Camunda 8 Helm Chart

License Apache 2.0 Test - Unit failing Camunda Platform 10.0.4

Please also refer to the documentation on how to use Helm charts.

Architecture

Requirements

- Helm >= 3.9.x
- Kubernetes >= 1.20+
- Minimum cluster requirements include the following to run this chart with default settings.
  - All of these settings are configurable.
  - Three Kubernetes nodes to respect the default "hard" affinity settings
  - 2GB of RAM for the JVM heap

Dependencies

Camunda 8 Helm chart is an umbrella chart for different components. Some are internal (sub-charts), and some are external (third-party). The dependency management is fully automated and managed by Helm itself; however, it's good to understand the dependency structure. This third-party dependency is reflected in the Helm chart as follows:

```
camunda-platform
  |_ elasticsearch
  |_ identity
    |_ keycloak
      |_ postgresql
  |_ optimize
  |_ operate
  |_ tasklist
  |_ zeebe
  |_ postgresql
```

> [!NOTE] Please note that the Connectors and Web Modeler components are part of the main chart and not implemented as sub-charts.

For example, Camunda Identity utilizes Keycloak and allows you to manage users, roles, and permissions for Camunda 8 components.

- Keycloak is a dependency for Camunda Identity, and PostgreSQL is a dependency for Keycloak.
- Elasticsearch is a dependency for the Camunda chart, which is used in Zeebe, Operate, Tasklist, and Optimize.
- PostgreSQL is an optional dependency for the Camunda chart and is used by Web Modeler.

The values for the dependencies Keycloak and PostgreSQL can be set in the same hierarchy:

```
identity:
  [identity values]
  keycloak:
    [keycloak values]
    postgresql:
      [postgresql values]
postgresql:
  [postgresql values]
```

Versioning

After the 8.4 release (January 2024), the Camunda Helm chart version is **decoupled** from the version of the application (e.g., the chart version is `9.0.0` and the application version is `8.4.x`).

Before the 8.4 release, the Camunda Helm chart version was **coupled** with the applications version (e.g., chart version is `8.3.x` and applications version is `8.3.x`).

For more details, check out the full version matrix.

Installation

The first command adds the official Camunda Helm charts repo, and the second installs the Camunda chart to your current Kubernetes context.

```
helm repo add camunda https://helm.camunda.io
helm install camunda-platform camunda/camunda-platform
```

Although the Camunda 8 Helm chart gets the latest version of Camunda 8 applications, the version is still possible to diverge slightly between the chart and the apps (more details about that can be found in versioning).

To have the latest version of the chart and apps at any time, install the chart as follows:

```
helm install camunda-platform camunda/camunda-platform \
    --values https://helm.camunda.io/camunda-platform/values/values-latest.yaml
```

For the previous version, you can get the latest applications patch version using our backporting mechanism.

Local Kubernetes

We recommend using Helm on KIND for local environments, as the Helm configurations are battle-tested and much closer to production systems.

For more details, follow the Camunda 8 local Kubernetes cluster guide.

OpenShift

Check out OpenShift Support to get started with deploying the charts on Red Hat OpenShift.

Backporting

Our Helm chart is highly customizable and constantly evolving. Hence, currently, we backport the older charts by providing an extra value file per version. That covers most backporting cases, like updating the application's image tags to the latest patch version, setting env var, etc.

To install a previous chart version with the latest app patch image tags for that version, use the values file for the minor release. For example (the values file could also be downloaded):

```
helm install camunda-platform camunda/camunda-platform --version 8.1 \
    --values https://helm.camunda.io/camunda-platform/values/values-v8.1.yaml
```

Uninstalling Charts

You can remove these charts by running:

```
helm uninstall camunda
```

> [!NOTE]
>
> Notice that all the Services and Pods will be deleted, but not the PersistentVolumeClaims (PVC) which are used to hold the storage for the data generated by the cluster and Elasticsearch.

To free up the storage, you need to delete all the PVCs manually.

First, view the PVCs:

```
kubectl get pvc -l app.kubernetes.io/instance=camunda
kubectl get pvc -l release=camunda
```

Then delete the ones that you don't want to keep:

```
kubectl delete pvc -l app.kubernetes.io/instance=camunda
kubectl delete pvc -l release=camunda
```

Or you can delete the related Kubernetes namespace, which contains all PVCs.

Configuration

The following sections contain the configuration values for the chart and each sub-chart. All of them can be overwritten via a separate `values.yaml` file.

Check out the default values.yaml file, which contains the same content and documentation.

> [!NOTE]
>
> For more details about deploying Camunda 8 on Kubernetes, please visit the Helm/Kubernetes installation instructions docs.

Notes on Configuration

Web Modeler

> [!NOTE]
>
> Web Modeler Self-Managed is available to Camunda enterprise customers only.

Docker registry The Docker images for Web Modeler are available in a private registry. Enterprise customers either already have credentials to this registry, or they can request access to this registry through their CSM contact at Camunda. To enable Kubernetes to pull the images from Camunda's registry, you'll need to:

- create an image pull secret using the provided credentials
- configure the Web Modeler pods to use the secret:

  ```
  webModeler:
    image:
      pullSecrets:
        - name: <SECRET_NAME>
  ```

Database Web Modeler requires a PostgreSQL database to store the data. You can either:

- Deploy a PostgreSQL instance as part of the Helm release by setting `postgresql.enabled` to `true` (which will enable the postgresql chart dependency).
- Configure a connection to an (existing) external database by setting `postgresql.enabled` to `false` and providing the values under `restapi.externalDatabase`.

SMTP server Web Modeler requires an SMTP server to send (notification) emails to users. The SMTP connection can be configured with the values under `restapi.mail`.

Updating Environment Variables When configuring the `env` options in the settings listed above, the environment variables you specify in values.yaml may show up twice when running `kubectl describe deployment <deployment>`. However, the environment variable is specified in values.yaml will have precedence when the pod actually runs. To verify this, you can check the output from the following command:

```
kubectl exec pod/<podName> -- env
```

Outbound Connectors To learn more about outbound connectors, visit related documentation article.

Inbound Connectors To learn more about inbound connectors, visit related documentation article.

Using Connector Secrets Connector secrets are generally configured via environment variables.

You can set them via `values.yaml`, or command line. For example, if you need to set a Slack token, you should configure the following:

```
connectors:
  env:
    - name: SLACK_TOKEN
      value: <your actual token value>
```

After that, a Modeler user can set in their BPMN diagram a value `secrets.SLACK_TOKEN` without ever knowing the actual token.

Visit using secrets in manual installation to learn more.

Elasticsearch

Camunda 8 Helm chart has a dependency on the Elasticsearch 8 Helm Chart. All variables related to Elasticsearch can be set under `elasticsearch`.

> [!NOTE]
>
> The default setup of the Elasticsearch 8 part of Camunda 8 uses nodes that have all roles (master, data, coordinating, and ingest). For high-demand deployments, it's recommended to deploy the Elasticsearch master-eligible nodes as master-only nodes.

Section	Parameter	Description	Default
`elasticsearch`	`enabled`	If true, enables Elasticsearch deployment as part of the Camunda Helm chart	`true`

Example:

```
elasticsearch:
  enabled: true
  image:
    tag: <YOUR_VERSION_HERE>
```

Elasticsearch Retention Since moving to Elasticsearch 8, Curator is deprecated in favor of Manage the index lifecycle (ILM). Hence, each component in Camunda 8 controls its Elasticsearch index retention.

Keycloak

When Camunda 8 Identity component is enabled by default, and it depends on Bitnami Keycloak chart. Since Keycloak is a dependency for Identity, all variables related to Keycloak can be found in bitnami/keycloak/values.yaml and can be set under `identity.keycloak`.

Section	Parameter	Description	Default
`identity.keycloak`	`enabled`	If true, enables Keycloak chart deployment as part of the Camunda Helm chart	`true`

Example:

```
identity:
  keycloak:
    enabled: true
```

Keycloak Theme Camunda provides a custom theme for the login page used in all apps. The theme is copied from the Identity image.

The theme is added to Keycloak by default, however, since Helm v3 (the latest checked 3.10.x) doesn't merge lists with custom values files, then you will need to add this to your own values file if you override any of `extraVolumes`, `initContainers`, or `extraVolumeMounts`.

```
identity:
  keycloak:
    extraVolumes:
    - name: camunda-theme
      emptyDir:
        sizeLimit: 10Mi
    initContainers:
    - name: copy-camunda-theme
      image: >-
        {{- $identityImageParams := (dict "base" .Values.global "overlay" .Values.global.identity) -}}
        {{- include "camundaPlatform.imageByParams" $identityImageParams }}
      imagePullPolicy: "{{ .Values.global.image.pullPolicy }}"
      command: ["sh", "-c", "cp -a /app/keycloak-theme/* /mnt"]
      volumeMounts:
      - name: camunda-theme
        mountPath: /mnt
    extraVolumeMounts:
    - name: camunda-theme
      mountPath: /opt/bitnami/keycloak/themes/identity
```

Development

For development purposes, you might want to deploy and test the charts without creating a new helm chart release. To do this you can run the following:

```
helm install camunda --atomic --debug ./charts/camunda-platform
```

- `--atomic if set, the installation process deletes the installation on failure. The --wait flag will be set automatically if --atomic is used`
- `--debug enable verbose output`

To generate the resources/manifests without really installing them, you can use:

- `--dry-run simulate an install`

If you see errors like:

```
Error: found in Chart.yaml, but missing in charts/ directory: elasticsearch
```

Then you need to download the dependencies first.

Run the following to add resolve the dependencies:

```
make helm.repos-add
```

After this, you can run: `make helm.dependency-update`, which will update and download the dependencies for all charts.

The execution should look like this:

```
$ make helm.dependency-update
helm dependency update charts/camunda-platform
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "camunda-platform" chart repository
...Successfully got an update from the "bitnami" chart repository
Update Complete.  Happy Helming!
Saving 6 charts
Dependency zeebe did not declare a repository. Assuming it exists in the charts directory
Dependency zeebe-gateway did not declare a repository. Assuming it exists in the charts directory
Dependency operate did not declare a repository. Assuming it exists in the charts directory
Dependency tasklist did not declare a repository. Assuming it exists in the charts directory
Dependency identity did not declare a repository. Assuming it exists in the charts directory
Deleting outdated charts
helm dependency update charts/camunda-platform/charts/identity
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "camunda-platform" chart repository
...Successfully got an update from the "bitnami" chart repository
Update Complete.  Happy Helming!
Saving 2 charts
Downloading keycloak from repo https://charts.bitnami.com/bitnami
Downloading common from repo https://charts.bitnami.com/bitnami
```

Releasing the Charts

Please see the corresponding release guide to find out how to release the chart.

Parameters

Global parameters

`global`		
`global.multitenancy`		
`global.multitenancy.enabled`	if true, then enable multitenancy in all applicable components.	`false`
`global.createReleaseInfo`	Create config that will be used in Camunda Console.	`true`
`global.annotations`	Annotations can be used to define common annotations, which should be applied to all deployments	`{}`
`global.labels.app`	Name of the application	`camunda-platform`
`global.image.registry`	Can be used to set container image registry.	`""`
`global.image.tag`	defines the tag / version which should be used in the most of the apps.	`8.5.0`
`global.image.pullPolicy`	defines the image pull policy which should be used https://kubernetes.io/docs/concepts/containers/images/#image-pull-policy	`IfNotPresent`
`global.image.pullSecrets`	can be used to configure image pull secrets https://kubernetes.io/docs/concepts/containers/images/#specifying-imagepullsecrets-on-a-pod	`[]`

`global.ingress`		
`global.ingress.enabled`	if true, an ingress resource is deployed. Only useful if an ingress controller is available, like Ingress-NGINX.	`false`
`global.ingress.className`	Ingress.className defines the class or configuration of ingress which should be used by the controller	`nginx`
`global.ingress.annotations`	defines the ingress related annotations, consumed mostly by the ingress controller	`{}`
`global.ingress.host`	If not specified the rules applies to all inbound http traffic, if specified the rule applies to that host.	`""`
`global.ingress.pathType`	can be used to define the Ingress path type. https://kubernetes.io/docs/concepts/services-networking/ingress/#path-types	`Prefix`
`global.ingress.tls`	configuration for tls on the ingress resource https://kubernetes.io/docs/concepts/services-networking/ingress/#tls	
`global.ingress.tls.enabled`	if true, then tls is configured on the ingress resource. If enabled the Ingress.host need to be defined.	`false`
`global.ingress.tls.secretName`	defines the secret name which contains the TLS private key and certificate	`camunda-platform`
`global.elasticsearch`		
`global.elasticsearch.enabled`	if true, enables elasticsearch for all components	`true`
`global.elasticsearch.external`	if true, tries to connect to an external elasticsearch	`false`
`global.elasticsearch.tls`		
`global.elasticsearch.tls.enabled`	enable tls for external elasticsearch	`false`
`global.elasticsearch.tls.existingSecret`	provide an already existing tls secret for connecting to external elasticsearch	`nil`
`global.elasticsearch.auth`		
`global.elasticsearch.auth.username`	the username for external elasticsearch	`nil`
`global.elasticsearch.auth.password`	the password for external elasticsearch	`nil`
`global.elasticsearch.auth.existingSecret`	you can provide an existing secret for the external elasticsearch password	`nil`
`global.elasticsearch.auth.existingSecretKey`	you can provide an existing secret key for the external elasticsearch password	`nil`
`global.elasticsearch.disableExporter`	DEPRECATED: this value is not needed anymore. Use global.elasticsearch.enabled	`false`
`global.elasticsearch.url`	Configuration to configure elasticsearch url	
`global.elasticsearch.url.protocol`	defines the elasticsearch access protocol.	`http`
`global.elasticsearch.url.host`	Elasticsearch.host defines the elasticsearch host, ideally the service name inside the namespace	`{{ .Release.Name }}-elasticsearch`
`global.elasticsearch.url.port`	Elasticsearch.port defines the elasticsearch port, under which elasticsearch can be accessed	`9200`
`global.elasticsearch.clusterName`	Elasticsearch.clusterName defines the cluster name which is used by Elasticsearch	`elasticsearch`
`global.elasticsearch.prefix`	Elasticsearch.prefix defines the prefix which is used by the Zeebe Elasticsearch Exporter to create Elasticsearch indexes	`zeebe-record`
`global.opensearch`		
`global.opensearch.enabled`	enabled external opensearch	`false`
`global.opensearch.aws.enabled`	Enabling AWS IRSA	`false`
`global.opensearch.tls`		
`global.opensearch.tls.enabled`	enable tls for external opensearch	`false`
`global.opensearch.tls.existingSecret`	provide an already existing tls secret for connecting to external opensearch	`nil`
`global.opensearch.auth`		
`global.opensearch.auth.username`	the username for external opensearch	`nil`
`global.opensearch.auth.password`	the password for external opensearch	`nil`
`global.opensearch.auth.existingSecret`	you can provide an existing secret for the external opensearch password	`nil`
`global.opensearch.auth.existingSecretKey`	you can provide an existing secret key for the external opensearch password	`nil`
`global.opensearch.url`	Configuration to configure opensearch url	
`global.opensearch.url.protocol`	defines the external opensearch access protocol	`https`
`global.opensearch.url.host`	defines the external opensearch host, ideally the service name inside the namespace	`nil`
`global.opensearch.url.port`	defines the external opensearch port, under which opensearch can be accessed	`443`
`global.zeebeClusterName`	ZeebeClusterName defines the cluster name for the Zeebe cluster. All Zeebe pods get this prefix in their name and the brokers uses that as cluster name.	`{{ .Release.Name }}-zeebe`
`global.identity.keycloak.internal`	It's useful for using existing Keycloak in another namespace with and access it with the combined Ingress.	`false`
`global.identity.keycloak.url`	can be used incorporate with "identityKeycloak.enabled: false" to use your own Keycloak instead of the one comes with Camunda Helm chart.	`{}`
`global.identity.keycloak.contextPath`	In Keycloak v16.x.x it's hard-coded as '/auth', but in v19.x.x it's '/'.	`/auth`
`global.identity.keycloak.realm`	defines Keycloak realm path used for Camunda.	`/realms/camunda-platform`
`global.identity.keycloak.auth`	same as "identityKeycloak.auth" but it's used for existing Keycloak.	`{}`
`global.identity.auth`	configuration, to configure identity authentication setup	
`global.identity.auth.enabled`	if true, enables the identity authentication otherwise basic-auth will be used on all services.	`true`
`global.identity.auth.issuer`	defines the issuer name, which is used by the services to validate the JWT tokens.	`""`

`global.identity.auth.issuerBackendUrl`	defines the issuer backend URL, which is used by the services to validate the JWT tokens in a container to container context.	`""`
`global.identity.auth.tokenUrl`	defines the token URL, which is used by the services to request JWT tokens.	`""`
`global.identity.auth.jwksUrl`	defines the JWKS URL, which is used by the services to validate the JWT tokens.	`""`
`global.identity.auth.type`	defines the type of authentication which should be used. Defaults to Keycloak	`KEYCLOAK`
`global.identity.auth.publicIssuerUrl`	Can be overwritten if ingress is in use and an external IP is available.	`http://localhost:18080/auth/realms/camunda-platform`
`global.identity.auth.connectors`	configuration to configure Connectors authentication specifics on global level, which can be accessed by other sub-charts	
`global.identity.auth.connectors.clientId`	defines the client id, which is used by Connectors in authentication flows.	`connectors`
`global.identity.auth.connectors.existingSecret`	can be used to use an own existing secret. If not set a random secret is generated.	`""`
`global.identity.auth.identity`	configuration to configure Identity authentication specifics on global level, which can be accessed by other sub-charts	
`global.identity.auth.identity.clientId`	defines the client id, which is used by Identity in authentication flows.	`identity`
`global.identity.auth.identity.audience`	defines the audience, which is used by Identity.	`camunda-identity-resource-server`
`global.identity.auth.identity.existingSecret`	can be used to reference an existing secret. If not set, a random secret is generated.	`nil`
`global.identity.auth.identity.redirectUrl`	defines the redirect URL, which is used by the auth platform to access Identity.	`http://localhost:8085`
`global.identity.auth.identity.initialClaimName`	defines the initial claim name, which is used by Identity to configure initial mapping rules,	`oid`
`global.identity.auth.identity.initialClaimValue`	defines the initial claim value, which is used by Identity to configure initial mapping rules.	`nil`
`global.identity.auth.operate`	configuration to configure Operate authentication specifics on global level, which can be accessed by other sub-charts	
`global.identity.auth.operate.clientId`	defines the client id, which is used by Operate in authentication flows.	`operate`
`global.identity.auth.operate.audience`	defines the audience, which is used by Operate.	`operate-api`
`global.identity.auth.operate.existingSecret`	can be used to reference an existing secret. If not set, a random secret is generated.	`nil`
`global.identity.auth.operate.redirectUrl`	defines the redirect URL, which is used by Keycloak to access Operate.	`http://localhost:8081`
`global.identity.auth.tasklist`	configuration to configure Tasklist authentication specifics on global level, which can be accessed by other sub-charts	
`global.identity.auth.tasklist.clientId`	defines the client id, which is used by Tasklist in authentication flows.	`tasklist`
`global.identity.auth.tasklist.audience`	defines the audience, which is used by Tasklist.	`tasklist-api`
`global.identity.auth.tasklist.existingSecret`	can be used to use an own existing secret. If not set a random secret is generated.	`nil`
`global.identity.auth.tasklist.redirectUrl`	defines the root (or redirect) URL, which is used by Keycloak to access Tasklist.	`http://localhost:8082`
`global.identity.auth.optimize`	configuration to configure Optimize authentication specifics on global level, which can be accessed by other sub-charts	
`global.identity.auth.optimize.clientId`	defines the client id, which is used by Optimize in authentication flows.	`optimize`
`global.identity.auth.optimize.audience`	defines the audience, which is used by Optimize.	`optimize-api`
`global.identity.auth.optimize.existingSecret`	can be used to use an own existing secret. If not set a random secret is generated.	`nil`
`global.identity.auth.optimize.redirectUrl`	defines the root (or redirect) URL, which is used by Keycloak to access Optimize.	`http://localhost:8083`
`global.identity.auth.webModeler`	configuration to configure WebModeler authentication specifics on global level, which can be accessed by other sub-charts	
`global.identity.auth.webModeler.clientId`	defines the client id, which is used by WebModeler in authentication flows.	`web-modeler`
`global.identity.auth.webModeler.clientApiAudience`	defines the audience which is used by WebModeler's client API.	`web-modeler-api`
`global.identity.auth.webModeler.publicApiAudience`	defines the audience which is used by WebModeler's public API.	`web-modeler-public-api`
`global.identity.auth.webModeler.redirectUrl`	defines the root URL which is used by Keycloak to access WebModeler.	`http://localhost:8084`
`global.identity.auth.console`	configuration to configure Console authentication specifics on global level, which can be accessed by other sub-charts	
`global.identity.auth.console.existingSecret`	can be used to use an own existing secret. If not set a random secret is generated.	`nil`
`global.identity.auth.console.redirectUrl`	defines the root URL which is used by Keycloak to access WebModeler.	`http://localhost:8080`
`global.identity.auth.console.audience`	can be used to Console audience in Identity.	`console-api`
`global.identity.auth.zeebe`	configuration to configure Zeebe authentication specifics on global level, which can be accessed by other sub-charts	
`global.identity.auth.zeebe.clientId`	defines the client id, which is used by Zeebe in authentication flows.	`zeebe`
`global.identity.auth.zeebe.existingSecret`	can be used to use an own existing secret. If not set a random secret is generated.	`""`
`global.identity.auth.zeebe.audience`	defines the audience, which is used by Zeebe.	`zeebe-api`
`global.identity.auth.zeebe.tokenScope`	defines the token scope, which is used by Zeebe.	`nil`

Console Parameters

Name	Description	Value
`console`	configuration for the Console.	
`console.enabled`	if true, the Console deployment and its related resources are deployed via a helm release	`false`

Name	Description	Value
`console.configuration`	Configuration passed directly to Console as YAML file. More details on Console official documenations	`""`
`console.image.registry`	can be used to set container image registry.	`registry.camunda.cloud`
`console.image.repository`	defines which image repository to use	`console/console-sm`
`console.image.tag`	can be used to set the Docker image tag for the Console image (overwrites global.image.tag)	`8.5.4`
`console.image.pullSecrets`	can be used to configure image pull secrets https://kubernetes.io/docs/concepts/containers/images/#specifying-imagepullsecrets-on-a-pod	`[]`
`console.sidecars`	can be used to attach extra containers to the console deployment	`[]`
`console.replicas`	Number of Console replicas	`1`
`console.contextPath`	can be used to make Console web application works on a custom sub-path. This is mainly used to run Camunda web applications under a single domain.	`""`
`console.initContainers`	can be used to set up extra init containers for the application Pod	`[]`
`console.podAnnotations`	can be used to define extra Console pod annotations	`{}`
`console.podLabels`	can be used to define extra Console pod labels	`{}`
`console.logging`	configuration for the Console logging. This template will be directly included in the Operate configuration YAML file	`{}`
`console.service.annotations`	can be used to define annotations, which will be applied to the Console service	`{}`
`console.service.type`	defines the type of the service https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types	`ClusterIP`
`console.service.port`	defines the port number where the web application will be available	`80`
`console.service.serverName`	defines the port name where the web application will be available	`http`
`console.service.managementPort`	defines the management port used to access metrics and app status	`9100`
`console.resources.requests.memory`		`1Gi`
`console.resources.limits.cpu`		`2`
`console.resources.limits.memory`		`2Gi`
`console.resources.requests.cpu`		`1`
`console.env`	can be used to set extra environment variables in each app container	`[]`
`console.command`	can be used to override the default command provided by the container image. See https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/	`[]`
`console.extraVolumes`	can be used to define extra volumes for the Console pods, useful for TLS and self-signed certificates	`[]`
`console.extraVolumeMounts`	can be used to mount extra volumes for the Console pods, useful for TLS and self-signed certificates	`[]`
`console.startupProbe.enabled`	if true, the startup probe is enabled in app container	`false`
`console.startupProbe.scheme`	defines the startup probe scheme used on calling the probePath	`HTTP`
`console.startupProbe.probePath`	defines the startup probe route used on the app	`/health/readiness`
`console.startupProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`console.startupProbe.periodSeconds`	defines how often the probe is executed	`30`
`console.startupProbe.successThreshold`	defines how often it needs to be true to be marked as ready, after failure	`1`
`console.startupProbe.failureThreshold`	defines when the probe is considered as failed so the Pod will be marked Unready	`5`
`console.startupProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`console.readinessProbe.enabled`	if true, the readiness probe is enabled in app container	`true`
`console.readinessProbe.scheme`	defines the startup probe scheme used on calling the probePath	`HTTP`
`console.readinessProbe.probePath`	defines the readiness probe route used on the app	`/health/readiness`
`console.readinessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`console.readinessProbe.periodSeconds`	defines how often the probe is executed	`30`
`console.readinessProbe.successThreshold`	defines how often it needs to be true to be marked as ready, after failure	`1`
`console.readinessProbe.failureThreshold`	defines when the probe is considered as failed so the Pod will be marked Unready	`5`
`console.readinessProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`console.livenessProbe.enabled`	if true, the liveness probe is enabled in app container	`false`
`console.livenessProbe.scheme`	defines the startup probe scheme used on calling the probePath	`HTTP`
`console.livenessProbe.probePath`	defines the liveness probe route used on the app	`/health/liveness`
`console.livenessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`console.livenessProbe.periodSeconds`	defines how often the probe is executed	`30`
`console.livenessProbe.successThreshold`	defines how often it needs to be true to be considered successful after having failed	`1`
`console.livenessProbe.failureThreshold`	defines when the probe is considered as failed so the container will be restarted	`5`
`console.livenessProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`console.metrics.prometheus`	Prometheus metrics endpoint	`/prometheus`
`console.serviceAccount.enabled`	if true, enables the Console service account	`true`
`console.serviceAccount.name`	can be used to set the name of the Console service account	`""`
`console.serviceAccount.annotations`	can be used to set the annotations of the Operate service account	`{}`

Name	Description	Value
`console.serviceAccount.automountServiceAccountToken`	can be used to control whether the service account token should be automatically mounted	`false`
`console.ingress.enabled`	if true, an ingress resource is deployed with the Console deployment. Only useful if an ingress controller is available, like nginx.	`false`
`console.ingress.className`	defines the class or configuration of ingress which should be used by the controller	`nginx`
`console.ingress.annotations`	defines the ingress related annotations, consumed mostly by the ingress controller	`{}`
`console.ingress.path`	defines the path which is associated with the Console service and port https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules	`/`
`console.ingress.pathType`	can be used to define the Ingress path type. https://kubernetes.io/docs/concepts/services-networking/ingress/#path-types	`Prefix`
`console.ingress.host`	can be used to define the host of the ingress rule. https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules	`""`
`console.ingress.tls.enabled`	if true, then tls is configured on the ingress resource. If enabled the Ingress.host need to be defined.	`false`
`console.ingress.tls.secretName`	defines the secret name which contains the TLS private key and certificate	`camunda-platform-console`
`console.podSecurityContext`	defines the security options the Console broker pod should be run with	
`console.podSecurityContext.runAsNonRoot`	run as non root	`true`
`console.podSecurityContext.fsGroup`		`1001`
`console.containerSecurityContext.allowPrivilegeEscalation`		`false`
`console.containerSecurityContext.privileged`		`false`
`console.containerSecurityContext.readOnlyRootFilesystem`		`true`
`console.containerSecurityContext.runAsNonRoot`		`true`
`console.containerSecurityContext.runAsUser`		`1001`
`console.nodeSelector`	can be used to define on which nodes the Console pods should run	`{}`
`console.tolerations`	can be used to define pod toleration’s https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/	`[]`
`console.affinity`	can be used to define pod affinity or anti-affinity https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity	`{}`

Zeebe Parameters

Name	Description	Value
`zeebe`	configuration for the Zeebe sub chart. Contains configuration for the Zeebe broker and related resources.	
`zeebe.enabled`	if true, all zeebe related resources are deployed via the helm release	`true`
`zeebe.debug`	if true, extra info is printed.	`false`
`zeebe.image`	configuration to configure the zeebe image specifics	
`zeebe.image.registry`	can be used to set container image registry.	`""`
`zeebe.image.repository`	defines which image repository to use	`camunda/zeebe`
`zeebe.image.tag`	can be set to overwrite the global tag, which should be used in that chart	`nil`
`zeebe.image.pullSecrets`	can be used to configure image pull secrets https://kubernetes.io/docs/concepts/containers/images/#specifying-imagepullsecrets-on-a-pod	`[]`
`zeebe.sidecars`	can be used to attach extra containers to the zeebe deployment	`[]`
`zeebe.clusterSize`	defines the amount of brokers (=replicas), which are deployed via helm	`3`
`zeebe.partitionCount`	defines how many zeebe partitions are set up in the cluster	`3`
`zeebe.replicationFactor`	defines how each partition is replicated, the value defines the number of nodes	`3`
`zeebe.env`	can be used to set extra environment variables in each zeebe broker container	
`zeebe.env[0].name`		`ZEEBE_BROKER_DATA_SNAPSHOTPERIOD`
`zeebe.env[0].value`		`5m`
`zeebe.env[1].name`		`ZEEBE_BROKER_DATA_DISK_FREESPACE_REPLICATION`
`zeebe.env[1].value`		`2GB`
`zeebe.env[2].name`		`ZEEBE_BROKER_DATA_DISK_FREESPACE_PROCESSING`
`zeebe.env[2].value`		`3GB`
`zeebe.configMap`	configuration which will be applied to the mounted config map.	
`zeebe.configMap.defaultMode`	can be used to set permissions on created files by default. Must be an octal value between 0000 and 0777 or a decimal value between 0 and 511. see https://github.com/kubernetes/api/blob/master/core/v1/types.go#L1615-L1623	`754`
`zeebe.command`	can be used to override the default command provided by the container image. See https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/	`[]`

Name	Description	Value
`zeebe.logLevel`	defines the log level which is used by the zeebe brokers	`info`
`zeebe.log4j2`	can be used to overwrite the log4j2 configuration of the zeebe brokers	`""`
`zeebe.javaOpts`	can be used to set java options for the zeebe brokers	`-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/usr/local/zeebe/data -XX:ErrorFile=/usr/local/zeebe/data/zeebe_error%p.log -XX:+ExitOnOutOfMemoryError`
`zeebe.service`	configuration for the broker service	
`zeebe.service.annotations`	can be used to define annotations, which will be applied to the Zeebe service	`{}`
`zeebe.service.type`	defines the type of the service https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types	`ClusterIP`
`zeebe.service.httpPort`	defines the port of the http endpoint, where for example metrics are provided	`9600`
`zeebe.service.httpName`	defines the name of the http endpoint, where for example metrics are provided	`http`
`zeebe.service.commandPort`	defines the port of the command api endpoint, where the broker commands are sent to	`26501`
`zeebe.service.commandName`	defines the name of the command api endpoint, where the broker commands are sent to	`command`
`zeebe.service.internalPort`	defines the port of the internal api endpoint, which is used for internal communication	`26502`
`zeebe.service.internalName`	defines the name of the internal api endpoint, which is used for internal communication	`internal`
`zeebe.service.extraPorts`	can be used to expose any other ports which are required. Can be useful for exporters	`[]`
`global.zeebe.ServiceAccount`	configuration for the service account where the broker pods are assigned to	
`zeebe.serviceAccount.enabled`	if true, enables the broker service account	`true`
`zeebe.serviceAccount.name`	can be used to set the name of the broker service account	`""`
`zeebe.serviceAccount.annotations`	can be used to set the annotations of the broker service account	`{}`
`zeebe.serviceAccount.automountServiceAccountToken`	can be used to control whether the service account token should be automatically mounted	`false`
`zeebe.cpuThreadCount`	defines how many threads can be used for the processing on each broker pod	`3`
`zeebe.ioThreadCount`	defines how many threads can be used for the exporting on each broker pod	`3`
`zeebe.resources`	configuration to set request and limit configuration for the container https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#requests-and-limits	
`zeebe.resources.requests`		
`zeebe.resources.requests.cpu`		`800m`
`zeebe.resources.requests.memory`		`1200Mi`
`zeebe.resources.limits.cpu`		`960m`
`zeebe.resources.limits.memory`		`1920Mi`
`zeebe.persistenceType`	defines the type of persistence which is used by Zeebe. Possible values are: disk, local and memory.	`disk`
`zeebe.pvcSize`	defines the persistent volume claim size, which is used by each broker pod https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims	`32Gi`
`zeebe.pvcAccessModes`	can be used to configure the persistent volume claim access mode https://kubernetes.io/docs/concepts/storage/persistent-volumes/#access-modes	`["ReadWriteOnce"]`
`zeebe.pvcStorageClassName`	can be used to set the storage class name which should be used by the persistent volume claim. It is recommended to use a storage class, which is backed with a SSD.	`""`
`zeebe.pvcAnnotations`	can be used to specify custom annotations for Zeebe's persistent volume claims, enhancing storage configuration flexibility.	`{}`
`zeebe.extraVolumes`	can be used to define extra volumes for the broker pods, useful for additional exporters	`[]`
`zeebe.extraVolumeMounts`	can be used to mount extra volumes for the broker pods, useful for additional exporters	`[]`
`zeebe.extraInitContainers`	(Deprecated - use `initContainers` instead) ExtraInitContainers can be used to set up extra init containers for the broker pods, useful for additional exporters	`[]`
`zeebe.initContainers`	can be used to set up extra init containers for the application Pod	`[]`
`zeebe.podAnnotations`	can be used to define extra broker pod annotations	`{}`
`zeebe.podLabels`	can be used to define extra broker pod labels	`{}`
`zeebe.podDisruptionBudget`	configuration to configure a pod disruption budget for the broker pods https://kubernetes.io/docs/tasks/run-application/configure-pdb/	
`zeebe.podDisruptionBudget.enabled`	if true a pod disruption budget is defined for the brokers	`false`
`zeebe.podDisruptionBudget.minAvailable`	can be used to set how many pods should be available. Be aware that if minAvailable is set, maxUnavailable will not be set (they are mutually exclusive).	`nil`
`zeebe.podDisruptionBudget.maxUnavailable`	can be used to set how many pods should be at max. unavailable	`1`
`zeebe.podSecurityContext`	defines the security options the Zeebe broker pod should be run with	

Name	Description	Value
`zeebe.podSecurityContext.runAsNonRoot`	run as non root	`true`
`zeebe.podSecurityContext.fsGroup`		`1001`
`zeebe.containerSecurityContext`	defines the security options the Zeebe broker container should be run with	
`zeebe.containerSecurityContext.allowPrivilegeEscalation`		`false`
`zeebe.containerSecurityContext.privileged`		`false`
`zeebe.containerSecurityContext.readOnlyRootFilesystem`		`true`
`zeebe.containerSecurityContext.runAsNonRoot`		`true`
`zeebe.containerSecurityContext.runAsUser`		`1001`
`zeebe.startupProbe`	configuration	
`zeebe.startupProbe.enabled`	if true, the startup probe is enabled in app container	`false`
`zeebe.startupProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`zeebe.startupProbe.probePath`	defines the startup probe route used on the app	`/actuator/health/startup`
`zeebe.startupProbe.initialDelaySeconds`	defines the number of seconds after the container has started before the probe is initiated.	`30`
`zeebe.startupProbe.periodSeconds`	defines how often the probe is executed	`30`
`zeebe.startupProbe.successThreshold`	defines how often it needs to be true to be marked as ready, after failure	`1`
`zeebe.startupProbe.failureThreshold`	defines when the probe is considered as failed so the Pod will be marked Unready	`5`
`zeebe.startupProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`zeebe.readinessProbe`	configuration	
`zeebe.readinessProbe.enabled`	if true, the readiness probe is enabled in app container	`true`
`zeebe.readinessProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`zeebe.readinessProbe.probePath`	defines the readiness probe route used on the app	`/actuator/health/readiness`
`zeebe.readinessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`zeebe.readinessProbe.periodSeconds`	defines how often the probe is executed	`30`
`zeebe.readinessProbe.successThreshold`	defines how often it needs to be true to be marked as ready, after failure	`1`
`zeebe.readinessProbe.failureThreshold`	defines when the probe is considered as failed so the Pod will be marked Unready	`5`
`zeebe.readinessProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`zeebe.livenessProbe`	configuration	
`zeebe.livenessProbe.enabled`	if true, the liveness probe is enabled in app container	`false`
`zeebe.livenessProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`zeebe.livenessProbe.probePath`	defines the liveness probe route used on the app	`/actuator/health/liveness`
`zeebe.livenessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`zeebe.livenessProbe.periodSeconds`	defines how often the probe is executed	`30`
`zeebe.livenessProbe.successThreshold`	defines how often it needs to be true to be considered successful after having failed	`1`
`zeebe.livenessProbe.failureThreshold`	defines when the probe is considered as failed so the container will be restarted	`5`
`zeebe.livenessProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`zeebe.metrics.prometheus`	Prometheus metrics endpoint	`/actuator/prometheus`
`zeebe.nodeSelector`	can be used to define on which nodes the broker pods should run	`{}`
`zeebe.tolerations`	can be used to define pod toleration's https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/	`[]`
`global.zeebe.Affinity`	can be used to define pod affinity or anti-affinity https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity	
`zeebe.priorityClassName`	can be used to define the broker pods priority https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#priorityclass	`""`
`zeebe.retention.enabled`	if true, the ILM Policy is created and applied to the index templates.	`false`
`zeebe.retention.minimumAge`	defines how old the data must be, before the data is deleted as a duration.	`30d`
`zeebe.retention.policyName`	defines the name of the created and applied ILM policy.	`zeebe-record-retention-policy`
`zeebe.configuration`	if specified, contents will be used as the application.yaml	`""`
`zeebe.extraConfiguration`	if specified, contents will be used for any extra configuration files such as log4j2.xml	`{}`

ZeebeGateway Parameters

Name	Description	Value
`Gateway`	configuration to define properties related to the standalone gateway	
`zeebeGateway.replicas`	defines how many standalone gateways are deployed	`2`
`zeebeGateway.image`	configuration to configure the ZeebeGateway image specifics	
`zeebeGateway.image.registry`	can be used to set container image registry.	`""`
`zeebeGateway.image.repository`	defines which image repository to use	`camunda/zeebe`
`zeebeGateway.image.tag`	can be set to overwrite the global tag, which should be used in that chart	`nil`

Name	Description	Value
`zeebeGateway.image.pullSecrets`	can be used to configure image pull secrets https://kubernetes.io/docs/concepts/containers/images/#specifying-imagepullsecrets-on-a-pod	`[]`
`zeebeGateway.sidecars`	can be used to attach extra containers to the ZeebeGateway deployment	`[]`
`zeebeGateway.podAnnotations`	can be used to define extra gateway pod annotations	`{}`
`zeebeGateway.podLabels`	can be used to define extra gateway pod labels	`{}`
`zeebeGateway.logLevel`	defines the log level which is used by the gateway	`info`
`zeebeGateway.log4j2`	can be used to overwrite the log4j2 configuration of the gateway	`""`
`zeebeGateway.javaOpts`	can be used to set java options for the ZeebeGateway	`-XX:+ExitOnOutOfMemoryError`
`zeebeGateway.env`	can be used to set extra environment variables in each gateway container	`[]`
`zeebeGateway.configMap`	configuration which will be applied to the mounted config map.	
`zeebeGateway.configMap.defaultMode`	can be used to set permissions on created files by default. Must be an octal value between 0000 and 0777 or a decimal value between 0 and 511.	`744`
`zeebeGateway.command`	can be used to override the default command provided by the container image. See https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/	`[]`
`zeebeGateway.podDisruptionBudget`	configuration to configure a pod disruption budget for the gateway pods https://kubernetes.io/docs/tasks/run-application/configure-pdb/	
`zeebeGateway.podDisruptionBudget.enabled`	if true a pod disruption budget is defined for the gateways	`false`
`zeebeGateway.podDisruptionBudget.minAvailable`	can be used to set how many pods should be available. Be aware that if minAvailable is set, maxUnavailable will not be set (they are mutually exclusive).	`1`
`zeebeGateway.podDisruptionBudget.maxUnavailable`	can be used to set how many pods should be at max. unavailable	`nil`
`zeebeGateway.resources`	configuration to set request and limit configuration for the container https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#requests-and-limits	
`zeebeGateway.resources.requests.cpu`		`400m`
`zeebeGateway.resources.requests.memory`		`450Mi`
`zeebeGateway.resources.limits.cpu`		`400m`
`zeebeGateway.resources.limits.memory`		`450Mi`
`zeebeGateway.priorityClassName`	can be used to define the gateway pods priority https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#priorityclass	`""`
`zeebeGateway.podSecurityContext`	defines the security options the gateway pod should be run wit	
`zeebeGateway.podSecurityContext.runAsNonRoot`		`true`
`zeebeGateway.podSecurityContext.fsGroup`		`1001`
`zeebeGateway.containerSecurityContext`	defines the security options the gateway container should be run with	
`zeebeGateway.containerSecurityContext.allowPrivilegeEscalation`		`false`
`zeebeGateway.containerSecurityContext.privileged`		`false`
`zeebeGateway.containerSecurityContext.readOnlyRootFilesystem`		`true`
`zeebeGateway.containerSecurityContext.runAsNonRoot`		`true`
`zeebeGateway.containerSecurityContext.runAsUser`		`1001`
`zeebeGateway.startupProbe`	configuration	
`zeebeGateway.startupProbe.enabled`	if true, the startup probe is enabled in app container	`false`
`zeebeGateway.startupProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`zeebeGateway.startupProbe.probePath`	defines the startup probe route used on the app	`/actuator/health/startup`
`zeebeGateway.startupProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`zeebeGateway.startupProbe.periodSeconds`	defines how often the probe is executed	`30`
`zeebeGateway.startupProbe.successThreshold`	defines how often it needs to be true to be marked as ready, after failure	`1`
`zeebeGateway.startupProbe.failureThreshold`	defines when the probe is considered as failed so the Pod will be marked Unready	`5`
`zeebeGateway.startupProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`zeebeGateway.readinessProbe`	configuration	
`zeebeGateway.readinessProbe.enabled`	if true, the readiness probe is enabled in app container	`true`
`zeebeGateway.readinessProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`zeebeGateway.readinessProbe.probePath`	defines the readiness probe route used on the app	`/actuator/health/readiness`
`zeebeGateway.readinessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`zeebeGateway.the`	probe is initiated.	
`zeebeGateway.readinessProbe.periodSeconds`	defines how often the probe is executed	`30`
`zeebeGateway.readinessProbe.successThreshold`	defines how often it needs to be true to be marked as ready, after failure	`1`
`zeebeGateway.readinessProbe.failureThreshold`	defines when the probe is considered as failed so the Pod will be marked Unready	`5`
`zeebeGateway.readinessProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`zeebeGateway.livenessProbe`	configuration	
`zeebeGateway.livenessProbe.enabled`	if true, the liveness probe is enabled in app container	`false`
`zeebeGateway.livenessProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`zeebeGateway.livenessProbe.probePath`	defines the liveness probe route used on the app	`/actuator/health/liveness`

Name	Description	Value
`zeebeGateway.livenessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`zeebeGateway.livenessProbe.periodSeconds`	defines how often the probe is executed	`30`
`zeebeGateway.livenessProbe.successThreshold`	defines how often it needs to be true to be considered successful after having failed	`1`
`zeebeGateway.livenessProbe.failureThreshold`	defines when the probe is considered as failed so the container will be restarted	`5`
`zeebeGateway.livenessProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`zeebeGateway.metrics.prometheus`	Prometheus metrics endpoint	`/actuator/prometheus`
`zeebeGateway.nodeSelector`	can be used to define on which nodes the gateway pods should run	`{}`
`zeebeGateway.tolerations`	can be used to define pod toleration's https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/	`[]`
`zeebeGateway.affinity`	can be used to define pod affinity or anti-affinity https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity	
`zeebeGateway.extraVolumeMounts`	can be used to mount extra volumes for the gateway pods, useful for enabling tls between gateway and broker	`[]`
`zeebeGateway.extraVolumes`	can be used to define extra volumes for the gateway pods, useful for enabling tls between gateway and broker	`[]`
`zeebeGateway.extraInitContainers`	(Deprecated - use `initContainers` instead) can be used to set up extra init containers for the gateway pods, useful for adding interceptors	`[]`
`zeebeGateway.initContainers`	can be used to set up extra init containers for the application Pod	`[]`
`zeebeGateway.service`	configuration for the gateway service	
`zeebeGateway.service.annotations`	can be used to define annotations, which will be applied to the zeebe-gateway service	`{}`
`zeebeGateway.service.type`	defines the type of the service https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types	`ClusterIP`
`zeebeGateway.service.loadBalancerIP`	defines public ip of the load balancer if the type is LoadBalancer	`""`
`zeebeGateway.service.loadBalancerSourceRanges`	defines list of allowed source ip address ranges if the type is LoadBalancer	`[]`
`zeebeGateway.service.httpPort`	defines the port of the http endpoint, where for example metrics are provided	`9600`
`zeebeGateway.service.httpName`	defines the name of the http endpoint, where for example metrics are provided	`http`
`zeebeGateway.service.grpcPort`	defines the port of the gateway gRPC endpoint, where client commands (grpc) are sent to	`26500`
`zeebeGateway.service.grpcName`	defines the name of the gateway gRPC endpoint, where client commands (grpc) are sent to	`gateway`
`zeebeGateway.service.restPort`	defines the REST port of the gateway REST endpoint, where client commands (REST) are sent to	`8080`
`zeebeGateway.service.restName`	defines the name of the gateway REST endpoint, where client commands (REST) are sent to	`rest`
`zeebeGateway.service.internalPort`	defines the port of the internal api endpoint, which is used for internal communication	`26502`
`zeebeGateway.service.internalName`	defines the name of the internal api endpoint, which is used for internal communication	`internal`
`zeebeGateway.serviceAccount`	configuration for the service account where the gateway pods are assigned to	
`zeebeGateway.serviceAccount.enabled`	if true, enables the gateway service account	`true`
`zeebeGateway.serviceAccount.name`	can be used to set the name of the gateway service account	`""`
`zeebeGateway.serviceAccount.annotations`	can be used to set the annotations of the gateway service account	`{}`
`zeebeGateway.serviceAccount.automountServiceAccountToken`	can be used to control whether the service account token should be automatically mounted	`false`
`zeebeGateway.ingress.grpc.enabled`	if true, an ingress resource is deployed with the Zeebe gateway deployment. Only useful if an ingress controller is available, like nginx.	`false`
`zeebeGateway.ingress.grpc.className`	defines the class or configuration of ingress which should be used by the controller	`nginx`
`zeebeGateway.ingress.grpc.annotations`	defines the ingress related annotations, consumed mostly by the ingress controller	`{}`
`zeebeGateway.ingress.grpc.path`	defines the path which is associated with the Zeebe gateway's gRPC service and port https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules	`/`
`zeebeGateway.ingress.grpc.pathType`	can be used to define the Ingress path type. https://kubernetes.io/docs/concepts/services-networking/ingress/#path-types	`Prefix`
`zeebeGateway.ingress.grpc.host`	can be used to define the host of the ingress rule. https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules	`""`
`zeebeGateway.ingress.grpc.tls`	configuration for tls on the ingress resource https://kubernetes.io/docs/concepts/services-networking/ingress/#tls	
`zeebeGateway.ingress.grpc.tls.enabled`	if true, then tls is configured on the ingress resource. If enabled the Ingress.host need to be defined.	`false`
`zeebeGateway.ingress.grpc.tls.secretName`	defines the secret name which contains the TLS private key and certificate	`camunda-platform-zeebe-gateway-grpc`
`zeebeGateway.ingress.rest.enabled`	if true, an ingress resource is deployed with the Zeebe gateway deployment. Only useful if an ingress controller is available, like nginx.	`false`
`zeebeGateway.ingress.rest.className`	defines the class or configuration of ingress which should be used by the controller	`nginx`
`zeebeGateway.ingress.rest.annotations`	defines the ingress related annotations, consumed mostly by the ingress controller	`{}`

Name	Description	Value
`zeebeGateway.ingress.rest.path`	defines the path which is associated with the Zeebe gateway's REST service and port https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules	`/`
`zeebeGateway.ingress.rest.pathType`	can be used to define the Ingress path type. https://kubernetes.io/docs/concepts/services-networking/ingress/#path-types	`Prefix`
`zeebeGateway.ingress.rest.host`	can be used to define the host of the ingress rule. https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules	`""`
`zeebeGateway.ingress.rest.tls`	configuration for tls on the ingress resource https://kubernetes.io/docs/concepts/services-networking/ingress/#tls	
`zeebeGateway.ingress.rest.tls.enabled`	if true, then tls is configured on the ingress resource. If enabled the Ingress.host need to be defined.	`false`
`zeebeGateway.ingress.rest.tls.secretName`	defines the secret name which contains the TLS private key and certificate	`camunda-platform-zeebe-gateway-rest`
`zeebeGateway.contextPath`	can be used to make Zeebe web application works on a custom sub-path. This is mainly used to run Camunda web applications under a single domain.	`""`
`zeebeGateway.configuration`	if specified, contents will be used as the application.yaml	`""`
`zeebeGateway.extraConfiguration`	if specified, contents will be used for any extra configuration files such as log4j2.xml	`{}`

Operate Parameters

Name	Description	Value
`.operate`	configuration for the Operate sub chart.	
`operate.enabled`	if true, the Operate deployment and its related resources are deployed via a helm release	`true`
`operate.image`	configuration to configure the Operate image specifics	
`operate.image.registry`	can be used to set container image registry.	`""`
`operate.image.repository`	defines which image repository to use	`camunda/operate`
`operate.image.tag`	can be set to overwrite the global tag, which should be used in that chart	`nil`
`operate.image.pullSecrets`	can be used to configure image pull secrets https://kubernetes.io/docs/concepts/containers/images/#specifying-imagepullsecrets-on-a-pod	`[]`
`operate.sidecars`	can be used to attach extra containers to the operate deployment	`[]`
`operate.initContainers`	can be used to set up extra init containers for the application Pod	`[]`
`operate.contextPath`	can be used to make Operate web application works on a custom sub-path. This is mainly used to run Camunda web applications under a single domain.	`""`
`operate.podAnnotations`	can be used to define extra Operate pod annotations	`{}`
`operate.podLabels`	can be used to define extra Operate pod labels	`{}`
`operate.logging`	configuration for the Operate logging. This template will be directly included in the Operate configuration YAML file	
`operate.logging.level.ROOT`		`INFO`
`operate.logging.level.io.camunda.operate`		`INFO`
`operate.service`	configuration to configure the Operate service.	
`operate.service.annotations`	can be used to define annotations, which will be applied to the Operate service	`{}`
`operate.service.type`	defines the type of the service https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types	`ClusterIP`
`operate.service.port`	defines the port of the service, where the Operate web application will be available	`80`
`operate.resources`	configuration to set request and limit configuration for the container https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#requests-and-limits	
`operate.resources.requests.cpu`		`600m`
`operate.resources.requests.memory`		`400Mi`
`operate.resources.limits.cpu`		`2000m`
`operate.resources.limits.memory`		`2Gi`
`operate.env`	can be used to set extra environment variables in each Operate container	`[]`
`operate.configMap`	configuration which will be applied to the mounted config map.	
`operate.configMap.defaultMode`	can be used to set permissions on created files by default. Must be an octal value between 0000 and 0777 or a decimal value between 0 and 511.	`744`
`operate.command`	can be used to override the default command provided by the container image. See https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/	`[]`
`operate.extraVolumes`	can be used to define extra volumes for the Operate pods, useful for tls and self-signed certificates	`[]`
`operate.extraVolumeMounts`	can be used to mount extra volumes for the Operate pods, useful for tls and self-signed certificates	`[]`

Name	Description	Value
`operate.serviceAccount`	configuration for the service account where the Operate pods are assigned to	
`operate.serviceAccount.enabled`	if true, enables the Operate service account	`true`
`operate.serviceAccount.name`	can be used to set the name of the Operate service account	`""`
`operate.serviceAccount.annotations`	can be used to set the annotations of the Operate service account	`{}`
`operate.serviceAccount.automountServiceAccountToken`	can be used to control whether the service account token should be automatically mounted	`false`
`operate.ingress.enabled`	if true, an ingress resource is deployed with the Operate deployment. Only useful if an ingress controller is available, like nginx.	`false`
`operate.ingress.className`	defines the class or configuration of ingress which should be used by the controller	`nginx`
`operate.ingress.annotations`	defines the ingress related annotations, consumed mostly by the ingress controller	`{}`
`operate.ingress.path`	defines the path which is associated with the Operate service and port https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules	`/`
`operate.ingress.pathType`	can be used to define the Ingress path type. https://kubernetes.io/docs/concepts/services-networking/ingress/#path-types	`Prefix`
`operate.ingress.host`	can be used to define the host of the ingress rule. https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules	`""`
`Ingress.tls`	configuration for tls on the ingress resource https://kubernetes.io/docs/concepts/services-networking/ingress/#tls	
`operate.ingress.tls.enabled`	if true, then tls is configured on the ingress resource. If enabled the Ingress.host need to be defined.	`false`
`operate.ingress.tls.secretName`	defines the secret name which contains the TLS private key and certificate	`camunda-platform-operate`
`operate.podSecurityContext`	defines the security options the Operate pod should be run with	
`operate.podSecurityContext.runAsNonRoot`		`true`
`operate.podSecurityContext.fsGroup`		`1001`
`operate.containerSecurityContext`	defines the security options the Operate container should be run with	
`operate.containerSecurityContext.allowPrivilegeEscalation`		`false`
`operate.containerSecurityContext.privileged`		`false`
`operate.containerSecurityContext.readOnlyRootFilesystem`		`true`
`operate.containerSecurityContext.runAsNonRoot`		`true`
`operate.containerSecurityContext.runAsUser`		`1001`
`operate.startupProbe`	configuration	
`operate.startupProbe.enabled`	if true, the startup probe is enabled in app container	`false`
`operate.startupProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`operate.startupProbe.probePath`	defines the startup probe route used on the app	`/actuator/health/readiness`
`operate.startupProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`operate.startupProbe.periodSeconds`	defines how often the probe is executed	`30`
`operate.startupProbe.successThreshold`	defines how often it needs to be true to be marked as ready, after failure	`1`
`operate.startupProbe.failureThreshold`	defines when the probe is considered as failed so the Pod will be marked Unready	`5`
`operate.startupProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`operate.readinessProbe`	configuration	
`operate.readinessProbe.enabled`	if true, the readiness probe is enabled in app container	`true`
`operate.readinessProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`operate.readinessProbe.probePath`	defines the readiness probe route used on the app	`/actuator/health/readiness`
`operate.readinessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`operate.readinessProbe.periodSeconds`	defines how often the probe is executed	`30`
`operate.readinessProbe.successThreshold`	defines how often it needs to be true to be marked as ready, after failure	`1`
`operate.readinessProbe.failureThreshold`	defines when the probe is considered as failed so the Pod will be marked Unready	`5`
`operate.readinessProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`operate.livenessProbe`	configuration	
`operate.livenessProbe.enabled`	if true, the liveness probe is enabled in app container	`false`
`operate.livenessProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`operate.livenessProbe.probePath`	defines the liveness probe route used on the app	`/actuator/health/liveness`
`operate.livenessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`operate.livenessProbe.periodSeconds`	defines how often the probe is executed	`30`
`operate.livenessProbe.successThreshold`	defines how often it needs to be true to be considered successful after having failed	`1`
`operate.livenessProbe.failureThreshold`	defines when the probe is considered as failed so the container will be restarted	`5`
`operate.livenessProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`operate.metrics.prometheus`	Prometheus metrics endpoint	`/actuator/prometheus`
`operate.nodeSelector`	can be used to define on which nodes the Operate pods should run	`{}`
`operate.tolerations`	can be used to define pod toleration's https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/	`[]`
`operate.affinity`	can be used to define pod affinity or anti-affinity https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity	`{}`

Name	Description	Value
`operate.retention.enabled`	if true, the ILM Policy is created and applied to the index templates.	`false`
`operate.retention.minimumAge`	defines how old the data must be, before the data is deleted as a duration.	`30d`
`operate.configuration`	if specified, contents will be used as the application.yaml	`""`
`operate.extraConfiguration`	if specified, contents will be used for any extra configuration files such as the log4j2.xml	`{}`

Tasklist Parameters

Name	Description	Value
`tasklist.enabled`	if true, the tasklist deployment and its related resources are deployed via a helm release	`true`
`tasklist.image`	configuration to configure the tasklist image specifics	
`tasklist.image.registry`	can be used to set container image registry.	`""`
`tasklist.image.repository`	defines which image repository to use	`camunda/tasklist`
`tasklist.image.tag`	can be set to overwrite the global tag, which should be used in that chart	`nil`
`tasklist.image.pullSecrets`	can be used to configure image pull secrets https://kubernetes.io/docs/concepts/containers/images/#specifying-imagepullsecrets-on-a-pod	`[]`
`tasklist.sidecars`	can be used to attach extra containers to the tasklist deployment	`[]`
`tasklist.initContainers`	can be used to set up extra init containers for the application Pod	`[]`
`tasklist.contextPath`	can be used to make Tasklist web application works on a custom sub-path. This is mainly used to run Camunda web applications under a single domain.	`""`
`tasklist.env`	can be used to set extra environment variables on each Tasklist container	`[]`
`tasklist.podAnnotations`	can be used to define extra Tasklist pod annotations	`{}`
`tasklist.podLabels`	can be used to define extra tasklist pod labels	`{}`
`tasklist.configMap`	configuration which will be applied to the mounted config map.	
`tasklist.configMap.defaultMode`	can be used to set permissions on created files by default. Must be an octal value between 0000 and 0777 or a decimal value between 0 and 511.	`744`
`tasklist.command`	can be used to override the default command provided by the container image. See https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/	`[]`
`tasklist.service`	configuration to configure the tasklist service.	
`tasklist.service.annotations`	can be used to define annotations, which will be applied to the Tasklist service	`{}`
`tasklist.service.type`	defines the type of the service https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types	`ClusterIP`
`tasklist.service.port`	defines the port of the service, where the tasklist web application will be available	`80`
`tasklist.identity`	configures app user management.	
`tasklist.identity.userAccessRestrictions.enabled`	if true, enables the identity user access restrictions	`true`
`tasklist.extraVolumes`	can be used to define extra volumes for the Tasklist pods, useful for tls and self-signed certificates	`[]`
`tasklist.extraVolumeMounts`	can be used to mount extra volumes for the Tasklist pods, useful for tls and self-signed certificates	`[]`
`tasklist.serviceAccount`	configuration for the service account where the Tasklist pods are assigned to	
`tasklist.serviceAccount.enabled`	if true, enables the Tasklist service account	`true`
`tasklist.serviceAccount.name`	can be used to set the name of the Tasklist service account	`""`
`tasklist.serviceAccount.annotations`	can be used to set the annotations of the Tasklist service account	`{}`
`tasklist.serviceAccount.automountServiceAccountToken`	can be used to control whether the service account token should be automatically mounted	`false`
`tasklist.podSecurityContext`	defines the security options the Tasklist pod should be run with	
`tasklist.podSecurityContext.runAsNonRoot`		`true`
`tasklist.podSecurityContext.fsGroup`		`1001`
`tasklist.containerSecurityContext`	defines the security options the Tasklist container should be run with	
`tasklist.containerSecurityContext.allowPrivilegeEscalation`		`false`
`tasklist.containerSecurityContext.privileged`		`false`
`tasklist.containerSecurityContext.readOnlyRootFilesystem`		`true`
`tasklist.containerSecurityContext.runAsNonRoot`		`true`
`tasklist.containerSecurityContext.runAsUser`		`1001`
`tasklist.startupProbe`	configuration	
`tasklist.startupProbe.enabled`	if true, the startup probe is enabled in app container	`false`
`tasklist.startupProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`tasklist.startupProbe.probePath`	defines the startup probe route used on the app	`/actuator/health/readiness`
`tasklist.startupProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`

Name	Description	Value
`tasklist.startupProbe.periodSeconds`	defines how often the probe is executed	`30`
`tasklist.startupProbe.successThreshold`	defines how often it needs to be true to be marked as ready, after failure	`1`
`tasklist.startupProbe.failureThreshold`	defines when the probe is considered as failed so the Pod will be marked Unready	`5`
`tasklist.startupProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`tasklist.readinessProbe`	configuration	
`tasklist.readinessProbe.enabled`	if true, the readiness probe is enabled in app container	`true`
`tasklist.readinessProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`tasklist.readinessProbe.probePath`	defines the readiness probe route used on the app	`/actuator/health/readiness`
`tasklist.readinessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`tasklist.readinessProbe.periodSeconds`	defines how often the probe is executed	`30`
`tasklist.readinessProbe.successThreshold`	defines how often it needs to be true to be marked as ready, after failure	`1`
`tasklist.readinessProbe.failureThreshold`	defines when the probe is considered as failed so the Pod will be marked Unready	`5`
`tasklist.readinessProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`tasklist.livenessProbe`	configuration	
`tasklist.livenessProbe.enabled`	if true, the liveness probe is enabled in app container	`false`
`tasklist.livenessProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`tasklist.livenessProbe.probePath`	defines the liveness probe route used on the app	`/actuator/health/liveness`
`tasklist.livenessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`tasklist.livenessProbe.periodSeconds`	defines how often the probe is executed	`30`
`tasklist.livenessProbe.successThreshold`	defines how often it needs to be true to be considered successful after having failed	`1`
`tasklist.livenessProbe.failureThreshold`	defines when the probe is considered as failed so the container will be restarted	`5`
`tasklist.livenessProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`tasklist.metrics.prometheus`	Prometheus metrics endpoint	`/actuator/prometheus`
`tasklist.nodeSelector`	can be used to define on which nodes the Tasklist pods should run	`{}`
`tasklist.tolerations`	can be used to define pod toleration's https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/	`[]`
`tasklist.affinity`	can be used to define pod affinity or anti-affinity https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity	`{}`
`tasklist.resources`	configuration to set request and limit configuration for the container https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#requests-and-limits	
`tasklist.resources.requests.cpu`		`400m`
`tasklist.resources.requests.memory`		`1Gi`
`tasklist.resources.limits.cpu`		`1000m`
`tasklist.resources.limits.memory`		`2Gi`
`tasklist.ingress.enabled`	if true, an ingress resource is deployed with the tasklist deployment. Only useful if an ingress controller is available, like nginx.	`false`
`tasklist.ingress.className`	defines the class or configuration of ingress which should be used by the controller	`nginx`
`tasklist.ingress.annotations`	defines the ingress related annotations, consumed mostly by the ingress controller	`{}`
`tasklist.ingress.path`	defines the path which is associated with the operate service and port https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules	`/`
`tasklist.ingress.pathType`	can be used to define the Ingress path type. https://kubernetes.io/docs/concepts/services-networking/ingress/#path-types	`Prefix`
`tasklist.ingress.host`	can be used to define the host of the ingress rule. https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules	`""`
`tasklist.ingress.tls.enabled`	if true, then tls is configured on the ingress resource. If enabled the Ingress.host need to be defined.	`false`
`tasklist.ingress.tls.secretName`	defines the secret name which contains the TLS private key and certificate	`camunda-platform-tasklist`
`tasklist.retention.enabled`	if true, the ILM Policy is created and applied to the index templates.	`false`
`tasklist.retention.minimumAge`	defines how old the data must be, before the data is deleted as a duration.	`30d`
`tasklist.configuration`	if specified, contents will be used as the application.yaml	`""`
`tasklist.extraConfiguration`	if specified, contents will be used for any extra configuration files such as log4j2.xml	`{}`

Optimize Parameters

Name	Description	Value
`optimize.enabled`	if true, the Optimize deployment and its related resources are deployed via a helm release	`true`
`optimize.image`	configuration to configure the Optimize image specifics	
`optimize.image.registry`	can be used to set container image registry	`""`
`optimize.image.repository`	defines which image repository to use	`camunda/optimize`

Name	Description	Value
`optimize.image.tag`	can be set to overwrite the global tag, which should be used in that chart	`8.5.0`
`optimize.image.pullSecrets`	can be used to configure image pull secrets https://kubernetes.io/docs/concepts/containers/images/#specifying-imagepullsecrets-on-a-pod	`[]`
`optimize.migration`	configuration for Optimize migration	
`optimize.migration.enabled`	if true, run Optimize migration script as an init container	`true`
`optimize.migration.env`	can be used to set environment variables for Optimize migration init container	`[]`
`optimize.migration.resources`	configuration to set request and limit configuration for the migration container https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#requests-and-limits	
`optimize.migration.resources.requests.cpu`		`600m`
`optimize.migration.resources.requests.memory`		`1Gi`
`optimize.migration.resources.limits.cpu`		`2000m`
`optimize.migration.resources.limits.memory`		`2Gi`
`optimize.sidecars`	can be used to attach extra containers to the optimize deployment	`[]`
`optimize.contextPath`	can be used to make Optimize web application works on a custom sub-path. This is mainly used to run Camunda web applications under a single domain.	`""`
`optimize.configMap`	configuration which will be applied to the mounted config map.	
`optimize.configMap.defaultMode`	can be used to set permissions on created files by default. Must be an octal value between 0000 and 0777 or a decimal value between 0 and 511.	`754`
`optimize.podAnnotations`	can be used to define extra Optimize pod annotations	`{}`
`optimize.podLabels`	can be used to define extra Optimize pod labels	`{}`
`optimize.partitionCount`	defines how many Zeebe partitions are set up in the cluster and which should be imported by Optimize	`3`
`optimize.env`	can be used to set extra environment variables in each Optimize container	`[]`
`optimize.command`	can be used to override the default command provided by the container image. See https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/	`[]`
`optimize.extraVolumes`	can be used to define extra volumes for the Optimize pods, useful for tls and self-signed certificates	`[]`
`optimize.extraVolumeMounts`	can be used to mount extra volumes for the Optimize pods, useful for tls and self-signed certificates	`[]`
`optimize.initContainers`	can be used to set up extra init containers for the application Pod	`[]`
`optimize.serviceAccount`	configuration for the service account where the Optimize pods are assigned to	
`optimize.serviceAccount.enabled`	if true, enables the Optimize service account	`true`
`optimize.serviceAccount.name`	can be used to set the name of the Optimize service account	`""`
`optimize.serviceAccount.annotations`	can be used to set the annotations of the Optimize service account	`{}`
`optimize.serviceAccount.automountServiceAccountToken`	can be used to control whether the service account token should be automatically mounted	`false`
`optimize.service`	configuration to configure the Optimize service.	
`optimize.service.annotations`	can be used to define annotations, which will be applied to the Optimize service	`{}`
`optimize.service.type`	defines the type of the service https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types	`ClusterIP`
`optimize.service.port`	defines the port of the service, where the Optimize web application will be available	`80`
`optimize.service.managementPort`	defines the port where actuator will be available. Also required to reach backup API	`8092`
`optimize.podSecurityContext`	defines the security options the Optimize pod should be run with	
`optimize.podSecurityContext.runAsNonRoot`		`true`
`optimize.podSecurityContext.fsGroup`		`1001`
`optimize.containerSecurityContext`	defines the security options the Optimize container should be run with	
`optimize.containerSecurityContext.allowPrivilegeEscalation`		`false`
`optimize.containerSecurityContext.privileged`		`false`
`optimize.containerSecurityContext.readOnlyRootFilesystem`		`true`
`optimize.containerSecurityContext.runAsNonRoot`		`true`
`optimize.containerSecurityContext.runAsUser`		`1001`
`optimize.startupProbe`	configuration	
`optimize.startupProbe.enabled`	if true, the startup probe is enabled in app container	`false`
`optimize.startupProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`optimize.startupProbe.probePath`	defines the startup probe route used on the app	`/api/readyz`
`optimize.startupProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`optimize.startupProbe.periodSeconds`	defines how often the probe is executed	`30`
`optimize.startupProbe.successThreshold`	defines how often it needs to be true to be marked as ready, after failure	`1`
`optimize.startupProbe.failureThreshold`	defines when the probe is considered as failed so the Pod will be marked Unready	`5`
`optimize.startupProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`optimize.readinessProbe`	configuration	

Name	Description	Value
`optimize.readinessProbe.enabled`	if true, the readiness probe is enabled in app container	`true`
`optimize.readinessProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`optimize.readinessProbe.probePath`	defines the readiness probe route used on the app	`/api/readyz`
`optimize.readinessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`optimize.readinessProbe.periodSeconds`	defines how often the probe is executed	`30`
`optimize.readinessProbe.successThreshold`	defines how often it needs to be true to be marked as ready, after failure	`1`
`optimize.readinessProbe.failureThreshold`	defines when the probe is considered as failed so the Pod will be marked Unready	`5`
`optimize.readinessProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`optimize.livenessProbe`	configuration	
`optimize.livenessProbe.enabled`	if true, the liveness probe is enabled in app container	`false`
`optimize.livenessProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`optimize.livenessProbe.probePath`	defines the liveness probe route used on the app	`/api/readyz`
`optimize.livenessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`optimize.livenessProbe.periodSeconds`	defines how often the probe is executed	`30`
`optimize.livenessProbe.successThreshold`	defines how often it needs to be true to be considered successful after having failed	`1`
`optimize.livenessProbe.failureThreshold`	defines when the probe is considered as failed so the container will be restarted	`5`
`optimize.livenessProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`optimize.metrics.prometheus`	Prometheus metrics endpoint	`/actuator/prometheus`
`optimize.nodeSelector`	can be used to define on which nodes the Optimize pods should run	`{}`
`optimize.tolerations`	can be used to define pod toleration's https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/	`[]`
`optimize.affinity`	can be used to define pod affinity or anti-affinity https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity	`{}`
`optimize.resources`	configuration to set request and limit configuration for the container https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#requests-and-limits	
`optimize.resources.requests.cpu`		`600m`
`optimize.resources.requests.memory`		`1Gi`
`optimize.resources.limits.cpu`		`2000m`
`optimize.resources.limits.memory`		`2Gi`
`optimize.ingress.enabled`	if true, an ingress resource is deployed with the Optimize deployment. Only useful if an ingress controller is available, like nginx.	`false`
`optimize.ingress.className`	defines the class or configuration of ingress which should be used by the controller	`nginx`
`optimize.ingress.annotations`	defines the ingress related annotations, consumed mostly by the ingress controller	`{}`
`optimize.ingress.path`	defines the path which is associated with the operate service and port https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules	`/`
`optimize.ingress.pathType`	can be used to define the Ingress path type. https://kubernetes.io/docs/concepts/services-networking/ingress/#path-types	`Prefix`
`optimize.ingress.host`	can be used to define the host of the ingress rule. https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules	`""`
`optimize.ingress.tls`	configuration for tls on the ingress resource https://kubernetes.io/docs/concepts/services-networking/ingress/#tls	
`optimize.ingress.tls.enabled`	if true, then tls is configured on the ingress resource. If enabled the Ingress.host need to be defined.	`false`
`optimize.ingress.tls.secretName`	defines the secret name which contains the TLS private key and certificate	`camunda-platform-optimize`
`optimize.configuration`	if specified, contents will be used as the environment-config.yaml	`""`
`optimize.extraConfiguration`	if specified, contents will be used for any extra configuration files such as environment-logback.xml	`{}`

Identity Parameters

Name	Description	Value
`identity.enabled`	if true, the identity deployment and its related resources are deployed via a helm release	`true`
`identity.fullnameOverride`	can be used to override the full name of the Identity resources	`""`
`identity.nameOverride`	can be used to partly override the name of the Identity resources (names will still be prefixed with the release name)	`""`
`identity.firstUser`	configuration to configure properties of the first Identity user, which can be used to access all	
`identity.firstUser.enabled`	if true, Identity will seed the first user in Keycloak.	`true`
`identity.firstUser.username`	defines the username of the first user, needed to log in into the web applications	`demo`

Name	Description	Value
`identity.firstUser.password`	defines the password of the first user, needed to log in into the web applications	`demo`
`identity.firstUser.email`	defines the email address of the first user; a valid email address is required to use WebModeler	`demo@example.org`
`identity.firstUser.firstName`	defines the first name of the first user; a name is required to use WebModeler	`Demo`
`identity.firstUser.lastName`	defines the last name of the first user; a name is required to use WebModeler	`User`
`identity.firstUser.existingSecret`	can be used to use an own existing secret for Identity first user.	`""`
`identity.image`	configuration to configure the identity image specifics	
`identity.image.registry`	can be used to set container image registry.	`""`
`identity.image.repository`	defines which image repository to use	`camunda/identity`
`identity.image.tag`	can be set to overwrite the global tag, which should be used in that chart	`nil`
`identity.image.pullSecrets`	can be used to configure image pull secrets https://kubernetes.io/docs/concepts/containers/images/#specifying-imagepullsecrets-on-a-pod	`[]`
`identity.sidecars`	can be used to attach extra containers to the identity deployment	`[]`
`identity.initContainers`	can be used to set up extra init containers for the application Pod	`[]`
`identity.fullURL`	can be used when Ingress is configured (for both multi and single domain setup).	`""`
`identity.contextPath`	can be used to make Identity web application works on a custom sub-path. This is mainly used to run Camunda web applications under a single domain.	`""`
`identity.podAnnotations`	can be used to define extra Identity pod annotations	`{}`
`identity.podLabels`	can be used to define extra Identity pod labels	`{}`
`identity.service`	configuration to configure the identity service.	
`identity.service.annotations`	can be used to define annotations, which will be applied to the identity service	`{}`
`identity.service.type`	defines the type of the service https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types	`ClusterIP`
`identity.service.port`	defines the port of the service on which the identity application will be available	`80`
`identity.service.metricsPort`	defines the port of the service on which the identity metrics will be available	`82`
`identity.service.metricsName`	defines the name of the service on which the identity metrics will be available	`metrics`
`identity.podSecurityContext`	defines the security options the Identity pod should be run with	
`identity.podSecurityContext.runAsNonRoot`		`true`
`identity.podSecurityContext.fsGroup`		`1001`
`identity.containerSecurityContext`	defines the security options the Identity container should be run with	
`identity.containerSecurityContext.allowPrivilegeEscalation`		`false`
`identity.containerSecurityContext.privileged`		`false`
`identity.containerSecurityContext.readOnlyRootFilesystem`		`true`
`identity.containerSecurityContext.runAsNonRoot`		`true`
`identity.containerSecurityContext.runAsUser`		`1001`
`identity.startupProbe`	configuration	
`identity.startupProbe.enabled`	if true, the startup probe is enabled in app container	`false`
`identity.startupProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`identity.startupProbe.probePath`	defines the startup probe route used on the app	`/actuator/health`
`identity.startupProbe.initialDelaySeconds`	defines the number of seconds after the container has started before the probe is initiated.	`30`
`identity.startupProbe.periodSeconds`	defines how often the probe is executed	`30`
`identity.startupProbe.successThreshold`	defines how often it needs to be true to be marked as ready, after failure	`1`
`identity.startupProbe.failureThreshold`	defines when the probe is considered as failed so the Pod will be marked Unready	`5`
`identity.startupProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`identity.readinessProbe`	configuration	
`identity.readinessProbe.enabled`	if true, the readiness probe is enabled in app container	`true`
`identity.readinessProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`identity.readinessProbe.probePath`	defines the readiness probe route used on the app	`/actuator/health`
`identity.readinessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before the probe is initiated.	`30`
`identity.readinessProbe.periodSeconds`	defines how often the probe is executed	`30`
`identity.readinessProbe.successThreshold`	defines how often it needs to be true to be marked as ready, after failure	`1`
`identity.readinessProbe.failureThreshold`	defines when the probe is considered as failed so the Pod will be marked Unready	`5`
`identity.readinessProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`identity.livenessProbe`	configuration	
`identity.livenessProbe.enabled`	if true, the liveness probe is enabled in app container	`false`
`identity.livenessProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`identity.livenessProbe.probePath`	defines the liveness probe route used on the app	`/actuator/health`
`identity.livenessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`identity.livenessProbe.periodSeconds`	defines how often the probe is executed	`30`
`identity.livenessProbe.successThreshold`	defines how often it needs to be true to be considered successful after having failed	`1`
`identity.livenessProbe.failureThreshold`	defines when the probe is considered as failed so the container will be restarted	`5`

Name	Description	Value
`identity.livenessProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`identity.metrics.prometheus`	Prometheus metrics endpoint	`/actuator/prometheus`
`identity.nodeSelector`	can be used to define on which nodes the Identity pods should run	`{}`
`identity.tolerations`	can be used to define pod toleration's https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/	`[]`
`identity.affinity`	can be used to define pod affinity or anti-affinity https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity	`{}`
`identity.resources`	configuration to set request and limit configuration for the container https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#requests-and-limits	
`identity.resources.requests.memory`		`400Mi`
`identity.resources.limits.cpu`		`2000m`
`identity.resources.requests.cpu`		`600m`
`identity.resources.limits.memory`		`2Gi`
`identity.env`	can be used to set extra environment variables in each identity container. See the documentation https://docs.camunda.io/docs/self-managed/identity/deployment/configuration-variables/ for more details.	`[]`
`identity.command`	can be used to override the default command provided by the container image. See https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/	`[]`
`identity.extraVolumes`	can be used to define extra volumes for the identity pods, useful for tls and self-signed certificates	`[]`
`identity.extraVolumeMounts`	can be used to mount extra volumes for the identity pods, useful for tls and self-signed certificates	`[]`
`identity.serviceAccount`	configuration for the service account where the identity pods are assigned to	
`identity.serviceAccount.enabled`	if true, enables the identity service account	`true`
`identity.serviceAccount.name`	can be used to set the name of the identity service account	`""`
`identity.serviceAccount.annotations`	can be used to set the annotations of the identity service account	`{}`
`identity.serviceAccount.automountServiceAccountToken`	can be used to control whether the service account token should be automatically mounted	`true`
`identity.ingress.enabled`	if true, an ingress resource is deployed with the identity deployment. Only useful if an ingress controller is available, like nginx.	`false`
`identity.ingress.className`	defines the class or configuration of ingress which should be used by the controller	`nginx`
`identity.ingress.annotations`	defines the ingress related annotations, consumed mostly by the ingress controller	`{}`
`identity.ingress.path`	defines the path which is associated with the operate service and port https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules	`/`
`identity.ingress.pathType`	can be used to define the Ingress path type. https://kubernetes.io/docs/concepts/services-networking/ingress/#path-types	`Prefix`
`identity.ingress.host`	can be used to define the host of the ingress rule. https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules	`""`
`identity.ingress.tls`	configuration for tls on the ingress resource https://kubernetes.io/docs/concepts/services-networking/ingress/#tls	
`identity.ingress.tls.enabled`	if true, then tls is configured on the ingress resource. If enabled the Ingress.host need to be defined.	`false`
`identity.ingress.tls.secretName`	defines the secret name which contains the TLS private key and certificate	`camunda-platform-identity`
`identity.externalDatabase.enabled`		`false`
`identity.externalDatabase.host`	Database host	`nil`
`identity.externalDatabase.port`	Database port number	`nil`
`identity.externalDatabase.username`	Non-root username	`nil`
`identity.externalDatabase.password`	Password for the non-root username	`nil`
`identity.externalDatabase.database`	The database name	`nil`
`identity.externalDatabase.existingSecret`	Name of an existing secret resource containing the database credentials	`nil`
`identity.externalDatabase.existingSecretPasswordKey`	Name of an existing secret key containing the database credentials	`nil`
`identity.configuration`	if specified, contents will be used as the application.yaml	`""`
`identity.extraConfiguration`	if specified, contents will be used for any extra configuration files such as the log4j2.xml	`{}`

Identity - PostgreSQL Parameters

Name	Description	Value
`identityPostgresql`	configuration for the PostgreSQL dependency chart used by Identity. For more details, check Bitnami package for PostgreSQL documentation.	
`identityPostgresql.enabled`	Enable Identity PostgreSQL Helm chart. Required for Multi-Tenancy.	`false`
`identityPostgresql.image.repository`	PostgreSQL repo	`bitnami/postgresql`
`identityPostgresql.image.tag`	PostgreSQL image tag	`15.6.0`
`identityPostgresql.nameOverride`	the name used for Identity PostgreSQL.	`identity-postgresql`
`identityPostgresql.auth.username`	Non-root username	`identity`
`identityPostgresql.auth.database`	The database name	`identity`
`identityPostgresql.auth.password`	Password for the non-root username	`nil`
`identityPostgresql.auth.existingSecret`	Name of an existing secret resource containing the database credentials	`nil`

Identity - Keycloak Parameters

Name	Description	Value
`identityKeycloak`	configuration, for the Keycloak dependency chart which is used by Identity. For more details, check Bitnami package for Keycloak documentation.	
`identityKeycloak.enabled`	Enable Identity Keycloak Helm chart. It is used incorporate with "global.identity.keycloak" to use your own Keycloak instead of the one comes with Camunda Helm chart	`true`
`identityKeycloak.nameOverride`	the name used for Keycloak.	`keycloak`
`identityKeycloak.image`	configuration.	
`identityKeycloak.image.repository`	image repo	`bitnami/keycloak`
`identityKeycloak.image.tag`	image tag	`23.0.7`
`identityKeycloak.postgresql`	configuration.	
`identityKeycloak.postgresql.image.repository`	image repo	`bitnami/postgresql`
`identityKeycloak.postgresql.image.tag`	image tag	`15.6.0`
`identityKeycloak.postgresql.primary.containerSecurityContext.enabled`		`true`
`identityKeycloak.postgresql.primary.containerSecurityContext.privileged`		`false`
`identityKeycloak.postgresql.primary.containerSecurityContext.readOnlyRootFilesystem`		`true`
`identityKeycloak.postgresql.primary.containerSecurityContext.allowPrivilegeEscalation`		`false`
`identityKeycloak.postgresql.primary.containerSecurityContext.runAsNonRoot`		`true`
`identityKeycloak.postgresql.primary.containerSecurityContext.runAsUser`		`1001`
`identityKeycloak.postgresql.primary.containerSecurityContext.capabilities.drop`		`["ALL"]`
`identityKeycloak.postgresql.primary.containerSecurityContext.seccompProfile.type`		`RuntimeDefault`
`identityKeycloak.postgresql.primary.podSecurityContext.enabled`		`true`
`identityKeycloak.postgresql.primary.podSecurityContext.runAsNonRoot`		`true`
`identityKeycloak.postgresql.primary.podSecurityContext.fsGroup`		`1001`
`identityKeycloak.proxy`	keycloak proxy	`edge`
`identityKeycloak.tls`	can be used to enable TLS encryption. Required for HTTPs traffic.	
`identityKeycloak.tls.enabled`	enabling tls	`false`
`identityKeycloak.initContainers[0].name`		`copy-camunda-theme`
`identityKeycloak.initContainers[0].image`		`{{ .Values.global.identity.image | default "camunda/identity:latest" }}`
`identityKeycloak.initContainers[0].imagePullPolicy`		`{{ .Values.global.identity.imagePullPolicy | default "Always" }}`
`identityKeycloak.initContainers[0].command`		`["sh","-c","cp -a /app/keycloak-theme/* /mnt"]`
`identityKeycloak.initContainers[0].securityContext.privileged`		`false`
`identityKeycloak.initContainers[0].securityContext.readOnlyRootFilesystem`		`true`
`identityKeycloak.initContainers[0].securityContext.allowPrivilegeEscalation`		`false`
`identityKeycloak.initContainers[0].securityContext.runAsNonRoot`		`true`
`identityKeycloak.initContainers[0].securityContext.runAsUser`		`1001`
`identityKeycloak.initContainers[0].securityContext.capabilities.drop`		`["ALL"]`
`identityKeycloak.initContainers[0].securityContext.seccompProfile.type`		`RuntimeDefault`
`identityKeycloak.initContainers[0].volumeMounts[0].name`		`camunda-theme`
`identityKeycloak.initContainers[0].volumeMounts[0].mountPath`		`/mnt`
`identityKeycloak.extraVolumeMounts[0].name`		`camunda-theme`
`identityKeycloak.extraVolumeMounts[0].mountPath`		`/opt/bitnami/keycloak/themes/identity`
`identityKeycloak.extraVolumeMounts[1].mountPath`		`/opt/bitnami/keycloak/data/tmp`
`identityKeycloak.extraVolumeMounts[1].name`		`data-tmp`
`identityKeycloak.containerSecurityContext.privileged`		`false`
`identityKeycloak.containerSecurityContext.readOnlyRootFilesystem`		`true`
`identityKeycloak.containerSecurityContext.allowPrivilegeEscalation`		`false`
`identityKeycloak.containerSecurityContext.runAsNonRoot`		`true`

Name	Description	Value
`identityKeycloak.containerSecurityContext.runAsUser`		`1001`
`identityKeycloak.containerSecurityContext.capabilities.drop`		`["ALL"]`
`identityKeycloak.containerSecurityContext.seccompProfile.type`		`RuntimeDefault`
`identityKeycloak.podSecurityContext.runAsNonRoot`		`true`
`identityKeycloak.podSecurityContext.fsGroup`		`1001`
`identityKeycloak.httpRelativePath`	defines the context for Keycloak. This config is valid for Keycloak v19.x.x only	`/auth/`
`identityKeycloak.extraEnvVars`		
`identityKeycloak.extraEnvVars[0].name`		`KEYCLOAK_PROXY_ADDRESS_FORWARDING`
`identityKeycloak.extraEnvVars[0].value`		`{{ .Values.global.ingress.tls.enabled }}`
`identityKeycloak.ingress.enabled`	can be used enable ingress record generation for Keycloak.	`false`
`identityKeycloak.ingress.tls`	can be used to enable TLS configuration for the host defined at ingress.hostname parameter.	`false`
`identityKeycloak.ingress.extraTls`	configuration for additional hostnames to be covered with this ingress record.	`[]`
`identityKeycloak.ingress.annotations`	configures annotations to be applied to the ingress record.	`{}`
`identityKeycloak.service`	configuration, to configure the service which is deployed along with keycloak	
`identityKeycloak.service.type`	can be set to change the service type.	`ClusterIP`
`identityKeycloak.auth`	uses the secrets generated by keycloak, to access keycloak.	
`identityKeycloak.auth.adminUser`	defines the keycloak administrator user	`admin`
`identityKeycloak.auth.existingSecret`	can be used to reuse an existing secret containing authentication information.	`""`

WebModeler Parameters

Name	Description	Value
`webModeler.enabled`	if true, the WebModeler deployment and its related resources are deployed via a helm release	`false`
`webModeler.fullnameOverride`	can be used to override the full name of the WebModeler resources	`""`
`webModeler.nameOverride`	can be used to partly override the name of the WebModeler resources (names will still be prefixed with the release name)	`""`
`webModeler.image`	configuration of the WebModeler Docker images	
`webModeler.image.registry`	can be used to set the Docker registry for the WebModeler images (overwrites global.image.registry)	`registry.camunda.cloud`
`webModeler.image.tag`	can be used to set the Docker image tag for the WebModeler images (overwrites global.image.tag)	`8.5.0`
`webModeler.image.pullSecrets`	can be used to configure image pull secrets, see https://kubernetes.io/docs/concepts/containers/images/#specifying-imagepullsecrets-on-a-pod	`[]`
`webModeler.contextPath`	can be used to make WebModeler available on a custom sub-path. This is mainly used to run the Camunda web applications under a single domain.	`""`

WebModeler - RestAPI Parameters

Name	Description	Value
`webModeler.restapi`	configuration of the WebModeler restapi component	
`webModeler.restapi.image`	configuration of the restapi Docker image	
`webModeler.restapi.image.repository`	defines which image repository to use for the restapi Docker image	`web-modeler-ee/modeler-restapi`
`webModeler.restapi.sidecars`	can be used to attach extra containers to the restapi deployment	`[]`
`webModeler.restapi.initContainers`	can be used to set up extra init containers for the application Pod	`[]`
`webModeler.restapi.externalDatabase`	can be used to configure a connection to an external database. This will only be applied	
`webModeler.restapi.externalDatabase.url`	defines the JDBC url of the database instance	`""`
`webModeler.restapi.externalDatabase.user`		`""`
`webModeler.restapi.externalDatabase.password`	defines the database user's password	`""`
`webModeler.restapi.mail`	configuration for emails sent by WebModeler	
`webModeler.restapi.mail.smtpHost`	defines the host name of the SMTP server to be used by WebModeler	`""`
`webModeler.restapi.mail.smtpPort`	defines the port number of the SMTP server	`587`
`webModeler.restapi.mail.smtpUser`	can be used to provide a user for the SMTP server	`""`
`webModeler.restapi.mail.smtpPassword`	can be used to provide a password for the SMTP server	`""`
`webModeler.restapi.mail.smtpTlsEnabled`	if true, enforces TLS encryption for SMTP connections (using STARTTLS)	`true`
`webModeler.restapi.mail.fromAddress`	defines the email address that will be displayed as the sender of emails sent by WebModeler	`""`
`webModeler.restapi.mail.fromName`	defines the name that will be displayed as the sender of emails sent by WebModeler	`Camunda 8`

Name	Description	Value
`webModeler.restapi.podAnnotations`	can be used to define extra restapi pod annotations	`{}`
`webModeler.restapi.podLabels`	can be used to define extra restapi pod labels	`{}`
`webModeler.restapi.env`	can be used to set extra environment variables in each restapi container	`[]`
`webModeler.restapi.command`	can be used to override the default command provided by the container image, see https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/	`[]`
`webModeler.restapi.extraVolumes`	can be used to define extra volumes for the restapi pods, useful for TLS and self-signed certificates	`[]`
`webModeler.restapi.extraVolumeMounts`	can be used to mount extra volumes for the restapi pods, useful for TLS and self-signed certificates	`[]`
`webModeler.restapi.podSecurityContext`	can be used to define the security options the restapi pod should be run with	
`webModeler.restapi.podSecurityContext.runAsNonRoot`		`true`
`webModeler.restapi.podSecurityContext.fsGroup`		`1001`
`webModeler.restapi.containerSecurityContext`	can be used to define the security options the restapi container should be run with	
`webModeler.restapi.containerSecurityContext.privileged`		`false`
`webModeler.restapi.containerSecurityContext.readOnlyRootFilesystem`		`true`
`webModeler.restapi.containerSecurityContext.allowPrivilegeEscalation`		`false`
`webModeler.restapi.containerSecurityContext.runAsNonRoot`		`true`
`webModeler.restapi.containerSecurityContext.runAsUser`		`1001`
`webModeler.restapi.startupProbe`	configuration of the restapi startup probe	
`webModeler.restapi.startupProbe.enabled`	if true, the startup probe will be enabled for the restapi container	`false`
`webModeler.restapi.startupProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`webModeler.restapi.startupProbe.probePath`	defines the HTTP endpoint used for the startup probe	`/health/liveness`
`webModeler.restapi.startupProbe.initialDelaySeconds`	defines the number of seconds after the container has started before the probe is initiated	`30`
`webModeler.restapi.startupProbe.periodSeconds`	defines how often the probe is executed	`30`
`webModeler.restapi.startupProbe.successThreshold`	defines how often the probe needs to succeed to be considered successful after having failed	`1`
`webModeler.restapi.startupProbe.failureThreshold`	defines when the probe is considered failed so the container will be restarted	`5`
`webModeler.restapi.startupProbe.timeoutSeconds`	defines the number of seconds after which the probe times out	`1`
`webModeler.restapi.readinessProbe`	configuration of the restapi readiness probe	
`webModeler.restapi.readinessProbe.enabled`	if true, the readiness probe will be enabled for the restapi container	`true`
`webModeler.restapi.readinessProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`webModeler.restapi.readinessProbe.probePath`	defines the HTTP endpoint used for the readiness probe	`/health/readiness`
`webModeler.restapi.readinessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before the probe is initiated	`30`
`webModeler.restapi.readinessProbe.periodSeconds`	defines how often the probe is executed	`30`
`webModeler.restapi.readinessProbe.successThreshold`	defines how often the probe needs to succeed to be considered successful after having failed	`1`
`webModeler.restapi.readinessProbe.failureThreshold`	defines when the probe is considered failed so the Pod will be marked unready	`5`
`webModeler.restapi.readinessProbe.timeoutSeconds`	defines the number of seconds after which the probe times out	`1`
`webModeler.restapi.livenessProbe`	configuration of the restapi liveness probe	
`webModeler.restapi.livenessProbe.enabled`	if true, the liveness probe will be enabled for the restapi container	`false`
`webModeler.restapi.livenessProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`webModeler.restapi.livenessProbe.probePath`	defines the HTTP endpoint used for the liveness probe	`/health/liveness`
`webModeler.restapi.livenessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before the probe is initiated	`30`
`webModeler.restapi.livenessProbe.periodSeconds`	defines how often the probe is executed	`30`
`webModeler.restapi.livenessProbe.successThreshold`	defines how often the probe needs to succeed to be considered successful after having failed	`1`
`webModeler.restapi.livenessProbe.failureThreshold`	defines when the probe is considered failed so the container will be restarted	`5`
`webModeler.restapi.livenessProbe.timeoutSeconds`	defines the number of seconds after which the probe times out	`1`
`webModeler.restapi.metrics.prometheus`	Prometheus metrics endpoint	`/metrics`
`webModeler.restapi.nodeSelector`	can be used to select the nodes the restapi pods should run on	`{}`
`webModeler.restapi.tolerations`	can be used to define pod tolerations, see https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/	`[]`
`webModeler.restapi.affinity`	can be used to define pod affinity or anti-affinity, see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity	`{}`
`webModeler.restapi.resources`	configuration of resource requests and limits for the container, see https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#requests-and-limits	
`webModeler.restapi.resources.requests.cpu`		`500m`
`webModeler.restapi.resources.requests.memory`		`1Gi`

Name	Description	Value
`webModeler.restapi.resources.limits.cpu`		`1000m`
`webModeler.restapi.resources.limits.memory`		`2Gi`
`webModeler.restapi.service`	configuration of the WebModeler restapi service	
`webModeler.restapi.service.annotations`	can be used to define annotations which will be applied to the service	`{}`
`webModeler.restapi.service.type`	defines the type of the service, see https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types	`ClusterIP`
`webModeler.restapi.service.port`	defines the default port of the service	`80`
`webModeler.restapi.service.managementPort`	defines the management port of the service	`8091`
`webModeler.restapi.configuration`	if specified, contents will be used as the application.yaml	`""`
`webModeler.restapi.extraConfiguration`	if specified, contents will be used for any extra configuration files such as log4j2.xml	`{}`

WebModeler - WebApp Parameters

Name	Description	Value
`webModeler.webapp.`	configuration of the WebModeler webapp component	
`webModeler.webapp.image`	configuration of the webapp Docker image	
`webModeler.webapp.image.repository`	defines which image repository to use for the webapp Docker image	`web-modeler-ee/modeler-webapp`
`webModeler.webapp.sidecars`	can be used to attach extra containers to the modeler webapp deployment	`[]`
`webModeler.webapp.initContainers`	can be used to set up extra init containers for the application Pod	`[]`
`webModeler.webapp.podAnnotations`	can be used to define extra webapp pod annotations	`{}`
`webModeler.webapp.podLabels`	can be used to define extra webapp pod labels	`{}`
`webModeler.webapp.env`	can be used to set extra environment variables in each webapp container	`[]`
`webModeler.webapp.command`	can be used to override the default command provided by the container image, see https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/	`[]`
`webModeler.webapp.extraVolumes`	can be used to define extra volumes for the webapp pods, useful for TLS and self-signed certificates	`[]`
`webModeler.webapp.extraVolumeMounts`	can be used to mount extra volumes for the webapp pods, useful for TLS and self-signed certificates	`[]`
`webModeler.webapp.podSecurityContext`	can be used to define the security options the webapp pod should be run with	
`webModeler.webapp.podSecurityContext.runAsNonRoot`		`true`
`webModeler.webapp.podSecurityContext.fsGroup`		`1001`
`webModeler.webapp.containerSecurityContext`	can be used to define the security options the webapp container should be run with	
`webModeler.webapp.containerSecurityContext.privileged`		`false`
`webModeler.webapp.containerSecurityContext.readOnlyRootFilesystem`		`true`
`webModeler.webapp.containerSecurityContext.allowPrivilegeEscalation`		`false`
`webModeler.webapp.containerSecurityContext.runAsNonRoot`		`true`
`webModeler.webapp.containerSecurityContext.runAsUser`		`1001`
`webModeler.webapp.startupProbe`	configuration of the webapp startup probe	
`webModeler.webapp.startupProbe.enabled`	if true, the startup probe will be enabled for the webapp container	`false`
`webModeler.webapp.startupProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`webModeler.webapp.startupProbe.probePath`	defines the HTTP endpoint used for the startup probe	`/health/liveness`
`webModeler.webapp.startupProbe.initialDelaySeconds`	defines the number of seconds after the container has started before the probe is initiated	`15`
`webModeler.webapp.startupProbe.periodSeconds`	defines how often the probe is executed	`30`
`webModeler.webapp.startupProbe.successThreshold`	defines how often the probe needs to succeed to be considered successful after having failed	`1`
`webModeler.webapp.startupProbe.failureThreshold`	defines when the probe is considered failed so the container will be restarted	`5`
`webModeler.webapp.startupProbe.timeoutSeconds`	defines the number of seconds after which the probe times out	`1`
`webModeler.webapp.readinessProbe`	configuration of the webapp readiness probe	
`webModeler.webapp.readinessProbe.enabled`	if true, the readiness probe will be enabled for the webapp container	`true`
`webModeler.webapp.readinessProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`webModeler.webapp.readinessProbe.probePath`	defines the HTTP endpoint used for the readiness probe	`/health/readiness`
`webModeler.webapp.readinessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before the probe is initiated	`15`
`webModeler.webapp.readinessProbe.periodSeconds`	defines how often the probe is executed	`30`
`webModeler.webapp.readinessProbe.successThreshold`	defines how often the probe needs to succeed to be considered successful after having failed	`1`
`webModeler.webapp.readinessProbe.failureThreshold`	defines when the probe is considered failed so the Pod will be marked unready	`5`
`webModeler.webapp.readinessProbe.timeoutSeconds`	defines the number of seconds after which the probe times out	`1`
`webModeler.webapp.livenessProbe`	configuration of the webapp liveness probe	
`webModeler.webapp.livenessProbe.enabled`	if true, the liveness probe will be enabled for the webapp container	`false`

Name	Description	Value
`webModeler.webapp.livenessProbe.scheme`	defines the startup probe schema used on calling the probePath	`HTTP`
`webModeler.webapp.livenessProbe.probePath`	defines the HTTP endpoint used for the liveness probe	`/health/liveness`
`webModeler.webapp.livenessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before the probe is initiated	`15`
`webModeler.webapp.livenessProbe.periodSeconds`	defines how often the probe is executed	`30`
`webModeler.webapp.livenessProbe.successThreshold`	defines how often the probe needs to succeed to be considered successful after having failed	`1`
`webModeler.webapp.livenessProbe.failureThreshold`	defines when the probe is considered failed so the container will be restarted	`5`
`webModeler.webapp.livenessProbe.timeoutSeconds`	defines the number of seconds after which the probe times out	`1`
`webModeler.webapp.metrics.prometheus`	Prometheus metrics endpoint	`/metrics`
`webModeler.webapp.nodeSelector`	can be used to select the nodes the webapp pods should run on	`{}`
`webModeler.webapp.tolerations`	can be used to define pod tolerations, see https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/	`[]`
`webModeler.webapp.affinity`	can be used to define pod affinity or anti-affinity, see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity	`{}`
`webModeler.webapp.resources`	configuration of resource requests and limits for the container, see https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#requests-and-limits	
`webModeler.webapp.resources.requests.cpu`		`400m`
`webModeler.webapp.resources.requests.memory`		`256Mi`
`webModeler.webapp.resources.limits.cpu`		`800m`
`webModeler.webapp.resources.limits.memory`		`512Mi`
`webModeler.webapp.service`	configuration of the WebModeler webapp service	
`webModeler.webapp.service.annotations`	can be used to define annotations which will be applied to the service	`{}`
`webModeler.webapp.service.type`	defines the type of the service, see https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types	`ClusterIP`
`webModeler.webapp.service.port`	defines the port of the service	`80`
`webModeler.webapp.service.managementPort`	defines the management port of the service	`8071`
`webModeler.webapp.configuration`	if specified, contents will be used as the application.yaml	`""`
`webModeler.webapp.extraConfiguration`	if specified, contents will be used for any extra configuration files such as log4j2.xml	`{}`

WebModeler - WebSockets Parameters

Name	Description	Value
`webModeler.websockets`	configuration of the WebModeler websockets component	
`webModeler.websockets.image`	configuration of the websockets Docker image	
`webModeler.websockets.image.repository`	defines which image repository to use for the websockets Docker image	`web-modeler-ee/modeler-websockets`
`webModeler.websockets.sidecars`	can be used to attach extra containers to the modeler websockets deployment	`[]`
`webModeler.websockets.initContainers`	can be used to set up extra init containers for the application Pod	`[]`
`webModeler.websockets.publicHost`	can be used to define the host on which the WebSockets server can be reached from the WebModeler client in the browser.	`localhost`
`webModeler.websockets.publicPort`	can be used to define the port number on which the WebSockets server can be reached from the WebModeler client in the browser.	`8085`
`webModeler.websockets.podAnnotations`	can be used to define extra websockets pod annotations	`{}`
`webModeler.websockets.podLabels`	can be used to define extra websockets pod labels	`{}`
`webModeler.websockets.env`	can be used to set extra environment variables in each websockets container	`[]`
`webModeler.websockets.command`	can be used to override the default command provided by the container image, see https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/	`[]`
`webModeler.websockets.extraVolumes`	can be used to define extra volumes for the websockets pod; useful for logging to a file	`[]`
`webModeler.websockets.extraVolumeMounts`	can be used to mount extra volumes for the websockets pod; useful for logging to a file	`[]`
`webModeler.websockets.podSecurityContext`	can be used to define the security options the websockets pod should be run with	
`webModeler.websockets.podSecurityContext.runAsNonRoot`		`true`
`webModeler.websockets.podSecurityContext.fsGroup`		`1001`
`webModeler.websockets.containerSecurityContext`	can be used to define the security options the websockets container should be run with	
`webModeler.websockets.containerSecurityContext.privileged`		`false`
`webModeler.websockets.containerSecurityContext.readOnlyRootFilesystem`		`true`
`webModeler.websockets.containerSecurityContext.allowPrivilegeEscalation`		`false`

Name	Description	Value
webModeler.websockets.containerSecurityContext.runAsNonRoot		true
webModeler.websockets.containerSecurityContext.runAsUser		1001
webModeler.websockets.startupProbe	configuration of the websockets startup probe	
webModeler.websockets.startupProbe.enabled	if true, the startup probe will be enabled for the websockets container	false
webModeler.websockets.startupProbe.initialDelaySeconds	defines the number of seconds after the container has started before the probe is initiated	10
webModeler.websockets.startupProbe.periodSeconds	defines how often the probe is executed	30
webModeler.websockets.startupProbe.successThreshold	defines how often the probe needs to succeed to be considered successful after having failed	1
webModeler.websockets.startupProbe.failureThreshold	defines when the probe is considered failed so the container will be restarted	5
webModeler.websockets.startupProbe.timeoutSeconds	defines the number of seconds after which the probe times out	1
webModeler.websockets.readinessProbe	configuration of the websockets readiness probe	
webModeler.websockets.readinessProbe.enabled	if true, the readiness probe will be enabled for the websockets container	true
webModeler.websockets.readinessProbe.initialDelaySeconds	defines the number of seconds after the container has started before the probe is initiated	10
webModeler.websockets.readinessProbe.periodSeconds	defines how often the probe is executed	30
webModeler.websockets.readinessProbe.successThreshold	defines how often the probe needs to succeed to be considered successful after having failed	1
webModeler.websockets.readinessProbe.failureThreshold	defines when the probe is considered failed so the Pod will be marked unready	5
webModeler.websockets.readinessProbe.timeoutSeconds	defines the number of seconds after which the probe times out	1
webModeler.websockets.livenessProbe	configuration of the websockets liveness probe	
webModeler.websockets.livenessProbe.enabled	if true, the liveness probe will be enabled for the websockets container	false
webModeler.websockets.livenessProbe.initialDelaySeconds	defines the number of seconds after the container has started before the probe is initiated	10
webModeler.websockets.livenessProbe.periodSeconds	defines how often the probe is executed	30
webModeler.websockets.livenessProbe.successThreshold	defines how often the probe needs to succeed to be considered successful after having failed	1
webModeler.websockets.livenessProbe.failureThreshold	defines when the probe is considered failed so the container will be restarted	5
webModeler.websockets.livenessProbe.timeoutSeconds	defines the number of seconds after which the probe times out	1
webModeler.websockets.nodeSelector	can be used to select the nodes the websockets pods should run on	{}
webModeler.websockets.tolerations	can be used to define pod tolerations, see https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/	[]
webModeler.websockets.affinity	can be used to define pod affinity or anti-affinity, see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity	{}
webModeler.websockets.resources	configuration of resource requests and limits for the container, see https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#requests-and-limits	
webModeler.websockets.resources.requests.cpu		100m
webModeler.websockets.resources.requests.memory		64Mi
webModeler.websockets.resources.limits.cpu		200m
webModeler.websockets.resources.limits.memory		128Mi
webModeler.websockets.service	configuration of the WebModeler websockets service	
webModeler.websockets.service.annotations	can be used to define annotations which will be applied to the service	{}
webModeler.websockets.service.type	defines the type of the service, see https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types	ClusterIP
webModeler.websockets.service.port	defines the port of the service	80
webModeler.websockets.configuration	if specified, contents will be used as the application.yaml	""
webModeler.websockets.extraConfiguration	if specified, contents will be used for any extra configuration files such as log4j2.xml	{}
webModeler.serviceAccount	configuration for the service account the WebModeler pods are assigned to	
webModeler.serviceAccount.enabled	if true, enables the WebModeler service account	true
webModeler.serviceAccount.name	can be used to set the name of the WebModeler service account	""
webModeler.serviceAccount.annotations	can be used to set the annotations of the WebModeler service account	{}
webModeler.serviceAccount.automountServiceAccountToken	can be used to control whether the service account token should be automatically mounted	false
webModeler.ingress.enabled	if true, an Ingress resource will be deployed with the WebModeler deployment. Only useful if an Ingress controller like NGINX is available.	false
webModeler.ingress.className	defines the class or configuration of ingress which should be used by the controller	nginx
webModeler.ingress.annotations	defines the ingress related annotations, consumed mostly by the ingress controller	{}
webModeler.ingress.webapp	configuration of the webapp ingress	
webModeler.ingress.webapp.host	defines the host of the ingress rule, see https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules; this is the host name on which the WebModeler web application will be available	""

Name	Description	Value
webModeler.ingress.webapp.pathType	can be used to define the Ingress path type. https://kubernetes.io/docs/concepts/services-networking/ingress/#path-types	Prefix
webModeler.ingress.webapp.tls	configuration for TLS on the ingress resource, see https://kubernetes.io/docs/concepts/services-networking/ingress/#tls	
webModeler.ingress.webapp.tls.enabled	if true, TLS will be configured on the ingress resource	false
webModeler.ingress.webapp.tls.secretName	defines the secret name which contains the TLS private key and certificate	camunda-platform-webmodeler-webapp
webModeler.ingress.websockets	configuration of the websockets ingress	
webModeler.ingress.websockets.host	defines the host of the ingress rule, see https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules; this is the host name the WebModeler client in the browser will use to connect to the WebSockets server	""
webModeler.ingress.websockets.pathType	can be used to define the Ingress path type. https://kubernetes.io/docs/concepts/services-networking/ingress/#path-types	Prefix
webModeler.ingress.websockets.tls	configuration for TLS on the ingress resource, see https://kubernetes.io/docs/concepts/services-networking/ingress/#tls	
webModeler.ingress.websockets.tls.enabled	if true, TLS will be configured on the ingress resource	false
webModeler.ingress.websockets.tls.secretName	defines the secret name which contains the TLS private key and certificate	camunda-platform-webmodeler-websockets

WebModeler - PostgreSQL Parameters

Name	Description	Value
postgresql	configuration for the postgresql dependency chart used by WebModeler. See the chart documentation https://github.com/bitnami/charts/tree/master/bitnami/postgresql#parameters for more details.	
postgresql.enabled	if true, a PostgreSQL database will be deployed as part of the Helm release by using the dependency chart	false
postgresql.nameOverride	defines the name of the Postgres resources (names will be prefixed with the release name), see https://github.com/bitnami/charts/tree/main/bitnami/postgresql#common-parameters	postgresql-web-modeler
postgresql.auth	configuration of the database authentication	
postgresql.auth.username	defines the name of the database user to be created for WebModeler	web-modeler
postgresql.auth.password	defines the database user's password; a random password will be generated if left empty	""
postgresql.auth.database	defines the name of the database to be created for WebModeler	web-modeler
postgresql.primary.extraVolumes[0].name		tmp
postgresql.primary.extraVolumes[0].emptyDir		{}
postgresql.primary.extraVolumes[1].name		config
postgresql.primary.extraVolumes[1].emptyDir		{}
postgresql.primary.extraVolumes[2].name		postgresql-tmp
postgresql.primary.extraVolumes[2].emptyDir		{}
postgresql.primary.extraVolumeMounts[0].mountPath		/tmp
postgresql.primary.extraVolumeMounts[0].name		tmp
postgresql.primary.extraVolumeMounts[1].mountPath		/opt/bitnami/postgresql/conf
postgresql.primary.extraVolumeMounts[1].name		config
postgresql.primary.extraVolumeMounts[2].mountPath		/opt/bitnami/postgresql/tmp
postgresql.primary.extraVolumeMounts[2].name		postgresql-tmp
postgresql.primary.containerSecurityContext.enabled		true
postgresql.primary.containerSecurityContext.privileged		false
postgresql.primary.containerSecurityContext.readOnlyRootFilesystem		true
postgresql.primary.containerSecurityContext.allowPrivilegeEscalation		false
postgresql.primary.containerSecurityContext.runAsNonRoot		true
postgresql.primary.containerSecurityContext.runAsUser		1001
postgresql.primary.containerSecurityContext.capabilities.drop		["ALL"]
postgresql.primary.containerSecurityContext.seccompProfile.type		RuntimeDefault
postgresql.primary.podSecurityContext.enabled		true
postgresql.primary.podSecurityContext.runAsNonRoot		true
postgresql.primary.podSecurityContext.fsGroup		1001

Connectors Parameters

Name	Description	Value
`connectors`	configuration for the Connectors.	
`connectors.enabled`	if true, the Connectors deployment and its related resources are deployed via a helm release	`true`
`connectors.inbound`	Switch for inbound mode (e.g., for webhook or polling)	
`connectors.inbound.mode`	acceptable values: disabled, credentials, or oauth	`oauth`
`connectors.inbound.auth`	configuration of the credentials authentication.	
`connectors.inbound.auth.existingSecret`	can be used to configure Secret name that contains Operate password (if inbound mode is credentials)	`""`
`connectors.image`	configuration to configure the Connectors image specifics	
`connectors.image.registry`	can be used to set container image registry.	`""`
`connectors.image.repository`	defines which image repository to use	`camunda/connectors-bundle`
`connectors.image.tag`	can be set to overwrite the global tag, which should be used in that chart	`8.5.0`
`connectors.image.pullSecrets`	can be used to configure image pull secrets https://kubernetes.io/docs/concepts/containers/images/#specifying-imagepullsecrets-on-a-pod	`[]`
`connectors.sidecars`	can be used to attach extra containers to the connectors deployment	`[]`
`connectors.initContainers`	can be used to set up extra init containers for the application Pod	`[]`
`connectors.replicas`	number of Connectors replicas	`1`
`connectors.contextPath`	can be used to make Connectors web application works on a custom sub-path. This is mainly used to run Camunda web applications under a single domain.	`""`
`connectors.podAnnotations`	can be used to define extra Connectors pod annotations	`{}`
`connectors.podLabels`	can be used to define extra Connectors pod labels	`{}`
`connectors.logging`	configuration for the Connectors logging. This template will be directly included in the Operate configuration YAML file	
`connectors.logging.level.io.camunda.connector`		`ERROR`
`connectors.service`	configuration to configure the Connectors service.	
`connectors.service.annotations`	can be used to define annotations, which will be applied to the Connectors service	`{}`
`connectors.service.type`	defines the type of the service https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types	`ClusterIP`
`connectors.service.serverPort`	defines the port number where the Connector web application will be available	`8080`
`connectors.service.serverName`	defines the port name where the Connector web application will be available	`http`
`connectors.resources`	configuration to set request and limit configuration for the container https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#requests-and-limits	
`connectors.resources.requests.cpu`		`1`
`connectors.resources.requests.memory`		`1Gi`
`connectors.resources.limits.cpu`		`2`
`connectors.resources.limits.memory`		`2Gi`
`connectors.env`	can be used to set extra environment variables in each Connector container	`[]`
`connectors.command`	can be used to override the default command provided by the container image. See https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/	`[]`
`connectors.extraVolumes`	can be used to define extra volumes for the Connectors pods, useful for TLS and self-signed certificates	`[]`
`connectors.extraVolumeMounts`	can be used to mount extra volumes for the Connectors pods, useful for TLS and self-signed certificates	`[]`
`connectors.startupProbe`	configuration	
`connectors.startupProbe.enabled`	if true, the startup probe is enabled in app container	`false`
`connectors.startupProbe.scheme`	defines the startup probe scheme used on calling the probePath	`HTTP`
`connectors.startupProbe.probePath`	defines the startup probe route used on the app	`/actuator/health/readiness`
`connectors.startupProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`connectors.startupProbe.periodSeconds`	defines how often the probe is executed	`30`
`connectors.startupProbe.successThreshold`	defines how often it needs to be true to be marked as ready, after failure	`1`
`connectors.startupProbe.failureThreshold`	defines when the probe is considered as failed so the Pod will be marked Unready	`5`
`connectors.startupProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`connectors.readinessProbe`	configuration	
`connectors.readinessProbe.enabled`	if true, the readiness probe is enabled in app container	`true`
`connectors.readinessProbe.scheme`	defines the startup probe scheme used on calling the probePath	`HTTP`
`connectors.readinessProbe.probePath`	defines the readiness probe route used on the app	`/actuator/health/readiness`
`connectors.readinessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before the probe is initiated.	`30`
`connectors.readinessProbe.periodSeconds`	defines how often the probe is executed	`30`
`connectors.readinessProbe.successThreshold`	defines how often it needs to be true to be marked as ready, after failure	`1`
`connectors.readinessProbe.failureThreshold`	defines when the probe is considered as failed so the Pod will be marked Unready	`5`

Name	Description	Value
`connectors.readinessProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`connectors.livenessProbe`	configuration	
`connectors.livenessProbe.enabled`	if true, the liveness probe is enabled in app container	`false`
`connectors.livenessProbe.scheme`	defines the startup probe scheme used on calling the probePath	`HTTP`
`connectors.livenessProbe.probePath`	defines the liveness probe route used on the app	`/actuator/health/liveness`
`connectors.livenessProbe.initialDelaySeconds`	defines the number of seconds after the container has started before	`30`
`connectors.livenessProbe.initialDelaySeconds`	the probe is initiated.	`30`
`connectors.livenessProbe.periodSeconds`	defines how often the probe is executed	`30`
`connectors.livenessProbe.successThreshold`	defines how often it needs to be true to be considered successful after having failed	`1`
`connectors.livenessProbe.failureThreshold`	defines when the probe is considered as failed so the container will be restarted	`5`
`connectors.livenessProbe.timeoutSeconds`	defines the seconds after the probe times out	`1`
`connectors.metrics.prometheus`	Prometheus metrics endpoint	`/actuator/prometheus`
`connectors.serviceAccount`	configuration for the service account where the Connectors pods are assigned to	
`connectors.serviceAccount.enabled`	if true, enables the Connectors service account	`false`
`connectors.serviceAccount.name`	can be used to set the name of the Connectors service account	`""`
`connectors.serviceAccount.annotations`	can be used to set the annotations of the Operate service account	`{}`
`connectors.serviceAccount.automountServiceAccountToken`	can be used to control whether the service account token should be automatically mounted	`false`
`connectors.ingress.enabled`	if true, an ingress resource is deployed with the Connectors deployment. Only useful if an ingress controller is available, like nginx.	`false`
`connectors.ingress.className`	defines the class or configuration of ingress which should be used by the controller	`nginx`
`connectors.ingress.annotations`	defines the ingress related annotations, consumed mostly by the ingress controller	`{}`
`connectors.ingress.path`	defines the path which is associated with the Connectors service and port https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules	`/`
`connectors.ingress.pathType`	can be used to define the Ingress path type. https://kubernetes.io/docs/concepts/services-networking/ingress/#path-types	`Prefix`
`connectors.ingress.host`	can be used to define the host of the ingress rule. https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-rules	`""`
`connectors.ingress.tls`	configuration for tls on the ingress resource https://kubernetes.io/docs/concepts/services-networking/ingress/#tls	
`connectors.ingress.tls.enabled`	if true, then tls is configured on the ingress resource. If enabled the Ingress.host need to be defined.	`false`
`connectors.ingress.tls.secretName`	defines the secret name which contains the TLS private key and certificate	`camunda-platform-connectors`
`connectors.podSecurityContext`	defines the security options the Connectors pod should be run with	
`connectors.podSecurityContext.runAsNonRoot`	run as non root	`true`
`connectors.podSecurityContext.fsGroup`		`1001`
`connectors.containerSecurityContext`	defines the security options the Connectors container should be run with	
`connectors.containerSecurityContext.privileged`		`false`
`connectors.containerSecurityContext.readOnlyRootFilesystem`		`true`
`connectors.containerSecurityContext.allowPrivilegeEscalation`		`false`
`connectors.containerSecurityContext.runAsNonRoot`		`true`
`connectors.containerSecurityContext.runAsUser`		`1001`
`connectors.nodeSelector`	can be used to define on which nodes the Connectors pods should run	`{}`
`connectors.tolerations`	can be used to define pod toleration's https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/	`[]`
`connectors.affinity`	can be used to define pod affinity or anti-affinity https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity	`{}`
`connectors.configuration`	if specified, contents will be used as the application.yaml	`""`
`connectors.extraConfiguration`	if specified, contents will be used for any extra configuration files such as the log4j2.xml	`{}`

Elasticsearch Parameters

Name	Description	Value
`elasticsearch`		
`elasticsearch.enabled`		`true`
`elasticsearch.image.repository`		`bitnami/elasticsearch`
`elasticsearch.image.tag`		`8.12.2`
`elasticsearch.master.containerSecurityContext.readOnlyRootFilesystem`		`true`
`elasticsearch.master.masterOnly`		`false`
`elasticsearch.master.heapSize`		`1024m`

Name	Description	Value
elasticsearch.master.persistence.size		64Gi
elasticsearch.master.resources.requests.cpu	cpu request	1
elasticsearch.master.resources.requests.memory	request	2Gi
elasticsearch.master.resources.limits.cpu	cpu limit	2
elasticsearch.master.resources.limits.memory	memory limit	2Gi
elasticsearch.master.extraEnvVars[0].name	env	ELASTICSEARCH_ENABLE_REST_TLS
elasticsearch.master.extraEnvVars[0].value	env value	false
elasticsearch.sysctlImage.enabled		true
elasticsearch.data.replicaCount		0
elasticsearch.coordinating.replicaCount		0
elasticsearch.ingest.enabled		false

Prometheus Parameters

Name	Description	Value
PrometheusServiceMonitor	configuration to configure a prometheus service monitor	
prometheusServiceMonitor.enabled	if true then a service monitor will be deployed, which allows an installed prometheus controller to scrape metrics from the deployed pods	false
promotheuServiceMonitor.labels	can be set to configure extra labels, which will be added to the servicemonitor and can be used on the prometheus controller for selecting the servicemonitors	
prometheusServiceMonitor.labels.release		metrics
prometheusServiceMonitor.scrapeInterval	can be set to configure the interval at which metrics should be scraped	10s