConfigOverride: "{{
.Cluster.IgnitionConfigOverride }}"
  nmStateConfigLabelSelector:
    matchLabels:
      nmstate-label: "{{ .Cluster.ClusterName }}"
  additionalNTPSources: "{{ .Cluster.AdditionalNTPSources
}}"
```

1 Specify the append operation to add a kernel argument.

2 Specify the kernel argument you want to configure. This example configures the audit kernel argument and the trace kernel argument.

- Commit the `InfraEnv-example.yaml` CR to the same location in your Git repository that has the `SiteConfig` CR and push your changes. The following example shows a sample Git repository structure:

```
~/example-ztp/install
          └── site-install
               ├── siteconfig-example.yaml
               ├── InfraEnv-example.yaml
               ...
```

- Edit the `spec.clusters.crTemplates` specification in the `SiteConfig` CR to reference the `InfraEnv-example.yaml` CR in your Git repository:

```
clusters:
  crTemplates:
    InfraEnv: "InfraEnv-example.yaml"
```

  When you are ready to deploy your cluster by committing and pushing the `SiteConfig` CR, the build pipeline uses the custom `InfraEnv-example` CR in your Git repository to configure the infrastructure environment, including the custom kernel arguments.

Verification

To verify that the kernel arguments are applied, after the Discovery image verifies that OpenShift Container Platform is ready for installation, you can SSH to the target host before the installation process begins. At that point, you can view the kernel arguments for the Discovery ISO in the `/proc/cmdline` file.

- Begin an SSH session with the target host:

```
$ ssh -i /path/to/privatekey core@<host_name>
```

- View the system's kernel arguments by using the following command:

```
$ cat /proc/cmdline
```

Deploying a managed cluster with SiteConfig and ZTP

Use the following procedure to create a `SiteConfig` custom resource (CR) and related files and initiate the zero touch provisioning (ZTP) cluster deployment.

Prerequisites

- You have installed the OpenShift CLI (`oc`).
- You have logged in to the hub cluster as a user with `cluster-admin` privileges.
- You configured the hub cluster for generating the required installation and policy CRs.

- You created a Git repository where you manage your custom site configuration data. The repository must be accessible from the hub cluster and you must configure it as a source repository for the ArgoCD application. See "Preparing the GitOps ZTP site configuration repository" for more information.

> 
>
> When you create the source repository, ensure that you patch the ArgoCD application with the `argocd/deployment/argocd-openshift-gitops-patch.json` patch-file that you extract from the `ztp-site-generate` container. See "Configuring the hub cluster with ArgoCD".

- To be ready for provisioning managed clusters, you require the following for each bare-metal host:

  Network connectivity
  Your network requires DNS. Managed cluster hosts should be reachable from the hub cluster. Ensure that Layer 3 connectivity exists between the hub cluster and the managed cluster host.

  Baseboard Management Controller (BMC) details
  ZTP uses BMC username and password details to connect to the BMC during cluster installation. The GitOps ZTP plugin manages the `ManagedCluster` CRs on the hub cluster based on the `SiteConfig` CR in your site Git repo. You create individual `BMCSecret` CRs for each host manually.

  Procedure

  - Create the required managed cluster secrets on the hub cluster. These resources must be in a namespace with a name matching the cluster name. For example, in `out/argocd/example/siteconfig/example-sno.yaml`, the cluster name and namespace is `example-sno`.
    - Export the cluster namespace by running the following command:

      ```
      $ export CLUSTERNS=example-sno
      ```

    - Create the namespace:

```
$ oc create namespace $CLUSTERNS
```

- Create pull secret and BMC `Secret` CRs for the managed cluster. The pull secret must contain all the credentials necessary for installing OpenShift Container Platform and all required Operators. See "Creating the managed bare-metal host secrets" for more information.

> **Note**
>
> The secrets are referenced from the `SiteConfig` custom resource (CR) by name. The namespace must match the `SiteConfig` namespace.

- Create a `SiteConfig` CR for your cluster in your local clone of the Git repository:
  - Choose the appropriate example for your CR from the `out/argocd/example/siteconfig/` folder. The folder includes example files for single node, three-node, and standard clusters:
    - `example-sno.yaml`
    - `example-3node.yaml`
    - `example-standard.yaml`
  - Change the cluster and host details in the example file to match the type of cluster you want. For example:

    Example single-node OpenShift cluster SiteConfig CR

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "<site_name>"
  namespace: "<site_name>"
spec:
  baseDomain: "example.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret" (1)
  clusterImageSetNameRef: "openshift-4.12" (2)
  sshPublicKey: "ssh-rsa AAAA..." (3)
  clusters:
  - clusterName: "<site_name>"
    networkType: "OVNKubernetes"
    clusterLabels: (4)
      common: true
      group-du-sno: ""
      sites : "<site_name>"
    clusterNetwork:
      - cidr: 1001:1::/48
        hostPrefix: 64
    machineNetwork:
      - cidr: 1111:2222:3333:4444::/64
    serviceNetwork:
      - 1001:2::/112
    additionalNTPSources:
      - 1111:2222:3333:4444::2
    #crTemplates:
    #  KlusterletAddonConfig:
"KlusterletAddonConfigOverride.yaml" (5)
    nodes:
      - hostName: "example-node.example.com" (6)
        role: "master"
        bmcAddress: idrac-
virtualmedia://<out_of_band_ip>/<system_id>/ (7)
        bmcCredentialsName:
          name: "bmh-secret" (8)
        bootMACAddress: "AA:BB:CC:DD:EE:11"
        bootMode: "UEFI" (9)
        rootDeviceHints:
          wwn: "0x11111000000asd123"
        cpuset: "0-1,52-53"  (10)
        nodeNetwork: (11)
```

```
          interfaces:
            - name: eno1
              macAddress: "AA:BB:CC:DD:EE:11"
          config:
            interfaces:
              - name: eno1
                type: ethernet
                state: up
                ipv4:
                  enabled: false
                ipv6: (12)
                  enabled: true
                  address:
                  - ip: 1111:2222:3333:4444::aaaa:1
                    prefix-length: 64
            dns-resolver:
              config:
                search:
                - example.com
                server:
                - 1111:2222:3333:4444::2
            routes:
              config:
              - destination: ::/0
                next-hop-interface: eno1
                next-hop-address:
1111:2222:3333:4444::1
                table-id: 254
```

1 Create the `assisted-deployment-pull-secret` CR with the same namespace as the `SiteConfig` CR.

2 `clusterImageSetNameRef` defines an image set available on the hub cluster. To see the list of supported versions on your hub cluster, run `oc get clusterimagesets`.

3 Configure the SSH public key used to access the cluster.

4 Cluster labels must correspond to the `bindingRules` field in the `PolicyGenTemplate` CRs that you define. For example, `policygentemplates/common-ranGen.yaml` applies to all clusters with `common: true` set, `policygentemplates/group-du-sno-ranGen.yaml` applies to all clusters with `group-du-sno: ""` set.

5 Optional. The CR specifed under `KlusterletAddonConfig` is used to override the default `KlusterletAddonConfig` that is created for the cluster.

6 For single-node deployments, define a single host. For three-node deployments, define three hosts. For standard deployments, define three hosts with `role: master` and two or more hosts defined with `role: worker` .

7 BMC address that you use to access the host. Applies to all cluster types.

8 Name of the `bmh-secret` CR that you separately create with the host BMC credentials. When creating the `bmh-secret` CR, use the same namespace as the `SiteConfig` CR that provisions the host.

9 Configures the boot mode for the host. The default value is `UEFI` . Use `UEFISecureBoot` to enable secure boot on the host.

10 `cpuset` must match the value set in the cluster `PerformanceProfile` CR `spec.cpu.reserved` field for workload partitioning.

11 Specifies the network settings for the node.

12 Configures the IPv6 address for the host. For single-node OpenShift clusters with static IP addresses, the node-specific API and Ingress IPs should be the same.

> For more information about BMC addressing, see the "Additional resources" section.

- You can inspect the default set of extra-manifest `MachineConfig` CRs in `out/argocd/extra-manifest` . It is automatically applied to the cluster when it is installed.
- Optional: To provision additional install-time manifests on the provisioned cluster, create a directory in your Git repository, for example, `sno-extra-manifest/` , and add your custom manifest CRs to this directory. If your `SiteConfig.yaml` refers to this directory in the `extraManifestPath` field, any CRs in this referenced directory are appended to the default set of extra manifests.

- Add the `SiteConfig` CR to the `kustomization.yaml` file in the `generators` section, similar to the example shown in `out/argocd/example/siteconfig/kustomization.yaml`.
- Commit the `SiteConfig` CR and associated `kustomization.yaml` changes in your Git repository and push the changes.

  The ArgoCD pipeline detects the changes and begins the managed cluster deployment.

Additional resources

- Preparing the GitOps ZTP site configuration repository
- Configuring the hub cluster with ArgoCD
- Signalling ZTP cluster deployment completion with validator inform policies
- Creating the managed bare-metal host secrets
- BMC addressing

Monitoring managed cluster installation progress

The ArgoCD pipeline uses the `SiteConfig` CR to generate the cluster configuration CRs and syncs it with the hub cluster. You can monitor the progress of the synchronization in the ArgoCD dashboard.

Prerequisites

- You have installed the OpenShift CLI (`oc`).
- You have logged in to the hub cluster as a user with `cluster-admin` privileges.

Procedure

When the synchronization is complete, the installation generally proceeds as follows:

- The Assisted Service Operator installs OpenShift Container Platform on the cluster. You can monitor the progress of cluster installation from the RHACM dashboard or from the command line by running the following commands:
  - Export the cluster name:

```
$ export CLUSTER=<clusterName>
```

- Query the `AgentClusterInstall` CR for the managed cluster:

```
$ oc get agentclusterinstall -n $CLUSTER $CLUSTER -o
jsonpath='{.status.conditions[?(@.type=="Completed")]}' |
jq
```

- Get the installation events for the cluster:

```
$ curl -sk $(oc get agentclusterinstall -n $CLUSTER
$CLUSTER -o jsonpath='{.status.debugInfo.eventsURL}')  | jq
'.[-2,-1]'
```

Troubleshooting GitOps ZTP by validating the installation CRs

The ArgoCD pipeline uses the `SiteConfig` and `PolicyGenTemplate` custom resources (CRs) to generate the cluster configuration CRs and Red Hat Advanced Cluster Management (RHACM) policies. Use the following steps to troubleshoot issues that might occur during this process.

Prerequisites

- You have installed the OpenShift CLI (`oc`).
- You have logged in to the hub cluster as a user with `cluster-admin` privileges.

Procedure

- Check that the installation CRs were created by using the following command:

```
$ oc get AgentClusterInstall -n <cluster_name>
```

  If no object is returned, use the following steps to troubleshoot the ArgoCD pipeline flow from `SiteConfig` files to the installation CRs.

- Verify that the `ManagedCluster` CR was generated using the `SiteConfig` CR on the hub cluster:

```
$ oc get managedcluster
```

- If the `ManagedCluster` is missing, check if the `clusters` application failed to synchronize the files from the Git repository to the hub cluster:

```
$ oc describe -n openshift-gitops application clusters
```

- Check for the `Status.Conditions` field to view the error logs for the managed cluster. For example, setting an invalid value for `extraManifestPath:` in the `SiteConfig` CR raises the following error:

```
Status:
  Conditions:
    Last Transition Time:  2021-11-26T17:21:39Z
    Message:               rpc error: code = Unknown desc =
`kustomize build /tmp/https___git.com/ran-
sites/siteconfigs/ --enable-alpha-plugins` failed exit
status 1: 2021/11/26 17:21:40 Error could not create extra-
manifest ranSite1.extra-manifest3 stat extra-manifest3: no
such file or directory 2021/11/26 17:21:40 Error: could not
build the entire SiteConfig defined by /tmp/kust-plugin-
config-913473579: stat extra-manifest3: no such file or
directory Error: failure in plugin configured via
/tmp/kust-plugin-config-913473579; exit status 1: exit
status 1
    Type:  ComparisonError
```

- Check the `Status.Sync` field. If there are log errors, the `Status.Sync` field could indicate an `Unknown` error:

```
Status:
  Sync:
    Compared To:
      Destination:
        Namespace:  clusters-sub
        Server:     https://kubernetes.default.svc
      Source:
        Path:             sites-config
        Repo URL:         https://git.com/ran-
sites/siteconfigs/.git
        Target Revision:  master
    Status:               Unknown
```

Removing a managed cluster site from the ZTP pipeline

You can remove a managed site and the associated installation and configuration policy CRs from the ZTP pipeline.

Prerequisites

- You have installed the OpenShift CLI (`oc`).
- You have logged in to the hub cluster as a user with `cluster-admin` privileges.

Procedure

- Remove a site and the associated CRs by removing the associated `SiteConfig` and `PolicyGenTemplate` files from the `kustomization.yaml` file.

  When you run the ZTP pipeline again, the generated CRs are removed.

- Optional: If you want to permanently remove a site, you should also remove the `SiteConfig` and site-specific `PolicyGenTemplate` files from the Git repository.
- Optional: If you want to remove a site temporarily, for example when redeploying a site, you can leave the `SiteConfig` and site-specific `PolicyGenTemplate` CRs in the Git repository.

After removing the `SiteConfig` file from the Git repository, if the corresponding clusters get stuck in the detach process, check Red Hat Advanced Cluster Management (RHACM) on the hub cluster for information about cleaning up the detached cluster.

Additional resources

- For information about removing a cluster, see Removing a cluster from management.

Removing obsolete content from the ZTP pipeline

If a change to the `PolicyGenTemplate` configuration results in obsolete policies, for example, if you rename policies, use the following procedure to remove the obsolete policies.

Prerequisites

- You have installed the OpenShift CLI (`oc`).
- You have logged in to the hub cluster as a user with `cluster-admin` privileges.

Procedure

- Remove the affected `PolicyGenTemplate` files from the Git repository, commit and push to the remote repository.
- Wait for the changes to synchronize through the application and the affected policies to be removed from the hub cluster.
- Add the updated `PolicyGenTemplate` files back to the Git repository, and then commit and push to the remote repository.

Removing zero touch provisioning (ZTP) policies from the Git repository, and as a result also removing them from the hub cluster, does not affect the configuration of the managed cluster. The policy and CRs managed by that policy remains in place on the managed cluster.

- Optional: As an alternative, after making changes to `PolicyGenTemplate` CRs that result in obsolete policies, you can remove these policies from the hub cluster manually. You can delete policies from the RHACM console using the **Governance** tab or by running the following command:

```
$ oc delete policy -n <namespace> <policy_name>
```

Tearing down the ZTP pipeline

You can remove the ArgoCD pipeline and all generated ZTP artifacts.

Prerequisites

- You have installed the OpenShift CLI (`oc`).
- You have logged in to the hub cluster as a user with `cluster-admin` privileges.

Procedure

- Detach all clusters from Red Hat Advanced Cluster Management (RHACM) on the hub cluster.
- Delete the `kustomization.yaml` file in the `deployment` directory using the following command:

```
$ oc delete -k out/argocd/deployment
```

- Commit and push your changes to the site repository.