opentext™

OpenText™ Information Archive

Cloud Deployment Guide

Deploy and configure OpenText Information Archive in certified cloud environments.

EARCORE250400-ICD-EN-01

OpenText™ Information Archive
Cloud Deployment Guide

EARCORE250400-ICD-EN-01
Rev.: 2025-Sept-22

This documentation has been created for OpenText™ Information Archive CE 25.4.
It is also valid for subsequent software releases unless OpenText has made newer documentation available with the product, on an OpenText website, or by any other means.

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1

Tel: +1-519-888-7111
Toll Free Canada/USA: 1-800-499-6544 International: +800-4996-5440
Fax: +1-519-888-0677
Support: https://support.opentext.com
For more information, visit https://www.opentext.com

Table of Contents

Chapter 1

Building Docker images from the OpenText Information Archive distribution

If you want to deploy OpenText Information Archive to the cloud, but you do not want to use the prebuilt Docker images (available at `registry.opentext.com`), which are based on an internally built base image using Orcale Linux and Eclipse Temurin JRE 21, then you can build Docker images from the OpenText Information Archive distribution (`infoarchive.zip` and `infoarchive-support.zip`). For example, you might want to use Red Hat Enterprise Linux rather than Oracle Linux.

Note: If using the prebuilt Docker images is acceptable to you, then you do not need to perform the tasks in this chapter.

The steps described in this document were tested with GKE, GCR, CFCR, AWS EKS, and Azure AKS. You will have to adjust the steps for other environments as required.

1.1 Prerequisites

1. If you are working with GKE you will need to have a Google Cloud Platform (GCP) account and a project to access the associated Google Container Registry (GCR).
2. Make sure to enable Google Container Registry (GCR) services.
3. Make sure that your local docker command can push images to GCR.
4. If you are using your own docker registry, make sure that your local docker command can push images to your Docker registry.
5. Do the following to set up Docker for Desktop:
   a. Download, install, and configure Docker Desktop from the Docker website.
   b. Add the directory that contains the docker and docker-compose commands to your PATH system variable. For more information see the docker website.
   c. Make sure that the Docker Desktop service is running.

   For more information about setting up Docker for Desktop, see the Docker for Desktop documentation.
6. Download the OpenText Information Archive distribution (`infoarchive.zip` and `infoarchive-support.zip`).
7. Prepare a base image with Linux of choice with JRE21 installed and `JAVA_HOME` set correctly. If you are using a non-Alpine base image, you will have touse the OS-specific package manager in place of apk (Alpine package manager) in the `ias/Dockerfile` and `posgres/Dockerfile` files.

8. Download the following packages:

   `infoarchive-<CURRENT_VERSION>-n-k8s-docker-compose.zip`
   This file contains the Docker files and docker compose commands to build the Docker images from scratch using the downloaded OpenText Information Archive current distribution (`infoarchive.zip` and `infoarchive-support.zip`).

   `infoarchive-<CURRENT_VERSION>-m-n.tgz`
   The current OpenText Information Archive CE Helm chart is available at this location (https://registry.opentext.com/). You will have to download and extract it in folder where you will extract the above `ZIP` archive.

1.2 Previous Helm updates

The following updates were made as part of a previous release:

- `isIAUpgrade` key has been removed (as the upgrade is a single-step now).
- logging level for all components has been externalized and is controlled via following keys: `ias.logging.level`, `iawa.logging.level` and `mgmt.logging.level`. The default value is WARN. To help with some troubleshooting the threshold can be increased to DEBUG for example. Additionally, for the IA Web App, you are able to provide a list of additional appenders that may help to see more details in incoming or outgoing requests (use key: `iawa.logging.level.appenders`). Example of additional appenders: `reactor.netty: DEBUG` or `reactor.netty.http.client.HttpClientOperations: DEBUG`.
- IA Server's configuration now can include preload – see keys: `ias.preload.*` and OpenText Information Archive documentation for more details.
- IA Server and the IA Web App now support node taints. Both components have their respective toleration keys which can be used with node taints to allow deployment onto specific nodes in the cluster.
- IA Server exposes rules profile configuration. By default, it is disabled and can be enabled by setting `ias.rules.profile.enabled` key.
- By default, password encryption is disabled. By default, the key `security.passwordEncryption.enabled` is set to false. This works well when storing secrets in the Vault. If you are not using Vault, ensure you set password encryption to true and use encrypted secrets only.

1.3 Building Docker images for GKE and GCR

The following steps use the examples of GKE and GCR. For your own Docker registry, adjust the tags accordingly.

To build docker images for GKE and GCR:

1. Extract the `infoarchive-<CURRENT_VERSION>-n-k8s-docker-compose.zip` file to the `infoarchive-<CURRENT_VERSION>-n-k8s-docker-compose` directory.
2. In a command prompt, go to the directory `infoarchive-<CURRENT_VERSION>-n-k8s-docker-compose`.
3. Extract the `infoarchive.zip` and `infoarchive-support.zip` file into the `docker-compose/context` directory. You should end up with the `docker-compose/context/infoarchive` directory containing the extracted OpenText Information Archive distribution.
4. Extract the `docker-compose/context/infoarchive/first-time-setup/PSQL_Linux_*_IA.txz` file (the PostgreSQL distribution) into the `docker-compose/context/ infoarchive` directory. You should end up with the `docker-compose\context\infoarchive\psql` directory containing the extracted PostgreSQL files. You may need XZ Utils to handle this file type.
5. Delete the `docker-compose\context\infoarchive\first-time-setup\PSQL_Linux_*_IA.txz` file.

   Note: It is important to delete this file because it will speed up building the Docker images.

   Even though the PostgreSQL Docker image is built, it is not supported in the current version of OpenText Information Archive.
6. Now you are ready to create Docker images using this prepared OpenText Information Archive distribution. The next step is to build the following images that are, by default, tagged as follows:

   - `base:<CURRENT_VERSION>.n-m`: This does not need to be uploaded to the Docker registry.
   - `iawa:<CURRENT_VERSION>.n-m`
   - `ias:<CURRENT_VERSION>.n-m`
   - `mgmt:<CURRENT_VERSION>.n-m`: This Docker image is used in multiple pods used to run HELM job hooks that are necessary to bring the system up. Here are the examples of the jobs that use the Docker image:
     - Initialize PVCs
     - Install first time applications

     These Helm Job hook pods run the above jobs to completion and, when the Helm install or upgrade finishes the deployment, they are cleaned up.

Notes

- For the full names of these images, with the tag numbers in the names, see *OpenText Information Archive Release Notes* on support.opentext.com (https://support.opentext.com/).
- One key image to notice is `base:<CURRENT_VERSION>.n-m`. This image uses the base OS image with `JRE21` installed in it. In case you may want to use a different base OS image, you can replace the `FROM` tag with the Docker file `docker-compose/base/Dockerfile` to start from the desired base OS image with `JRE17` installed on it. The details of this are outside the scope of this document. The `base:<CURRENT_VERSION>.n-m` image does not need to be pushed to the Docker registry.
- If you use a non-Oracle-Linux-based image, you may have to adjust the instructions to install `postgresql15—client` in `ias/Dockerfile` to use OS-specific package manager commands.

7. Make sure to set the `IA_TAG` environment variable to the correct value.

   Tags are specified using the `IA_TAG` environment variable. For example:

   ```
   IA_TAG-<CURRENT_VERSION>.n-m
   ```

   Note: It is highly recommended to keep value of `n-m` in sync with the tags of the corresponding images on `registry.opentext.com` for traceability reasons.

   After the images are built, they need be tagged to get them ready to push to your container registry. For example, for `GKE`, the container registry is `gcr.io`, and the project is `iacustomer`). You must have a different project name (instead of `iacustomer`) on GCP. For example, the `ias:<CURRENT_VERSION>.n-m` image needs to be tagged as follows:

   ```
   gcr.io/iacustomer/infoarchive/ias:<CURRENT_VERSION>.n-m
   ```

8. Edit the `docker-compose/base/Dockerfile` file to use the correct base image of the OS (for example, Red Hat Enterprise Linux). The details of this step are outside the scope of this document.

9. Run the following commands to build the Docker images. You can directly edit and adjust the image name in the `docker-compose.yml` file so that the images can be pushed to your Docker registry. Alternatively, you can tag them later before pushing to the Docker registry.

   ```
   cd docker-compose
   docker-compose build base mgmt ias iawa
   ```

10. List the Docker images using the following command:

    ```
    docker images
    ```

11. Authenticate with your cloud provider and its container registry so that you can push images to the container registry. For GCP, you must use the `gcloud` command for this. You may also need Image Pull Secret, which you will have to specify when deploying the Helm chart.

12. Push the docker images to GCR.

    Note: If you are using a totally different Docker registry, you will have to adjust the tags accordingly. Please make a note of these tags and keep an eye on this in subsequent steps and Helm charts. Also – if you are leveraging OpenText Information Archive integration with Vault, and using CUBBYHOLE authentication method, you will additionally need to push Vault Agent image into the GCR and place it at the same level where you would place BusyBox image (for more information, see Using the OpenText Information Archive distribution).

```
cd docker-compose
.\pushtoGCP.bat
```

    Note: Make sure you can see the images in GCR using the Google Cloud Console web interface. For more information see the Google Cloud Platform website.

    Now you can deploy OpenText Information Archive to GKE using a Helm chart. For more information, see the rest of this document.

Chapter 2

Deploying OpenText Information Archive to the cloud

You must make sure that you go through the following prerequisites and preparation, regardless of the type of cloud deployment you choose.

2.1 Prerequisites

1. If you are working with Google Kubernetes Engine (GKE), you will need to have a Google Cloud Platform (GCP) account and a project to create your Kubernetes cluster in.
   a. Make sure to enable Google Container Registry (GCR) services.
   b. Make sure that your local docker command can push images to GCR.
2. If you are working with Azure Kubernetes Service (AKS), you will need to have an Azure account and a resource group to create your Kubernetes cluster in.
   a. Make sure to enable Container Registry services.
   b. Make sure that your local docker command can push images to Azure Container Registry.
3. If you are working with EKS, you will need to have an AWS account to create your Kubernetes cluster in.
   a. Make sure to enable Elastic Container Registry services.
   b. Make sure that your local docker command can push images to Elastic Container Registry.
4. If you are using your own docker registry, make sure that your local docker command can push images to your Docker registry and also any required image pull secrets.
5. Do the following to set up Docker Desktop:
   a. Download, install, and configure Docker Desktop(https://www.docker.com/products/docker-desktop) (https://www.docker.com/products/docker-desktop).
   b. Add the directory that contains the docker(https://docs.docker.com/engine/reference/commandline/docker/) (https://docs.docker.com/engine/reference/commandline/docker/) and docker-compose (https:// docs.docker.com/compose/) (https://docs.docker.com/compose/) commands to your PATH system variable.
   c. Make sure that the Docker Desktop service is running.

6. Download and install Helm 3.x. Make sure to add the Helm binary helm to your PATH system variable.
7. If you want to use the prebuilt Docker images, then download the following images from `registry.opentext.com`:

   - `otia-ias:<CURRENT_VERSION>.n-m`
   - `otia-iawa:<CURRENT_VERSION>.n-m`
   - `otia-mgmt:<CURRENT_VERSION>.n-m`

   Make sure to pull these images to the local registry. The `otia-` prefix is due to the naming conventions of `registry.opentext.com` and is not required.

   Note: For the full names of these images, with the tag numbers in the names, see *OpenText Information Archive Release Notes* on support.opentext.com (https://support.opentext.com/).
8. Download the following package:

   `infoarchive-<CURRENT_VERSION>-n-k8s-helm-support.zip`
   Contains the utilities and configuration files.

   `infoarchive-<CURRENT_VERSION>-n.tgz`
   The current OpenText Information Archive CE Helm chart is available at this location (https://registry.opentext.com/). You will have to download and extract it in folder where you will extract the above `ZIP` archive.

2.2 Preparation

2.2.1 Deploying OTDS

OpenText Directory Services (OTDS) deployment is not covered in this guide. The OpenText Information Archive Helm package now contains an artifact that needs to be used during OTDS deployment, namely OTDS bootstrapping YAML template. Bootstrapping template (`otds-ia-bootstrap-config.yml`) is populated when you run the password generation utility. For more information, see Generating and encrypting passwords.

Once password generation is successfully run, update the YAML bootstrapping template with the following:

- Set the `resources.secretKey` parameter. This is similar to the `cryptKey` parameter from the OTDS configuration. This can be any string, 16 characters in length that has been Base64–encoded. The following is an example of how to Base64 encode that string:

  ```
  $ echo -n "123456789abcdefg"|base64 -w 0&& echo
  MTIzNDU2Nzg5YWJjZGVmZw==
  ```

  The following is an example of how this should appear in the YAML template after editing the `secretKey` value:

```
resources:
  - resourceName: infoarchive
    description: "The resource representing InfoArchive Product in OTDS."
    displayName: "InfoArchive Resource"
    secretKey: !!binary |
        YWJjZGVmZzEyMzQ1Njc4OQ==
```

- Set the `redirectURLs` for the `infoarchive.iawa` `oauthClient` parameter. Determine the URL for the IA Web App after OpenText Information Archive has been deployed and set the `redirectURL` parameter, accordingly. Since this value is a list, set the first entry to the IA Web App public URL and leave the other entry to be `http://localhost:8080/`:

```
oauthClients:
  - name: infoarchive.gateway
...
  - name: infoarchive.iawa
    # Optionally prespecify client secret
    secret: "xxxxx"
    description: "InfoArchive Web Application (IAWA) OAuth2 Client"
    confidential: true
    accessTokenLifeTime: 1800
    allowRefreshToken: true
    useSessionRefreshTokenLifeTime: true
    refreshTokenLifeTime: 0
    redirectURLs:
      - "https://iawa.acme.infoarchive.ot.net/"
      - "http://localhost:8080/"
    allowedScopes:
....
```

- Save the YAML bootstrap template and copy it to the `config` location for OTDS chart under `<OTDS root>/otds/charts/otdsws/config`, as illustrated below:

```
otds
└── charts
    ├── otdsws
    │   └── config
    │       └── otds-ia-bootstrap-config.yml
    ├── Chart.yaml
    └── values.yaml
```

  Delete any other file (for example, `config.yml`) from the `otds/charts/otdsws/config` location. When configured for bootstrapping, OTDS picks up the first YAML file from this location, so you need to ensure that the bootstrapping template is the only file in there.

- Lastly edit main values file for OTDS (or your override file) and ensure that `otdsws.enableBootstrapConfig`is set to true:

```
##############################################################################
# Directory Services server subchart:
# Don't remove the YAML anchors - only change the parameters behind!
# (they are used for cross-referencing to avoid repetitive settings)
##############################################################################
otdsws:
  ## Pod Security Context'
 …
  ## enableBootstrapConfig enables the use of the config.yml file in the
  ## otdsws chart directory to apply a specific set of configuration options
  ## on the initial run when the DB is populated.
  ## A sample config.yml file is in the otdsws chart directory
  enableBootstrapConfig: true
...
```

Follow the rest of the OTDS documentation to complete its deployment.

After OTDS is deployed, login to the OTDS interface and confirm following:

- Under Partitions, there are `infoarchive` and `infoarchive.bootstrap` partitions.
- Under Resources, there is an OpenText Information Archive resource.
- Under OAuth Clients, OpenText Information Archive related OAuth Clients were created.

If you do not see any or all of these artifacts, see OTDS bootstrap template troubleshooting.

2.2.2 OTDS bootstrap template troubleshooting

If any of the system's related artifacts were not created, examine the log of OTDS POD. First, look for the message: "Migrating schema `otds` with repeatable migration `BootstrapFromFiles`":

```
...
[main] INFO org.flywaydb.core.internal.command.DbMigrate - Current version of schema
"otds": 1
[main] INFO org.flywaydb.core.internal.command.DbMigrate - Migrating schema "otds"
with repeatable migration "BootstrapFromFiles"
2024-10-08 16:52:43.085|OTDS        |INFO  |[main]|R__BootstrapFromFiles||
Bootstrapping config from file: /opt/config/bootstrap/otds-ia-bootstrap-config.yml...
...
2024-10-08 16:52:44.356|OTDS-AUDIT  |INFO  ||0|0|Configuration Service|Information|
65,System Attribute Set||""||""|"The system attribute
otdsautoconfig.deployment.infoarchive-init was set to 24.4"
2024-10-08 16:52:44.452|OTDS-PROVN  |INFO  ||515344112227109049|515344112227109049|
Recycle Bin|Success Provenance|90,Scheduled Cleanup||""||""|"Started"
2024-10-08 16:52:44.753|OTDS-PROVN  |INFO  ||7056156529330747472|7056156529330747472|
Directory Server|Success Provenance|5,Object Create||""||""|"name=infoarchive;
entryDN=oTResource=23b75fe9-
feee-4356-817c-3c8efc263ecf,dc=identity,dc=opentext,dc=net; objectClass=oTResource"
2024-10-08 16:52:44.754|OTDS-AUDIT  |INFO  ||0|0|Configuration Service|Information|
31,Resource Create||""||""|"Resource 'infoarchive' created with ID '23b75fe9-
feee-4356-817c-3c8efc263ecf'"
2024-10-08 16:52:44.874|OTDS-PROVN  |INFO  ||6020400807195316786|6020400807195316786|
Directory Server|Success Provenance|5,Object Create||""||""|"name=23b75fe9-
feee-4356-817c-3c8efc263ecf; entryDN=oTPerson=8bc6db22-
e4ce-4a61-9cd6-363eb40236da,orgunit=Root,partition=otds.admin,dc=identity,dc=opentext,d
c=net; objectClass=oTPerson"
2024-10-08 16:52:45.057|OTDS-AUDIT  |INFO  ||0|0|UserGroup Service|Information|1,User
Create||""||""|"The user: 23b75fe9-feee-4356-817c-3c8efc263ecf was created in the
partition: otds.admin"
...
```

If the bootstrap template was placed in the correct folder, it is picked up by OTDS during deployment. It is then mounted to the POD under in the following `/opt/config` folder, as illustrated in the example above. The OpenText Information Archive bootstrapping template (`otds-ia-bootstrap-config.yml`) was correctly placed. If you do not see the message as above - or you see a different template name, such as `config.yml`, ensure the original OTDS's bootstrap template was removed, as outlined in this guide, and the only YAML template in that folder is the system's bootstrap template (`otds-ia-bootstrap-config.yml`).

The name of the bootstrap file is not important. What is important is that bootstrap file has the correct extension, which is `.yml`.

If you do not see a reference in OTDS's log to the OpenText Information Archive bootstrap template, complete the configuration steps again. Un-install OTDS and reinstall it after ensuring all configuration steps were followed correctly.

Additionally, after referencing the bootstrapping template, there should be plenty of additional logging related to OTDS creating various system resources, as can be seen from the log snippet sample above.

Note: During installation, OTDS persists some metadata related to the bootstrap file (for example, it records its checksum). If you are re-installing OTDS without dropping its database schema, it may run the checksum on the bootstrap file again. If it is unchanged, it may not use the bootstrap file during subsequent deployment(s). To work around it, either drop its database and then create it from scratch again before deploying, or make some changes to the bootstrap file's contents so that OTDS calculates it (for example, add some comments to the file, re-generate the secretKey, *etc.*).

2.2.3 Collecting information about OTDS

The Helm chart does not deploy OTDS. The current version of OpenText Information Archive was tested with the current version of OTDS. Be sure to install the latest version of OTDS for the latest version of OpenText Information Archive to use. Also note that the latest version of OTDS uses its own PostgreSQL database. You must have OTDS deployed separately, with access to the following information:

- The URL of OTDS, including the following:
  - The protocol: `https` (recommended) or `http` (discouraged).
  - The host: the FQDN or IP address that is reachable from inside the Kubernetes cluster that OpenText Information Archive is deployed to.
  - The port: for example, `443` (the default port for the `https` protocol).
- The OTDS administrator username.
- The OTDS administrator password.
- This is only applicable if you are using HTTPS and certificates that were **not** signed by valid/trusted CA. A valid TLS/SSL certificate for OTDS when the `https` protocol is in use. Later, this will be imported into the truststore used by the OpenText Information Archive components that connect to OTDS using the `https` protocol.

2.2.4 Preparing OpenText Information Archive

The Helm chart for OpenText Information Archive allows you to configure the following parameters based on your license and contract:

- `transactionOption`: Configures the resource allocation and limits for CPU and memory. It also configures the number of instances of IA Web App and IA Server. This can take the following values:
  - `m1`
  - `m2`
  - `m3`
  - `m4`
  - `ms` (flexible)

  For more information about `m1`, `m2`, `m3`, `m4`, and `ms`, see Transaction options.
- `AS Data Root PVC size`: For m1 through m4 transaction options, set `resources.ias.pvcSizes.iasDataRootPvc` or for ms transaction, `ms.ias.pvcSizes.iasDataRootPvc`, set this to required value. The default value is 200 GB. Adjust, as required, based on the selected SKU. The `storageInTB` key has been removed.

You can configure these parameters in your customer-specific file. For example:

`helm/customers/iacustomer/overrides-general.yaml`

You can use a different customer name than `iacustomer`. In that case, copy the contents of the above folder to your customer folder and proceed to use that folder.

The Helm chart also allows you to configure how to run components of OpenText Information Archive on different nodes or node pools, as applicable to your platform. You can configure the `nodeSelectors` and/or tolerations in the customer-specific `overrides-general.yaml` file. The `nodeSelectors` and tolerations give you a powerful way to organize node allocation.

You must configure the cluster and the associated storage consistent with the settings above.

For more information about the parameters above and the distinct node selector keys that are available, see Customer values file.

2.3 Configuring the cluster

The node pools designated for the IA Server pods may require an adjustment to the `vm.max_map_count` system property. On some Kubernetes clusters, the default value for that property is `65530`, which is too restrictive and can cause exceptions when using Lucene indexes. It should be increased to at least `262144`. It can be done by adding the following lines to the node pool configuration YAML file:

```
linuxConfig:
   sysctl:
     vm.max_map_count: 262144
```

For more information, see the chapter for your cloud provider:

- Deploying OpenText Information Archive in a private cloud
- Deploying OpenText Information Archive on Microsoft Azure
- Deploying OpenText Information Archive on GCP
- Deploying OpenText Information Archive on AWS
- Deploying OpenText Information Archive on OpenShift
- Deploying OpenText Information Archive on CFCR

2.4 Configuring ReadWriteMany storage

ReadWriteMany storage is required for several `PersistentVolumeClaims`. The NFS-based `PersistentVolumes` is one such provider. Other alternatives are also available depending on the platform. For example:

- On Azure, there is Azure Files
- On AWS, there is Elastic File Storage

You may have predefined Kubernetes storage classes to support the ReadWriteMany storage in your Kubernetes cluster. Use of cluster administrator-defined classes is highly recommended.

For more information, see the chapter for your cloud provider:

- Deploying OpenText Information Archive in a private cloud
- Deploying OpenText Information Archive on Microsoft Azure
- Deploying OpenText Information Archive on GCP
- Deploying OpenText Information Archive on AWS
- Deploying OpenText Information Archive on OpenShift
- Deploying OpenText Information Archive on CFCR

2.5 Extracting the packages

Extract the contents of downloaded packages into the `helm` directory. Download the current version of the OpenText Information Archive CE Helm Chart `infoarchive-<CURRENT_VERSION>-m-n.tgz` from the Registry and extract it in the above folder next to the `customers`, `platforms`, and `password-generator` folders. The chart folder name is `infoarchive`.

To download the helm chart first ensure opentext registry was added to helm. For example:

```
helm repo add opentext https://registry.opentext.com/helm --username <myuser
emailaddress> --password <mypassword>
```

Once the OpenText Registry is added to Helm, download the Helm chart using following `pull` command:

```
helm pull opentext/infoarchive --version <CURRENT_VERSION>
```

2.6 Pushing Docker images to the Docker registry

2.6.1 Pulling Docker images into the local Docker environment

To pull Docker images into the local Docker environment:

1. If you want to use the pre-built Docker images available on `registry.opentext.com`, pull them into your local Docker environment using the following command:

   ```
   docker pull registry.opentext.com/otia-imagename:tag
   ```

2. Note the tag of each pulled image after running the previous command. The `otia-` prefix is due to the naming conventions of `registry.opentext.com` and is not required when you push the images to your Docker registry.

2.6.2 Tagging the images

To tag the images:

1. Determine the URL of the Docker registry associated with your platform.

2. Tag the images using the following command:

   ```
   docker tag <pulled image tag> <tag for pushing to your Docker registry>
   ```

3. Ensure the tags look like the following for all the images:

   ```
   .../infoarchive/mgmt:<CURRENT_VERSION>.n-m
   .../infoarchive/ias:<CURRENT_VERSION>.n-m
   .../infoarchive/iawa:<CURRENT_VERSION>.n-m
   ```

Note: For the full names of these images, with the full build numbers in the names, see *OpenText Information Archive Release Notes* on support.opentext.com (https://support.opentext.com/).

Your prefix will depend on the Docker container registry associated with your platform. For example, it may have the host name of the Docker container registry and project name (GCP case) or resource group name for the Azure case. The prefix `/infoarchive/` is based on settings in your platform values file. For more information, see Platform values files.

4. You will also have to pull the following image from Docker hub and tag them for pushing to your Docker registry. It is used by init containers.

   - `busybox/latest`

5. If leveraging integration with Vault, and using vault authentication mode CUBBYHOLE, you will additionally need Vault Agent image. Place that image in artifactory in a structure that is similar to how other images, such as BusyBox, are placed (where BusyBox or vault would be placed at the top level). It should be at the same level as BusyBox, OTDS, or OpenText Information Archive prefix level.

2.6.3 Pushing tagged images to the Docker registry

Push the tagged images to the Docker registry.

If using integration with Vault and Vault authentication method CUBBYHOLE, Vault agent container should also be pushed to the registry.

2.7 Preparing the platform values file

You must configure the following for the platform you are using:

- The location of the Docker registry
- The paths inside the Docker registry
- StorageClass names for RWM PVCs.

For more information about the values files for each platform, see Platform values files. You can opt into the **Host:** header injection support in OpenText Information Archive.

2.8 Configuring PostgreSQL

The current version of OpenText Information Archive requires a PostgreSQL database. For production deployment, we recommend using external instances of PostgreSQL but, for the demo deployments, an in-cluster PostgreSQL instance can be used. Currently we are using PostgreSQL 16.8.

2.8.1 Configuring PostgreSQL externally

For the current version of OpenText Information Archive, it is strongly recommended to use an external PostgreSQL database in production. In the Helm chart for the current version, you have the option to use an externally configured PostgreSQL database.

When using an externally deployed PostgreSQL database, you must use the PostgreSQL passwords that you generated and encrypted when you installed PostgreSQL. For more information on how to install the PostgreSQL database, see the PostgreSQL website. If the PosgreSQL administrator created the passwords and they were given to you, you will have to enter them while generating the password. The pre-specified passwords will only be encrypted by the utility below.

Before configuring PostgreSQL, make sure that you know the following information about externally deployed PostgreSQL components:

- FQDN for the PostgreSQL components for the following:
  - PosgreSQL instance

    Note: Although FQDN is still a recommended, an alternative way of connecting to your PostgreSQL instance is via an IP address. The system additionally supports adding an IP address instead of host name. There is a specific key (`ip`) at the same level as host key and, if set, the system attempts to connect to your database using the IP address. Providing both fields will additionally insert a new entry into the PODs hosts file. You can also set only an IP key if the host name is not resolvable via DNS.
- Port
  - Port on which the PosgreSQL instance has been configured to run
- The super user name. Although we refer to it as a super user, it does not have to be an actual super user account. It has be to an account with the following privileges:
- – `CREATEDB`
  - `CREATEROLE`
  - `LOGIN`

- Superuser password that you generated and encrypted earlier (the password for the `admin` user above). Be sure to use these passwords in the following procedure.
- Database owner name. Database owner has be an account with the following privileges:
  - `CREATEDB`
  - `LOGIN`
- Database owner password that you generated and encrypted earlier. Be sure to use these passwords in the following procedure.

If you are using the SSL protocol, you will have to get the TLS/SSL certificates for the external PostgreSQL server as well as the client and client's certificate private key. For details on how to configure PostgreSQL over SSL, see the PostgreSQL website.

OpenText Information Archive uses two PostgreSQL nodes, each running as separate instances. One node is dedicated to system data while the other is dedicated to structured data. Additionally, each instance requires two users/roles configured for OpenText Information Archive – one user/role for a system user (called internally as `superuser`, but it does not have to be a `superuser` account) and one for a database owner user.

Because the IA Server will be connecting as a client to the PostgreSQL database, it will need these artifacts to successfully make the TLS/SSL connection:

- `sslrootcert`: PostgreSQL server's root certificate used by the client to certify the server's identity.
- `sslcert`: Client's certificate sent to PostgreSQL.
- `sslkey`: Private key matching the client's certificate.
- `sslpassword`: If the private key is password-protected, this is the password to the key.

```
postgres: deployment:
type: external image:
tag: <docker image tag> system:
external:
host: postgresql-system.iacustomer.com port: 5432
superuser:
username: ia-admin
# This is used only for external case as database owner # This MUST be different
from superuser
dbowner:
username: ia-user sslcert: ia-user.crt
sslkey: ia-user.key.pk8
# Postgres SSL configuration sslConnection:
enabled: true
sslrootcert: system-root.crt sslcert: ia-admin.crt sslkey: ia-admin.key.pk8
sslpassword:
sslmode: structuredData:
external:
host: postgresql-structured-data.iacustomer.com port: 5432
superuser:
username: sd-admin
# This is used only for external case as database owner # This MUST be different
```

```
from superuser
dbowner:
username: sd-user sslcert: sd-user.crt sslkey: sd-user.key.pk8
# Postgres SSL configuration sslConnection:
enabled: true sslrootcert: sd-root.crt sslcert: sd-admin.crt sslkey: sd-
admin.key.pk8 sslpassword:
sslmode:
```

Note that `sslrootcert`, as it is a certificate for the PostgreSQL server, is global, meaning only one instance is defined. However, `sslcert`, `sslkey` and `sslpassword` are specific to user/roles required by the OpenText Information Archive Server, which acts as a client to the PostgreSQL server. As such, these artifacts are specific to each user/role instance, meaning this set of artifacts is per each system and per each database owner user.

- enabled – set to true if leveraging TLS/SSL

Just like for the truststore, we will need to generate base64–encoded versions of the root certificate, client certificate, and the private key if using TLS/SSL to connect to PostgreSQL. Make sure to use the base64 utility flag `–w 0` so that the generated `.base64` file has a single line. Create a base64–encoded form of the certificate and the key file with the extension `.base64`. For example:

- `root.crt.base64`

To use the externally deployed PostgreSQL database, set the following configuration. Make sure to also correctly set the FQDN (`postgres.system.external.host` and `postgres.structuredData.external.host`) port (`postgres.system.port` and `postgres.structuredData.port`), admin and database users (`postgres.system.superuser.username`, `postgres.system.dbowner.username`, `postgres.structuredData.superuser.username` and `postgres.structuredData.dbowner.username`), as well as the required values for TLS/SSL connectivity (all keys for `postgres.system.sslConnection.*`, `postgres.structuredData.sslConnection.*` and respective `sslcert` and `sslkey` for `postgres.system.dbowner` and `postgres.structuredData.dbowner`), as described below:

```
postgres:
  deployment:
    type: external
  system:
    superuser:
      username: ia-admin
    dbowner:
      username: ia-user
      sslcert: ia-user.cert.crt
      sslkey: ia-user.key.pk8
    external:
      host: <FQDN>
      ip: [optional ip address if no FQDN available]
    port: <port>
    sslConnection:
      enabled: true
      certAuth: true
      sslrootcert: ca.crt
      sslcert: ia-admin.cert.crt
      sslkey: ia-admin.key.pk8
      sslmode:
  structuredData:
    superuser:
      username: sd-admin
    dbowner:
```

```
    username: sd-user
    sslcert: sd-user.cert.crt
    sslkey: sd-user.key.pk8
  external:
    host: <FQDN>
    ip: [optional ip address if no FQDN available]
  port: <port>
  sslConnection:
    enabled: true
    certAuth: true
    sslrootcert: sd.ca.crt
    sslcert: sd-admin.cert.crt
    sslkey: sd-admin.key.pk8
    sslmode:
```

2.8.2 Rotating CA certificates for IA Server to communicate with PostgreSQL

The following two options are available to update or refresh PostgreSQL certificates for IA Servers:

- Run the Helm `upgrade` command. The command should be identical to Helm install/upgrade command that deployed the latest version of the release. This option is easier as long as you have access to the script that was used to deploy/upgrade your latest release.

  Tip: If you do not have access to last-used Helm install/upgrade script, you may want to follow the other option, as trying to re-create the command from scratch on a previously deployed release may alter the configuration and break the release. For such scenarios - it is best to patch the release which is described in the next option.

  Edit the relevant ConfigMap `k8s` resources.

Caution

Since both options modify resources, do not proceed without taking a full backup of the system. In the event that something goes wrong. you can restore the system from the backup.

The following provides more information about each option:

Running the Helm `upgrade` command

To run Helm upgrade command:

1. Provide a path to the updated certificates for all external keys related to PostgreSQL running over TLS. The following `external` keys are related to PostgreSQL running over TLS. The keys that are set/used depend on type of PostgreSQL TLS configuration:

```
Root CA:
- external.system.sslrootcert
IA Server's cert + key for the system role:
- external.system.sslcert
```

```
- external.system.sslkey
IA Server's cert + key for the database owner role:
- external.system.dbowner.sslcert
- external.system.dbowner.sslkey

The same for the structured data database:
Root CA:
- external.structuredData.sslrootcert
IA Server's cert + key for the system role:
- external.structuredData.sslcert
- external.structuredData.sslkey
IA Server's cert + key for the database owner role:
- external.structuredData.dbowner.sslcert
- external.structuredData.dbowner.sslkey
```

2. The last used Helm install/upgrade command should already contain the configured settings for the above keys. Replace their values with the paths to the updated certificates and keys, if applicable.

3. Once the script's external keys for PostgreSQL over TLS have been updated, run the Helm `upgrade` command. The command updates all resources accordingly. Depending on the `k8s` environment, relevant pods may or may not be restarted. Check if the pods were restarted by running the following command (ensure you are in the correct OpenText Information Archive namespace):

```
> kubectl get pod
```

4. The `upgrade` command lists all running pods. Check the Age column of the listing. If the pods have been restarted, the age should specify the time unit in seconds/minutes (for example, 30s or 3m, which indicates 30 seconds or 3 minutest, respectively).

   If the age indicates the pods restarted, no further action is required, as the pods have updated versions of the certificates/keys. However, if the age indicates that pods were not restarted (for example, age is 30h or 30d, which indicates 30 hours or 30 days, respectively), you must manually scale the pods.

   Only the IA Server's pods need to be restarted. Use the `kubectl scale` command to scale the IA Server's `statefulset` resources down and then scale them up. Ensure all IA Server resources were properly restarted after scaling up.

Editing ConfigMap `k8s` resources

This option only updates specific resources that need to be updated and does not change the rest of the resources. While this option can be seen as a safer alternative, it includes base64 encoding of the certificates and requires you to directly edit deployed resources, so it is also more complicated.

1. Update the following `configmap` resources:

```
- configmap-postgres-system-tls
- configmap-postgres-structured-data-tls
```

2. To ensure you have these `configmap` resources, once you are in the correct namespace, run following command:

```
> kubectl get configmap
NAME                                    DATA   AGE
configmap-first-time-setup              1      3m2s
configmap-http-proxies                  0      3m2s
configmap-ias-config-iaserver           1      3m2s
configmap-ias-config-iaserver-otds      2      3m2s
configmap-iashell-config-iashell        5      3m2s
configmap-iawa-config-iawebapp          3      3m2s
configmap-iawa-config-iawebapp-otds     1      3m2s
configmap-otds-ia-init-config           3      3m2s
configmap-password-encryption           3      3m2s
configmap-postgres-structured-data-tls  5      3m2s
configmap-postgres-system-tls           5      3m2s
kube-root-ca.crt                        1      473d
```

In the example above, notice that you can see both resources. Each contains five pieces of binary (encoded) data, three certificates, and two keys.

The deployed `configmap` has following structure:

```
#
apiVersion: v1
binaryData:
      <name of system role certificate>:
LSOtLS1CRUdJ...SUNBVEUtLSOtLQ==
    <name of system role key>:
MIIEvQIBA...ZjWO8qkJc=
    <name of database owner role certificate>:
LZOwLS1CRUdT...KOLEdfuiosd==
    <name of database owner role key>:
UIdoidEW...JDoekwjOKjw==
    <name of root CA certificate>:
MKodwoKoK...QRDeifowE==
kind: ConfigMap
metadata:
  annotations:
    meta.helm.sh/release-name: infoarchive
    meta.helm.sh/release-namespace: infoarchive
  creationTimestamp: "2024-06-17T21:05:03Z"
  ...
```

The number of pieces of encoded data per `configmap` depends on how PostgreSQL was configured with TLS. It ranges from a single binary piece to five binary data pieces. After replacing all existing binary pieces, you should have the same number of binary pieces that were originally there.

There are two ConfigMaps, one with certificates/keys for system data and the other with the certificate/key for the structured data database.

3. The ConfigMap contains five base64–encoded entities, corresponding to three certificates and two keys. Replace each encoded value with the new base64–encoded certificate and/or key.

4. Update the ConfigMaps related to PostgreSQL TLS and then restart the IA Server pod(s).

5. Edit the `configmap-postgres-system-tls` and `configmap-postgres-structured-data-tls` (among others). Data within those ConfigMaps are in binary format and then base64–encoded. Depending on how your TLS for PostgreSQL has been configured, you may have between one and three certificates and, potentially, two private keys encoded in each of the `configmap` resources. Prepare your new certificates and keys. For each certificate/key, get the binary value and then base64 encode it. For example,

suppose your root certificate is named `root.crt`, and you have a superuser certificate: `super.crt` with its corresponding key: `super.key.pk8`. You would encode the certificate as follows:

```
>cat root.crt | base64 | tr -d '\n'
>cat super.crt | base64 | tr -d '\n'
>cat super.key.pk8 | base64 | tr -d '\n'
```

Suppose your `configmap` looks like the following with the encoded `root.crt`:

```
...
root.crt: LSOtLS1CRUd...VJUSUZJQOFURSOtLSOt
kind: ConfigMap
metadata:
  annotations:
  ....
```

The structure is as follows: name of the certificate, followed by the colon, followed by the space, followed by the encoded certificate. Only update section for the encoded certificate. Do not change the name of the certificate or colon or the (white) space. Once you replaced section for the root CA certificate, replace other sections, as applicable. Ensure you match each certificate with its replacement value. Do not mix certificates, as that will lead to TLS issues and will break communication. Replace all configured certificates and keys. Once all applicable certificates and keys were updated, save the `configmap` resource. As long as there are no formatting issues, your changes will save correctly. If it does not save, fix the formatting issues and then save again.

6. Update another `configmap: configmap-postgres-structured-data-tls` and save your changes.
7. Once both resources are updated successfully, scale down and then scale up each IA Server to ensure all the certificates are reloaded.

2.9 Integration with Vault

OpenText Information Archive supports integration with HashiCorp Vault. Each OpenText Information Archive component: IA Server, IA Web App, IA Shell can be integrated with Vault. That means that all secrets/passwords required for a specific component's initialization, currently found in configuration files (either YML or properties), can be moved to Vault. This is an opt-in process which, by default, is not enabled. When enabled, OpenText Information Archive components will reach out to a configured Vault instance to gather all secrets/passwords and will silently merge those with the rest of the configuration properties.

The integration between OpenText Information Archive and Vault can be broken into four separate phases:

- Generation of Vault JSON templates containing secrets/passwords per component
- Configuration of Vault instance

- Populating Vault with secrets/passwords
- Enabling Vault integration in Helm templates and providing necessary configuration details for Vault

2.9.1 Authentication with Vault

OpenText Information Archive supports multiple ways of authenticating with Vault:

- TOKEN: This is the simplest way of authentication with the Vault. All you need to provide is a valid token generated by the role with permissions for the locations where all secrets are stored. Once provided to OpenText Information Archive configuration, each component is able to authenticate/login (to Vault) and fetch all necessary secrets/passwords. However, the tokens have expiry dates and, additionally, this method of authentication does expose Vault's token in plain text view in the configuration file, so token itself can be compromised.
- APPROLE: This requires provisioning a special role in Vault, with access to all required secrets, and then providing role ID and secret ID of such role, to OpenText Information Archive configuration. Each component, at run time, uses these credentials to login to the Vault and obtain its own token. This way, you do not have to worry about a token's expiry but, since role's (both) credentials are stored in configuration files, they can be compromised.
- CUBBYHOLE* (actually it is a combination of AppRole and Cubbyhole auth methods): This is the most secure way of authenticating to the Vault but also the most complex. It requires two roles provisioned with the Vault: one role (main role) with access to all required secrets and the other role (helper role) with no access to OpenText Information Archive specific secrets, but with access to another location inside the Vault where secret ID for the main role has been stored. In OpenText Information Archive configuration, you provide role ID + secret ID for the helper role, but only a role ID for the main role, as secret ID for the main role will be retrieved from the Vault dynamically at run time. Before PODs start, `init-container`, dynamically, using Vault's agent, calls out to the Vault instance, using helper's role, to pre-fetch secret ID for the main role, and then use the main role's credentials to login to Vault, prepare wrapped token, and lastly update POD's configuration with wrapped token. It is dynamically injected to the Vault's configuration for OpenText Information Archive, for each component. When this pre-fetching is done, POD starts and exchanges wrapped token, which is a single use token, for an actual token, and then fetches all required secrets from the Vault.
- Due to overall complexity of CUBBYHOLE flow, it is recommended to first test deployment using TOKEN, then APPROLE authentications. Only after ensuring that both these authentication methods work, try testing CUBBYHOLE authentication, as this greatly simplifies any potential troubleshooting.

2.9.2 Configuration of Vault

Full configuration of Vault is beyond the scope of this document. This section focuses on what configuration is expected from a given Vault instance to effectively enable integration with OpenText Information Archive.

- Secrets Engine: OpenText Information Archive supports KV Secrets Engine (both V1 and V2), which needs to be enabled on a Vault instance. When enabling KV Secrets Engine, make a note of the path. By default, it is `kv` but could be anything, such as `secret`, *etc*. This value will be needed later when configuring Helm and corresponds to the `vault.kv.backend` key. For some of the examples that follow, assume the path has been set as `secret`.
- At this point, you have a running Vault instance with enabled KV Secrets Engine. You will populate Vault with actual values later when you have generated JSON templates.
- Ensure `approle` authentication is enabled. Using the Vault interface, go to **Access** tab and check if the `approle` authentication method has been enabled. If it is, make a note of the path component, as you will need it later when preparing Helm configuration for Vault. The path corresponds to `vault.appRolePath` key. If it is not enabled, enable AppRole authentication providing the path and making note of the value.
- Later, you will create a policy with access to all paths for your secrets, and then use that policy to create a main role. Once the role is created, keep track of the role ID and its secret ID values.
- Optionally, if using `CUBBYHOLE` authentication mode, create an additional policy with access to another area of the KV Secrets Engine. Store the main role's secret ID in that location. Then you will create a (helper) role with that policy.

Note: IA Shell is referred to interchangeably as shell or cli. They both mean the same component representing command line interface or shell integration of OpenText Information Archive.

2.9.3 Generation of Vault JSON templates

You will run a password generation utility that generates Vault JSON templates (if configured as such). Refer to that section for more information, but among the artifacts that it generates, you should find four JSON templates: `server`, `iawa`, and `cli JSON` template. The first three (`server`, `iawa`, `cli JSON`) come in two flavours: encrypted or plain. Encrypted is the default and contains no postfix in the name: `server.json`, `iawa.json`, and `cli.json` versions have ready-to-ingest Vault JSON templates and should be used if you enable password encryption. This should be the default for a production environment. These templates come also with `_plain` postfix (`server_plain.json`, `iawa_plain.json`, and `cli_plain.json`) that contain un-encrypted versions of the passwords. These are just for your reference but, alternatively, can be used to populate Vault if password encryption has been disabled at the Helm-level.

2.9.4 Creating Vault's policy and a role

Before storing secrets in the Vault, you need to create a dedicated access policy and then create a role with that policy. This ensures that you will assign appropriate access to the secrets.

Using Vault's interface, click the **Policies** link at the top of the screen. On the **ACL Policies** screen, select the **Create ACL policy** action. Enter a name for your policy (for example, `infoarchive`) and, in the **Policy data** section, define the policy. You can set any policy rules you like but, essentially, you will want to allow it read access to the secrets' location(s). The actual policy, however, can be tailored to your specific needs. For example, if you are storing secrets in the **iacustomer** location (refer to the next section for more information), along with the KV Secrets Engine path (for example, `secret`), the full path would be: `secret/data/iacustomer`. For the subcomponent path `data`, note the particular behavior of paths to secrets when using KV Secrets Engine). You need to allow read access to that location. The following is an example of such a policy:

```
# Allow read access to iacustomer/*
path "secret/data/iacustomer/*" {
    capabilities = ["read"]
}

# Allow read access to secret/data/application
path "secret/data/application" {
    capabilities = ["read"]
}

# Allow read access to auth/approle/login
path "auth/approle/login" {
  capabilities = [ "read" ]
}

# Allow read access to ACL policies
path "sys/policies/acl/*" {
  capabilities = [ "read"]
}
```

Next, create a role with that policy. Since creation of roles is not supported in the interface, create it using Vault's CLI. You can look up the details on Vault's CLI on the Hashicorp Developer website.

The following is an example of the Vault write command to create `infoarchive` role using the `infoarchive` policy:

```
>vault write -address=https://<vault host here>:8200 -ca-cert=vault_ca.crt auth/approle/
role/infoarchive secret_id_ttl=360d token_num_uses=10000 token_ttl=360d
token_max_ttl=390d secret_id_num_uses=10000 token_policies="infoarchive"
```

Tailor all options for `vault write` call to your specific environment, including duration, number of uses, *etc*.

Next, retrieve the role ID and secret ID for that role, and keep a note of it. The following are examples of such interactions using Vault's CLI:

```
>vault read -address=https://<vault host>:8200 -ca-cert=vault_ca.crt auth/approle/role/
infoarchive/role-id
Key        Value
```

```
---         -----
role_id     7aaf461a-4112-ff41-5d1e-72de0be47f8c

Read secret id for that role:
>vault write -address=https://<vault host>:8200 -ca-cert=vault_ca.crt -f auth/approle/
role/infoarchive/secret-id
Key                   Value
---                   -----
secret_id             c5962170-063e-76fc-9793-d7efdba02eda
```

2.9.5 Populating Vault with secrets

Once you have your Vault JSON templates, you can use Vault's interface to navigate to the KV Secrets Engine, as it can be now populated with JSON data.

- When using Vault's interface, go to **Secrets** and select the KV mounting path. In this scenario, would is `secret`. That takes you to the **Secret Configuration** screen. Here, create a new secret. Since you are preparing a location for secrets for a specific customer, choose a name for the secrets path that is meaningful for your customer (for example, the customer's name, *etc*.). For this scenario, we will use `iacustomer`.

  Note: This location should correspond precisely to a location chosen in the previous step when you created a role and policy for the `infoarchive` client.

- The secrets can be created in JSON or simple properties flavour. For now, however, select JSON. The customer's name becomes the first part of the actual path to the secret's location. Under the same path, create four actual folders, each corresponding to one of OpenText Information Archive components. For example, the first path will be `iacustomer/ias`. This denotes the `ias` (IA Server) folder within the `iacustomer` path. For the actual data, remove anything that was pre-populated in the Data section of the interface screen by Vault (for example, remove curly brackets, *etc*.), and then copy content of the `server.json` template (or `server_plain.json` if not using encryption) and paste it to the Data portion of the interface. Vault validates JSON for errors so, if the copy/paste action resulted in some issues (for example, bad formatting) that will need to be corrected before you can successfully save it. Once all potential JSON issues are resolved, Save the secret. Now go back to the **Secrets** view and select `secret` KV path (or, alternatively, click **secret** part of the breadcrumb view shown in upper-left portion of the interface). Create another secret, this time choosing path `iacustomer/shell`, and populate it now with content of `cli.json` (or `cli_plain.json`), then save it. Go back to the `secret` again, create another secret: `iacustomer/iawa` (and use this time `iawa.json` or `iawa_plain.json`), then save it. Keep track of these paths as they will be needed later when configuring Helm. They will correspond to the properties under the `vault.kv.application` key: `ias`, `iawa`, and `shell` properties.

2.9.6 Enabling CUBBYHOLE authentication

If you are planning to use CUBBYHOLE authentication, you need to store the secret ID for the main role in some location in the Vault from which you will be able to retrieve it at run time. CUBBYHOLE authentication requires two roles: a main role with access to the OpenText Information Archive secrets, and a helper role with access only to the location where you store the secret ID. Refer to the previous section where you created role (main) for `infoarchive` client. Using similar steps, create an access policy for a helper role and then create a helper role using that policy. This policy can be very simple, since you will only be granting read access to one location where you will store the secret ID for the main role.

First, store the secret ID for the main role in some location. Using Vault's interface, go to **Secrets** tab and select path to our KV Secrets Engine (for example, `secret` in our previous examples) and create new a secret with path like the following: `helper/iacustomer`. Turn off JSON radio control, as the secret will be a simple key-value pair. In the **Secret** field, for the key, input `secret-id`. For the corresponding value field, paste the secret ID of the main role you created earlier. With this, you have now preserved secret ID for the main role in this location. Save your new secret.

Next, define the policy with access to that secret:

```
# Allow read access to helper/iacustomer/*
path "secret/data/helper/iacustomer" {
    capabilities = ["read"]
}
```

Then create a new role (for example, `infoarchive-helper` using this new policy). Refer to the previous examples on creating role, as the steps are very similar.

Finally, make a note of the role ID and secret ID for your new helper role.

Note: As can be seen in this example, the `infoarchive-helper` role has no access to `infoarchive` secrets. That is the key idea behind creating dual role access. As the helper's access is only allowed to one specific location where we store secret ID, it cannot compromise any of the actual secrets.

2.9.7 Enabling Vault integration in Helm

Below you can see an example of the Vault's configuration section as part of Helm deployment. By default, `vault.enabled` is set to false. Set this to true if leveraging integration with Vault. Here is a brief description of relevant properties for the Vault:

- `protocol`: Can be either http or https, but depends on how your Vault's instance has been configured. Most likely, it will work over the https protocol.
- Set `hostname` to FQDN of the machine where Vault is running. Note that this FQDN must be accessible from where the PODs will be running.
- `port`: Set this to the value of the port on which your instance is running. Default is 8200 but Vault can be run on any port.

- `cacert`: Obtain a CA certificate from Vault's instance, as this will be used to validate the server's identity and establish trust between OpenText Information Archive (Vault's client) and the Vault server. Provide the name of your certificate here.
- `skipTlsVerification`: Set it to false. If you have trouble getting TLS handshake to work between OpenText Information Archive client and Vault instance, which is running over TLS, you can temporarily set this to true to ignore some certificate-related checks, which could be failing due to issues with the certificate.

  Important
  OpenText Information Archive should never run with this value set to true in production so ensure it is switched back to false as soon as possible.
- `namespace`: If using Vault Enterprise and namespace was assigned, set this value here.
- `token`: When using the TOKEN authentication method, set this value to non-expired token issued by the role with full access to secrets required by the OpenText Information Archive components
- `roleId`: When using APPROLE or CUBBYHOLE authentication methods, set it to the value for the role ID of the main role
- `secretId`: When using APPROLE authentication only, set it to value of secret ID for the main role. For CUBBYHOLE authentication, this value can be blank, as it will be retrieved from Vault dynamically at run time.
- `roleName`: Name of the main role
- `appRolePath`: When enabling APPROLE authentication in Vault, this is the path selected for it. By default, it is set to `approle`, but can be anything. If unsure, look this up in Vault under **Access > Authentication Methods**. Details of the APPROLE authentication, including its path, can be found there.
- `helper`: This section is only used for CUBBYHOLE authentication
  - `roleId`: Role id for the helper role
  - `secretId`: Secret ID for the helper role
  - `hasWriteAccess`: For most purposes, set this to false
  - `secretPath` and `secretKeyName`: When configuring the helper role, you also create secret (location) where to store the secret ID for the main role. `secretPath` should correspond to the path where the secret is stored, and `secretKeyName` is the name of the property for which the value is the secret ID of the main role. For example, if the secret is stored under `helper/iacustomer`, `secretPath` should be set to that value. If the property name is `secret-id`, `secretKeyName` should be set to that value.
- `authentication`: Sets the authentication mode for all OpenText Information Archive components. Valid values are either: TOKEN, APPROLE, CUBBYHOLE or NONE

- `wrappedTokenTtl`: Time to live, for the wrapped token, when using CUBBYHOLE authentication, in seconds. After such time, the wrapped token expires. This value should be long enough to allow for the starting POD to use it but should not be too long. Typically, should be no longer than few minutes.
- `kv`: This section is used for configuration related to KV Secrets Engine:
  - `enabled`: This should be always set to true
  - `backend`: When configuring KV Secrets Engine, this is the value of the `path` component selected for the engine. By default, it is `kv` but can be set to anything.
  - `application`: This subsection contains secrets location for all three components:
    - `ias`: Location of the secrets for IA server (`iacustomer/ias`)
    - `iawa`: Same but for the IA Web App (`iacustomer/iawa`)
    - `shell`: location for secrets for IA Shell (`iacustomer/shell`)
- `metadata/annotations`: list of all annotations as required
- `serviceAccountName`: if required, name of service account that is expected for the Vault integration. Additionally, that service account needs to be created in the K8's namespace in which OpenText Information Archive is deployed.
- `image`: Subsection name/tag should correspond to the location of the Vault Agent's image in repository, similar to how there are image sections for all other containers

The following is an example of `YAML` section for the Vault's configuration from Helm's values file:

```
vault:
  enabled: true
  protocol: https
  hostname: vault-infoarchive.us-west1-a.c.otl-eng-cs-ia.internal
  port: 8200
  # for vault https
  cacert: ca.crt

  skipTlsVerification: false
  namespace:
  token:
  roleId: 7aaf461a-4112-ff41-5d1e-72de0be47f8c
  #secretId: c5962170-063e-76fc-9793-d7efdba02eda
  roleName: infoarchive
  appRolePath: approle
  helper:
    roleId: 0e191143-e8c3-7f8a-17a4-cb3c03c0f394
    secretId: a430d7be-3804-6eed-0bce-61f9899c649e
    hasWriteAccess: false
    secretKeyName: secret-id
    secretPath: helper/iacustomer
  authentication: CUBBYHOLE
  wrappedTokenTtl: 600
  kv:
    enabled: true
    backend: secret
    application:
      ias: iacustomer/iaserver
```

```
    iawa: iacustomer/iawa
    shell: iacustomer/cli
metadata:
  annotations: {}
serviceAccountName:
image:
  name: vault
  tag: 1.12.2
  pullPolicy: Always
```

Ensure all secrets are pre-loaded to the Vault before trying to deploy OpenText Information Archive.

Just like for the truststore and PostgreSQL certificates, you need to generate Base64 encoded versions of the Vault's CA certificate if connecting to Vault instance over TLS. Make sure to use the Base64 utility flag `–w 0` so that the generated .base64 file has a single line. Create a Base64–encoded form of the certificate with the extension `.base64`. For example:

- `vault_ca.crt.base64`

2.9.8 Configuring horizontal pod autoscaling

OpenText Information Archive supports HPA and supports both standard and custom metrics. HPA works by specifying minimum and maximum values for Pod replicas and, depending on type of HPA, it uses specific metric(s) to scale number of running replicas either up or down, as needed. HPA is only supported for transaction type `ms`.

HPA can be configured by modifying the `autoscaling` section in Helm's values file, ideally by copying that section into one of your override files and updating it there.

HPA, by default, is disabled. It can be enabled as needed and per-component. Specifically, there is a a separate subsection for each of IA Server instance (bp, search or ingestion) and for each of IA Web App instance (search and ingestion). Each instance can be separately enabled or disabled.

Standard metrics: These are controlled by threshold set on the CPU utilization (`targetCPUUtilizationPercentage`). By default, this is set at 80 (%), meaning that if CPU processing exceeds 80%, and HPA is enabled, the system will start new replica(s) to better support load balancing features. Similarly, when threshold falls down, some replica(s) may be scaled down. This is all done automatically by the system.

As long as metric server is deployed and available in your cluster, you can enable HPA standard metrics and adjust behavior, as required.

Custom metrics: In addition to standard metrics, you can also leverage HPA based on custom metrics, which are metrics that are not supported natively by built-in Kubernetes but provided by custom metric servers, such as Prometheus. Setting up such a server is beyond the scope of this documentation. If you have a metric server running that supports custom queries, you can enable the IA Server again – per component, just like for standard metrics – as required. You will notice there is a

`customMetrics` section in HPA's configuration, and each IA Server instance (bp, search, and ingest) can be enabled/disabled separately. Note that the IA Web App cannot be scaled based on custom metrics. This is only available for the IA Server.

Each IA Server instance has a specific `metricName`, which becomes a query in custom metrics server, and each instance of the IA Server can be queried based on that value.

There is a separate section for enabling metrics in OpenText Information Archive. The `metrics` section, by default, is disabled. To leverage custom metrics, this needs to be enabled first. Then each IA Server instance under custom metrics can be enabled/disabled, as necessary. Once custom metrics are enabled per given IA Server instance, you can further tweak target value property (OOTB this is set to 10), which means if the query returns count higher than 10, scaling up should commence. Similarly, if the target value returns integer smaller than target value, scaling down may take place depending on current count of replicas.

2.10 Integration with Google Cloud Platform® service GCP

Like Vault, integration with GCP Secret Manager is optional. The opt-in model is based on the Spring profile.

The `gcp-secrets-manager` profile is similar to a Vault profile, and contains a small set of properties that allow each component to connect to Secret Manager and read configured secrets.

Example 2-1:

```
spring.cloud.gcp.secretmanager.enabled=true
spring.cloud.gcp.core.enabled=true
spring.cloud.gcp.secretmanager.allow-default-secret=true
spring.cloud.gcp.project-id=<GCP project ID>
spring.cloud.gcp.secretmanager.project-id=<GCP project ID>
spring.config.import=sm@
systemData.psql.database.admin.password=${sm@systemData_psql_database_admin_password}
crypto.keyStore.keyStorePass=${sm@crypto_keyStore_keyStorePass}
…
infoarchive.gateway.token.secret=${sm@infoarchive_gateway_token_secret}
OTDS.password=${sm@OTDS_password}
OTDS.infoarchive.clients.server.clientSecret=$
{sm@OTDS_infoarchive_clients_server_clientSecret}
```

In the example above, the first six sets of properties are configuration properties that control whether GCP Secret Manager integration is enabled, whether to allow default secrets, and provide the ID of the GCP service.

The last configuration property (`spring.config.import`) contains a default prefix for the secrets stored in GCP Secret Manager. It is recommended that you keep it as it is displayed in the example above.

All these configuration properties will be the same for each OpenText Information Archive component: IA Server, Gateway, or IA Shell.

What follows these initial set of configuration-level properties are the references to actual properties dedicated to each component. Each component has a separate set of properties.

2.10.1 Configuration in Helm's values file

The following section of Helm's values file is dedicated to configuring the GCP Secret Manager:

Example 2-2:

```
gcpSecretsManager:
  # set to true if using integration with GCP SM
  enabled: true
  #
  allowDefaultSecrets: true
  # Supported authentication methods: envVar
  # For envVar type: GOOGLE_APPLICATION_CREDENTIALS env variable will have json access
key
  authentication: "GOOGLE_APPLICATION_CREDENTIALS"
  # project id - most times gcp and secret manager project ids will be the same
  projectId:
  secretmanager:
    projectId:
  # Google application credentials, this should be set externally as it is a JSON file.
  accessKey:
  # Prefix for all the secrets, default is empty. Set to some value, i.e. customer name,
  # to allow for multiple deployments in the same SM. Use only alphanumeric characters,
  # preferably end with an underscore, i.e. acme_
  secretsPrefix:
```

Currently, the only authentication scheme supported is `GOOGLE_APPLICATION_CREDENTIALS`.

For most deployments, `projectId` and `secretmanager.projectId` will be the same.

Set the `accessKey` property to reference the JSON file that contains the JSON representation of the GCP service account's key that accesses the GCP Secret Manager, which contains secrets for your Information Archive deployment. The service account requires appropriate access to the GCP Secret Manager. At the very least, you need to assign the Secret Manager Secret Accessor Role to the service account; otherwise the service account will not have access to read secrets from Secret Manager.

By default, the `secretsPrefix` property can be left as is for most deployments. If you use the same Secret Manager to manage secrets for multiple OpenText Information Archive deployments, ensure each deployment has a unique prefix. For example, if Secret Manager is shared for two deployments – those could be two deployments for the same customer or for two different customers – in each case, the prefix should be unique for the deployment. You could use the prefix: `acme_stage_` for one deployment and `acme_prod_` for the other deployment.

2.10.2 Storing secrets in GCP Secret Manager

Use the Password Generation utility as you normally would to generate all the secrets. Notice that one of the new artifacts generated by the utility is a text template (`gcp_secret_manager.txt`). This template contains all the secrets that need to be stored in GCP Secret Manager .

The template contains a list of key-value properties in which the key corresponds to the name of the secret and the value contains the value of the secret itself.

Currently, secrets can only be created using the Google API or Google UI. Use either of these methods to create secrets. Both methods allow you to create one secret at a time.

Note: If you are using `secretsPrefix`, add the prefix to each name of the secret. For example, instead of creating a secret with the name `OTDS_password`, if the `secretsPrefix` property is set to `acme_prod_`, the name of the secret would be `acme_prod_OTDS_password`. Repeat this for every secret that needs to be created.

The structure of the template itself can easily be used by a script if you were going to use Google API to automate the process of secret creation.

Since the template contains actual passwords/secrets, ensure you protect access to it and delete the file (or archive, *etc.*) if you are done creating secrets.

2.11 Integration with AWS Secrets Manager

Like Vault, integration with AWS Secrets Manager is optional. Opt-in is based on the Spring profile. The `aws-secret-manager` profile is similar to a Vault profile, and contains minimal set of properties that allow each component to connect to Secrets Manager and read configured secrets.

Example 2-3:

```
aws.secretsmanager.enabled=true
aws.secretsmanager.failFast=true
spring.cloud.aws.region.static=us-east-1

spring.config.import=aws-secretsmanager:prod1/iacustomer/server
```

`aws.secretsmanager.enabled` / `aws.secretsmanager.failFast` / `spring.cloud.aws.region.static` properties

- The `aws.secretsmanager.enabled` property allows you to allow Secrets Manager integration.
- The `aws.secretsmanager.failFast` property allows you to configure components to quickly fail if there are any issues accessing Secrets Manager.

- The `spring.cloud.aws.region.static` property allows you to configure regional settings.

`spring.config.import` property

This property contains the actual path to the secrets (preceded by the `aws-secretmanager:` prefix). The path itself (`prod1/iacustomer/server` in Example 2–1 above) should correspond exactly to the secrets path (referred sometimes as a secret name), as defined in Secrets Manager.

The secret value is a JSON structure and looks exactly like the secret value for HashiCorp Vault:

Example 2-4:

```
{
  "OTDS.password": "otds",
  "crypto.keyStore.keyStorePass": "NgVS5IgfbM",
  "infoarchive.gateway.token.secret": "YJe4Yaikngs",
  "systemData.psql.database.admin.password": "Ga#Rd#1RT7t9ogGdI7Rm",
  "systemData.psql.database.connectionProperties.sslpassword": "",
  "systemData.psql.databaseCluster.connectionProperties.sslpassword": "",
  "systemData.psql.databaseCluster.superuser.password": "Ga#Rd#1RT7t9ogGdI7Rm"
}
```

As with Vault, Secrets Manager breaks the secret values down to key-value pairs.

2.11.1 Importing secrets

As with Vault, you can determine any preferred method of importing secrets into Secrets Manager, as long as they are ingested using the expected JSON templates. These templates are the same as for Vault integration.

Example 2-5: IA Server secrets as JSON

```
{
  "OTDS.password": "otds",
  "crypto.keyStore.keyStorePass": "XXXXXXXXXX",
  "infoarchive.gateway.token.secret": "XXXXXXXXXXXX",
  "systemData.psql.database.admin.password": "XXXXXXXXXXXXXXXXXXXX",
  "systemData.psql.database.connectionProperties.sslpassword": "",
  "systemData.psql.databaseCluster.connectionProperties.sslpassword": "",
  "systemData.psql.databaseCluster.superuser.password": "XXXXXXXXXXXXXXXXXXXX"
}
```

Example 2-6: IA Web App secrets as JSON

```
{
  "OTDS.infoarchive.clients.gateway.clientSecret": "XXXXXXXXXX",
  "OTDS.infoarchive.clients.iawa.clientSecret": "XXXXXXXXXX",
  "OTDS.password": "otds",
  "infoarchive.gateway.client.ssl.key-store-password": "",
  "infoarchive.gateway.client.ssl.trust-store-password": "XXXXXXX",
```

```
  "infoarchive.gateway.token.secret": "XXXXXXXXXX"
}
```

Example 2-7: IA Shell secrets as JSON

```
{
  "connection.clientSecret": "XXXXXXXXXX",
  "connection.userPassword": "XXXXXXXXXX",
  "rdbDataNode.connectionProperties.sslpassword": "",
  "rdbDataNode.password": "XXXXXXXXXXXXXXXXXXXX",
  "rdbDatabase.connectionProperties.sslpassword": "",
  "rdbDatabase.password": "XXXXXXXXXXXXXXXXXXX",
  "ssl.trustStorePassword": "XXXXXXX"
}
```

2.11.2 Authentication

Unlike Vault integration, Spring only supports a limited number of authentication methods for Secrets Manager:

- Authentication using environment variables,
- Authentication using system properties, or
- Authentication using profiles.

For on-prem deployment, it does not matter which authentication method is used, as Spring uses an authentication chain. As long as one of the above methods are selected, Secrets Manager will be able to fetch secrets.

Cloud deployment uses the environment variables method to authenticate Secrets Manager. An infrastructure was created to configure the access key ID (environment variable: `AWS_ACCESS_KEY_ID`) and secret access key (environment variable: `AWS_SECRET_ACCESS_KEY`) and set them as K8's secrets, which are then available to each component's container.

2.12 Preparing a customer directory

Create a customer directory by making a copy of the provided example customer directory:

```
helm\customers\iacustomer
```

For the purposes of this document, we will just use the above directory name.

2.12.1 Preparing the truststore with an imported OTDS certificate

The truststore is an optional element. As long as you are using proper Certificate Authority (CA) certificates with a full chain, you do not need to generate a custom truststore. In this case, the system's truststore is used. A custom truststore is for scenarios in which custom certificates issued by private CAs are used.

You must have the TLS/SSL certificate for OTDS available. This certificate needs to be imported into the truststore.

To prepare the truststore with an imported OTDS certificate:

1. Create the truststore in the `helm/customers/iacustomer/tls/client` folder. For example:

   ```
   helm/customers/iacustomer/tls/client/truststore.pkcs12
   ```

   You can create the truststore using the JDK provided `keytool`. The details of how to create the truststore are outside the scope of this document.

2. Note the truststore type (for example, PKCS12) and password. You will need the password in the next step.

3. Import the OTDS certificate into the truststore.

4. Note down the truststore password.

5. If you choose to use externally deployed PostgreSQL database components with the TLS/SSL protocol, you also need to import the TLS/SSL certificates for the PostgreSQL components into the truststore, before completing the next step.

6. Optional If you are leveraging Vault integration and Vault instance runs over TLS (most likely scenario), you'll also need to import Vault CA certificate into the truststore so that all components will be able to establish trust with Vault instance.

7. Create a base64 encoded form of the truststore file with the extension `.base64`. For example:

   ```
   truststore.pkcs12.base64
   ```

   Note: Make sure to use a single-line format (typically using the `–w 0` flag of the base64 utility) while creating the above file.

2.12.2 Generating and encrypting passwords

Before you begin: If you are using Windows, you must do the following:

- Install Windows Subsystem for Linux (`wsl`) because this step uses a BASH script.
- Install OpenJDK11 inside `wsl`.

To generate and encrypt passwords:

1. If you are on Windows, start `wsl` using the following command:

```
wsl
```

2. The OTDS administrator password is configured externally. Similarly, the truststore password was configured in the previous step. You do not need to generate these passwords. Instead, you need to specify them directly. You will still generate the encrypted form of these passwords. To specify the known passwords for encryption, in a text editor, open the following file:

   `helm\password-generator\config\helm-password-gen\passwordsToEncrypt`

3. Specify the passwords as follows:

```
otds.password=security.otds.encryptedPassword=<OTDS_PASSWORD>
security.tls.client.trustStorePassword=security.tls.client.encryptedTrustStorePasswo
rd=<TRUSTSTORE_PASSWORD>

...

security.crypto.keyStore.keyStorePass=security.crypto.keyStore.encryptedKeyStorePass
security.postgres.system.superuser.password=security.postgres.system.superuser.encry
ptedPassword=admin_one
security.postgres.system.dbowner.password=security.postgres.system.dbowner.encrypted
Password=db_owner_one
security.postgres.system.dbowner.sslpassword.password=security.postgres.system.dbown
er.sslpassword.encryptedPassword
security.postgres.structuredData.superuser.password=security.postgres.structuredData
.superuser.encryptedPassword=admin_two
security.postgres.structuredData.dbowner.password=security.postgres.structuredData.d
bowner.encryptedPassword=db_owner_two
security.postgres.structuredData.dbowner.sslpassword.password=security.postgres.stru
cturedData.dbowner.sslpassword.encryptedPassword
```

   The PostgreSQL database passwords need to be entered above only if they were given to you by the PostgreSQL administrator.

4. If you want to use your own passwords and secrets for other entries, you can use a similar technique.

   

   Note: The = sign is not allowed in any password.

5. Run the following commands. Depending on whether you are integrating with Vault the command may require additional argument at the end. If not using Vault integration use following command:

```
cd helm/password-generator
./bin/helm-password-gen.sh 40.0 20 ALPHANUMERICSYMBOL ../customers/iacustomer
```

   And if integrating with Vault, add “vault” at the end of your command:

```
cd helm/password-generator
./bin/helm-password-gen.sh 40.0 20 ALPHANUMERICSYMBOL ../customers/iacustomer vault
```

This will generate the following files:

- `../customers/iacustomer/overrides-passwords.yaml`: This file contains the generated passwords and encrypted passwords. The passwords pre-specified in the file are not generated but only encrypted by the above step. In OpenText Information Archive 24.2 you will need to specify PostgreSQL database passwords for corresponding keys in this file as shown above.

  Note: If using Vault integration, since all the secrets/passwords will be moved to the vault, the `override-passwords.yml` file is rather empty – it contains only a single password for `tls.client.trustStore` component which is the only secret/password that doesn't go into Vault.

- Optionally, if integrating with Vault, these additional Vault JSON templates will be generated in the `../customers/iacustomer/` folder: `server.json`, `server_plain.json`, `iawa.json`, `iawa_plain.json`, `cli.json`, and `cli_plain.json`.

- `helm/customers/iacustomer/password-encrypt/**`: This directory contains the generated keystore and associated files that allow non-interactive handling of the decryption of passwords.

  Note: If you do not want to use generated or pre-specified passwords the process is more complicated. In this case, you can copy the template from the following file:

  `helm\password-generator\config\helm-password-gen\overrides-passwords.yaml`

  You can then enter the unencrypted passwords and encrypt the passwords manually using the next step. It is recommended to use strong passwords.

- `../customers/iacustomer/otds-ia-bootstrap-config.yml`: This file is a bootstrapping YAML template. It is used during OTDS deployment to create the required artifacts for OpenText Information Archive.

  Note: Check the value of the password generated for `security.tls.client.trustStorePassword` (in the `overrides-passwords.yaml` file) to ensure it does not contain a `$` character. If it does, re-run the password generation utility until that password does not contain any `$` characters.

6. Use the following utility to encrypt the passwords:

   `helm/password-generator/bin/password-encrypt`

Caution

It is strongly recommended to encrypt the passwords.

Note: On a Windows computer, you must run the `password-encrypt` utility in Windows Subsystem for Linux (`wsl`). You must also install OpenJDK 1 on wsl. For more information about the password-encrypt utility, see section 7.2.3 "Using the password encryption utility to manually encrypt passwords and tokens" in *OpenText Information Archive - Encryption Guide (EARCORE-AGE)*.

The password-encrypt utility generates the following files:

- `creds`
- `keystore.jceks`
- `secretStore.uber`

It also generates a Base64 encoded form of these files with the extension `.base64` and copies them to the following directory:

`helm/customers/iacustomer/password-encrypt/`

2.12.3 Generating certificates and keys for Kubernetes ingress host names (FQDN)

The Kubernetes Ingress resource is used to enable access to IA Web App (IA Web App) for search or ingestion. Although there may be many different ways in which HTTPS is configured for a given cluster, when Ingress is used, there are two cases:

- HTTPS is terminated at the Ingress, or
- HTTPS may be terminated in another appliance fronting our cluster, in which case, Ingress traffic is coming over HTTP.

In case the HTTPS is terminated in front of the cluster and Ingress receives only HTTP traffic, you do not need to create K8's TLS secret for Ingress, as the certificate would be applied directly on the fronting appliance. However, if Ingress receives HTTPS traffic, you need to provide a TLS secret with certificate and key to the Ingress.

The Kubernetes Ingress resource works out of the box on GCP. You may have to deploy NGINX or some other Ingress controller on your platform. The details of that are outside the scope of this document.

2.12.3.1 Determining the FQDN for IA Web App

This document assumes the FQDN for IA Web App to be `iawa.iacustomer-cloud.net` when you are not using the `transactionOption` equal to `ms`.

If you are using the `transactionOption` equal to `ms`, then separate instances of IA Web App are configured for search and ingestion. In that case, the FQDN for the IA Web App that is configured for search is assumed to be `iawa-search.iacustomer-cloud.net`. The FQDN for the IA Web App that is configured for ingestion is assumed to be `iawa-ingestion.iacustomer-cloud.net`. You should make a note of these FQDNs.

Make sure to create a DNS record for these FQDNs so that the users of IA Web App will be able to access it using the IA Web App FQDNs. The details of this are dependent on your environment and are out of the scope of this document. Alternatively, you can add the external IP address associated with Ingress to the FQDN mapping in your `etc/hosts` file.

These FQDNs should be specified in the customer-specific values file:

```
helm\customers\iacustomer\overrides-general.yaml
```

For more information, see Customer values file.

2.12.3.2 Obtaining or generating the key and certificate for the IA Web App FQDNs

Note: This section is applicable only if HTTPS traffic ends at Ingress.

If you are not using the `transactionOption` equal to `ms`, generate the TLS/SSL key consistent with the FQDN `iawa.iacustomer-cloud.net`. Save it in the following file in a non-encrypted `.pem` format:

```
helm\customers\iacustomer\ingress\https\iawa.key
```

Obtain or generate a corresponding certificate. Save it in the following file:

```
helm\customers\iacustomer\ingress\https\iawa.cer
```

These file locations are later used while installing the Helm chart.

If you are using the `transactionOption` equal to `ms`, then generate the key and certificate consistent with the FQDN names. For example:

- `iawa-search.iacustomer-cloud.net`
- `iawa-ingestion.iacustomer-cloud.net`

You can use tools like OpenSSL or Java JDK's Keytool or the KeyExplorer GUI to generate the key and certificate and get it issued by the certificate authority (CA). Follow the best practices of your IT department.

2.12.3.3 Configuring the FQDN for PostgreSQL databases

Configure the `postgres:` section in the `helm\customers\iacustomer\overrides-general.yaml` file. You must set `postgres:deployment.external` to true. Then configure the rest of the `postgres:` section.

2.12.3.4 Configuring the FQDN for OTDS

In version 20.4 and later, OpenText Information Archive uses an external OTDS. Make sure that this FQDN name is resolvable from inside the cluster where you will deploy OpenText Information Archive.

Specify the OTDS FQDN in the customer-specific values file:

`helm/customers/iacustomer/overrides-general.yaml`

For more information, see Customer values file.

2.12.3.5 Configuring the FQDN for Vault

If leveraging integration with Vault, configure vault section in the `helm\customers\iacustomer\overrides- general.yml` file. You must set `vault: hostname`. Then configure the rest of the `vault:` section. Refer to the Integration with Vault chapter for more information.

2.13 Making sure Helm 3 is working in your cluster

Make sure you have installed Helm 3.x. Make sure the `helm` command is added to your PATH environment variables.

2.14 Configuring resources

It is good practice to set resources for the Kubernetes pods. This ensures that we have control over the amount of CPU as well as memory that will be carved out for each container at the time of creation, and that the containers won't be able to exhaust all available resources if things go wrong. Resource enablement is controlled via key `resources.enabled`. By default, it is set to `true` and there are default values for each resource. You are encouraged to adjust those values (such as `cpu` and `memory`) for each resource, based on your expectations and on the cluster's configuration.

There are two types of resource control: requests and limits. Resource-requests controls how much of CPU and memory is allocated to the container at the time of the container's creation. Resource-limits on the other hand establish a hard ceiling for CPU and memory. Typically, a container starts off with a lower CPU and memory values, and Kubernetes starts allocating more as resource consumption increases. This typically continues until we hit a limit, and if any container hits the limit, it will be killed by Kubernetes. It is therefore crucial to be able to evaluate what are good starting points for CPU and memory (resource requests) and what are the limits that the container should not be crossing.

It is difficult to predict what are good starting points and/or limits for a given deployment as it will depend on many factors, including types of containers/pods, day-to-day pod usages, etc. Ideally, you would start with some values and allow the system to run and observe resource consumption and tune from there. Providing not enough resources may cause pods to crash, and providing too much resource freedom may not be efficient from a cost perspective. For each type of resource, there is the option to disable/enable resources and, if they are enabled, there are the following options:

- Providing a value only for resource-requests for memory. In this case, we provide only the actual memory amount expected to be allocated for the container at the time of creation. It has no limit for memory and no CPU allocation or limit. Example: memory: 1Gi (one giga byte of memory).
- Providing a value for the CPU. This value is in mili-slices (for example, `500m`), but can be also provided in a fraction form (`0.5`). Both values indicate half of the virtual CPU core. In this case, we provide allocation for CPU (and memory), but no limits, meaning usage may grow indefinitely until some other hard limit is met.
- Specifying resource-limits for memory. This gives us an upper bound for memory consumption.
- Specifying resource-limits for the CPU, getting an upper bound for the CPU.

Note that we use the terms container and POD interchangeably but, depending on Kubernetes resources, this may or may not mean the same thing. This depends on whether the pod consists of a single container or multiple containers, including init containers.

Establishing resource allocation and limits further depends on the transaction type. The resources key is followed by transaction type: m1 through m4 so, depending on the transaction type used, it should be set accordingly.

Furthermore, there are additional resource allocations and limits that can be individually set for each Kubernetes job and one setting for all `init` containers.

Lastly, the ms transaction type has resources set via key, starting with `ms` prefix and followed by either `ias` or `iawa`, and then the resource type (for example `ms.ias.ingestion.resources.requests.memory`). See these keys in the values file and, if required, override them in your override file. You are encouraged to familiarize yourself with K8's resources and setting requests/limits so that you can provide best configuration for your environment.

2.15 Installing the OpenText Information Archive Helm chart

Before you begin:

- Make sure that OTDS is running.
- If you are using an external PostgreSQL database, make sure that all instances of it are running.

Install the Helm chart using the following commands:

```
cd helm
helm version
```

On Windows:

- The ^ at the end of the line is required to specify a multiline command for the Windows command prompt.
- In the example below, we are assuming a truststore in PKCS12 format, and assuming the GCP platform as indicated by `platforms\gcp.yaml`. Make sure to use your platform's values file.

  For non-TLS/SSL PostgreSQL:

```
helm install ^
--namespace ia ^
--timeout 1500Os ^
--set-file external.creds=customers/iacustomer/password-encrypt/creds.base64 ^
--set-file external.keystore=customers/iacustomer/password-encrypt/
keystore.jceks.base64 ^
--set-file external.secretStore=customers/iacustomer/password-encrypt/
secretStore.uber.base64 ^
--set-file external.iawaKey=customers/iacustomer /ingress/https/iawa.key ^
--set-file external.iawaCert=customers/iacustomer /ingress/https/iawa.cer ^
--values platforms\gcp.yaml ^
--values customers\iacustomer\overrides-general.yaml ^
--values customers\iacustomer\overrides-passwords.yaml ^
infoarchive ^
infoarchive
```

  For PostgreSQL over TLS/SSL:

```
helm install ^
--namespace ia ^
--timeout 1500Os ^
--set-file external.creds=customers/iacustomer/password-encrypt/creds.base64 ^
--set-file external.keystore=customers/iacustomer/password-encrypt/
keystore.jceks.base64 ^
--set-file external.secretStore=customers/iacustomer/password-encrypt/
secretStore.uber.base64 ^
--set-file external.iawaKey=customers/iacustomer /ingress/https/iawa.key ^
--set-file external.iawaCert=customers/iacustomer /ingress/https/iawa.cer ^
--set-file external.system.sslrootcert=customers/iacustomer/tls/postgres/
root.crt.base64 ^
--set-file external.system.sslcert=customers/iacustomer/tls/postgres/
iasuper.crt.base64 ^
--set-file external.system.sslkey=customers/iacustomer/tls/postgres/
iasuper.key.pk8.base64 ^
--set-file external.system.dbowner.sslcert=customers/iacustomer/tls/postgres/
iadbowner.crt.base64 ^
--set-file external.system.dbowner.sslkey=customers/iacustomer/tls/postgres/
```

```
iadbowner.key.pk8.base64 ^
--set-file external.structuredData.sslrootcert=customers/iacustomer/tls/postgres/
sdroot.crt.base64 ^
--set-file external.structuredData.sslcert=customers/iacustomer/tls/postgres/
sdiasuper.crt.base64 ^
--set-file external.structuredData.sslkey=customers/iacustomer/tls/postgres/
sdiasuper.key.pk8.base64 ^
--set-file external.structuredData.dbowner.sslcert=customers/iacustomer/tls/
postgres/sdiadbowner.crt.base64 ^
--set-file external.structuredData.dbowner.sslkey=customers/iacustomer/tls/postgres/
sdiadbowner.key.pk8.base64 ^
--values platforms\gcp.yaml ^
--values customers\iacustomer\overrides-general.yaml ^
--values customers\iacustomer\overrides-passwords.yaml ^
infoarchive ^
infoarchive
```

When leveraging Vault integration, additionally set following flag (if Vault is accessible over TLS):

```
--set-file external.vaultCaCert=customers/iacustomer/tls/vault/ca.crt.base64 ^
```

When using a custom truststore, add the following:

```
--set-file external.truststore=customers/iacustomer/tls/client/
truststore.pkcs12.base64 ^
```

On Linux:

- The \ at the end of the line is required to specify a multiline command on Linux.
- In the example below, we are assuming a truststore in PKCS12 format, and assuming the GCP platform as indicated by `platforms/gcp.yaml`. Make sure to use your platform's values file.

For non-TLS/SSL PostgreSQL:

```
helm install \
--namespace ia \
--timeout 15000s \
--set-file external.creds=customers/iacustomer/password-encrypt/creds.base64 \
--set-file external.keystore=customers/iacustomer/password-encrypt/
keystore.jceks.base64 \
--set-file external.secretStore=customers/iacustomer/password-encrypt/
secretStore.uber.base64 \
--set-file external.iawaKey=customers/iacustomer /ingress/https/iawa.key \
--set-file external.iawaCert=customers/iacustomer /ingress/https/iawa.cer \
--values platforms/gcp.yaml \
--values customers/iacustomer/overrides-general.yaml \
--values customers/iacustomer/overrides-passwords.yaml \
infoarchive \
infoarchive
```

For PostgreSQL over TLS/SSL:

```
helm install \
--namespace ia \
--timeout 15000s \
--set-file external.creds=customers/iacustomer/password-encrypt/creds.base64 \
--set-file external.keystore=customers/iacustomer/password-encrypt/
keystore.jceks.base64 \
--set-file external.secretStore=customers/iacustomer/password-encrypt/
secretStore.uber.base64 \
--set-file external.iawaKey=customers/iacustomer /ingress/https/iawa.key \
--set-file external.iawaCert=customers/iacustomer /ingress/https/iawa.cer \
--set-file external.system.sslcert=customers/iacustomer/tls/postgres/
iasuper.crt.base64 \
--set-file external.system.sslkey=customers/iacustomer/tls/postgres/
```

```
iasuper.key.pk8.base64 \
--set-file external.system.dbowner.sslcert=customers/iacustomer/tls/postgres/
iadbowner.crt.base64 \
--set-file external.system.dbowner.sslkey=customers/iacustomer/tls/postgres/
iadbowner.key.pk8.base64 \
--set-file external.structuredData.sslrootcert=customers/iacustomer/tls/postgres/
sdroot.crt.base64 \
--set-file external.structuredData.sslcert=customers/iacustomer/tls/postgres/
sdiasuper.crt.base64 \
--set-file external.structuredData.sslkey=customers/iacustomer/tls/postgres/
sdiasuper.key.pk8.base64 \
--set-file external.structuredData.dbowner.sslcert=customers/iacustomer/tls/
postgres/sdiadbowner.crt.base64 \
--set-file external.structuredData.dbowner.sslkey=customers/iacustomer/tls/postgres/
sdiadbowner.key.pk8.base64 \

--values platforms/gcp.yaml \
--values customers/iacustomer/overrides-general.yaml \
--values customers/iacustomer/overrides-passwords.yaml \
infoarchive \
infoarchive
```

When leveraging Vault integration, additionally set following flag (if Vault is accessible over TLS):

```
--set-file external.vaultCaCert=customers/iacustomer/tls/vault/ca.crt.base64 \
```

When using a custom truststore, add the following:

```
--set-file external.truststore=customers/iacustomer/tls/client/
truststore.pkcs12.base64 \
```

Notes

- You can use the `helm lint` command to pre-check the configuration of the values files.
- You can use the `helm template --debug` or `helm install -– debug –- dry-run` command to pre-check the configuration of values files and validate the generated Kubernetes manifest files. Perform this step until the generated manifest has the values you have configured.
- You are free to choose a namespace other than `ia` as shown in the example above.

This might take as much as 15 to 20 minutes, or as little as a few minutes. At the end you will receive a report of what Kubernetes resources were configured. While the chart is installing you can monitor the progress using Google Cloud Console. If you are deploying to Azure, AWS, or CFCR you can use the Kubernetes Dashboard to monitor the progress, if it is available. You can use the respective web consoles for Azure or AWS.

Tip: Using Visual Studio Code editor with the Kubernetes extension is another alternative.

2.16 Validating the installation

2.16.1 Connecting to OTDS

To connect to OTDS:

1. In a private browser window, connect to OTDS using the following URL:

   `https://otds.iacustomer-cloud.net/otds-admin`

   Note: Use a private browser window so that this session does not interfere with connecting to IA Web App, which follows connecting to OTDS.

2. Sign in to OTDS using your credentials.
3. In OTDS, disable the `http.negotiate` authentication handler in case it is still enabled.

2.16.2 Connecting to IA Web App

Connect to IA Web App using the following URL:

`https://iawa.iacustomer-cloud.net`

Use any of the following bootstrap usernames that are pre-populated in the `infoarchive.bootstrap` partition of OTDS:

- `adam@iacustomer.com`
- `audrey@iacustomer.com`
- `bob@iacustomer.com`
- `connie@iacustomer.com`
- `emma@iacustomer.com`
- `imran@iacustomer.com`
- `rita@iacustomer.com`
- `sue@iacustomer.com`

The unencrypted passwords for these users can be found in the following file:

`helm/customers/iacustomer/overrides-passwords.yaml`

Caution

Storing passwords in clear text (unencrypted) is a security risk. You should protect the `overrides-passwords.yaml` file in some way.

It is highly recommended to configure an additional partition in OTDS for your users and groups and associate it with the `infoarchive` access role.

For a fresh install, the groups configured in the `infoarchive` bootstrap partition are correctly mapped to the appropriate roles.

After configuring the additional partition, make sure to map those groups to IA Web App roles using the **Administration > Groups** page in IA Web App. After that, you can disable the `infoarchive.bootstrap` partition.

2.16.3 Optional: Validating Vault if enabled

Vault is acting as another property source so most of the integration, when working correctly, is not verbose at any level. It is important to note a few points. If PostgreSQL passwords for super user and DB owner were moved to Vault, and are no longer in `override-passwords.yml` file, the IA Server will not connect successfully to PostgreSQL (assuming server is expecting passwords from clients and not using trust blindly) unless it gets correct values for these passwords from the Vault. So if IA Server was able to connect to PostgreSQL database, it validates that it was able to pull passwords successfully from the Vault.

For each user, such as in the above example – for user adam@iacustomer.com – it should be reported that system found no password, is attempting to find it, then returning it and lastly updating it. That confirms that list of users coming from the Vault was merged with list of users coming from `YAML` configuration file. When not using Vault integration – none of these statements should be reported in the log.

2.17 Configuring the local environment to ingest applications

2.17.1 Downloading the pre-configured IA Shell

This option is applicable for ingesting your applications using IA Shell. If you want to ingest the example applications that are available in the distribution, see Using the OpenText Information Archive distribution.

Using this IA Shell, you will be able to connect to a running OpenText Information Archive application and ingest data. For more information about how to use IA Shell, see *OpenText Information Archive - IA Shell Guide (EARCORE-ARE)*.

To download the pre-configured IA Shell:

- In IA Web App (IA Web App), in the top-right corner of the page, click your user name, and then select **Download IA Shell**.

2.17.2 Using the OpenText Information Archive distribution

You can download the OpenText Information Archive distribution and install a local, simple configuration in the `infoarchive` directory (referred to as `<IA_ROOT>` below). Then you can configure IA Shell in the following configuration files:

- `<IA_ROOT>/config/iashell/application.yml`
  - FQDN of IA Web App, set with the properties `gatewayUrl` (the home page of IA Web App) and `restApiUrl` (typically the `gatewayUrl` with the postfix of `/restapi/services`)
  - `infoarchive.cli clientSecret`
  - Password for `sue@iacustomer.com`
  - `defaultSettings.other.uploadGatewaySocketTimeout`: value in milliseconds, specifying how long the socket timeout should be configured for the upload of large files. By default it is set to 60000 - corresponding to 60 seconds. This value may need to be adjusted depending on how big a file we're uploading and what are the network conditions.
- `<IA_ROOT>/config/iashell/application-https.yml`
  - FQDN of IA Web App, set with the properties `gatewayUrl` (the home page of IA Web App) and `restApiUrl` (typically the `gatewayUrl` with the postfix of `/restapi/services`)
  - Truststore information where the certificate for IA Web App has been imported
- `<IA_ROOT>/config/iashell/default.properties`:
  - The superuser and admin passwords for Structured Data and Search Results PostgreSQL
  - The path to the dataFileSystem directory (`/opt/iadata/defaultFileSystemRoot/data/root`)
  - Also change the values for the following parameters:

Parameter	Value
`rdbDataNode.name`	`structuredData`
`rdbDataNode.bootstrap`	`jdbc:postgresql://<FQDN of Postgres>:5432/`
`rdbDataNode.username`	`<postgres superuser username>`
`rdbDataNode.password`	`<postgres superuser password>`
`rdbDatabase.username`	`<database owner name>`
`rdbDatabase.password`	`<database owner password>`

If you are using an external PostgreSQL database configured over TLS/SSL, you'll need to specify the following additional parameters:

Parameter	Value
rdbDataNode. connectionProperties.ssl	true
rdbDataNode. connectionProperties.sslrootcert	Root certificate for structured database, i.e., root.crt.
rdbDataNode. connectionProperties.sslcert	Client certificate for structured database superuser, i.e., client.crt
rdbDataNode. connectionProperties.sslkey	Private key matching client certificate, i.e.,client.key.pk8
rdbDataNode. connectionProperties.sslpassword	<sslkey password if the key is password-protected>
rdbDatabase. connectionProperties.sslcert	Client certificate for structured database owner (*i.e.*, dbowner.crt)
rdbDatabase. connectionProperties.sslkey	Private key matching client certificate (*i.e.*, dbowner.key.pk8)
rdbDatabase. connectionProperties.sslpassword	<sslkey password if the key is password-protected>

You can find the passwords mentioned above in the following file:

```
helm/customers/iacustomer/overrides-passwords.yaml
```

Once configured, you can put your standard application configuration into the following directory and install it like the other example applications:

```
<IA_ROOT>/example/applications
```

You can also install the other example applications.

If you have any custom presentation or branding files that are copied into the <IA_ROOT>/config/iawebapp/customization directory, then you will have to copy them into the IA Web App Pod.

Use a BusyBox image-based deployment with iawa-config-iawebapp-customization-pvc PVC mounted in it. The BusyBox image is also used by the init containers inside the IA Web App and IA Server pods.

Use the following procedure to copy custom logos and custom presentations for IA Web App deployed as a pod if the tar command is not available inside the IA Web App pod. The 24.2 images do not have the tar command.

1. Download the deployment-cp.yml file.
2. Edit and adjust the image property to point to the BusyBox image they use in the IA Web App pod init containers. Get this information by running the kubectl describe command on any of the IA Web App pods.

   Be sure to specify any pull secrets, if applicable.

3. Make sure to switch to the same Kubernetes namespace in which the OpenText Information Archive Helm chart is deployed.

4. Apply the deployment manifest:

```
kubectl apply -f deployment-cp.yml
```

5. Get the name of the pod deployed by the deployment mentioned above . It will have the name of the form `infoarchoive-cp-nnnnnnn`.

```
kubectl get pods
NAME                                                READY   STATUS
RESTARTS   AGE
infoarchive-cp-7df74db9d7-vj55w                     1/1     Running
0          13m      <---------------------- for example
infoarchive-ias-bp-0                                1/1     Running
0          112m
infoarchive-ias-ingestion-0                         1/1     Running
0          112m
infoarchive-ias-search-0                            1/1     Running
0          112m
infoarchive-iawa-ingestion-7ff588488f-j9ww4         1/1     Running
0          112m
infoarchive-iawa-search-b79c888bc-dtz95             1/1     Running
0          112m
```

6. Terminal into the pod in case you need to create any folder under `/opt/ia/config/iawebapp/customization`.

```
kubectl exec -it infoarchive-cp-7df74db9d7-vj55w -- /bin/sh
/ $  cd /opt/ia/config/iawebapp/customization
/opt/ia/config/iawebapp/customization $ # Create any folders
/opt/ia/config/iawebapp/customization $ exit
```

7. Copy files into the `infoarchive-cp pod`:

```
kubectl cp logo.png infoarchive-cp-7df74db9d7-vj55w:/opt/ia/config/iawebapp/
customization/branding/...
```

8. Once finished, delete the above deployment:

```
kubectl delete -f deployment-cp.yml
```

2.17.2.1 Upload of large content

In addition to configuring `uploadGatewaySocketTimeout`, if there are any other proxies, load balancers, etc., those components may come pre-configured with their own timeouts that play a role during file upload and those may need to be adjusted, especially if we're ingesting very large files. During the file upload, IA Shell is initiating file push from the client machine. Depending on the network topology, in the simples example, the push goes directly into OpenText Information Archive Gateway, from which it is being streamed to the IA Server, which ultimately processes and persists the file . During that processing, both: IA Shell and Gateway need to wait until IA Server finishes. It is for that wait specifically that sometimes we may need to adjust the timeout to a larger value. We'll notice that as if the timeout is not sufficient, IA Shell will terminate prematurely. Depending on conditions, it may be reporting that "Target server didn't respond" and may end up getting a 504 HTTP status code in response as well. Note that in some cases, depending on where in the process of file upload the error occurred, the file may actually end up being

successfully processed on the IA Server. In any case, assigning a larger value to the timeout is usually required in such cases.

Additionally, if there are any proxies, load balancers, etc., involved in the upload chain, those also have to be checked for timeouts and extend these timeouts as necessary. For example, in cloud deployment we usually have an Ingress that plays a role of a proxy. Those often come configured OOTB with timeouts around 60 seconds as well (i.e., Nginx Ingress controller comes configured with `proxy-read-timeout` set at 60). Any load balancers, if they have their own timeouts related to traffic (i.e., Elastic Load Balancer in AWS comes with Idle timeout set to 60 seconds as part of the Traffic configuration), those also may need to be adjusted if there are issues with file uploads. If there are timeouts experienced during file upload, network configuration will have to be well understood and all potential timeouts configurations for various components may need to be adjusted as required.

Note that the actual timeout value may be proportional to the size of the content being uploaded and the overall network conditions. The rule of thumb is that the larger the file, the longer the timeout value may be required in order for successful upload to complete without any errors. As an example, in our clusters, when uploading 1.5 GB file, default timeouts of 60 seconds had to be extended to 90 seconds on each component (including IA Shell, Ingress and Load Balancers) before upload is successful.

Some load balancers or proxies may not support streaming out of the box. For example, the NGINX Ingress Controller, by default, temporarily caches large content into local storage. This behavior can be tweaked by setting an ingress annotation proxy-request-buffering to off:

```
nginx.ingress.kubernetes.io/proxy-request-buffering: "off"
```

Chapter 3

Deploying OpenText Information Archive in a private cloud

When deploying to your private cloud Kubernetes environment, make sure you have the following:

- Administrative access to your Kubernetes environment. This is required to do the following:
  - Create the cluster
  - Create the required storage classes for provisioning the `PersistentVolumeClaims`
  - Configure a suitable ingress controller to enable access to IA Web App from outside the cluster
- DNS configurations so that you can configure mappings for IA Web App FQDNs, the external OTDS FQDN, and if using an external PostgreSQL Server, FQDNs of the external PostgreSQL components
  - Creation of the required TLS/SSL keys and certificates
- The Docker images pushed to the Docker registry that the OpenText Information Archive deployment can pull the Docker images from
  - Creation of any pull secrets to enable pulling of Docker images from the Container Registry

Chapter 4

Deploying OpenText Information Archive on Microsoft Azure

In addition to the considerations mentioned in this chapter, all of the considerations mentioned in Deploying OpenText Information Archive in a private cloud apply.

4.1 Configuring the Azure Kubernetes service cluster

Configure the cluster of an appropriate size, with the CPU and memory configuration consistent with the `transactionOption` settings, using Azure Console or the `az` CLI in your resource group.

Ideally, run each of the distinct runtime components on separate node-pools and `nodeSelector` configurations:

- IA Web App
- IA Server

You may run more instances of the same type of components on the same or different nodes.

Make sure to enable access to the Docker registry services to push and pull the OpenText Information Archive Docker images.

Configure a suitable ingress controller (for example, NGINX) to allow access to IA Web App from outside the cluster.

4.2 Configuring ReadWriteMany storage for Azure

On AKS, you can use Azure Files to implement ReadWriteMany PersistentVolumeClaims. Once configured, note the storage class for Azure Files. You will use this later while configuring the values for the Azure platform. This option is available on Azure only. Note the name of storage class name (for example, `azurefile-standard`). You will specify it in the customer-specific `overrides-general.yaml` file.

For more information, see Customer values file.

4.3 Configuring the Azure Gateway Ingress Controller

The Azure Gateway Ingress Controller (AGIC) injects `X-Forwarded-For` header with some additional details, such as port, which breaks some interactions with our IA Web App/Gateway component. To ensure AGIC works with OpenText Information Archive, an additional re-writing rule on AGIC is required. For more information, refer to the article Rewrite HTTP request and response headers – Azure Application Gateway (https://learn.microsoft.com/en-us/azure/application-gateway/rewrite-http-headers-portal) on the Microsoft Learn site. Follow the instructions outlined in the article to add an additional re-write rule for `X-Forwarded-For` header and set it to the following value:

```
{var_add_x_forwarded_for_proxy}
```

Chapter 5

Deploying OpenText Information Archive on GCP

In addition to the considerations mentioned in this chapter, all of the considerations mentioned in Deploying OpenText Information Archive in a private cloud apply.

5.1 Configuring the GKE cluster

Configure the cluster of an appropriate size, with the CPU and memory configuration consistent with the `transactionOption` settings, using Google Cloud Console or the `gcloud` CLI in your GCP project. For more information, see the GKE documentation.

Ideally you can run each of the distinct runtime components on separate node-pools and nodeSelector configurations:

- IA Web App
- IA Server

You may run more instances of the same type of components on the same or different nodes.

Make sure to enable access to the Docker registry (for example, GCR) services to push and pull the OpenText Information Archive Docker images.

GKE provides a default ingress controller to allow access to IA Web App from outside the cluster. You may also configure a suitable ingress controller (for example, NGINX) to allow access to IA Web App from outside the cluster.

5.2 Configuring ReadWriteMany storage for GKE

Make sure that your cluster has the Storage Classes that support RWM PVCs. Specify the Storage Class names in the platform values file.

Chapter 6

Deploying OpenText Information Archive on AWS

In addition to the considerations mentioned in this chapter, all of the considerations mentioned in Deploying OpenText Information Archive in a private cloud apply.

6.1 Configuring the EKS cluster

Configure the cluster of an appropriate size, with a CPU and memory configuration consistent with the `transactionOption` settings, using AWS Management Console or the `eksctl` CLI in your resource group.

Ideally you can run each of the distinct runtime components on separate node-pools and nodeSelector configurations:

- IA Web App
- IA Server

You may run more instances of the same type of components on the same or different nodes.

Make sure to enable access to the Docker registry (Elastic Container Registry) services to push and pull the OpenText Information Archive Docker images.

Configure a suitable ingress controller (for example, NGINX) to allow access to IA Web App from outside the cluster.

6.2 Configuring ReadWriteMany storage for AWS

On AKS, you can use the Elastic File System to implement `ReadWriteMany PersistentVolumeClaims`. Once configured, note the storage class for the Elastic File System. You will use this later while configuring the values for the AWS platform. This option is available on AWS only. Note the name of storage class name (for example, `efs-standard`). You will specify it in the customer-specific `overrides-general.yal` file.

For more information, see Customer values file.

Chapter 7

Deploying OpenText Information Archive on OpenShift

In addition to the considerations mentioned in this chapter, all of the considerations mentioned in Deploying OpenText Information Archive in a private cloud apply. If you are deploying to OpenShift on AWS, then all considerations for AWS apply here.

On OpenShift, you might have to make use of OpenShift concepts such as Projects and Routes, which are over and above Kubernetes concepts like namespace and Ingress.

The default route created for the Ingress may not support HTTPS. You might have to explicitly create this route using the OpenShift console or command-line interface (CLI). The details of this are outside the scope of this document.

Access to the Docker registry will depend on the underlying Kubernetes platform. The creation of storage classes that support `ReadWriteMany PersistentVolumes` will depend on the underlying Kubernetes cluster. For more information, consult your cluster administrator.

Chapter 8

Deploying OpenText Information Archive on CFCR

8.1 Configuring the CFCR cluster

Configure the cluster of an appropriate size, with the CPU and memory configuration consistent with the `transactionOption` settings, using the Cloud Foundry `cf` CLI in your organization and space.

Configure a suitable ingress controller (for example, NGINX) to allow access to IA Web App from outside the cluster.

If you encounter a 504 Gateway Timeout error coming back from NGINX, you might want to extend the timeout to a value that is larger than the default of 60 seconds. You can change the timeout in the `platforms.local/cfcr-infoarchive.yaml` file, in the `ingressAnnotations` section. For example:

```
nginx.ingress.kubernetes.io/proxy-read-timeout: "600"
```

8.2 Configuring ReadWriteMany storage for CFCR

Note the storage class that implements the NFS-based PersistentVolumes. You will specify it in the customer- specific `overrides-general.yaml` file.

For more information, see Customer values file.

Chapter 9

Upgrading a cloud deployment

9.1 Upgrading OTDS

9.1.1 Upgrading OTDS from previous OpenText Information Archive deployments

The 24.4 version of OpenText Information Archive deployment works with the 24.3 version of OTDS. The 24.3 version of OpenText Information Archive used the OTDS 24.3.0 Helm chart. Download the OTDS 24.4.0 Helm chart from OpenText Support.

Like previous versions of OTDS, the current version of OTDS uses PostgreSQL database to store data. During the upgrade, you will need to use the same PostgreSQL database with the following details:

- Database name: `otdsdb`
- OTDS database user name: `otds`
- OTDS user password: Set the password based on your best practices. Avoid the use of – in the password.

To perform the upgrade:

1. Download and extract the current version of the OTDS Helm chart.
2. Make sure to pull the current version of the OTDS Docker image from registry.opentext.com (https://registry.opentext.com/) and push it to your Container registry.
3. Configure the values file based on the values file from OTDS 24.3.0 Helm chart. Specifically, make sure to copy over `otdsws.cryptKey` value chart.
4. Run the Helm upgrade command with the current version of the OTDS Helm chart.
5. Verify the data is imported into the current OTDS deployment:
   - Partitions
     - `infoarchive`
     - `infoarchive.bootstrap`
     - Any other partitions you have configured
   - Resource
   - Access Role

- OAuth2 clients
  - `infoarchive.gateway`
  - `infoarchive.iawa`
  - `infoarchive.cli`
  - `infoarchive.jdbc`
  - `infoarchive.sap`
  - `infoarchive.server`
  - Any other OAuth2 clients used for OpenText Information Archive

Use the section 4 “Upgrading to OTDS 25.4.0” in *OpenText Directory Services - Cloud Deployment Guide (OTDS-CGD)* as a reference when upgrading an existing OTDS deployment.

9.2 Preparing for upgrade to the current version using the OpenText Information Archive Helm chart

9.2.1 Downloading and extracting the current version OpenText Information Archive Helm chart

1. Download the current version of the OpenText Information Archive Helm chart from the Registry. For more information, see Extracting the packages.
2. Extract the package next to your previous OpenText Information Archive Helm chart. You should have the following structure:

```
infoarchive-<CURRENT_VERSION>-n-k8s-helm
├── customers
│ └── iacustomer
│ ├── ingress
│ └── overrides-general.yaml
│
├── infoarchive
│ └── templates
│
├──infoarchive-automation
│ └── bin
│ ├── configuration.yml
│ ├── infoarchive-automation-linux
│ └── infoarchive-automation-win.exe
│
├── password-generator
│ ├── bin
│ │ ├── helm-password-gen.sh
│ │ └── password-encrypt
│ ├── config
│ └── lib
│
└── platforms
  ├── cfcr.yaml
  ├── gcp.yaml
```

```
  └── …
infoarchive-<PREVIOUS_VERSION>.n-x-k8s-helm
```

3. Copy all override values from override files from 24.3 folders (customers, platforms, *etc.*) into their corresponding locations in the values files for the current release.

9.2.2 Preparing the customer folder

To prepare the customer folder:

1. Create the customer folder:

```
> cd infoarchive-<CURRENT_VERSION>-n-k8s-helm/customers
> mkdir <CUSTOMER_NAME)
> cp iacustomer/* <CUSTOMER_NAME)
```

2. Copy the customer-specific configuration from the 24.3 customer `../customer/<CUSTOMER_NAME>/overrides-general.yml` file into the `helm/customers/acme/overrides-general.yml` file.

Below is a snippet of the current release's `customer/acme/overrides-general.yml` file (for `transactionOption` `m1`, `m2`, `m3`, `m4`; for `ms` `transactionOption`, make sure to configure the `ms:` section).

```
iawaPublicHostname: iawa.acme.infoarchive.ot.net
otdsPublicHostname: otds.acme.infoarchive.ot.net
:
:
postgres:
  deployment:
    type: external
  system:
    external:
      host: FQDN for postgres-system.acme.infoarchive.ot.net
    port: 5432
    superuser:
      username: name of postgres user
    dbowner:
      username: name of database owner user
  structuredData:
    external:
      host: FQDN for postgres-structured-data.acme.infoarchive.ot.net
    port: 5432
    superuser:
      username: name of postgres user
    dbowner:
      username: name of database owner us
```

By default, the support for XProc is disabled. If you are upgrading from 23.4, you should keep it enabled using the key `ias.xproc.support.disabled` by setting it to `false`.

Depending on existing definitions of your PVC definitions, you may or may not need to set the `pvcMetadataAnnotationsEnabled` flag. That will depend on whether you are upgrading the system-deployed OOTB as of version 21.4 or 21.2 or before. You can always describe on your PVCs to see if `annotation: volume.beta.kubernetes.io/storage-class` is on your PVCs.

If your PVCs contain `annotations->volume.beta.kubernetes.io/storage-class` as, for example, shown below, you will need to set `pvcMetadataAnnotationsEnabled` to true.

```
apiVersion: v1
 kind: PersistentVolumeClaim
  metadata:
  name: ias-data-root-pvc
  annotations:
    volume.beta.kubernetes.io/storage-class: <your storage class>
```

If your PVCs do not have that annotation, you do not need to set the property. Note that the value of `storage-class` may vary from deployment to deployment.

9.2.3 Optional – NewRelic APM support

OpenText Information Archive supports NewRelic Application Performance Monitoring (APM). In order to enable NewRelic APM, set `newRelic.enabled` to true and ensure there are no empty/blank values for any of the specific `newRelic` keys. Ensure you have a proper NewRelic licence key.

The only keys that are optional and may be left blank are proxy user and password. However, when you are leveraging proxy, ensure you set both user and password to the correct credentials for the proxy user.

Please see below for the relevant keys. Add these keys to the `customer/overrides-general.yaml` file and make sure you set the right values for them. `ENV`, `PLATFORM`, `CELL`, `ZONE`, `REGION`, `DC`, `BU` and `CUSTOMER` are used to name the application in NewRelic and assigning labels to it using this pattern: `Name: NewRelicQueueMetricCollector-$(ENV)-$(PLATFORM)_$(CELL)_$(ZONE)_$(DC)-$(BU) Labels: ENV:$(ENV);Platform:$(PLATFORM);Cell:$(CELL);Zone:$(ZONE);DC:$(DC);BU:$(BU).`

```
newRelic:
  #is newRelic enabled or not
  enabled: false
  ENV: ""
  PLATFORM: ""
  CELL: ""
  ZONE: ""
  REGION: ""
  DC: ""
  BU: ""
  CUSTOMER: ""
  NEW_RELIC_LOG_FILE_NAME: ""
  NEW_RELIC_PROXY_SCHEME: ""
  NEW_RELIC_PROXY_HOST: ""
  NEW_RELIC_PROXY_PORT: ""
  NEW_RELIC_DISTRIBUTED_TRACING_ENABLED: "true"
  NEW_RELIC_SEND_DATA_ON_EXIT: "true"
  NEW_RELIC_EXPLAIN_ENABLED: "false"
  NEW_RELIC_RECORD_SQL: "off"

  NEW_RELIC_LICENSE_KEY: ""
  # proxy settings
  NEW_RELIC_PROXY_USER:
  NEW_RELIC_PROXY_PASSWORD:
```

9.2.4 Preparing for the upgrade

Scale down and delete `infoarchive-ias statefulset(s)` and `infoarchive` deployment resources. Ideally, this should be done when there are no jobs, ingestions, etc. running. For example:

```
>    kubectl scale --replicas=0 statefulset/infoarchive-ias
>    kubectl delete statefulset infoarchive-ias
>    kubectl scale --replicas=0 deployment/infoarchive-iawa
```

9.2.4.1 Backup PostgreSQL and PVCs

With IAS instance(s) down, we can now safely backup the current state of the PostgreSQL databases. Please work with your database provisioner to take a backup of every PostgreSQL instance. These backups will be crucial in case the upgrade fails and/or we have to roll back to this point in time for any reason. You can only proceed further if PostgreSQL data has been backed up.

Also, backup your PVCs, especially `ia-data-root-pvc`.

9.2.5 Upgrading using the current OpenText Information Archive Helm chart

Ensure all PostgreSQL instances are running.

9.2.5.1 Running the Helm upgrade

To run the Helm upgrade:

1. Use the following command to run the Helm upgrade.

   You will need to set a replica count for the IA Server to `1` instead of the usual number. The following example shows how to set the number of replicas for ms transaction type to 1. If running m2 through m4 transaction type, adjust accordingly (for example, `--set replicas.m1.ias=1`). If you are running ms-transaction, set the following instead: `--set ms.ias.ingestion.replicas=1`.

```
> cd infoarchive-<CURRENT_VERSION>-n-k8s-helm
> helm upgrade \
    --namesapce <CUSTOMER_NAMESPACE> \
    --timeout 30000s \
    --set-file
external.creds=customers/<CUSTOMER-NAME>/password-encrypt/creds.base64 \
    --set-file
external.keystore=customers/<CUSTOMER_NAME>/password-encrypt/keystore.jceks.base64 \
    --set-file
external.truststore=customers/<CUSTOMER_NAME>/tls/client/truststore.pkcs12.base64 \
    --set-file
external.iawaCert=customers/<CUSTOMER_NAME>/ingress/https/iawa.cer \
    --set-file
external.secretStore=customers/<CUSTOMER_NAME>/password-encrypt/
secretStore.uber.base64 \
    --values platforms\<PLATFORM>.yaml \
    --values customers/<CUSTOMER_NAME>/overrides-general.yaml \
    --values customers/<CUSTOMER_NAME>/overrides-passwords.yaml \
    --set replicas.m1.ias=1 \
```

```
    infoarchive \
    infoarchive
```

2. Check that the container images have the tag `<CURRENT_VERSION>.x-y`.
3. Verify that IA Server instances have started running.
4. Verify that once the IA Server instance(s) is/are running, job: `infoarchive-first-time-setup-upgrade-*` started and ran as well. You can tail the log of that job to confirm. When successful, the job will exit at the end.
5. Once the job is completed, set the number of IA Server replicas to the desired number instead of 1. Use following command to set the IA Server to 3 replicas for m1 through m4 transaction types:

```
> kubectl scale --replicas=3 statefulset/infoarchive-ias
```

For ms-transaction type, use the following instead:

```
> kubectl scale --replicas=3 statefulset/infoarchive-ias-ingestion
```

9.2.6 Verifying access to OpenText Information Archive

To verify access to OpenText Information Archive:

1. Log in to the IA Web App.
2. Click  and select **About Information Archive**. Ensure that the current version is listed for the **IAWA Version** and **IAS Version**.
3. Ensure that you can access your applications in the IA Web App and to run searches and jobs.

9.2.7 Restoring after a failed upgrade

If the upgrade failed, perform the following steps to restore the previous version.

Note: Below you will see examples for commands for `m1 transactionOption`, which has one IA Server and one IA Web App running. If you are running another `transactionOption` (for example, `ms`) and, therefore, you have multiple instances of IA Server or IA Web App PODs, you will need to account for that when scaling and deleting various Kubernetes resources.

To restore the previous version:

1. Scale down all components:

```
> kubectl scale --replicas=0 deployment/infoarchive-iawa
> kubectl scale --replicas=0 statefulset/infoarchive-ias
```

2. Delete `IAS statfulset infoarchive-ias`:

```
> kubectl delete statefulset infoarchive-ias
```

3. Delete services to prevent `ClusterIP-related` errors reported by Helm3 rollback. The services will be added back when the rollback operation is performed.

```
> kubectl delete service ias
> kubectl delete service iawa
```

4. Delete IA Server's temporary PVCs:

```
> kubectl delete pvc ia-logs-infoarchive-ias-0
> kubectl delete pvc ia-ls-node-infoarchive-ias-0
> kubectl delete pvc ia-ls-temp-infoarchive-ias-0
```

To restore the previous OTDS version:

1. Uninstall the <OTDS_CURRENT_VERSION> version to free up the public host name.
2. Drop the `otdsdb` PostgreSQL database.
3. Restore <OTDS_PREVIOUS_VERSION> `otdsdb` from your backup.
4. Redeploy the <OTDS_PREVIOUS_VERSION> using the previous OTDS version's Helm chart.

9.2.7.1 Restoring PostgreSQL instances and PVCs

Before we can rollback Helm templates to the previous version's snapshots, we will have to restore all PostgreSQL instances to the pre-upgrade backups. Work with your provisioner to ensure PostgreSQL instances are restored to the last backups taken during our pre-upgrade step. When all PostgreSQL instances have been restored to the pre-upgrade state, which is a last snapshot of the previous version's state, we can go ahead and do the Helm rollback in the next section. Also restore your PVCs (especially `ias-data-root-pvc`).

9.2.7.2 Rolling back OpenText Information Archive to the previous version

To roll back to the previous version:

1. Run Helm 3 rollback.

Note: In the example below, we are rolling back to version 1. This may be different in your environment. You will have to roll back to the version corresponding to the previous version.

```
> helm list
> helm history infoarchive
# Look at the output and choose the release number to roll back to. Normally this
value should be 1, which we will use in the next command.
> helm rollback infoarchive <release number to roll back to>
```

2. Make sure OTDS is accessible.

3. Make sure the services are present:

```
> kubectl get service ias
> kubectl get service iawa
```

4. Scale down all components again:

```
> kubectl scale --replicas=0 deployment/infoarchive-iawa
> kubectl scale --replicas=0 statefulset/infoarchive-ias
```

5. Scale up all component:

```
> kubectl scale --replicas=1 statefulset/infoarchive-ias
> kubectl scale --replicas=1 deployment/infoarchive-iawa
```

6. Check that the container images have the tag corresponding to your previous version's configuration.
7. Verify that you can access the previous version's IA Web App.
8. Ensure that you can access your applications in IA Web App and run searches and jobs.

Chapter 10

Appendix A – Platform values files

10.1 GCP

See the `helm/platforms/gcp.yaml` file for a sample for GCP and GCE. This file shows the use of `NFS Fileshare`.

10.2 Azure

See the `helm/platforms/azure.yaml` file for a sample for Azure and AKS. This file shows use of `azurefile-standard` storage class.

10.3 AWS

See the `helm/platforms/aws.yaml` file for a sample for AWS and EKS.

10.4 OpenShift on AWS

See the `helm/platforms/ocp.yaml` file for a sample for OCP on AWS.

10.5 CFCR

See the `helm/platforms/cfcr.yaml` file for a sample for CFCR.

10.6 Customer values file

See the `.../customers/iacustomer/overrides-general.yaml` file for a sample of a customer override file.

Chapter 11

Appendix B – Troubleshooting

- Make sure the FQDNs of IA Web App and OTDS are reachable from your browser, and make sure to use `https://` in the URL.
- Make sure you can log into OTDS using the `admin` credentials.
- Use the `helm lint` command with the same applicable parameters as the `helm install` command to check validity of Helm values files.
- Use the `helm template --debug` command with same, applicable parameters as the `helm install` command to generate and inspect the generated `kubectl config` files.
- You can view the `kubectl` configuration of the deployed Helm chart using the following command:

```
> helm get infoarchive
```

11.1 Troubleshooting Vault integration

Vault has rather complex configuration so troubleshooting integration may be somewhat tricky. Here are some main points that can be typically looked it:

- Incorrect Vault's hostname, port or not reachable host – PODs for the IA Server will be stuck on the initialization step. One of the `init` containers for these PODs will attempt to check whether the host/port is accessible and if these values are not correct or if the host i.e. will be blocked by network configuration – these PODs will just continue to be stuck during initialization. You can do "describe" on such POD and look for step `infoarchive-wait-for-vault`. Normally this step is run very quickly as long as host/port are reachable. If you see the `state` of that step is stuck on **Running** phase for some time – that is an indication that Vault, as currently configured, is not accessible. Check hostname and port to ensure values are correct. If they are, then ensure there is a proper network connectivity between PODs in your cluster and the Vault. Below is the step that is stuck on Running phase just like that:

```
infoarchive-wait-for-vault:
    Container ID:  docker://
fb75537d7b44431e81efabdaac79ec33dab024107e1b58088ae560e162e3e4df
    Image:         <cloud>/<your cluster>/busybox:latest
    Image ID:      docker-pullable://<cluster>/
busybox@sha256:a7766145a775d39e53a713c75b6fd6d318740e70327aaa3ed5d09e0ef33fc3df
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
      -c
    Args:
      until nc -w 3 -z [vault host name here] 8200 ; do echo Waiting for vault... ;
sleep 5 ; done
>       State:          Running
```

- Wrong protocol (http or https). When specifying incorrect protocol type, PODs will fail to connect displaying error message that will depend on type of authentication method being utilized. For example, when using CUBBYHOLE authentication, the message in the log may look similar to the one below (note – due to errors, PODs in example below would just error out and crash. Also note that this particular error message is very generic and can be indicative of several other issues as well so it may not necessarily mean that protocol is specifically wrong here):

```
14:30:01.859 [main] ERROR org.springframework.boot.SpringApplication - Application
run failed
java.lang.IllegalArgumentException: Initial Token (spring.cloud.vault.token) for
Cubbyhole authentication must not be empty
>    at ….
```

- Wrong protocol for APPROLE authentication may look as follows:

```
4:53:49.084 [main] ERROR org.springframework.boot.SpringApplication - Application
run failed
org.springframework.vault.authentication.VaultLoginException: Cannot login using
AppRole: Client sent an HTTP request to an HTTPS server.
>    ; nested exception is org.springframework.web.client.HttpClientErrorException
$BadRequest: 400 Bad Request: "Client sent an HTTP request to an HTTPS server.<EOL>"
```

- Wrong protocol for TOKEN authentication will throw similar error message:

```
14:58:15.641 [main] ERROR org.springframework.boot.SpringApplication - Application
run failed
org.springframework.vault.VaultException: Status 400 Bad Request [secret/
iacustomer/ias/vault]: Client sent an HTTP request to an HTTPS server.
>    ; nested exception is org.springframework.web.client.HttpClientErrorException
$BadRequest: 400 Bad Request: "Client sent an HTTP request to an HTTPS server.<EOL>"
```

- Incorrect TOKEN (non-existing, expired, etc.) may produce error message similar to this one:

```
15:03:12.629 [main] ERROR org.springframework.boot.SpringApplication - Application
run failed
org.springframework.vault.VaultException: Status 403 Forbidden [secret/
iacustomer/ias/vault]: permission denied; nested exception is
org.springframework.web.client.HttpClientErrorException$Forbidden: 403 Forbidden:
"{"errors":["permission denied"]}<EOL>"
```

- Incorrect path to secrets for properties specified using keys `vault.kv.application`, such as `ias`, `iawa`. `shell`, or `otdsinit`. If the path specified or the `vault.kv.backend` property are not correct, an error message similar to the following is issued (note the `iacustomer` in path is misspelled as `iacustomerx`):

```
15:11:12.082 [main] ERROR org.springframework.boot.SpringApplication - Application
run failed
org.springframework.vault.VaultException: Status 403 Forbidden [secret/data/
iacustomerx/ias/vault]: 1 error occurred:
* permission denied
; nested exception is org.springframework.web.client.HttpClientErrorException
$Forbidden: 403 Forbidden: "{"errors":["1 error occurred:\n\t* permission denied\n
\n"]}<EOL>"
```

11.2 Cubbyhole troubleshooting

- Since Cubbyhole authentication is the most complex, troubleshooting it is also the most complicated. The majority of incorrect configurations for Cubbyhole impact `init-container`, as it is responsible for pre-fetching the secret ID, fetching the token, and preparing configuration for the POD. All this work may fail without any visible errors, but the errors will be reported shortly after POD's startup. The `init-container` is currently performing the following steps:
  - Logging in using helper's role credentials to obtain valid token
  - Using that token, it attempts to fetch secret id for the main role
  - Uses main role's ID and secret ID (retrieved in the previous step) to obtain a wrapped token and prepares POD's configuration using wrapped token
- Any of these steps may fail and currently some of these failures may be silent. The end result of all these failures is a lack of token in POD's configuration, which produces error like the one below:

```
5:18:36.426 [main] INFO infoarchive.ias.console - Loading Spring application
context...
15:18:38.828 [main] ERROR org.springframework.boot.SpringApplication - Application
run failed
java.lang.IllegalArgumentException: Initial Token (spring.cloud.vault.token) for
Cubbyhole authentication must not be empty
>
```

Ensure the helper's role ID and secret ID, as specified in configuration, are valid. Use the Vault CLI to verify the IDs.

Example 11-1: Sample values for the role_id and secret_id:

```
>vault write -address=https://<vault host>:8200 -ca-cert=vault_ca.crt auth/approle/
login  role_id="0e191143-e8c3-7f8a-17a4-cb3c03c0f394"
secret_id="a430d7be-3804-6eed-0bce-61f9899c649e"
>    helm get infoarchive
```

Ensure that you will receive a token in response. If not, perhaps the role ID or secret ID for the helper role are incorrect. Re-fetch them.

- Check the rest of the helper's role configuration, namely `secretKeyName` and `secretPath`. Ensure both values are correct. Use the Vault CLI to further confirm that you can fetch secret ID for the main role from Vault. See an example of the call below. In the example: `secretKeyName` is `secret-id`, `secretPath` is `helper/iacustomer`, and `vault.kv.backend` is `secret`. Also note that the call below reads the token from `VAULT_TOKEN` environment, which has been pre-seeded with the value for the token obtained in previous step:

  Example 11-2: Call example:

```
>vault kv get -address=https://<vault host>:8200 -ca-cert=vault_ca.crt -
field=secret-id secret/helper/iacustomer
>
```

– Ensure you will receive the secret ID for the main role in response. If not, ensure the paths and secret ID key names are correct.

  Lastly, if you are connecting to Vault over TLS/HTTPS, ensure that the `vault.cacert` key is set correctly and that it has corresponding Base64–encoded file set as part of Helm install command using syntax: `--set-file external.vaultCaCert=…` as explained in the configuration steps.

  Similarly, if you see errors in the log with message similar to: `PKIX path building failed`, this is most likely due to either Vault's CA certificate missing from configured truststore or problems with your Vault's CA certificate, – as that indicates that OpenText Information Archive components cannot establish the TLS handshake with Vault's instance due to lack of trust. Ensure Vault CA certificate is correct and ensure it has been added to our truststore.

11.3 Errors received while configuring internalProxies as part of IA Web App/Gateway Tomcat

Issue: The following error messages are received: "Load Balancer Bad Request" and "OTDS Invalid Request":

You may encounter these two errors if you have an untrusted Load Balancer or an internal proxy in front of the IA Web App. To resolve the issue, follow the instructions documenting how to properly handle these proxies in section 9.3.1 "Handling X-Forwarded-Host and host headers" in *OpenText Information Archive - Installation Guide (EARCORE-IGD)*.

Chapter 12

Appendix C – Transaction options

12.1 M1

This is a pre-configured option, and its values should not be changed, as per the license.

Component	Element	Value
IA Web App	POD # of Replicas	1
	CPU(m)/POD	2000
	RAM(G)/POD	4
IA Server	POD # of Replicas	3
	CPU(m)/POD	2000
	RAM(G)/POD	4

12.2 M2

This is a pre-configured option, and its values should not be changed, as per the license.

Component	Element	Value
IA Web App	POD # of Replicas	2
	CPU(m)/POD	2000
	RAM(G)/POD	4
IA Server	POD # of Replicas	6
	CPU(m)/POD	2000
	RAM(G)/POD	4

12.3 M3

This is a pre-configured option, and its values should not be changed, as per the license.

Component	Element	Value
IA Web App	POD # of Replicas	4
	CPU(m)/POD	2000
	RAM(G)/POD	4
IA Server	POD # of Replicas	12
	CPU(m)/POD	2000
	RAM(G)/POD	4

12.4 M4

This is a preconfigured option, and its values should not be changed as per the license.

Component	Element	Value
IA Web App	POD # of Replicas	4
	CPU(m)/POD	2000
	RAM(G)/POD	4
IA Server	POD # of Replicas	16
	CPU(m)/POD	2000
	RAM(G)/POD	4

12.5 MS

This is a flexible option for deploying OpenText Information Archive to Kubernetes. In this option IA Web App and IA Server are deployed in two distinct tracks:

- Search
- Ingestion

In addition, a separate IA Server is deployed for background processing.

For an example configuration, see the `.../infoarchive/values.yaml` file.

Component		Element	Value
IA Web App	Search	POD # of Replicas	See the values file for sample values.
		CPU(m)/POD	

Component		Element	Value
		RAM(G)/POD	
	Ingestion	POD # of Replicas	
		CPU(m)/POD	
		RAM(G)/POD	
IA Server	Search	POD # of Replicas	
		CPU(m)/POD	
		RAM(G)/POD	
	Ingestion	POD # of Replicas	
		CPU(m)/POD	
		RAM(G)/POD	
	Background Processing	POD # of Replicas	
		CPU(m)/POD	
		RAM(G)/POD	

Chapter 13

Appendix D – Acronyms glossary

Acronym	Expansion
AKS	Azure Kubernetes Service
AWS	Amazon Web Services
CFCR	Cloud Foundry Container Runtime
CLI	Command Line Interface
EKS	Elastic Kubernetes Service
FQDN	Fully Qualified Domain Name
GCP	Google Cloud Platform
GCR	Google Container Registry
GKE	Google Kubernetes Engine
IAS	OpenText Information Archive Server, also known as IA Server
IA Web App	OpenText Information Archive Web Application, also known as IA Web App
OCP	Red Hat OpenShift Container Platform
OTDS	OpenText Directory Services