Open for Innovation
KNIME

KNIME Edge Installation Guide

KNIME AG, Zurich, Switzerland
Version 1.0 (last updated on 2024-01-29)

Table of Contents

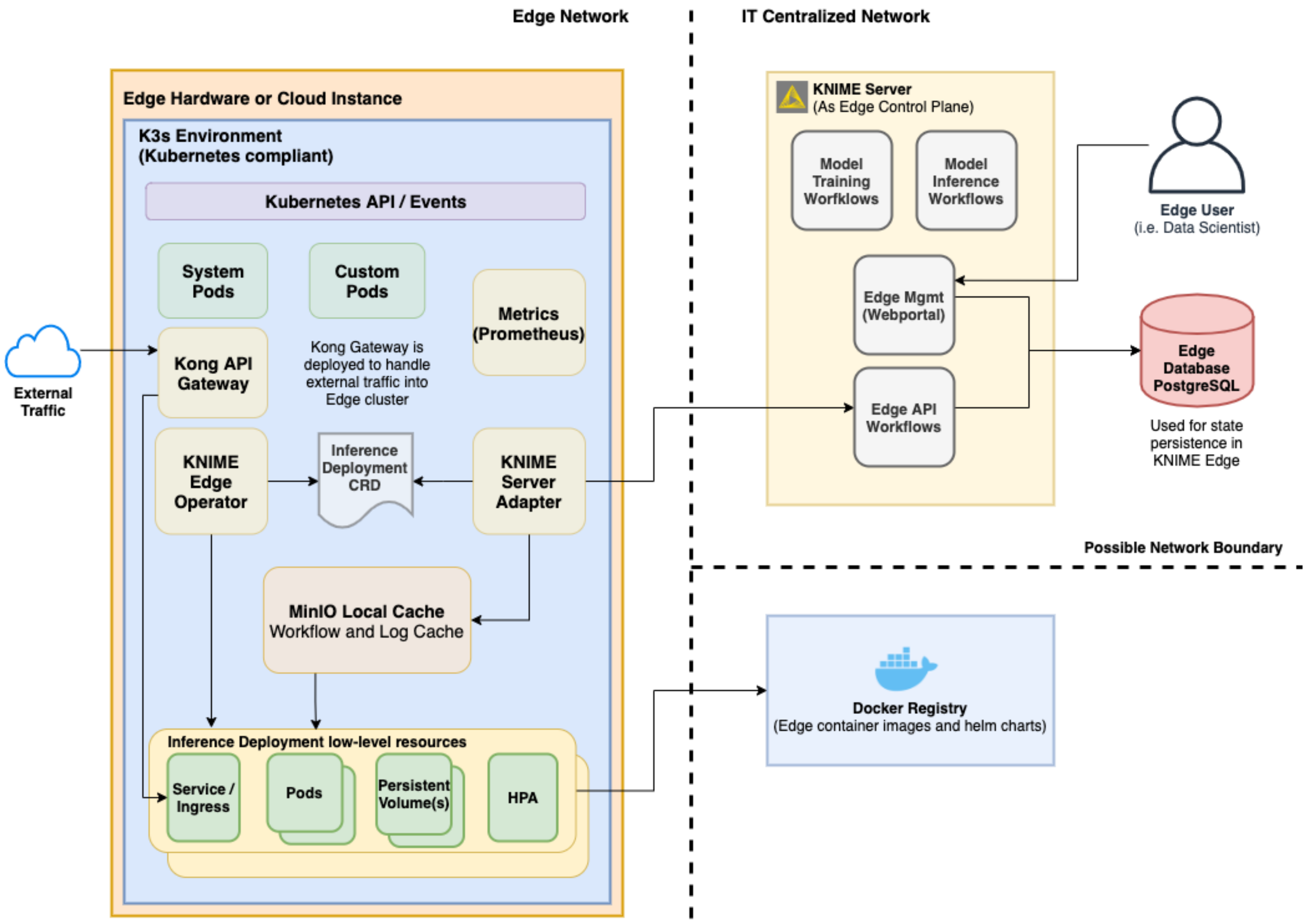
Introduction

This guide outlines the requirements, considerations, and steps for creating a KNIME Edge cluster.

KNIME Edge Architecture

Below is a high-level diagram of the architectural components within KNIME Edge.

KNIME Edge Architecture Diagram



Components of KNIME Edge

KNIME Server (Edge Control Plane)

KNIME Server is used as the "control plane" to manage one or more KNIME Edge clusters.

This is made possible by deploying a collection of workflows to KNIME Server for user interaction and management. Additionally, several of the workflows act as API endpoints that the KNIME Edge cluster(s) interact with.

State is maintained (in case of Server having multiple executors) by leveraging a PostgreSQL database.

KNIME Edge Server Adapter

The KNIME Edge Server Adapter is responsible for all communication between KNIME Edge and KNIME Server.

On startup, it will register the cluster to KNIME Server so that it may begin accepting work (i.e. Inference Deployments). When a deployment is assigned to a KNIME Edge cluster, the Server Adapter parses the instructions from KNIME Server and creates an "InferenceDeployment" resource in the cluster.

KNIME Edge Operator

The KNIME Edge Operator is responsible for initializing the KNIME Edge stack as well as reconciling / handling InferenceDeployments created by the Server Adapter. If a custom resource (e.g., InferenceDeployment) is created, updated, or deleted, the Operator responds by reconfiguring the low level resources to reconcile the cluster to an expected state.

KNIME Edge Inference Agent

The KNIME Edge Inference Agent provides a lightweight REST API on top of the KNIME Analytics Platform to execute one specific workflow that is loaded on initialization.

The purpose is to allow the solution to be built into a Docker container so the Inference Agent can scale as needed behind a network load balancer. Only one workflow can be specified, but multiple instances of that workflow can be loaded into memory. This allows the Inference Agent to handle multiple requests simultaneously.

Kong API Gateway

Kong is an open-source, third-party stack that provides KNIME Edge more flexibility and functionality in how external API requests are handled and proxied to the relevant deployments.

MinIO

MinIO is an embedded object store (which leverages the same API as the AWS S3 service). KNIME Edge uses MinIO to cache workflows, logs and other artifacts to optimize the amount of "fetching" required from KNIME Server.

Inference Deployment (Kubernetes Custom Resource)

Inference Deployment (`InferenceDeployment`) is a custom resource type that KNIME Edge adds to the Kubernetes API (thus making it a native resource type). It contains the definition for a KNIME Edge deployment, including:

- The KNIME workflow to deploy
- Resource allocation
- Scaling configuration

Installation Planning

KNIME Server Large

ℹ **Supported version(s):** KNIME Server Large 4.11.0+

KNIME Edge can be licensed to KNIME Server Large customers. See the Configuring KNIME Server for KNIME Edge Guide for details on configuring a KNIME Server as the control plane for one or more KNIME Edge cluster(s).

If you’re interested in KNIME Edge and want to understand more about the licensing, please contact our Customer Care team.

Kubernetes

ℹ **Supported version(s):** v1.18 to v1.21

A Kubernetes cluster must be available for KNIME Edge to function. See the Kubernetes documentation for details on installation and usage.

Helm CLI

Supported version(s): v3.0+

The Helm CLI is required for managing and utilizing the Helm charts for KNIME Edge. See the Helm documentation for details on installation and usage.

Kubectl CLI

The Kubectl CLI is required for executing various commands for controlling the KNIME Edge cluster. See the Kubectl CLI documentation for details on installation and usage.

Capacity planning & considerations

Memory per pod is the primary driver of infrastructure costs in a KNIME Edge cluster.

Warning: If insufficient memory is allocated to an Inference Agent pod, the pod will likely crash and Kubernetes will report an **OOMKilled** event for that pod.

When calculating needed capacity in Kubernetes for an Edge cluster, the **largest factor will be the expected number of Inference Deployment pods that will be running**.

The base components of Edge (`Server-Adapter`, `Edge-Operator`, `MinIO object cache`, `Prometheus`, etc.) require minimal resources to run. Combined, the base resources typically utilize a small portion of 1 CPU and around 1 GB of memory.

The KNIME Edge Inference Agent pod(s) that are deployed for each Inference Deployment require additional resources.

For most inference workflows, it is recommended that each corresponding KNIME Edge Inference Agent pod to be given an upper limit of **1 to 2 CPUs** and an upper limit of **2 GB of RAM**.

Note that each pod, once fully started, will typically average around **1.5 to 2 GB in active memory utilization**. This memory utilization is mostly static and does not typically increase or spike with request load.

CPU, by comparison, is heavily affected by request throughput. CPU usage will typically spike while requests are being processed and settle down to a lower level of utilization while waiting for new requests.

One useful method for gauging capacity requirements is to load test a single pod that utilizes the intended inference workflow and measure the number of requests handled and latency per request. By factoring in the expected number of concurrent Inference Deployments and number of pods running in each deployment, it's possible to estimate overall capacity needs.

It is recommended to add a ~20% buffer on top of estimated capacity for scalability. If a large-scale KNIME Edge cluster is being planned, the Considerations for large clusters page from the Kubernetes documentation may be useful.

Working with the KNIME Artifact Registry

Logging in with a knime.com user

The Docker images required for an Edge cluster are available in the KNIME Artifact Registry: https://registry.hub.knime.com

The `knime-edge` images and Helm charts can be found here.

Log into Harbor with a valid KNIME Hub user via the **OIDC Login** button.

> ℹ A valid KNIME Hub user (with elevated permissions) is required to access the `knime-edge` project, Docker images, and deployment charts in Harbor. Contact support@knime.com if help is needed accessing projects in Harbor.

Helm repository configuration

In order to install a KNIME Edge cluster directly from the KNIME Artifact Registry, the KNIME Edge chart repository will need to be added.

Authentication with the KNIME Artifact Registry and KNIME Edge chart repository requires a `username` and `password`. The `username` is the same as your Hub username. The `password` to specify is a CLI secret provided by Harbor. See the Using OIDC from the Docker or Helm CLI guide for information on how to retrieve the CLI secret (which acts as both the Helm and Docker password).

Add the Helm chart repository

To add the Helm chart repository, run the following command (with `kubectl` already available and configured), and substitute the `username` and `password`.

```
helm repo add --username <username> --password <password> knime-edge \
https://registry.hub.knime.com/chartrepo/knime-edge
```

Add Docker credential secret to Kubernetes

In order to pull Docker images directly from the KNIME Artifact Registry, a **secret** with the name `regcred` needs to be created in the KNIME Edge cluster. Creating the secret is performed using the Kubectl CLI.

```
kubectl create secret docker-registry regcred \
--docker-server=<your-registry-server> \
--docker-username=<your-name> \
--docker-password=<your-pword> \
--docker-email=<your-email>
```

> ℹ Make sure the secret is created in the same **namespace** where **Edge** is intended to be deployed. Run `kubectl get secret regcred --namespace <your-edge-namespace>` to verify.

Installing a new KNIME Edge cluster

Creating a KNIME Edge license secret

For the KNIME Edge cluster to work, a license is required. Further information about the licensing in KNIME Edge can be found in the KNIME Edge Licensing section of the KNIME Edge User Guide.

Once a license has been obtained, a **secret** with the name `knime-edge-license` that contains the license file needs to be created in the Kubernetes cluster.

```
kubectl create secret generic knime-edge-license --from-file=license=<path-to-license
-file>
```

> ℹ Make sure the secret is created in the same **namespace** where **Edge** is intended to be deployed.

KNIME Edge chart

The KNIME Edge Chart is a helm chart which defines the CRDs and dependencies needed by an Edge Cluster.

The parameters required to configure an Edge Cluster are specified within a standard Helm Values file, e.g. `values.yaml`, which is obtained from the `knime-edge` Helm repository.

The following steps outline the process of obtaining, understanding, customizing, and installing this chart.

Fetching the Helm values file for the KNIME Edge chart

Refresh Available Charts

```
helm repo update
```

Determine Version:

You can view all published versions of the KNIME Edge Helm charts in the Harbor registry. See the README in each chart version for the values file definition specific to that version. An overview of YAML techniques that can be used in the values.yaml file can be found in the Helm documentation.

(Advanced) All Chart Versions can be listed with the following command:

```
helm search repo knime-edge/knime-edge-operator --versions --devel
```

Values file configuration

Generate the file `values.yaml` to configure parameters for the KNIME Edge cluster:

For the latest stable release, run:

```
helm show values knime-edge/knime-edge-operator > edge-values.yaml
```

(Advanced) For a specific version, run:

```
helm show values knime-edge/knime-edge-operator --version <chart_version> > edge-
values.yaml
```

Installing the KNIME Edge chart

Once you have an updated values file relevant to your KNIME Server installation and environment, you can install Edge to your Kubernetes cluster with the following (add the namespace override flag if necessary).

```
helm install [-n <namespace>] knime-edge knime-edge/knime-edge-operator -f edge-
values.yaml
```

(Advanced) If using a specific Chart version, install with:

```
helm install [-n <namespace>] knime-edge knime-edge/knime-edge-operator --version
<chart_version> -f edge-values.yaml
```

Confirming a KNIME Edge cluster is operational

Verify Installation of Edge Cluster

Upon initial installation, the KNIME Edge Operator and KNIME Edge Server Adapter pods should be up and running. The Kubectl CLI can be used to check the status (the pod ID will be different in your cluster):

```
> kubectl [-n namespace] get pod <edge-name>-knime-edge-operator-6574d6fd79-stlss

NAME                                               READY   STATUS    RESTARTS   AGE
# KNIME Edge Operator
<edge-name>-knime-edge-operator-6574d6fd79-stlss   1/1     Running   0          7m43s
```

```
> kubectl [-n namespace] get pod server-adapter-697768bc8c-g9zjw

NAME                                               READY   STATUS    RESTARTS   AGE
# KNIME Edge Server Adapter
server-adapter-697768bc8c-g9zjw                    1/1     Running   0          7m40s
```

If the KNIME Edge Server Adapter does not exist, there might be an issue in the KNIME Edge Operator. To inspect the logs of the KNIME Edge Operator pod, run:

```
> kubectl [-n namespace] logs <edge-name>-knime-edge-operator-6574d6fd79-stlss
```

Additionally to checking that the pods are running, querying the endpoint should return an empty list for `inferenceDeployments`:

```
% curl -sL http://localhost:8081/ | jq
{
  "inferenceDeployments": []
}
```

The KNIME Edge Operator creates additional pods in the KNIME Edge cluster which should all be either in the Running or Completed state. For more information on these pods, see the section below.

Interpreting the pods in the cluster

The Kubectl CLI can be used to investigate the pods in a KNIME Edge cluster. Below is sample output from the `kubectl [-n <namespace>] get pods` command (where `-n <namespace>` defines the namespace) with comments added to help identify the various pods and their roles:

```
> kubectl [-n <namespace>] get pods

NAME                                              READY   STATUS    RESTARTS   AGE

# Tracks endpoints of Inference Deployments
endpoint-discovery-depl-6457b745b7-pkb24          1/1     Running   0          7m40s

# Logging Service
fluentd-ds-6q8l6                                  1/1     Running   0          7m41s
fluentd-ds-562zj                                  1/1     Running   0          7m41s

# KNIME Edge Operator
knime-edge-knime-edge-operator-6574d6fd79-stlss   1/1     Running   0          25m

# Kong
knime-edge-kong-5bb658458f-znwsg                  2/2     Running   0          25m

# Workflow and Log Cache
minio-b9b9547dd-xbvs2                             1/1     Running   0          7m41s

# Inference Deployment
sentiment-predictor-7d56b6cd47-lp8gp              1/1     Running   0          6m31s

# KNIME Edge Server Adapter
server-adapter-697768bc8c-g9zjw                   1/1     Running   0          7m40s

# Kong Gateway Loadbalancer
svclb-knime-edge-kong-proxy-bgqzc                 2/2     Running   0          25m
svclb-knime-edge-kong-proxy-jg8fg                 2/2     Running   0          25m

# Started by a cronjob to delete old logs
minio-log-cleaner-cronjob-1632700800-ssqsx        0/1     Completed 0          19m
```

Testing the KNIME Edge Server Adapter

The Server Adapter is an essential piece in the KNIME Edge architecture as it communicates with KNIME Server. If, e.g., Inference Deployments created in the control plane on KNIME Server are not showing up in the Edge cluster, a not properly working Server Adapter might be the issue.

First, test if the Server Adapter pod is running properly (the pod ID will be different in your cluster):

```
> kubectl [-n namespace] get pod server-adapter-697768bc8c-g9zjw
```

It should return the status "Running".

```
NAME                              READY   STATUS    RESTARTS   AGE
server-adapter-697768bc8c-g9zjw   1/1     Running   0          8m2s
```

To inspect the recent logs of the Server Adapter pod, run:

```
> kubectl [-n namespace] logs --since=30s server-adapter-697768bc8c-g9zjw
```

There are a few things to look for in the logs:

1. A line that says "Registration accepted" tells you that the KNIME Edge license is valid and set up correctly. Only if KNIME Server accepts the registration, the Server Adapter is able to pull information about Inference Deployments from KNIME Server.
2. A line that says "Unable to download file <knime-server>/workflow-paths.json" tells you that either a wrong KNIME Server URL or root path is set in the Helm values file.
3. The Server Adapter makes requests to different workflows on KNIME Server. For each request, it logs the HTTP response status. a) 200: the request was successful. b) 404: the workflow was not found. This could be caused by a wrong workflow root path being set in the Helm values file or by Edge workflows missing on KNIME Server. Make sure the Helm values file is configured correctly and run the Init workflow on KNIME Server again. c) 5xx: server error. Make sure KNIME Server is running properly.
4. A line that says "Error from workflow execute call to KNIME Server" followed by the body of the response means the workflow execution failed. This could be caused by incompatible Server Adapter and workflow versions. Run the Init workflow on KNIME Server again with the proper version set.

Advanced Operations and Troubleshooting

Error: Failed to pull image

Error Signature

```
Failed to pull image "registry.hub.knime.com/knime-edge/knime-edge-operator:0.0.1-beta-
20210802-161605-000061-4fa252d":\
 rpc error: code = Unknown desc = failed to pull and unpack image
"registry.hub.knime.com/knime-edge/knime-edge-operator:0.0.1-beta-20210802-161605-
000061-4fa252d":\
 failed to resolve reference "registry.hub.knime.com/knime-edge/knime-edge-
operator:0.0.1-beta-20210802-161605-000061-4fa252d":\
 unexpected status code [manifests 0.0.1-beta-20210802-161605-000061-4fa252d]: 401
Unauthorized
```

Interpretation:

- 401 Unauthorized
- image URI `registry.hub.knime.com/knime-edge/knime-edge-operator:0.0.1-beta-20210802-161605-000061-4fa252d`
    - web URL https://registry.hub.knime.com/harbor/projects/2/repositories → `knime-edge` → `knime-edge-operator` → search for tag `0.0.1-beta-20210802-161605-000061-4fa252d`

1. Log in to KNIME Artifact Registry

Log in to https://registry.hub.knime.com to refresh the validity of the access token.

2. Verify that the secrets are configured correctly

2a. Verify that the `imagePullSecrets` are configured

Look in deployment for imagePullSecrets; verify that `regcred` is present:

```
 % kubectl [-n <namespace>] get deployments knime-edge-knime-edge-operator -o json | jq
'.spec.template.spec.imagePullSecrets'
[
  {
    "name": "regcred"
  }
]
```

2b. Verify the secret exists in the correct namespace

```
% kubectl [-n <namespace>] get secret regcred
NAME      TYPE                             DATA   AGE
regcred   kubernetes.io/dockerconfigjson   1      50m
```

2c. Verify the value of the secret

Warning: Executing the command below prints the secret value to stdout

Examine Secret values:

```
% kubectl [-n <namespace>] get secret regcred -o json | jq -r
'.data.".dockerconfigjson"' | base64 -d | jq
{
  "auths": {
    "registry.hub.knime.com": {
      "username": "<harbor_user>",
      "password": "<harbor_secret>",
      "email": "<harbor_email>",
      "auth": "<auth>"
    }
  }
}
```

(Advanced) Examine Secret values without printing to standard out:

```
% cd ~/.shh
% diff <(kubectl [-n <namespace>] get secret regcred -o json | jq -r
'.data.".dockerconfigjson"' | base64 -d | jq -r '.auths[].password') <(cat
./harbor_secret); echo $?
0
```

Error: Unable to Access Host

Possible Causes:

- Misconfigured Ingress, i.e. Kong was not enabled or Traefik was left enabled

Error Signature: HTTP - Empty reply from server

```
% curl -L http://localhost:8081/
curl: (52) Empty reply from server
```

For reference, a completely misconfigured cluster would return: `curl: (7) Failed to connect to localhost port 8081: Connection refused`

1. Check the Kubernetes Cluster Networking settings

```
kubectl [-n <namespace>] get ingress
kubectl [-n <namespace>] get services
kubectl [-n <namespace>] get endpoints
```

2. Check the Kong Ingress Controller settings

Correct ingress configuration requires a helm install with `kong.enabled: true`.

Check for the following in the `values.yaml` used to deploy the cluster (or explore Kubernetes configuration to determine whether the value was set):

```
...
kong.enabled:
  true
```

Alternatively, check the manifest used by Helm at cluster install time:

```
helm status -o yaml [-n <namespace>] knime-edge | less
```

Resolution:

If necessary, set to `true` with:

```
helm upgrade [-n <namespace>] knime-edge knime-edge/knime-edge-operator --version
<chart_version> --reuse-values --set kong.enabled=true
```

Upgrading an existing KNIME Edge chart

To update to a new chart version, follow the instructions for obtaining an updated values file and then run:

```
helm upgrade [-n <namespace>] knime-edge knime-edge/knime-edge-operator --version
<chart_version> -f edge-values.yaml
```

If any errors occur, refer to the advanced troubleshooting section in this guide.

Error: Helm Upgrade is invalid due to Required Value

Error Signature: **`INSTALLATION FAILED: EdgeDeployment.edge.knime.com "edge" is invalid: ⋯ Required value`**

```
% helm upgrade [-n <namespace] edge knime-edge/knime-edge-operator --version
<chart_version> -f edge-values.yaml
...
    Error: INSTALLATION FAILED: EdgeDeployment.edge.knime.com "edge" is invalid: \
    [spec.knimeServerDeleteLogRequestWF: Required value, \
    spec.knimeServerDeployWF: Required value, \
    spec.knimeServerGetLogRequestsWF: Required value, \
    spec.knimeServerPushMetricsWF: Required value, \
    spec.knimeServerRegisterWF: Required value, \
    spec.knimeServerUploadLogFileWF: Required value]
```

Interpretation: These are all keys which are required by the currently installed CRDs which are used by the operator. The following questions explore potential root causes.

1. Are these values new in this version? Check the Chart

It is possible that a newer version may have updated values; check for this by following the steps to obtain a new copy of the `values.yaml` file and searching it for the mentioned values.

2. Are these values not present in the Chart for this Version? Delete the CRDs

The following command will remove all KNIME Edge CRDs:

```
kubectl delete customresourcedefinition $(kubectl get customresourcedefinition
-o=jsonpath='{.items[?(@.spec.group=="edge.knime.com")].metadata.name}')
```

Explanation: Helm deliberately will not modify CRDs, so if the required values change between versions, Helm cannot automatically manage the update.

KNIME AG
Talacker 50
8001 Zurich, Switzerland
www.knime.com
info@knime.com