# TIBCO WebFOCUS® Container Edition Installation and Deployment Guide

## Version 1.0.2 GA

*July 2022*

## Contents

## Introduction

TIBCO WebFOCUS® Container Edition (CE) provides a scalable, microservices-based analytical platform designed for container-based deployments. This new offering packages TIBCO WebFOCUS® software components in a container image, including all required OS components and libraries, which can be deployed and run in a Kubernetes cluster.

The following are several benefits and advantages you can leverage using TIBCO WebFOCUS® CE:

- **Automatic deployments**. Identify which (and how many) WebFOCUS components (for example, TIBCO WebFOCUS® Reporting Server) need to be deployed and where (e.g., specific clusters).

- **Reduce operational and infrastructure costs**. Add or remove services and resources to the Kubernetes cluster as needed. Data sources can exist in the cloud (private, public), or on-premises.

- **Increase adoption and usage**. Develop specific BI applications for targeted users on-premises and deploy to the cloud as needed. When you install WebFOCUS on Kubernetes, development and testing can occur within internal clusters or external clusters.

- **Portability**. Fully portable between cloud and on-premises environments.

- **Dynamic scaling**. Rapidly adjust to increased user and data demands.

Existing Kubernetes services that are provided by cloud providers can also be leveraged (for example, Amazon® Elastic Kubernetes Service (EKS)).

Deploying TIBCO WebFOCUS® CE on Kubernetes provides a variety of deployment options in the private cloud, which can be centrally managed and administered based on dynamic workloads.

Docker images enable key components and functionality of the TIBCO WebFOCUS® platform to be isolated, packaged into specific containers and run on Kubernetes. As a result, each WebFOCUS component can be scaled independently to accommodate users, data, and data sources on demand.

## Kubernetes Overview

Kubernetes® (K8s) is an open-source system for running, managing, and orchestrating containerized applications in a cluster of servers (known as a Kubernetes cluster). Kubernetes clusters can run in any cloud environment (e.g., private, public, hybrid) or on-premises.

A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node.

The size of a Kubernetes cluster (determined by the number of nodes) depends on the application workload(s) that will be running. For example, each node can represent an 8 core / 64 GB RAM system. Pods are the basic objects in a Kubernetes cluster, which consists of one or more software containers. The worker node(s) host the Pods that are the components of the application workload. Usually, only one container runs in a Pod. However, multiple containers can be run in a Pod if needed (depending on specific environment requirements). If one Pod fails, Kubernetes can automatically replace that Pod with a new instance.

Key benefits of Kubernetes include automatic scalability, efficient load balancing, high availability, failover/fault tolerance, and deploying updates across your environment without any disruptions to users.

## What's New in Version 1.0.2 GA?

This section provides a summary of the new components that are packaged with TIBCO WebFOCUS® Container Edition (CE) version 1.0.2 GA in the .tar file (*TIB_wfce_1.0.2.tar*).

- Includes the latest version (9.0.2) of TIBCO WebFOCUS® Client and TIBCO WebFOCUS® Reporting Server.
- Includes the Dockerfile for cachemanager. This custom cachemanager image contains all of the required libraries (for example, npm).
- Includes the Dockerfile for postgres-alpine. This image is used by PostgreSQL and contains all of the required libraries and dependencies (for example, jq and curl).
- Includes the Helm charts for postgresql, etcd, solr, prometheus, prometheus adapter, and raw. As a result, charts are not required to pull from Helm repositories and can use a local chart path.
- Supports air gap installation/deployment. The required steps and commands are provided, which allow you to use your own repository instead of the default repository path and tags.
- Includes five TIBCO images instead of three. The two new images are for cachemanager and postgres-alpine to support air gap installation/deployment.

## Requirements and Prerequisites

This section describes requirements and prerequisites for deploying TIBCO WebFOCUS® Container Edition (CE) in a Kubernetes cluster.

**Red Hat Enterprise Linux UBI Version 8.x**

A Linux environment is required to build your Docker images. TIBCO WebFOCUS® CE Release 1.0.2 GA is currently certified on Red Hat Enterprise Linux UBI Version 8.x.

**Docker Version 19 or Higher**

If Docker is currently not installed, use the links below and follow the steps to install Docker on your machine:

https://docs.docker.com/engine/install

- For Mac (macOS): https://docs.docker.com/docker-for-mac/install

To verify the Docker version currently installed, enter the following command in your terminal window:

```
$> docker version
```

**Kubernetes Cluster Version 1.19.7 or Higher**

Users can consult with their Kubernetes administrator to provision the best available Kubernetes cluster on their environment (such as HA, non-HA control-plane, and so on). A Kubernetes cluster running on Amazon® Elastic Kubernetes Service (EKS) is recommended.

- The Kubernetes cluster must use CNI that supports UDP transport.
- Kubernetes must have access to the registry where TIBCO WebFOCUS® CE component images are available.
- The Kubernetes cluster must support dynamic provision of volumes (pv).
- The common volume should be 30 to 40 GB.
- If you are using a back-end data source/database that is running outside (external) of the Kubernetes cluster, then ensure it is running somewhere where the cluster has access to it.

- Each node in the Kubernetes cluster must be sufficiently powerful:
    - 32 GB RAM
    - 8 CPU 3.x Ghz
    - 50 GB Available Disk Space

**Note:** The individual who is responsible for installing TIBCO WebFOCUS® CE must have admin privileges over the Kubernetes namespace where WebFOCUS CE® will be installed.

To verify the Kubernetes version currently installed, enter the following command in your terminal window:

```
$> kubectl version
```

**kubectl**

The Kubernetes command-line tool, kubectl, allows you to run commands against Kubernetes clusters. You can use kubectl to deploy applications, inspect and manage cluster resources, and view logs.

If kubectl is currently not installed, use the following link to download and install kubectl (Linux, macOS):

https://kubernetes.io/docs/tasks/tools

**PersistentVolume (PV) Storage Class Provisioner**

Ensure that a storage provisioner is available in the Kubernetes cluster. To verify that a storage class exists, enter the following command in your terminal window:

```
$> kubectl get sc
```

If a default storage class is currently not available, use the link below and follow the steps to enable *local storage* for your environment:

https://github.com/rancher/local-path-provisioner

To enable the default storage class, enter the following command in your terminal window:

```
$> kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-
provisioner/v0.0.21/deploy/local-path-storage.yaml
```

You can enter the following command in your terminal window to make *local-path-storage* (referenced in the above link) a default storage class:

```
kubectl patch storageclass local-path -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

### ReadWriteMany Access Mode

For deployment scaling and multi-node Kubernetes clusters, ensure your PersistentVolume (PV) is set to the *ReadWriteMany* access mode.

### Kubernetes Context

Ensure your context is set. To display a list of contexts currently being used, enter the following command in your terminal window:

```
$> kubectl config get-contexts
```

If required, enter the following command in your terminal window to set the context:

```
$> kubectl config use-context <my-cluster-name>
```

### Helm Version 3.5.0 or Higher

Helm is the “package manager” for Kubernetes (K8s), which is used to install charts (“packages”) into K8s. You can consider charts as packaged applications. Charts are your versioned, pre-configured application resources, which can be deployed as one unit. A Helm chart (.tgz file) contains templates that define Kubernetes resources to install.

For more information, reference the following link:

https://helm.sh/docs/intro/install

If Helm is currently not installed, use the links below and follow the steps to install Helm on your machine:

- For Linux:
    - https://helm.sh/docs/intro/install/#from-apt-debianubuntu
    - https://helm.sh/docs/intro/install/#from-pkg-freebsd
- For Mac (macOS): https://helm.sh/docs/intro/install/#from-homebrew-macos
- Github releases: https://github.com/helm/helm/releases

To verify the Helm version currently installed, enter the following command in your terminal window:

```
$> helm version
```

**Helmfile Version 0.138.7 or Higher**

Helmfile adds functionality to Helm by wrapping it in a declarative specification that allows you to compose several charts together to create a comprehensive deployment artifact, from a single application to your entire infrastructure stack.

For more information, reference the following link:

https://github.com/roboll/helmfile#installation

To download Helmfile, reference the following link:

https://github.com/roboll/helmfile/releases

- macOS (using homebrew): `brew install helmfile`

To verify the Helmfile version currently installed, enter the following command in your terminal window:

```
$> helmfile version
```

## Component Version Reference

The following table lists versions of all the ancillary components that were tested with TIBCO WebFOCUS® Container Edition (CE) for Release 1.0.2.

| Component | Version |
|---|---|
| Kubernetes | 1.20 |
| Docker | 20.10.8 |
| Kubeadm | 1.22.0 |
| Kubectl | 1.22.0 |
| Helm | 3.6.3 |
| Helmfile | 0.139.9 |
| Elastic Kubernetes Service (EKS) Platform | eks.2 |
| Eksctl | 0.61.0 |
| Kubernetes Storage Class (/local-path-provisioner) (Rancher) | 0.0.20 |

| | |
|---|---|
| Kubernetes Storage Class (efs.csi.aws.com) | 1.3.3 (Helm chart: 2.1.5) |
| Kubernetes NGINX Ingress controller | 0.40.2 (Helm chart: 3.4.1) |
| Swego | 1.16.0 |
| Octant | 0.13.1 |
| PostgreSQL | 11.12.0 (Helm chart: 10.4.5) |
| Apache Solr | 8.8.2 (Helm chart: 0.3.2) |
| Etcd | 3.4.14 (Helm chart: 5.3.0) |
| Node.js | 14.0.0 |
| Prometheus Server (prometheus) | 2.31.1 (Helm chart: 15.0.0) |
| Prometheus Adapter (prometheus-adapter) | 0.9.1 (Helm chart: 3.0.0) |

## Docker Images Used by TIBCO WebFOCUS® Container Edition

The following table provides a reference for the Docker images that are used (pulled) by TIBCO WebFOCUS® Container Edition (CE).

| **Service Pod Name** | **Docker Image** |
|---|---|
| appserver | redhat/ubi8 |
| cachemanager | node:14 |
| clm | redhat/ubi8 |
| edaserver | redhat/ubi8 |
| etcd | bitnami/etcd:3.4.14-debian-10-r0 |
| postgresql | postgres:alpine<br>postgresql:11.12.0-debian-10-r1 |
| prometheus-server | k8s.gcr.io/prometheus-adapter/prometheus-adapter:v0.9.1<br>quay.io/prometheus/prometheus:v2.31.1 |
| prometheus-adapter | k8s.gcr.io/prometheus-adapter/prometheus-adapter:v0.9.1 |

| | |
|---|---|
| reportcaster | redhat/ubi8 |
| solr | bitnami/solr:8.8.2-debian-10-r0 |
| zookeeper | bitnami/zookeeper:3.7.0-debian-10-r0 |
| swego | ishswar/swego:1.0.2<br>ployst/nginx-ssl-proxy<br>xmartlabs/htpasswd |
| metrics-server | metrics-server:v0.5.2 |

## Key Components/Systems

Prior to deploying TIBCO WebFOCUS® Container Edition (CE), ensure that the following systems are available in your environment with the corresponding component(s) installed accordingly.

**Note:** If required, you can also run everything from a single machine (from building the Docker images to deploying TIBCO WebFOCUS® CE in a Kubernetes cluster).

1. **Build Server.** After obtaining the *TIB_wfce_1.0.2.tar* file, this machine is used to build the required Docker images for TIBCO WebFOCUS® CE on the image registry.

   **Recommended Specifications:**

   - 16 GB Ram
   - 4 CPU 3.x Ghz
   - 50 GB Available Disk Space

   **Installed Software**

   - Docker

2. **Deployment Server.** Using the Helm charts that are provided with each Docker image (as is or modified as per end-user requirements), this machine is used to deploy TIBCO WebFOCUS® CE components on the Kubernetes cluster.

   **Recommended Specifications:**

   - 16 GB Ram
   - 4 CPU 3.x Ghz

- 20 GB Available Disk Space

**Installed Software**

- Helm and Helmfile
- Kubectl

3. **Kubernetes Cluster.** Ensure that your Kubernetes cluster is ready/available and each node in your cluster where TIBCO WebFOCUS® CE is being deployed supports the following recommended specifications:
   - 32 GB RAM
   - 8 CPU 3.x Ghz
   - 50 GB Available Disk Space

## Step 1: Unzip the TIB_wfce_1.0.2.tar File

TIBCO WebFOCUS® Container Edition (CE) is packaged and provided as a *TIB_wfce_1.0.2.tar* file. Unzip the *TIB_wfce_1.0.2.tar* file to a location on your file system.

## Step 2: Create Docker Images and Deploy Using the Helm Charts

Once you have downloaded and unzipped the *TIB_wfce_1.0.2.tar* file, you must build the Docker images that are required by TIBCO WebFOCUS® Container Edition (CE). There are two options you can use to build your Docker images:

- **Option 1:** Create all Docker images using the *build-image script.sh* script and deploy using Helm charts (*default, dev*, or *cloud* environment).
- **Option 2:** Create individual Docker images and deploy using Helm charts (*default, dev*, or *cloud* environment).

### Option 1: Create All Docker Images Using the build-images.sh Script

The commands in this procedure deploy TIBCO WebFOCUS® CE in the default configuration of a Kubernetes cluster. You can review default values that are defined in the *configuration.md* file. For more information, see Appendix A: Configuration Parameters Reference.

To create all Docker images for TIBCO WebFOCUS® CE using the *build-images.sh* script and deploy using Helm charts:

1. Enter the following commands in your terminal window/console:

```
cd <webfocus-ce>/scripts

./build-images.sh

cd <webfocus-ce>/scripts/helmfile/infra

source ../export-defaults.sh

helmfile sync

cd <webfocus-ce>/scripts/helmfile/

helmfile -e <env-name> sync
```

**Note:** You can also deploy using the `-e dev` environment option, which enables Swego and also opens ports for Cluster Manager (CLM) (Port: 31121) and WebFOCUS Reporting Server (Port: 31131):

```
helmfile -e dev sync
```

To list all deployed releases of TIBCO WebFOCUS® CE, enter the following command in your terminal window/console:

```
helm list
```

2. Access TIBCO® WebFOCUS using the following URL:

   `http://localhost:31080/webfocus`

   **Note:** If you need to access this URL from another machine, replace `localhost` with the hostname or IP of that machine, for example:

   `http://{hostname}:31080/webfocus`

   The default login credentials are:

   - Username: **secret**
   - Password: **terces**

   When deployed using the `-e dev` environment option, the URLs for CLM and WebFOCUS Reporting Server URLs are available as follows:

   - **CLM:** `http://localhost:31121`
   - **WebFOCUS Reporting Server:** `http://localhost:31131`

## Option 2: Create Individual Docker Images

The commands in this procedure deploy TIBCO WebFOCUS® CE in the default configuration of a Kubernetes cluster. You can review default values that are defined in the *configuration.md* file. For more information, see Appendix A: Configuration Parameters Reference.

To create individual Docker images for TIBCO WebFOCUS® CE and deploy using Helm charts:

1. Create the Docker image for WebFOCUS Reporting Server:

   ```
   cd <webfocus-ce>/wfs/iserver

   DOCKER_BUILDKIT=1 docker build --no-cache -t ibi2020/webfocus:wfs-9.0-1.0.2 .
   ```

2. Create the Docker image for iserver_etc and cachemanager:

   ```
   cd <webfocus-ce>/wfs/iserver_etc

   DOCKER_BUILDKIT=1 docker build --no-cache -t ibi2020/webfocus:wfs-etc-9.0-1.0.2 .

   DOCKER_BUILDKIT=1 docker build --no-cache -t ibi2020/webfocus:cachemanager -f
   Dockerfile.cachemanager .
   ```

3. Create the Docker image for WebFOCUS Client:

   ```
   cd <webfocus-ce>/wfc

   DOCKER_BUILDKIT=1 docker build --no-cache -t ibi2020/webfocus:wfc-9.0-1.0.2 .
   ```

4. Create Postgres-Alpine with the jq tool Docker image:

   ```
   cd <webfocus-ce>/scripts/helmfile/common/dockerfiles

   DOCKER_BUILDKIT=1 docker build --no-cache -t ibi2020/webfocus:postgres-alpine -f
   Dockerfile.postgres-alpine .
   ```

5. Enter the following commands in your terminal window/console:

   ```
   cd <webfocus-ce>/scripts/helmfile/infra

   source ../export-defaults.sh

   helmfile sync
   ```

   **Note:** You can skip the installation of some components from *infra* if they are not required (for example, Solr, Prometheus, Postgres). You can specify the setting to enable or disable installation for corresponding components in the configuration file. For more information, see Global Parameters.

```
cd <webfocus-ce>/scripts/helmfile/
```

```
helmfile -e <env-name> sync
```

**Note:** You can also deploy using the `-e dev` environment option, which enables Swego and also opens ports for Cluster Manager (CLM) (Port: 31121) and WebFOCUS Reporting Server (Port: 31131):

```
helmfile -e dev sync
```

To list all deployed releases of TIBCO WebFOCUS® CE, enter the following command in your terminal window/console:

```
helm list
```

6. Access TIBCO WebFOCUS® using the following URL:

   ```
   http://localhost:31080/webfocus
   ```

   The default login credentials are:

   - Username: **secret**
   - Password: **terces**

   When deployed using the `-e dev` environment option, the URLs for CLM and WebFOCUS Reporting Server URLs are available as follows:

   - **CLM:** `http://localhost:31121`
   - **WebFOCUS Reporting Server:** `http://localhost:31131`

## Changing the Default Namespace Used by Ancillary Software

Ancillary software (Apache Solr and PostgreSQL) that is provided with TIBCO WebFOCUS® CE is installed in the */infra* subfolder. By default, when you run the *helmfile sync* command on the */infra* subfolder, this subfolder is created in the *default* namespace.

To specify a different namespace:

1. Export the *INFRA_NS* environment variable to the required namespace as follows:

   ```
   export INFRA_NS=namespace_name
   ```

   This is the same technique used when exporting other environment variables (e.g., *WF_TAG*, *RS_TAG*, and so on).

2. Run the *helmfile sync* command from the */infra* subfolder.

Required pods will be created for the ancillary software under the specified namespace.

**Note:** If you currently have TIBCO WebFOCUS® CE deployed in a Kubernetes cluster, then you must first remove/delete this instance. For more information, see Removing (Undeploying) TIBCO® WebFOCUS Container Edition.

## Using the Swego Utility

Swego is a utility provided for TIBCO WebFOCUS® CE development (*dev*) environments that enables you to traverse through files in a container from a web browser.
It is useful for administrators who may not have access to containers that are running, but perform configuration modifications, and need to view resulting changes in the actual file system.

By default, the file system provides read-only access for users. Users can only view files from a web browser, but cannot make any changes directly.

You can change the default behavior from read-only (true) to false by modifying the Helm charts.

**Note:** It is not recommended to enable Swego for any other environments, especially *production* (*prod*), as it exposes configuration files to users. Swego is intended for testing purposes only.

Swego can be accessed using the following URL:

```
http://localhost:31081
```

## Deploying to an Amazon Web Services (AWS) Environment

You can also deploy TIBCO WebFOCUS® Container Edition (CE) in AWS if you are using Amazon Elastic Kubernetes Service (Amazon EKS).

To deploy in AWS, you must modify the following components in your configuration:

- **storage.storageClass** - Update this value with the correct storage class being used.
- **storage.access** - Must be set to *ReadWriteMany*.

During testing, the Amazon EFS Container Storage Interface (CSI) driver was used. This driver provides a CSI interface that allows Kubernetes clusters running on AWS to manage the lifecycle of Amazon EFS file systems.

For more information, reference the following link:

https://docs.aws.amazon.com/eks/latest/userguide/efs-csi.html

**Note:** Although there are other drivers available, the Amazon EFS CSI driver is recommended since it was tested with TIBCO WebFOCUS® CE. If you use any other storage driver, configuration settings may vary.

## Pulling an Image From a Private Docker Registry or Repository

If you are using a private Docker registry to pull images, then you must update the pull Secret in your configuration:

```
platform.dockerconfigjson
```

The value of the `.dockerconfigjson` field is a base64 representation of your Docker credentials.

## Configuring an Air Gap Installation Using Images From a Single Repository

An air gap installation places all required images and charts in a single repository, and only references this repository during deployment. It avoids downloading any component during runtime or from multiple repositories. Since all required charts are packaged in the *TIB_wfce_1.0.2.tar* file, no downloading occurs during runtime or from the respective repositories.

This section provides the required steps and commands to configure an air gap installation, which are summarized as follows:

1. Build/pull the required images.
2. Tag and push the images to your repository.
3. Update the repository path and tags in the *local.yaml.gotmpl* file.
4. Run the `sync` command with the `state-values-file` parameter.

For demonstration purposes, AWS ECR is used as an example repository where all images are stored.

**Step 1: Building/Pulling the Required Images**

The images referenced in the table below are required to run TIBCO WebFOCUS® CE. Five images are built from the *build-images.sh* script (*appserver*, *edaserver*, *clm*, *cachemanager*, and *postgres-alpine*). Any other (non-TIBCO) images that are required must be pulled from the respective repositories. The following table also includes the pull commands for non-TIBCO images.

| Service Pod Name | Description | Default Tag |
|---|---|---|
| appserver and reportcaster | Image is built from the *build_image* script. The base image used is “redhat/ubi8”. Once the image is created, *tag* the image and *push* the image to your repository. Scan the Docker image before issuing a *push* operation to the repository to check if the image has any vulnerabilities. | WF_TAG = wfc-9.0-1.0.2 |
| clm and edaserver | Image is built from the *build_image* script. The base image used is “redhat/ubi8”. Once the image is created, *tag* the image and *push* the image to your repository. Scan the Docker image before issuing a *push* operation to the repository to check if the image has any vulnerabilities. | RS_TAG = wfs-9.0-1.0.2 |
| eda-etc<br><br>(Static stored content of edaserver) | Image is built from the *build_image* script. The base image used is “redhat/ubi8”. Once the image is created, *tag* the image and *push* the image to your repository. Scan the Docker image before issuing a *push* operation to the repository to check if the image has any vulnerabilities. | ETC_TAG = wfs-etc-9.0-1.0.2<br><br>(From export-defaults.sh) |
| cachemanager | Same as above, additionally uses the node:14 image as well. | *cachemanager*, however you can overwrite before building the image or tag the image with a different name before issuing a push operation. |
| etcd | You must pull/download the **bitnami/etcd:3.4.14-debian-10-r0** image from Docker hub, tag the image, and push the image to your repository.<br><br>Pull command:<br><br>`docker pull bitnami/etcd:3.4.14-debian-10-r0` | 3.4.14-debian-10-r0 |
| postgresql | You must pull the **postgresql:11.12.0-debian-10-r1** image from Docker hub, tag the image, and push the image to your repository. | postgresql:11.12.0-debian-10-r1 |

| | | |
|---|---|---|
| | Pull command:<br>`docker pull bitnami/postgresql:11.12.0-debian-10-r1` | |
| postgresql-alpine | Image is built from the *build_image* script. The base image used is **postgres:alpine**. Once the image is created, *tag* the image and *push* the image to your repository. Scan the Docker image before issuing a *push* operation to the repository to check if the image has any vulnerabilities. | postgres:alpine |
| prometheus-server | Pull the **quay.io/prometheus/prometheus:v2.31.1** image from Docker hub.<br>Pull commands:<br>`docker pull quay.io/prometheus/prometheus:v2.31.1`<br>`docker pull jimmidyson/configmap-reload:v0.5.0` | |
| prometheus-adapter | Pull the **k8s.gcr.io/prometheus-adapter/prometheus-adapter:v0.9.1** image from Docker hub.<br>Pull command:<br>`docker pull k8s.gcr.io/prometheus-adapter/prometheus-adapter:v0.9.1` | v0.9.1 |
| solr | Pull the **bitnami/solr:8.8.2-debian-10-r0** image from Docker hub, tag the image, and push the image to your repository.<br>Pull command:<br>`docker pull bitnami/solr:8.8.2-debian-10-r0` | 8.8.2-debian-10-r0 |
| zookeeper | Pull the **bitnami/zookeeper:3.7.0-debian-10-r0** image from Docker hub, tag the image, and push the image to your repository.<br>Pull command: | 3.7.0-debian-10-r0 |

| | | |
|---|---|---|
| | `docker pull bitnami/zookeeper:3.7.0-debian-10-r0` | |
| metrics-server | metrics-server:v0.5.2<br><br>`docker pull rainbond/metrics-server:v0.5.2` | v0.5.2 |
| swego<br><br>(Optional if you enabled Swego) | Pull commands:<br><br>`docker pull ishswar/swego:1.0.0`<br><br>`docker pull ployst/nginx-ssl-proxy`<br><br>`docker pull xmartlabs/htpasswd` | |

**Step 2: Tagging and Pushing the Images to Your Repository**

Once you pull and build all the images, you must tag and push these images to your repository.

The following are examples of *pull*, *tag*, and *push* operations to an AWS ECR repository:

```
$> docker pull bitnami/etcd:3.4.14-debian-10-r0

$> docker tag bitnami/etcd:3.4.14-debian-10-r0 0123xxxxxx.dkr.ecr.us-west-
2.amazonaws.com/my_repo:tag_name

$> docker push 0123xxxxxx.dkr.ecr.us-west-2.amazonaws.com/my_repo:tag_name
```

**Note:** Ensure to scan the Docker image before issuing a *push* operation to the repository to check if the image has any vulnerabilities.

**Step 3: Updating the Repository Path and Tags in the local.yaml.gotmpl File**

A sample file (*local.yaml.gotmpl*) is provided in the environments directory:

```
/scripts/helmfile/environments/local.yaml.gotmpl
```

You can overwrite this sample file or create a new copy and update the file based on your environment settings. Optionally, you can place this file in any path (it is not required to be located only in the environments folder).

**Step 4: Running the Sync Command With the State-Values-File parameter**

This is an important step in the normal scenario where `helmfile sync` is run with only passing `-e <env-name>`. However, with the air gap installation/deployment, there is change in how the sync command is run. In this case, you must pass the file with the `state-values-file` parameter using the `helmfile sync` command as follows:

```
cd <webfocus-ce>/scripts/helmfile/infra

source ../export-defaults.sh

helmfile --state-values-file=<file-path> sync

cd <webfocus-ce>/scripts/helmfile

helmfile -e <env-name> --state-values-file=<file-path> sync
```

This command will deploy the cluster using the values that were specified in the file.

You can also overwrite other configuration parameters in the same file.

## Using Your Own PostgreSQL Database

You can use your own PostgreSQL database instance with TIBCO WebFOCUS® CE instead of the default. To point to your PostgreSQL database instance, edit the *wf.integ.yaml.gotmpl* environment file and update following *postgres* parameters with your host, username, and password.

```
- tenants:
     postgresUrl: jdbc:postgresql://postgresql.{{ env "INFRA_NS"  | default
"default" }}.svc.cluster.local:5432/{{ requiredEnv "PLATFORM_NAME" }}
eda?currentSchema=public
     postgresUser: {{ requiredEnv "PLATFORM_NAME" }}
     postgresPassword:

- global:
    config:
      DB_URL: "postgresql.{{ env "INFRA_NS"  | default "default"
      }}.svc.cluster.local:5432"

postgres:
     adminUsername: "postgres"
     adminPassword: "postgres123"
     host: "postgresql.{{ env "INFRA_NS"  | default "default"
     }}.svc.cluster.local"

- initHookDb:
     dbAdminUser: "postgres"
     dbAdminPass: "postgres123"
     dbHost: "postgresql.{{ env "INFRA_NS"  | default "default"
     }}.svc.cluster.local"
```

**Note:** Database changes are only applicable for new installations/deployments of TIBCO WebFOCUS® CE in a Kubernetes cluster.

## Using the NGINX Ingress Controller

The NGINX Ingress controller can be installed via Helm using the chart from the project repository. To install the chart with the release name *ingress-nginx*, enter the following commands in your terminal window/console:

```
$> helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

$> helm repo update

$> helm install ingress-nginx ingress-nginx/ingress-nginx
```

To detect which version of the ingress controller is running, enter the following commands in your terminal window/console:

```
$> POD_NAME=$(kubectl get pods -l app.kubernetes.io/name=ingress-nginx -o
jsonpath='{.items[0].metadata.name}')

$> kubectl exec -it $POD_NAME -- /nginx-ingress-controller --version
```

## Common Deployment Scenarios (Use Cases) for Kubernetes Clusters

This section describes common deployment scenarios (use cases) for TIBCO WebFOCUS® Container Edition (CE) when working with specific Kubernetes cluster configurations.

### Deploying to a Single-Node Kubernetes Cluster Running on the Same Machine

This scenario outlines how TIBCO WebFOCUS® CE can be deployed to a single-node Kubernetes cluster that is running on the same machine as all other ancillary components (for example, Helmfile, Docker, Kubernetes, and so on).

1. Install Docker.
2. Install Kubeadm.
3. Install Kubectl, Helm, and Helmfile.
4. Create a Kubernetes cluster.
5. Follow the installation steps described in Step 1: Unzip the TIB_wfce_1.0.2.tar File and Step 2: Create Docker Images and Deploy Helm Charts.

**Usage Considerations**

In a single-node Kubernetes cluster, you cannot scale the configuration (for example, adding a new node), unless you use images from a Docker registry and a storage provider that supports the *ReadWriteMany* access mode.

## Deploying to a Remote Kubernetes Cluster

This scenario outlines how TIBCO WebFOCUS® CE can be deployed to a remote Kubernetes cluster, where you will need to build Docker images and push these images to the registry.

In this scenario, access to the Kube config file (`~/.kube/config`) is required. You must ensure that the ancillary components (specifically, Kubectl, Helm, and Helmfile) on your local machine have connectivity to the remote Kubernetes cluster.

If your Kube config file has many cluster-info (contexts), then ensure you switch to the correct cluster context (the one you intend to use).

1. Create a Kubernetes cluster using Kubeadm or any of the supported cluster creation techniques.
2. On a local machine, install Docker, Kubectl, Helm, and Helmfile.
3. Follow the installation steps described inStep 1: Unzip the TIB_wfce_1.0.2.tar File and Step 2: Create Docker Images and Deploy Helm Charts.
4. Build your Docker images and push your images to the Docker registry to which your Kubernetes cluster has access.

   **Note:** You will need to update your pull Secret (*ibi-docker-hub*) and the repository name for your image registry (default is *ibi2020/webfocus*).
5. Run the Helmfile commands described in Step 2: Create Docker Images and Deploy Helm Charts.

**Usage Considerations**

If your remote Kubernetes cluster is multi-node and you want to scale TIBCO WebFOCUS® CE, then you must use a storage provider that supports the *ReadWriteMany* access mode.

If you are planning to scale the WebFOCUS Client (AppServer), then you will require an Ingress controller (for example, NGINX) so you can load balance incoming browser traffic requests between application servers. A sample Ingress controller YAML file is provided with the product.

You can generate a sample YAML file (Kubernetes manifest) for an Ingress controller and all other Kubernetes objects using the following command:

```
PLATFORM_NAME=dummy RS_TAG=dummy WF_TAG=dummy ETC_TAG=dummy  helmfile -e cloud
template > templates.yaml
```

**Note:** This command assumes you are in the `~/webfocus-ce/scripts/helmfile` folder.

When generated, open the *templates.yaml* file and search for *kind: Ingress*.

The pull Secret for your remote image registry can also be found in the *templates.yaml*. Search for *ibi-docker-hub*.

## Deploying to a Local or Remote Kubernetes Cluster Where Docker Images Exist

This scenario outlines how TIBCO WebFOCUS® CE can be deployed to a local or remote Kubernetes cluster, but you already have WebFOCUS® CE images pushed to the image registry. This is a typical use case where WebFOCUS® CE has been previously installed in an environment.

1. Ensure that a remote Kubernetes cluster (single-node or multi-node) is available.
2. On a local machine, install Kubectl, Helm, and Helmfile. Docker is not required.
3. Follow the installation steps described in Step 1: Unzip the TIB_wfce_1.0.2.tar File and Step 2: Create Docker Images and Deploy Helm Charts. Skip the steps on building Docker images and proceed directly to running the Helmfile commands.
4. Update your pull Secret (*ibi-docker-hub*) and the repository name for your image registry (default is *ibi2020/webfocus*).

**Usage Considerations**

Just as described in the previous topic Deploying to a Remote Kubernetes Cluster, if you are planning to use a multi-node Kubernetes cluster, then you must use a storage provider that supports the *ReadWriteMany* access mode. You will also require an Ingress controller (for example, NGINX) if you plan to scale the WebFOCUS Client (AppServer).

## Testing Specifications

The deployment scenarios in the previous sections consider Amazon® Elastic Kubernetes Service (EKS) as remote Kubernetes clusters. If there are any distinctions for particular managed Kubernetes clusters, ensure to incorporate those as well. EKS has been tested as follows:

1. EKS was created using *eksctl* with unmanaged nodes (version 1.20). Node EC2 instance types were *c5.2xlarge*.

2. Storage driver was Amazon EFS CSI, backed with an EFS drive in the same region/same VPC as the EKS cluster (un-encrypted).

3. All ancillary software, such as PostgreSQL for the WebFOCUS Client (AppServer), Apache Solr, and Zookeeper were running on the same cluster.

4. The database adapter for the WebFOCUS Reporting Server (Edaserver) was Amazon RDS for PostgreSQL.

5. Application server scaling was tested using the NGINX Ingress controller with AWS ELBv2 (network load balancer). Sticky session cookie support was provided by the NGINX Ingress controller.

6. While running Helmfile, the *cloud* environment file (shipped with the product) was used, as per the following command:

   ```
   helmfile -e cloud sync
   ```

7. By default, the swego web server is only enabled in a *dev* environment. If you require swego in your configuration, then add the following switch to your sync command:

   ```
   --state-values-set swego.enabled=true
   ```

   For example:

   ```
   helmfile -e cloud --state-values-set swego.enabled=true sync
   ```

## Helmfile Environments Reference

The following table provides a reference for the files used and Helmfile commands for each environment.

| Environment Name | Files Used | Sample Command | Notes |
|---|---|---|---|
| **default** | All values from *environments/wf.integ. yaml.gotmpl*. | `helmfile sync` | Assumes Docker images are available locally or they are available to the Kubernetes cluster locally. No image pull is required and the |

| | | | |
|---|---|---|---|
| | | | default Kubernetes storage driver is used. |
| **dev** | All values come from *environments/wf.integ.yaml.gotmpl* and *environments/dev-wf.integ.yaml.gotmpl*, which means last loaded file wins (overwrites) previous values. | `helmfile -e dev sync` | Assumes Docker images are available on a remote image registry. The default image repository name is *ibi2020/webfocus* and the image pull Secret is *ibi-docker-hub*.<br><br>The Kubernetes storage driver must support the *ReadWriteMany* access mode. |
| **cloud** | Just like *dev*, most values come from *environments/wf.integ.yaml.gotmpl*, but others come from *environments/cloud-wf.integ.yaml.gotmpl*. | `helmfile -e cloud sync` | Just like *dev*, but in this case the Kubernetes storage driver must support the *ReadWriteMany* access mode, and an Ingress controller object for the WebFOCUS Client (AppServer) will be created. |

## Post-Deployment Configuration Options

After deploying TIBCO WebFOCUS® Container Edition (CE), you can configure WebFOCUS® CE in your Kubernetes cluster.

### Scaling in Kubernetes

To scale your deployment (for example, by adding more WebFOCUS Reporting Servers), enter the following command in your terminal window/console:

```
$>kubectl scale statefulset.apps/edaserver --replicas=3 -n webfocus
```

**Note:** If required, you can also scale WebFOCUS Client (AppServer) and Cluster Manager (CLM) in your Kubernetes cluster.

## Configuring Horizontal Pod Autoscaling

TIBCO WebFOCUS® Container Edition (CE) supports Horizontal Pod Autoscaling in a Kubernetes cluster.

A *HorizontalPodAutoscaler* automatically updates a workload resource (such as a Deployment or StatefulSet), with the objective of automatically scaling the workload to match demand. In Kubernetes, horizontal scaling means that the response to increased load is to deploy more pods.

In TIBCO WebFOCUS® CE, the following components can be autoscaled:

- WebFOCUS Reporting Server (edaserver)
- WebFOCUS Client (appserver)
- Cache Manager (cachemanager)

For *appserver* and *cachemanager*, CPU-based autoscaling is implemented. In this scenario, if CPU usage is greater than the percentage set limit or threshold value for a specific period of time, then Kubernetes will automatically scale up by increasing the number of available pods.

Similarly, if the CPU threshold is under a specified percentage set limit for a specific period of time, then Kubernetes will automatically scale down by decreasing the number of available pods (to the minimum that is allowed).

For *edaserver*, custom metrics are implemented for autoscaling. This is a combination of CPU percentage utilization and *server average response time* data from the WebFOCUS Reporting Server, **or** the number of failed resources (*edaserver_resource_fails_total*) consistently for a specific period of time. If any of these conditions are met, then Kubernetes will autoscale the *edaserver* pod.

### Installing (Enabling) the Metrics Server

The CPU data that is used for custom metrics and CPU-based autoscaling calculations is provided by the Metrics server. To install the Metrics server with the Prometheus adapter, use the following flag:

```
global.prometheusAdapter.metricsServer=true
```

**Notes:**

- This flag is disabled by default (`=false`).
- If the Metrics server is already running in the Kubernetes cluster, then keep it disabled.

## Installing (Enabling) the Prometheus Server

Helmfile installs the Prometheus server. If the Prometheus server is already running, then disable it by using the following flag:

```
global.prometheus.enabled=false
```

**Note:** This flag is enabled by default (`=true`).

You can provide configuration details for the Prometheus server in:

```
global.prometheusAdapter.prometheus.url
```

```
global.prometheusAdapter.prometheus.port
```

## Modifying Threshold Values for Horizontal Pod Autoscaling

You can modify (overwrite) the default threshold values that are used for autoscaling pods.

For *appserver* and *cachemanager*, where CPU-based autoscaling is implemented:

```
autoscaling:
   enabled: false
   minReplicas: 1
   maxReplicas: 10
   targetCPUUtilizationPercentage: 80
```

For example: `appserver.autoscaling.targetCPUUtilizationPercentage=60`

This will set/update the threshold value for *appserver*. If the CPU usage goes above 60%, then Kubernetes will automatically scale up by increasing the number of available pods.

For *edaserver*, where custom metrics are implemented:

```
edaserver:
  autoscaling
    custom
      targetRunningResponseTime: 2
```

For example: `edaserver.autoscaling.custom.targetRunningResponseTime=10`

**Note:** You can also add this setting in a Go template file to be applied in a specific environment. For example:

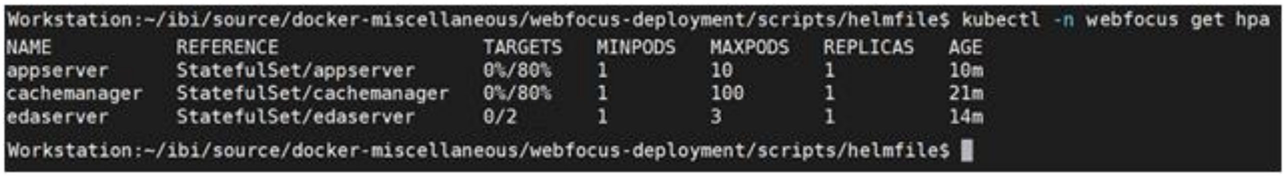
```
edaserver:
  imagePullSecrets:
  - name: ibi-docker-hub
  autoscaling:
   custom:
    targetRunningResponseTime: 2
```

## Retrieving a List of Horizontal Pod Autoscaling Components

Use the following command to retrieve a list (and current status) of WebFOCUS components impacted by Horizontal Pod Autoscaling:

```
$>kubectl -n webfocus get hpa
```

For example:



```
Workstation:~/ibi/source/docker-miscellaneous/webfocus-deployment/scripts/helmfile$ kubectl -n webfocus get hpa
NAME           REFERENCE                  TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
appserver      StatefulSet/appserver      0%/80%    1         10        1          10m
cachemanager   StatefulSet/cachemanager   0%/80%    1         100       1          21m
edaserver      StatefulSet/edaserver      0/2       1         3         1          14m
Workstation:~/ibi/source/docker-miscellaneous/webfocus-deployment/scripts/helmfile$ █
```

## Modifying Autoscaling Behavior

The following syntax can be added for each Horizontal Pod Autoscaling definition:

```
behavior:
  scaleDown:
    stabilizationWindowSeconds: 300 # (wait for 5 minutes before scaling down)
    policies:
    - type: Percent
      value: 100
      periodSeconds: 15 # (Scale down 1 pod every 15 seconds)
  scaleUp:
    stabilizationWindowSeconds: 120 # (wait for 2 minutes before scaling up)
    policies:
    - type: Percent
      value: 100
      periodSeconds: 15 # (Scale up 1 pod every 15 seconds)
    - type: Pods
      value: 1
      periodSeconds: 30 # (i.e., scale up 1 pod every 30 seconds)
    selectPolicy: Max
```

## Useful Commands

The following is a set of useful commands for debugging custom metrics:

```
$> kubectl get --raw /apis/custom.metrics.k8s.io/v1beta1 | jq .
$> kubectl get --raw
"/apis/custom.metrics.k8s.io/v1beta1/namespaces/webfocus/pods/*/edaserver_running_avg_response_time" | jq .
```

```
$> kubectl get --raw
"/apis/custom.metrics.k8s.io/v1beta1/namespaces/webfocus/pods/*/edaserver_resource_fails_total" | jq .
```

**Modifying the Graceful Shutdown Time of the WebFOCUS Reporting Server (edaserver) Pod**

When terminating pods during Horizontal Pod Autoscaling, to change the default shutdown (termination) time for the WebFOCUS Reporting Server (edaserver) pod, which is 60 seconds, change the following configuration value:

```
terminationGracePeriodSeconds: 60
```

This configuration value is located in the edaserver statefulset YAML file.

## Removing (Undeploying) TIBCO WebFOCUS® Container Edition

To remove (undeploy) TIBCO WebFOCUS® Container Edition (CE) from a Kubernetes cluster, enter the following commands in your terminal window/console:

```
cd <webfocus-ce>/scripts/helmfile/

source ../export-defaults.sh

helmfile -e <env-name> destroy

cd <webfocus-ce>/scripts/helmfile/infra

helmfile destroy
```

**Note:** The `<env-name>` parameter refers to the environment (*dev* or *cloud*). You must destroy the Kubernetes cluster in the same environment it was previously created.

## Upgrading TIBCO WebFOCUS® Container Edition

TIBCO WebFOCUS® Container Edition (CE) supports in-place upgrades in a Kubernetes cluster, where the previous configuration and logs are preserved. Note that Persistent Volumes (PVs) are not recreated.

The following components are updated during the in-place upgrade:

- WebFOCUS Reporting Server (Edaserver)
- WebFOCUS Client (AppServer)
- Cluster Manager (CLM)

- Cache Manager
- ReportCaster

Optionally, the following components can also be upgraded:

- PostgreSQL
- Apache Solr
- Prometheus (server and adapter)

To upgrade TIBCO® WebFOCUS CE:

1. Download the new .tar file.
2. Create (build) the Docker images. Ensure that you use the new image tag (for example, *ibi2020/webfocus:wfs-9.0-1.0.2*).
3. Enter the following command in your terminal window/console:
   ```
   TIB_wfce_1.0.2/scripts/build-images.sh
   ```
   The following images are created:
   ```
   REPOSITORY          TAG                IMAGE ID       CREATED          SIZE
   ibi2020/webfocus    wfc-9.0-1.0.2      d0ac3405fdba   31 seconds ago   1.96GB
   ibi2020/webfocus    wfs-etc-9.0-1.0.2  ef04c448df8e   2 minutes ago    3.93GB
   ibi2020/webfocus    wfs-9.0-1.0.2       82fa0b973ed0  4 minutes ago    1.05GB
   ibi2020/webfocus    cachemanager        071af2ccb3de  3 minutes ago    941MB
   ibi2020/webfocus    postgres-alpine     16abefd8e881  1 minutes ago    222MB
   ```
4. Export the new image tags:
   ```
   export PLATFORM_NAME=webfocus
   export INFRA_NS=webfocus
   export WF_TAG=wfc-9.0-1.0.2
   export RS_TAG=wfs-9.0-1.0.2
   export ETC_TAG=wfs-etc-9.0-1.0.2
   ```
5. **(Optional)** Ensure the *helm-diff* plugin (https://github.com/databus23/helm-diff) is available in your environment. To verify, run the following command:
   ```
   helmfile -e <env-name> diff
   ```
   where:

`<env-name>`

Is the specific environment (dev or cloud).

This command will show the changes to be reviewed in your environment before upgrading. It is recommended to run `diff` before upgrading to verify all changes.

6. **(Optional)** Upgrade infra by navigating to the `~/scripts/helmfile/infra` directory and running the following command:

   ```
   helmfile sync
   ```

   **Note:** Upgrading infra will not delete existing data in the database. This step is optional and executed if there is any change in `Postgres/solr/Prometheus` or `Prometheus-adapter`.

7. After reviewing the changes, run the following command:

   ```
   $> helmfile -e <env-name> -l upgradable=true sync
   ```

   After you run this command, all containers are restarted and the in-place upgrade is complete.

8. **(Optional)** To verify that the chart is updated to the latest version, run the following command:

   ```
   helm list -n <namespace>
   ```

   where:

   `<namespace>`

   Is the PLATFORM name set by the export variable in step 4 (for example, `PLATFORM_NAME=webfocus`).

   You will see output returned similar to the following:

   ```
   NAME        NAMESPACE  REVISION  STATUS     CHART             APP VERSION

   appserver   webfocus   1         deployed   appserver-1.0.2   1.0.2
   ```

9. **(Optional)** To verify that the required pods are upgraded correctly, run the following command:

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

   where:

   `<namespace>`

Is the PLATFORM name set by the export variable in step 4 (for example, `PLATFORM_NAME=webfocus`).

Ensure the following pods are in a running state after the upgrade:

- `appserver-0`
- `cachemanager-0`
- `clm-0`
- `edaserver-0`
- `etcd-0`
- `reportcaster-0`

**Upgrading Consideration for WebFOCUS Reporting Server (Edaserver), CLM, and Cache Manager**

These components have a dependency on the *eda-etc* persistent volume. During the upgrade process, content is deleted from this volume and copied from the new image. In addition, no configuration changes are made, only the binaries are updated.

**WebFOCUS Client (AppServer) Pre-upgrade Scripts**

The following scripts are run prior to upgrading WebFOCUS Client (AppServer):

```
/bin/sh-x
${IBI_DOCUMENT_ROOT}/utilities/dbupdate/db_check_version.shUSERNAME={configurat
ion} PASSWORD={configuration}
COMPLETION_STATUS_FILE=${IBI_DOCUMENT_ROOT}/logs/db_check_version_status.log
```

```
/bin/sh-x
${IBI_DOCUMENT_ROOT}/utilities/dbupdate/db_inplace_update.shUSERNAME={configura
tion} PASSWORD={configuration}
```

```
/bin/sh-x
${IBI_DOCUMENT_ROOT}/utilities/WFReposUtil/update_repos.shUSERNAME=$WF_ADMIN_US
ER PASSWORD=$WF_ADMIN_PASS
```

## Updating Docker Images Without Destroying a Kubernetes Cluster

This section describes how to update Docker images without destroying a Kubernetes cluster. There are two options you can use:

- **Option 1:** When not changing the container image name or tag.
- **Option 2:** When changing the container image name or tag.

## Option 1: When Not Changing the Container Image Name or Tag

To update Docker images without destroying a Kubernetes cluster when not changing the container image name or tag:

1. Ensure the *imagePullPolicy* property set to *Always* in the configuration.
2. Scale the application down to 0 and back to the original size using replicas.
   Note, the following commands use the WebFOCUS Client (AppServer) pod as an example.
   a. Scale down your application to 0:
   ```
   $> kubectl scale statefulset.apps/appserver --replicas=0 -n
   webfocus
   ```
   b. Scale up your application to the original size (for example, 1):
   ```
   $> kubectl scale statefulset.apps/appserver --replicas=1 -n
   webfocus
   ```

## Option 2: When Changing the Container Image Name or Tag

Running the `helmfile apply` command is useful when new versions of the images are available and you want to upgrade your Kubernetes cluster to use these images.

Before continuing, ensure the new Docker images are available in the registry. In addition, ensure the *helm-diff* plugin (https://github.com/databus23/helm-diff) is available in your environment.

If this plugin is currently not installed, enter the following command in your terminal window/console:

```
$> helm plugin install https://github.com/databus23/helm-diff
```

1. Export the following environment variables with the new image tag names:
   ```
   $> export WF_TAG=<new-image-tag>
   $> export RS_TAG=<new-image-tag>
   $> export ETC_TAG=<new-image-tag>
   $> export PLATFORM_NAME=webfocus
   ```
2. Before running `helmfile apply`, run a `diff` to verify that Helm picked up the new image tags. Navigate to the *scripts/helmfile* directory and enter the following commands:
   ```
   $> cd <webfocus-ce>/scripts/helmfile
   $> helmfile -e <env-name> diff
   ```
   **Note:** Ensure you reference the proper environment by using `-e`.

3. Run `helmfile apply` if you see the desired image in `diff`:

   ```
   $> helmfile -e <env-name> apply
   ```

   Once you run the `helmfile apply` command, the running pods are terminated and new pods are created using the updated images.

To check your result (if required), enter the following command in your terminal window/console:

```
$> helm list -n webfocus
```

You will see chart revisions increased by one.

You can also see the history of a particular service and roll back or apply a specific revision. This is useful in the event of a failure or undesired result and you need to revert/roll back changes.

To view a history of WebFOCUS Client (AppServer) Helm charts, enter the following command in your terminal window/console:

```
$> helm history appserver -n webfocus
```

For example, to roll back the *appserver* Helm chart, back to *n* revision, enter the following command in your terminal window/console:

```
$> helm rollback appserver 1 -n webfocus
```

In this example n = 1.

## Usage and Environment Considerations

This section describes usage and considerations for TIBCO WebFOCUS® Container Edition (CE) in your Kubernetes cluster.

### Scaling

If you are planning or have a requirement to scale your TIBCO WebFOCUS® Container Edition CE deployment, ensure the following:

1. You are using a private Docker registry to pull images.
2. Update the image pull Secret in your configuration (`platform.dockerconfigjson`).
3. Your storage provider is using the *ReadWriteMany* access mode.

### Resource Limits

Be aware of resource limits and make the appropriate adjustments according to your environment.

## Troubleshooting

This section provides notes for troubleshooting purposes as needed.

| Error / Issue | Workaround / Solution |
|---|---|
| `helmfile sync`<br>`in ./helmfile.yaml: error during helmfile.yaml.part.0 parsing:`<br>`template: stringTemplate:33:30: executing "stringTemplate" at <.Environment.Values.storage.nfs_server>: map has no entry for key "storage"` | Current environment variables that are set do not match the exported variables. For example:<br><br>`PLATFORM_NAME=webfocus`<br><br>Export all required variables as described. |

### Helmfile Times Out

If your Helmfile fails to deploy and times out, log into kubectl, and enter the following command in your terminal window/console to investigate:

```
$> kubectl get pods -n [platform]
```

This command returns the detailed status of the pods in your Kubernetes cluster that are deployed to run in the specified platform. When your Kubernetes cluster is running correctly, each of the pods should have:

- All pods counted in the READY column (for example, 1/1, 2/2, 3/3, and so on).
- The value *Running* listed in the STATUS column.

A value other than *Running* in the STATUS column indicates an issue with your Kubernetes cluster. A non-zero (0) or growing value (1, 2, 3, and so on) in the RESTARTS column indicates an issue with the corresponding Kubernetes pod.

## Appendix A: Configuration Parameters Reference

This section provides a reference for the configuration parameters that are included in the Helm charts packaged with TIBCO WebFOCUS® Container Edition (CE).

### Common Parameters

This section lists and describes the common parameters.

| **Parameter** | **Description** | **Default Value** |
|---|---|---|
| platform.servingDomain | Used when an ingress controller is enabled. | jenkins.dev-cloud.ibi.com |
| platform.rsNamespace | Not required for single tenant configuration. | N/A |
| platform.tlsSecretName | Used when an ingress controller is enabled. | N/A |
| platform.dockerconfigjson | If you are using a private Docker registry to pull images, then you must update the pull Secret in your configuration. The value of the `.dockerconfigjson` field is a base64 representation of your Docker credentials. | N/A |
| platform.tenants.name | Obtained from the *PLATFORM_NAME* environment variable. | webfocus |
| platform.tenants.postgresUrl | PostgreSQL database URL. | jdbc:postgresql://postgresql.default.svc.cluster.local:5432/{{ requiredEnv "PLATFORM_NAME" }}-eda?currentSchema=public |
| platform.tenants.postgresUser | PostgreSQL database user name (default is | webfocus |

| | | |
|---|---|---|
| | obtained from *PLATFORM_NAME*). | |
| platform.tenants.postgresPassword | PostgreSQL database user password. | lIPO0ytSDfk3u7sdfk |
| platform.tenants.solrUser | Solr user name (default is obtained from *PLATFORM_NAME*). | webfocus |
| platform.tenants.solrPassword | Solr user password. | lIPO0ytSDfk3u7sdfk |
| platform.tenants.solrCollection | Collection name in Solr (default is obtained from *PLATFORM_NAME*). | webfocus |

**Note:** The *PLATFORM_NAME* environment variable is set to *webfocus* by default in the *readme.md* file. This can be modified by exporting a different value for this environment variable.

## Global Parameters

This section lists and describes the global parameters.

| **Parameter** | **Description** | **Default Value** |
|---|---|---|
| global.solr.install | Determines whether to install the solr infra component or not. | true |
| global.config.IBI_INFOSEARCH_SOLR_URL | Solr search server URL. | http://solr.default.svc.cluster.local:8983/solr |
| global.config.DB_URL | Postgres database URL. | postgresql.default.svc.cluster.local:5432 |
| global.postgres.install | Determines whether to install | true |

| | | |
|---|---|---|
| | the postgres infra component or not. | |
| global.postgres.adminUsername | Postgres admin user name. | postgres |
| global.postgres.adminPassword | Postgres admin user password. | postgres123 |
| global.postgres.host | Postgres database hostname. | postgresql.default.svc.cluster.local |
| global.postgres.port | Postgres database port. | 5432 |
| global.solr.adminUser | Solr admin user name. | admin |
| global.solr.adminPassword | Solr admin user password. | solr123 |
| global.solr.host | Solr server hostname. | solr.default.svc.cluster.local |
| global.solr.port | Solr server port. | 8983 |
| global.prometheus.enabled | Determines whether to install prometheus server or not from infra. | true |
| global.prometheusAdapter.enabled | Determines whether to install prometheusAdapter or not from infra. | true |

## WebFOCUS Client (AppServer) Parameters

This section lists and describes the parameters that are used by the WebFOCUS Client (AppServer).

| Parameter | Description | Default Value |
|---|---|---|
| appserver.image.repository | Repository where the Docker image is stored. | ibi2020/webfocus |

| | | |
|---|---|---|
| appserver.image.tag | Docker image tag is obtained from the *WF_TAG* environment variable. | wfc-9.0-1.0.2 |
| appserver.image.pullPolicy | The imagePullPolicy and the tag of the image affect when the kubelet attempts to pull the specified image. For more information, reference the following link:<br><br>https://kubernetes.io/docs/concepts/configuration/overview/#container-images | IfNotPresent |
| appserver.replicaCount | Number of WebFOCUS Client (AppServer) copies (replicas). | 1 |
| appserver.service.type | Allows you to specify the type of service. NodePort is default, others are ClusterIP, ingress. | NodePort |
| appserver.service.nodePortWFC | WebFOCUS Client port exposed for NodePort service. | 31080 |
| appserver.config.install_cfg.IBI_IMPORT_DIRECTORY | The *import* directory location/path in the *install.cfg* file. | /opt/webfocus/cm/import |
| appserver.config.install_cfg.IBI_EXPORT_DIRECTORY | The *export* directory location/path in the *install.cfg* file. | /opt/webfocus/cm/export |
| appserver.config.install_cfg.IBI_SCM_STAGING_DIRECTORY | The *scm* directory location/path in the *install.cfg* file. | /opt/webfocus/scm |
| appserver.config.install_cfg.IBI_TRACE_DIRECTORY | The *traces* directory location/path in the *install.cfg* file. | /opt/webfocus/traces |
| appserver.config.install_cfg.IBI_TEMPORARY_DIRECTORY | The *temp* directory location/path in the *install.cfg* file. | /opt/webfocus/temp |
| appserver.config.install_cfg.IBI_MAGNIFY_CONFIG | The *magnify* directory location/path in the *install.cfg* file. | /opt/webfocus/config/magnify |

| | | |
|---|---|---|
| appserver.config.install_cfg.IBI_ADMIN_NAME | WebFOCUS Client admin user name. | secret |
| appserver.config.install_cfg.IBI_ADMIN_PASS | WebFOCUS Client admin user password. | terces |
| appserver.config.install_cfg.IBI_Eula_Acceptance | End-user license agreement. | TRUE |
| appserver.config.install_cfg.IBI_INFOSEARCH_SOLR_URL | Solr server URL. | http://solr.default.svc.cluster.local:8983/solr |
| appserver.config.install_cfg.IBI_EMAIL_SERVER | Email server URL. | http://solr.default.svc.cluster.local:8983/solr |
| appserver.config.install_cfg.IBI_EMAIL_SERVER_PORT | Email server port. | 587 |
| appserver.config.install_cfg.IBI_EMAIL_SMTP_USER | SMTP user email. | foo_user |
| appserver.config.install_cfg.IBI_EMAIL_SMTP_PASS | SMTP user password. | foo_pass |
| appserver.ingress.enabled | Set to *true* if you are using an ingress service type. Routing rules to manage external users' access to the services through HTTPS/HTTP. | false |

**Note:** The ingress is enabled (*true*) by default when using the *cloud* environment. It is disabled (*false*) for the *dev* environment.

## WebFOCUS Reporting Server (Edaserver) Parameters

This section lists and describes the parameters that are used by the WebFOCUS Reporting Server (Edaserver).

| Parameter | Description | Default Value |
|---|---|---|
| edaserver.image.repository | Repository where the Docker image is stored. | ibi2020/webfocus |
| edaserver.image.tag | Docker image tag is obtained from the *RS_TAG* environment variable. | wfs-9.0-1.0.2 |
| edaserver.EDAUSER | WebFOCUS Reporting Server admin user name. | secret |
| edaserver.EDAPASSWD | WebFOCUS Reporting Server admin user password. | terces |
| edaserver.etc.image.tag | Docker image tag for WebFOCUS Reporting Server static content is obtained from the *ETC_TAG* environment variable. | {{ requiredEnv "ETC_TAG" }} |
| edaserver.PYSERV_URL | URL of running DSML service. Recommend to update from here or from the web console. | http://dsml |
| edaserver.service.type | Allows you to specify the type of service. Set ClusterIP since we do not want to expose the WebFOCUS Reporting Server externally. | ClusterIP |
| edaserver.service.port1 | Used to map with the *nodeport* service. | 8120 |
| edaserver.service.port2 | Used to map with the *nodeport* service. | 8121 |
| edaserver.service.port3 | Used to map with the *nodeport* service. | 8122 |
| edaserver.service.port4 | Used to map with the *nodeport* service. | 8123 |
| edaserver.service.nodePort1 | Used to expose the port for external access. | 31120 |
| edaserver.service.nodePort2 | Used to expose the port for external access. | 31121 |
| edaserver.service.nodePort3 | Used to expose the port for external access. | 31122 |
| edaserver.service.nodePort4 | Used to expose the port for external access. | 31123 |

| | | |
|---|---|---|
| edaserver.replicaCount | Number of WebFOCUS Reporting Server copies (replicas). | 1 |
| edaserver.ingress.enabled | Set to *true* if you are using an ingress service type. Routing rules to manage external users' access to the services through HTTPS/HTTP. | false |

**Note:** By default and for convenience, for *service.nodePort*, the Kubernetes control plane will allocate a port from a range (default: 30000-32767).

## Cluster Manager (CLM) Parameters

This section lists and describes the parameters that are used by the Cluster Manager (CLM).

| Parameter | Description | Default Value |
|---|---|---|
| clm.image.repository | Repository where the Docker image is stored. | ibi2020/webfocus |
| clm.image.tag | The Docker image tag is obtained from the *RS_TAG* environment variable. | wfs-9.0-1.0.2 |
| clm.EDAUSER | CLM admin user name. | secret |
| clm.EDAPASSWD | CLM admin user password. | terces |
| clm.service.type | Allows you to specify the type of service. Set ClusterIP since we do not want to expose CLM externally. | ClusterIP |
| clm.service.port1 | Used to map with the *nodeport* service. | 8120 |
| clm.service.port2 | Used to map with the *nodeport* service. | 8121 |
| clm.service.port3 | Used to map with the *nodeport* service. | 8122 |
| clm.service.port4 | Used to map with the *nodeport* service. | 8123 |
| clm.service.nodePort1 | Used to expose the port for external access. | 31120 |
| clm.service.nodePort2 | Used to expose the port for external access. | 31121 |
| clm.service.nodePort3 | Used to expose the port for external access. | 31122 |

| | | |
|---|---|---|
| clm.service.nodePort4 | Used to expose the port for external access. | 31123 |
| clm.replicaCount | Number of CLM copies (replicas). | 1 |
| clm.ingress.enabled | Set to *true* if you are using an ingress service type. Routing rules to manage external users' access to the services through HTTPS/HTTP. | false |

## Initialized Database Parameters (Application Server / WebFOCUS Client)

This section lists and describes the initialized database parameters that impact the application server / WebFOCUS Client.

| **Parameter** | **Description** | **Default Value** |
|---|---|---|
| initHookDb.dbAdminUser | Database admin user name. | postgres |
| initHookDb.dbAdminPass | Database admin password. | postgres123 |
| initHookDb.dbHost | Database host. | postgresql.default.svc .cluster.local |
| initHookDb.dbPort | Database port. | 5432 |
| initHookDb.MAILFROM | Mail server "from" address. | info@dev-cloud.ibi.com |
| initHookDb.MAILSERVER | Mail server host. | email-smtp.us-east-1.amazonaws.com |
| initHookDb.MAILUSER | User name for mail server. | foo_user |
| initHookDb.MAILPASSWORD | Password for mail server. | foo_password |
| initHookSolr.SOLR_CONFIG | Solr configuration file. | ibi-protected |
| initHookSolr.SOLR_ADMIN_USER | Solr admin user name. | admin |
| initHookSolr.SOLR_ADMIN_PASSWORD | Solr admin password. | solr123 |
| storage.nfs_server | NFS server for storage. | localhost |
| storage.accessMode | Storage access mode. Options include: | ReadWriteMany |

|  |  |  |
|---|---|---|
|  | ReadWriteOnce, ReadOnlyMany, ReadWriteMany, and ReadWriteOncePod. It is recommended not to change this value. |  |
| storage.capacity | Capacity of storage. Gi refers to Gigabytes. You can also use Ei, Pi, Ti, Gi, Mi, or Ki. | 4Gi |

The following table lists and describes the configuration parameters that are used for volume permissions.

| **Parameter** | **Description** | **Default Value** |
|---|---|---|
| appserver.volumePermissions.enabled | Specify *true* to set the required file permissions on the persistent volume, specify *false* for local-storage. | false |
| edaserver.volumePermissions.enabled | Specify *true* to set the required file permissions on the persistent volume, specify *false* for local-storage. | false |
| clm.volumePermissions.enabled | Specify *true* to set the required file permissions on the persistent volume, specify *false* for local-storage. | false |

**Note:** These parameters are required only for NFS mounts. For more information, reference the following link:

https://github.com/kubernetes/examples/issues/260

## Resource (CPU and Memory) Parameters

This section lists and describes the resource parameters that control CPU and memory (RAM) usage.

By default, resource limits can only be applied to the *cloud* environment. There are no resource limits for the *dev* environment. However, you can modify this default behavior if required by including `resource-limit.yaml.gotmpl` for the *dev* environment in Helmfile.

**Note:** Limits and requests for ephemeral-storage are measured in bytes. You can express storage as a plain integer or as a fixed-point number using one of the following suffixes: E, P, T, G, M, K

You can also use the following power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki

For example, the following represent roughly the same value: 128974848, 129e6, 129M, 123Mi

| Parameter | Description | Default Value |
|---|---|---|
| appserver.resources.requests.memory | Appserver/WebFOCUS Client pod/container reserves requested memory in the system. | 1Gi |
| appserver.resources.requests.cpu | Appserver/WebFOCUS Client pod/container reserves requested CPU in the system (500m = 0.5 CPU). | 500m |
| appserver.resources.limits.memory | Appserver/WebFOCUS Client container max limit for the memory (RAM). | 2Gi |
| appserver.resources.limits.cpu | Edaserver/Reporting Server container max limit for the CPU. | 1000m |
| edaserver.resources.requests.memory | Edaserver/Reporting Server pod/container reserves requested memory in the system. | 512Mi |
| edaserver.resources.requests.cpu | Edaserver/Reporting Server pod/container reserves | 1 |

| | | |
|---|---|---|
| | requested CPU in the system (1 = 1 CPU). | |
| edaserver.resources.limits.memory | Edaserver/Reporting Server container max limit for the memory (RAM). | 4Gi |
| edaserver.resources.limits.cpu | Edaserver/Reporting Server container max limit for the CPU (2 = 2 CPU). | 2 |
| clm.resources.requests.memory | CLM pod/container reserves requested memory in the system. | 256Mi |
| clm.resources.requests.cpu | CLM pod/container reserves requested CPU in the system. | 100m |
| clm.resources.limits.memory | CLM container max limit for the memory (RAM). | 512Mi |
| clm.resources.limits.cpu | CLM container max limit for the CPU. | 500m |
| swego.resources.requests.memory | Swego pod/container reserves requested memory in the system. | 256Mi |
| swego.resources.requests.cpu | Swego pod/container reserves requested CPU in the system. | 100m |
| swego.resources.limits.memory | Swego container max limit for the memory (RAM). | 512Mi |
| swego.resources.limits.cpu | Swego container max limit for the CPU. | 250m |
| cachemanager.resources.requests.memory | Cache manager pod/container reserves requested memory in the system. | 256Mi |

| | | |
|---|---|---|
| cachemanager.resources.requests.cpu | Cache manager pod/container reserves requested CPU in the system. | 100m |
| cachemanager.resources.limits.memory | Cache manager container max limit for the memory (RAM). | 512Mi |
| cachemanager.resources.limits.cpu | Cache manager container max limit for the CPU. | 250m |