Threedy Documentation Portal

Threedy

May 15, 2024

DOCUMENTATION

Note: Release 3.7.0 includes possibly breaking changes.

For further details see *3.7.0 Migration Guide*

CHAPTER

ONE

TASKS

1.1 instant3Dhub Integration

This section includes information on how to install and conifgure instant3Dhub to match your needs and environment. For more information about the product visit www.threedy.io. For direct support please refer to our ServiceDesk.

1.1.1 Installation Variants

instant3Dhub is a containerized software that can be deployed on top of a container orchestration layer. The primary target is the Kubernetes platform, which is also highly recommended for productive environments. Having an up and running cluster is required to continue the installation. However, we also provide a quick start guide for running instant3Dhub on a single linux machine with minimal overhead.

- For details on how to deploy instant3Dhub on an existing cluster, please follow the guide for installation on k8s
- For a quick start please follow the guide for installation on a single node

1.1.2 Support

Thank you for working with instant3Dhub. For further help or to give much appreciated feedback contact your personal contact at Threedy or open a ticket at our ServiceDesk.

instant3Dhub requirements

Read First

This section contains a general overview of the resource requirements for running the containerized instant3Dhub. These specifications are based upon measurements of average CAD Data encountered in various scenarios in the Automotive and BIM industry. Depending on the sizes of the input data, these values may need to be adjusted. As it is difficult to give a one-fits-all recommendation, we do not enforce any limits or requests in k8s as of yet. Storage requirements are given as a minimum.

Provisioning

To following resources have to be provisioned to enable the cluster to run instant3Dhub

- Helm & k8s
- License Server
- Image Registry
- Storage
- Compute
- Database (optional)

K8s Requirements

- Kubernetes: v1.21
- kubectl: v1.21
- helm (CLI): v3.4.0

PostgreSQL Requirements

- PostgreSQL: v9.2 (Tested: 10 and 13)

Newer versions on minor levels might work, but were not tested!

License Server

To run an instant3Dhub installation access to an instant3Dhub License Server is required. The license server is installed separately from the instant3Dhub instance and can be shared with other instant3Dhub instances. A guide on how to set up the license server can be found here.

Image Registry

We encourage transfering all containers referenced by our Docker-Compose or Helm deploys into a local selfmanaged image repository. To load the images into the registry you can use tools like skopeo or docker. The references contain an example script to fill the local image registry.

The images of instant3Dhub are provided on a public registry: instant3Dhub-images

> **Warning:** It is highly discouraged to use the public registry for direct access by the nodes!

If you can not use an image registry, the nodes of the cluster can also be provisioned proactively. For more information see: Pre-pulled images

Storage

instant3Dhub requires a set of volumes for persistency and data exchange between services in the system. Kubernetes volumes need to be configured in your backend. Our helm charts come with PersistentVolumeClaims which need to be satisfied for basic functionality. Examples for PersistentVolume definitions can be found in ./reference/volumes/pv_kubectl.yml. These need to be adjusted for your deploy.

The following non ephemeral volumes are defined and required to run instant3Dhub:

Name	Size	accessMode	Description
cache_volume	20GB	ReadWriteMany*	Stores the caches. Size can be configured. (grows depending on used data)
postgres_volume	5GB	ReadWriteOnce	Stores cache metadata and indices. Is 25% of the cache size.
elastic_volume	5GB	ReadWriteOnce	Log data. Currently fixed size.
entrygw_volume	2GB	ReadWriteMany*	Code-on-demand distribution. Fixed size.
rabbitmq_volume	1GB	ReadWriteOnce	Event / Transaction data. Fixed size.

Note: If instant3Dhub is deployed on a cluster with multiple nodes some of the services require the access mode **ReadWriteMany** (marked with *) which means that the chosen storage provider has to have such an option. (see the official Kubernetes documentation). ReadWriteMany – the volume can be mounted as read-write by many nodes. By this method, multiple pods running on multiple nodes can use a single volume and read/write data.

Compute

The containers have requests and limits defined in the Helm Chart. In the current version they are disabled by default and can be enabled by setting "resourceLimits.ignore: false" in the values.yaml file.

Database

For all Database purposes instant3Dhub relies on the use of PostgreSQL. instant3Dhub deploys a PostgresSQL instance within its containers. The use of external databases is explained here.

Graphics Processing Unit

With GPUs on the service components we can computationally support clients and allow model tracking services. Currently we only support NVIDIA GPUs.

To install GPU support for k8s we recommend using the NVIDIA GPU Operator. Following the NVIDIA guide will allow an easy installation of a variety of GPUs. Please check if the GPU of your choice is in the supported GPUs list. It is our experience however that other reasonable recent consumer GPUs also work even if not listed.

instant3Dhub on Kubernetes

Read First

Kubernetes is the targeted platform for running instant3Dhub. Therefore, this guide assumes a k8s cluster exists and can be accessed for deployments. Details about the requirements for running instant3Dhub can be found here Instant3Dhub is shipped with all the necessery components and services to run completely in kubernetes without the need to setup external services. However not all services. However it is possible to use exernal services like PostgreSQL, ELK stack and RabbitMQ. Setting up these external services is recommend in some scenarios and specially in production environments. More information on how to configure Instant3Dhub to run with external services can be found below.

Limitations:

- **HTTPS**: HTTPS is not supported inside the cluster or at the gateway. Currently HTTPS can be used by adding a proxy outside of the cluster or as a sidecar on the apigw component.
- **Services**: Currently only the SharedSession, SpaceStore, Measurement and Query services are enabled. This means a range of functionalities will not be available in webvis and other API-libraries.
- **Security/Signatues**: The third main version instant3Dhub is designed to include security on all layers of the system like controlling the access to the management APIs, resource APIs, the services or the data that goes through the system. It is not yet possible to configure custom keys.

Installation

Transfer docker images

Our docker images are available on our public registry images.threey.io and can be pulled from there. However we recommend that you transfer the images to your own registry as we do not guarantee high availablity of our registry. For transfering images to another docker registry, we provide a script.

```
./transfer_images.sh images.threedy.io registry.yourdomain.com
```

Install Helm

We provide a Helm Chart repository for the deployment of instant3Dhub on Kubernetes.

Therefore, helm should be installed first:

```
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-
↪helm-3 \
 && chmod 700 get_helm.sh \
 && ./get_helm.sh
```

We provide three channels for helm packages:

- **stable**: This channel contains the stable release packages of instant3Dhub.
- **dev**: This channel contains all rc and dev packages.
- **trk_<track-name>**: Each track package has its own channel and is named using the suffix `trk_` followed by the name of the track.

Now, add the instant3Dhub Helm repository from one of the channels:

```
helm repo add instant3Dhub https://repo.threedy.io/api/v4/projects/2/packages/helm/
↪<channel> \
&& helm repo update
```

You can now deploy using helm install but please continue reading to know which configuration you need to set. There are four main configuration that are required and need to be adjusted before you deploy. These configuration are:

- licenseServer
- storage
- entrypoints
- registry

```
helm install [release-name] instant3Dhub/instant3Dhub
```

As an alternative please consider the reference scripts located here

License Server

Instant3Dhub license server must be installed outside your k8s cluster and shoud be accessible from within the cluster so you can use the features of instant3Dhub. For information on how to install the license server please follow these instructions

Now you can set the address to the license server with the –set parameter.

```
helm install [release-name] instant3Dhub/instant3Dhub \
  --set licenseServer=license.yourdomain.com:8200
```

Storage

instant3Dhub requires a set of volumes for persistency and data exchange between services in the system.

In case you have a storage class that provides PVs dynamically, the option storage.class can be set to to be the name of your storage class via the –set parameter.

```
helm install [release-name] instant3Dhub/instant3Dhub \
  --set licenseServer=license.yourdomain.com:8200 \
  --set storage.class=your-storage-class
```

Otherwise, make sure that you create the following static PVs with their corresonding labels:

Name	Size	Labels
cache_volume	20GB	app: idhub-cache & instance: <The-name-of-namespace>
postgres_volume	5GB	app: idhub-postgres & instance: <The-name-of-namespace>
elastic_volume	5GB	app: idhub-elastic & instance: <The-name-of-namespace>
entrygw_volume	2GB	app: idhub-entrygw & instance: <The-name-of-namespace>
rabbitmq_volume	1GB	app: idhub-rabbitmq & instance: <The-name-of-namespace>

If you create the PVs manually with the correct labels, the storage.class parameter does not need to be set then.

An example of static persistent volumes can be found here

Access

In order for the system to address its resources properly, the entry points for the system have to be set. An entry point is a URL from which the system is accessible. For example if you have proxy in front of your cluster, the entry point must be set to be the URL of the proxy.

```
helm install [release-name] instant3Dhub/instant3Dhub \
  --set licenseServer=license.yourdomain.com:8200 \
  --set storage.class=your-storage-class \
  --set entrypoints={https://proxy.yourdomain.com:30000}
```

Images Registry

You should make sure that you are using the correct registry for pulling the docker images. Once you have transfered the images to your registry, you can set the URL in helm chart.

```
helm install [release-name] instant3Dhub/instant3Dhub \
  --set licenseServer=license.yourdomain.com:8200 \
  --set storage.class=your-storage-class \
  --set entrypoints={https://proxy.yourdomain.com:30000} \
  --set registry=registry.yourdomain.com
```

At this point you should have all the requirements to run instant3Dhub. The startup and initialization of all containers can take a few minutes, depending on your cluster and registry connection speed. To check the status of the containers run:

```
watch kubectl get pods -n your-namespace
```

After all containers are running the system might take up to one minute to initialize correctly.

Using External Services

PostgreSQL

The system can optionally work with a previously provisioned PostgreSQL instance. This instance must be initialized corrctly with our schema and tables. For that we provide ansible playbooks to install and initialize postgres manually. For more information please *see our guide*.

If you are using external PostgresSQL, we recommend using k8s secrets to store the connection string. The secrets can be referenced then using these values in our helm chart.

Name	Description
credentials.postgres.system.secretKeyRef.name	The name of the secret where the connection string is stored
credentials.postgres.system.secretKeyRef.key	The name of the key in the secret's items.

RabbitMQ

The system uses rabbitmq to dispatch jobs. If you want to use external service for that, it is possible to change the configuration to point instant3Dhub to use the external service.

These values need to be set in our helm chart values then.

Name	Description
credentials.rabbitmq.secretKeyRef.name	The name of the secret where the connection string is stored
credentials.rabbitmq.secretKeyRef.key	The name of the key in the secret's items.

instant3Dhub on single node

Read First

This guide targets those who have neither an existing k8s cluster nor the resources to set up and manage one. While having a k8s cluster to run instant3Dhub is still a requirement, we provide tools to get all the requirements ready for you on a single CentOS 8 machine. For installing on WINDOWS, please follow the instructions here

Prerequisites

The machine where you want to run instant3Dhub should have the following requirements in order to run the system properly:

Resources

These minimal resources are required to run instant3Dhub

- CPU-only: CPU=4 Cores, Mem=16GB
- with GPU: CPU=8 Cores, Mem=32GB, GPU=1 NVIDIA GPU

System

- Snapd is installed and running
- SELinux is disabled
- The system has the correct Linux kernel sources from the CentOS repositories
- If you want to use GPU support, then make sure that the nouveau driver is disabled and that there is no other driver installed on the host system running microk8s.
- If your system uses firewalld then make sure that masquerade is enabled on the host.
- You have a running instant3Dhub License Server which can be accessed from the host. For installing instant3Dhub License Server please check the documenation here.

If you are not sure if or how to make your system meets these requirements, then please check and run the prerequisite.sh script which will reconfigure your system to meet the requirements 1-5. Please note that a restart might be required to make the changes persistent.

Installation

For installing instant3Dhub on a single machine:

Clone the release repository

```
git clone http://repo.threedy.io/instant3Dhub/release.git

cd release/references/singleNode/
```

The script expects these env variables to be set in the setup file:

- RELEASE_NAME: Is the name of the release to be used when deploying instant3Dhub. This can be any name of your choice. Example: instant3dhub-test. Default is: i3dhub-singlenode.
- ENTRYPOINT: The external exposure hosts or proxies. Example: http://host-name.your-domain.com:30000. Default is: http://your-hostname:30042. For more information on how to set up a proxy please see our *guide*.
- REGISTRY: Is the container registry where instant3Dhub images can be fetched. Example: hub.docker.com. Default is: images.threedy.io.
- LICENSE_SERVER: The address of the license server. Example: http://license.threedy.io:8200. Default is empty.
- ENABLE_GPU: Set to true if your system has a GPU and you want to enable it to be used by instant3Dhub visualization services. Default is false.

Please adjust these variables based on your setup.

Now you can install instant3Dhub

```
./install.sh
```

instant3Dhub on single node - WINDOWS

Read First

This guide targets those who wants to run instant3Dhub for local testing on a WINDOWS machine. Installing the license server is not part of this guide as it requires Linux machine and we do not support WINDOWS.

Prerequisites

The machine where you want to run instant3Dhub should have the following requirements in order to run the system properly:

These minimal resources are required to run instant3Dhub

- CPU-only: CPU=4 Cores, Mem=16GB
- with GPU: CPU=8 Cores, Mem=32GB, GPU=1 NVIDIA GPU

Installation

1. Install Docker Desktop: Docker Desktop is freely available in a community edition, for WINDOWS and Mac. Start by downloading and installing docker desktop: https://hub.docker.com/editions/community/docker-ce-desktop-windows.
2. Enable Kubernetes
    1. Make sure you have Docker Desktop running - in the taskbar in WINDOWS you'll see Docker's whale logo. Click the whale and select Settings.
    2. A new screen opens with all of Docker Desktop's configuration options. Click on Kubernetes and check the Enable Kubernetes checkbox.
    3. Docker Desktop will download all the Kubernetes images in the background and get everything started up. When it's ready you'll see two green lights in the bottom of the settings screen saying Docker running and Kubernetes running.
    4. Verify your Kubernetes cluster by running the following command

        ```
        kubectl get nodes
        ```

        You should see a single node in the output called docker-desktop. That's a full Kubernetes cluster, with a single node that runs the Kubernetes API and your own applications.
3. Install helm: We provide a Helm Chart repository for the deployment of instant3Dhub on Kubernetes. Therefore, helm should be installed first: Please follow the instructions from the official helm documentation to get helm on WINDOWS: https://helm.sh/docs/intro/install/
4. Now, open a command prompt and run the following command to add the instant3Dhub Helm repository:

```
helm repo add instant3Dhub https://repo.threedy.io/api/v4/projects/2/packages/helm/
↪stable \
&& helm repo update
```

5. Run helm install to deploy instant3Dhub

```
helm install -n i3dhub i3dhub-windows instant3Dhub/instant3Dhub \
 --set licenseServer=license.yourdomain.com:8200 \
 --set storage.class=hostpath \
 --set entrypoints={http://localhost:30042} \
 --set registry=images.threedy.io
```

6. The startup and initialization of all containers can take a few minutes, depending on your cluster and registry connection speed. To check the status of the containers run:

```
kubectl get pods -n i3dhub
```

#. Once all containers are running, you should be able to access the splash page at http://localhost:30042

install tools: Ansible

Install Ansible

For simplicity we provide an example with CentOS 7. Other Linux LSB distributions should be handled similarly. We assume that we are on a clean install as root.

```
yum -y update
yum -y install epel-release
yum -y update
```

(update cache)

```
yum -y install ansible
ansible --version
```

should show ansible 2.9.10 later versions should be fine aswell

instant3Dhub License Server

Read First

The instant3Dhub License Server needs to be accessible by the servers instant3Dhub is running on. The License Server cannot be run in container environments as it is bound to a host machine. There are two options for storing data (you need to use PostgreSQL 14):

- *PostgreSQL on the same server*
- *PostgreSQL managed remotely*

We recommend an externally managed PostgreSQL in order to avoid data loss.

Installation

First start by installing and setting up PostgresSQL and obtain a connection string.

Using local PostgreSQL instance

We assume that no instant3Dhub version 2 PostgreSQL installation is running on the same machine. If this is the case the port and socket directory used by the license database Postgres cluster must be changed from 5433 to another port, for example to 5435 and `/var/run/instant3DhubLicensePgsql`.

We recommend PostgreSQL version 14, but require a minimum version of 9.5. The minmum version may change in the future, so we recommend installing the highest version available in order to avoid time consuming upgrades.

For RedHat based systems, we recommend using the official PostgreSQL repository to download packages. This guide will exemplify the installation using RedHat.

First, add the official PostgreSQL repository to the package manager:

```
yum -y install https://download.postgresql.org/pub/repos/yum/reporpms/EL-$(rpm -E %{rhel}
↪)-x86_64/pgdg-redhat-repo-latest.noarch.rpm
yum -y update
```

Next, install PostgreSQL 14:

```
yum install -y postgresql14
yum install -y postgresql14-server
yum install -y postgresql14-contrib
```

Initialize and check the PostgreSQL-service (must be active and running):

```
/usr/pgsql-14/bin/postgresql-14-setup initdb
systemctl start postgresql-14
systemctl enable postgresql-14
systemctl status postgresql-14
```

Warning: If Postgres-service cannot be launched then set the correct permissions for PostgreSQL from `0755` to `2777`:

```
sed -i 's/postgresql 0755 postgres postgres - -/postgresql 2777 postgres postgres - -/g'␣
↪/usr/lib/tmpfiles.d/postgresql-14.conf
```

Now you may configure a connection string or use the defualt one (for details see *Configuration of Postgres connection string*)

Now you may configure a connection string or use the defualt one (for details see *Configuration of Postgres connection string*)

Using remote PostgreSQL instance

In case you have a PostgreSQL instance that is running on a differnt machine or provisioned externally, all you need is to *configure* the License Server to use the remote PostgreSQL instance by providing a connection string (see *Configuration of Postgres connection string*). The License Server will setup or migrate required tables on startup. To start the server, make sure to use the systemd unit file which does not depend on a local postgres instance:

```
systemctl start instant3DhubLicenseServerRemoteDB.service
systemctl status instant3DhubLicenseServerRemoteDB.service
```

Configuration of Postgres connection string

Initially Postgres service sets up a database `postgres` with a username `postgres` and a password `postgres`, which is being used as the default connection by the License Server as follows `postgresql://postgres:postgres@localhost:5432/postgres?sslmode=disable` (see *License Server configuration*). You may configure a new connection according to the pattern `postgres://{NEW_USER}:{PASSWORD}@{NEW_HOSTNAME}:{PORT}/{NEW_DATABASE}?sslmode=disable` (described here) To create a new database with a new user please refer to the official.

Install the License Server

The instant3Dhub License Server is installed from RPM files. The installation requires the OS dependent License Server package:

- **CentOS 7** : instant3Dhub-licenseServer-CentOS-Linux-7-x64*.rpm
- **SLES-12.3** : instant3Dhub-licenseServer-SLES-12.3-x64*.rpm
- **Fedora 23** : instant3Dhub-licenseServer-Fedora-23-x64*.rpm
- **Fedora 29** : instant3Dhub-licenseServer-Fedora-29-x64*.rpm

The latest installation files and a changelog can be found here.

Note: For OS where is the Debian package management tools are used installation process of the License Server and a Postgres service instance differs. Please refer to the page with example for Ubuntu (*instant3Dhub License Server on Ubuntu*).

First download the latest version of the License Server for your OS, for example:

```
wget https://repo.threedy.io/licenseserver/23.1/instant3Dhub-licenseServer-CentOS-Linux-
↪7-x64.23.1.rpm
```

And install the package:

```
yum install -y ./instant3Dhub-licenseServer-CentOS-Linux-7-x64.23.1.rpm
```

License Server configuration

Before starting the license server it must be properly configured. Configuration is provided by creating a configuration file at the `/opt/instant3Dhub.custom/license_config.yml`. Below is a sample file with available options and their defaults. If no file exists, the defaults seen below will be used:

```
# whether to use the BIOS ID of the current machine when performing host checks
use-bios-uuid: false

# interface and port to listen on
address: 0.0.0.0:8200

# log level to use. options: fatal panic error warning info debug trace
log-level: warning

# location of the license file
licenseFile: /opt/instant3Dhub.custom/license.xml

# location of tls certificate
tls-cert:

# location of tls private key
tls-key:

# postgres backend options
```

(continues on next page)

(continued from previous page)

```
postgres:
  # connection string describing the postgres location and user. the given
  # user must have permissions to create schemas, tables and stored procedures.
  # expected format is the URI format described here: https://www.postgresql.org/docs/
↪current/libpq-connect.html#LIBPQ-CONNSTRING
  connection-string: postgresql://postgres:postgres@localhost:5432/postgres?
↪sslmode=disable

  # user to use once migrations are complete. this user also receives table write␣
↪permissions during migrations
  # by default the user from the connection string is also used during runtime.
  runtime-user:

  # password for the runtime user to allow switching to this user once migrations are␣
↪complete
  # this value must be given if a runtime user is also provided
  runtime-password:
```

Note: You may configure your connection instead of using the default one (for details see *Configuration of Postgres connection string*).

HostID generation

Licenses are bound to a single host. We require a HostID to provide a license. Please invoke the following:

```
/opt/instant3Dhub/bin/licenseTool
```

The output should look similar to the following:

```
Current Host ID: 7bf3b23f5c6ff5a444a37f2dfc42ed34f5c470df
```

Please provide your threedy contact or if you have no valid license agreement yet `sales@threedy.io` with this output to start the license key aquisition process. In return a license file will be provided to you.

Warning: The hostID depends on the hardware configuration of your machine such as the network adapter and hard drive. In some virtualized environments, the hardware configuration can be dynamic by default and it might changes when restarting a VM. Therefore, please make sure that is not the case in your setup as this will result in a changing HostID.

License file placement

A received license file must be placed to:

```
/opt/instant3Dhub.custom/license.xml
```

Start the License Server

After getting a license and placing it to `/opt/instant3Dhub.custom/license.xml` the License Server can be launched (must be active and running):

```
systemctl restart instant3DhubLicenseServer{RemoteDB}.service
systemctl status instant3DhubLicenseServer{RemoteDB}.service
```

The status should now be active.

In default configuration this service exposes 8200.

An address to the License Server must be placed inside the setup file of your instant3Dhub deployment.

instant3Dhub License Server on Ubuntu

This document contains information on how to install Postgres service and License Server on Ubuntu. Here are reflected only the most crucial details specific to Ubuntu. Complete installation process can be found on the main page (*instant3Dhub License Server*).

Installation

Using local PostgreSQL instance

Install PostgreSQL (we suggest to install version 14):

```
apt install -y postgresql-14 postgresql-contrib
```

Initialize and check the PostgreSQL-service (must be active and running):

```
systemctl restart postgresql
systemctl enable postgresql
systemctl status postgresql
```

Install the License Server

Threedy does not provide Debian packages, therefore we will use the tool 'alien' to convert RPM package to DED format.

Download the desired version of the License Server package for CentOS:

```
wget https://repo.threedy.io/licenseserver/23.1/instant3Dhub-licenseServer-CentOS-Linux-
↪7-x64.x.x.rpm
```

Convert the RPM-package to DEB-format:

```
alien /root/source_files/license_server_ubuntu/instant3Dhub-licenseServer-CentOS-Linux-7-
↪x64.x.x.rpm
```

Install the package:

```
dpkg -i /root/source_files/license_server_ubuntu/instant3dhub-licenseserver-centos-linux-
↪7-x64_x.x-2_amd64.deb
```

The License Server can be launched (must be active and running):

```
systemctl restart instant3DhubLicenseServer{RemoteDB}.service
systemctl status instant3DhubLicenseServer{RemoteDB}.service
```

Warning: If the License Server cannot be launched then add the user `instant3dhub` with permissions as follows:

```
adduser -u 901 --gecos "" --disabled-password instant3dhub
```

Guide: running instant3Dhub with Proxy

Read First

This section contains a configuration examples for instant3Dhub with a proxy aswell as an example configuration for nginx. Use the nginx configuration at your own risk. This nginx configuration has no security. We support SSO mechanisms via cookie passthrough. Our backends authenticate to data backends via cookies or other custom HTTP headers.

instant3Dhub Configuration

Our deploys require a small set of variables which must be set for minimal deployments. This is done by adjusting the entrypoints variable either in our helmcharts or docker-compose variants to the external address under which the installation should run.

As an example, if the installation should run under a different path on a proxy server the entrypoints should be set as the full URL: https://example-proxy-server/extra/path/ This way internal components which deliver our webfrontend already know how to deliver addresses to get back to the the backend.

```
# entrypoints in values.yaml
entrypoints: [ "https://example-proxy-server/extra/path/" ]
```

For convenience we provide a nginx configuration as a reference below. It assumes the installation is running on `146.140.211.12:30101` and proxies anything under `/hubproxy/` to the instance.

```
user nginx;
worker_processes  3;
error_log  /var/log/nginx/error.log info;
events {
  worker_connections  10240;
}
```

(continues on next page)

(continued from previous page)

```
http {
  access_log  /var/log/nginx/access.log;
  server {
  client_max_body_size 100M;
  listen  80;
  proxy_read_timeout 7d;
  proxy_http_version 1.1;
  location ~ /hubproxy/(.*) {
    proxy_pass http://146.140.211.12:30101/$1$is_args$args;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "Upgrade";
  }
  client_body_temp_path      /var/cache/nginx/client_body_temp;
  proxy_temp_path            /var/cache/nginx/proxy_temp;
  fastcgi_temp_path          /var/cache/nginx/fastcgi_temp;
  uwsgi_temp_path            /var/cache/nginx/uwsgi_temp;
  scgi_temp_path             /var/cache/nginx/scgi_temp;
  }
}
```

Guide: running instant3Dhub with external PostgreSQL

Read First

instant3Dhub comes packaged with PostgreSQL. This guide explains how to provision a separate PostgreSQL instance. It is advised to remove the PostgresSQL services from your deploy configuration if you choose to host your own PostgresSQL instance. Although this might look similar to the LicenseServer install, it uses a different set of Ansible playbooks and does other things.

Installation

In order to run an external PostgreSQL with instant3Dhub you need Ansible. If you need guidance on how to install Ansible please refer to our Ansible install guide. The archive containing the Ansible-playbooks can be found in `./init/postgres/instant3Dhub-ansibledb-noarch-*.tgz`

Install PostgreSQL

In the following documentation we assume that the PostgreSQL is installed on the same host as the ansible control host. For that reason the following inventory is a localhost inventory.

First unzip the provided `instant3Dhub-ansible-noarch-\*.tgz`:

```
tar zxvf ./instant3Dhub-ansible-*.tgz
```

The folder where this is extracted should now contain a `./PostgreSQL/` folder aswell as `./inventories/` containing example Ansible Inventories.

Create the following `hosts.yml`:

```
hubs:
  hosts:
    localhost:
  vars:
    ansible_connection: local
```

Invoke the playbooks:

```
ansible-playbook -i hosts.yml ./PostgreSQL/pgsql.install.i3dhub3.yml
ansible-playbook -i hosts.yml ./PostgreSQL/pgsql.init.i3dhub3.yml
```

This installs and initializes a PostgreSQL instance.

Guide: running instant3Dhub with Grafana

Read First

This section explains how to enable Grafana for instant3Dhub. As Grafana uses AGPL it is not directly packaged with a regular installation. Grafana serves as a tracing and debugging tool to diagnose functional or performance issues.

instant3Dhub configuration

The Grafana image is not hosted on the Threedy image repository. The official docker image hosted on docker.io is used by default. In order to copy the image to a local registry, the following script can transfer the image to your repository via docker:

```
docker pull docker.io/grafana/grafana:9.1.6
docker image docker.io/grafana/grafana:9.1.6 ${TARGET_IMAGE_REGISTRY_HOST}/grafana/
↪grafana:9.1.6
docker push ${TARGET_IMAGE_REGISTRY_HOST}/grafana/grafana:9.1.6
```

Grafana is disabled by default. The following options must be set in `values.yaml` to enable Grafana support:

```
# Controls third party system tracing and logging.
tracing:

  # Options for deploying services required for integrated tracing and metrics
  # dashboards.
  withIntegrated:

    # Whether this option is enabled. This does not include Grafana. Metrics
    # must be manually explored via the Prometheus GUI. Traces must be manually
    # explored via the Jaeger GUI.
    enabled: true

    # Settings for deploying Grafana.
    grafana:

      # Whether grafana should be enabled. This also deploys dashboards detailing
      # system performance regarding transcoding, service runtimes, error rates
      # and memory usage. Additionally, metrics are linked to traces to allow
```

(continues on next page)

(continued from previous page)

```
    # detailed inspection of internals for given metrics.
    enabled: true

    # Which Grafana image to use. The image is not hosted on the Threedy registry
    # as it is AGPL licensed.
    image: docker.io/grafana/grafana:9.1.6
```

Or, alternatively, via Helm

```
# helm chart parameter
helm install --set tracing.withIntegrated.enabled=true --set tracing.withIntegrated.
↪grafana.enabled=true ...
```

1.2 Troubleshooting an instant3Dhub Instance

1.2.1 Introduction

Note: For troubleshooting we focus on Kubernetes as it is our recommended deploy format.

The following cases contain frequent issues encountered during the rollout of instant3Dhub. This content serves as an addition to the other integration guides and will be extended over time. In case you need support it is vital that you look into the following topics first.

1.2.2 License Server

The startup of all instant3Dhub components is independent of the license server setup. All pods will show `Running` even if the address to the license server is wrong or the license is expired. Errors will only become visible during transcoding or other transactions which require a license checkout.

1.2.3 Container Stuck in `ContainerCreating`

This behavior can have many causes. Its is likely that the root cause is not an instant3Dhub issue. Possible sources of this behavior:

- Volume mounts not setup correctly.
- Special capability nodes are all claimed by other pods.

Refer to *debugging steps below*, especially `kubecl describe` to determine exact failure causes.

1.2.4 Pods shows unusual amount of `CrashLoopBackOff`

As instant3Dhub does not feature init containers yet, it is normal to see at least a few restarts of pods until the system is fully operational. Should this count exceed 10 it is likely that something else went wrong. Usually the container logs provide a hint of the problem.

Several pods are required for the system to work, and containers which depend on these will be in a `CrashLoopBackOff`:

- `i3dhub-postgres`
- `i3dhub-rabbitmq`
- `i3dhub-consul`
- `i3dhub-keystore`

Once these have started succesfully the rest of the containers should start. If any containers are not starting, it is worth trying the *debugging steps below* to determine the root cause.

1.2.5 `kubectl` troubleshooting

If you prefer other tools to monitor or manage your cluster feel free to use them. As `kubectl` should be available anywhere, we use it as a baseline tool to explain our way of troubleshooting instant3Dhub. For extensive information use the official kubernetes documentation. The following commands should be enough to figure out most errorcases:

Determine pod states:

```
kubectl describe pod <name> -n <namespace>
```

Print logs of running or failed pods:

```
kubectl logs <podname> -n <namespace>
```

1.3 Data Gateway Integration

1.3.1 Introduction

The system offers two main APIs for integration into existing environments. The webVis-API allows the application developer to visualize different resources based on simple URI (can be URN or URL), and the Data Gateway API provides pull-access to the data based on a single URL per URI. The system automatically translates the URI to URL based on local configuration settings.

Resource Specification

A URI can either be a URL or a URN. URNs are mapped to URLs by the system. For example:

- URN: urn:company:backend:document:123456
- URL: https://backend.company.net/api/document/123456

The URL must be valid location with a known schema (e.g. HTTPS). The application also must provide authentication tokens if necessary. To support this, the system can transparently pass single-sign-on tokens (e.g. Cookies) along.

The URN can be an installation specific name which is automatically mapped to a configured location while processed. Exact mappings can be configured within the system via a mapping table.

Configuring a URN mapping rule instead of directly using URLs allows for decoupling of client applications interacting with the system and backend data gateway locations. This allows future relocation or replacement of backends without needing to modify client applications.

1.3.2 Data Gateway Interface

The Data Gateway Interface is a simple but powerful abstraction to access data and builds on best practice webserver techniques. It maps the webVis URI to a single URL which can be configured to specific needs. The resource URL can be any valid URL container. For example:

> https://backend.company.net/api/document/123456

The interface tries to achieve the following goals:

- Minimal interface to pull data from any data backend
- Based on standard HTTP techniques and best practices
- Standard Apache server should be able to provide sufficient functionality
- Provide single interface to all persistent and dynamic resources
- New backends can be added without changes to instant3Dhub components

Resource Concepts

Several key concepts are supported by the infrastructure and define how cached resources are handled by the system.

A resource can be:

- *static*, i.e. it never changes. Static resources are never checked for newer versions.
- *dynamic*, i.e. it can change over time. How often changes are expected to occur can be specified to control update checks.
- *public*, i.e. personalized authorization checks are skipped when providing clients with caches. This can only be set explicitly, as resources are protected by default.
- *protected*, i.e. personalized authorization must be done against the source when proving clients with caches. This is the default behavior and must be explicitly overridden.

Each of these properties can either be defined per URN, or set in the HTTP response headers of the source, as described in the next few sections.

The system accesses resources in two separate ways:

- Retrieving the resource itself to generate a cached representation.
- Retrieving properties about the resource, such as whether a certain user has the appropriate viewing rights.

It is important to make this distinction, as the resource will not need to be retrieved very often, while property requests can happen very frequently.

Accessing Resources & Authorization

Resources are accessed via HTTP GET and authorized via HTTP HEAD requests. The system assumes that a HEAD request is faster than a full GET request, as HEAD requests are made more frequently, and are also used to determine whether a cached representation is outdated, or whether a user is authorized to view a resource.

- **GET**: Requests a representation of the specified resource (header + body)
- **HEAD**: Expected to contain the same information as GET, only without a body

For both HEAD and GET requests, Single-Sign-On tokens in the form of cookies or other HTTP headers are supported. Both request types use the token of the user initiating the request.

Response Codes

Standard HTTP response codes are supported. The most essential are the following:

- **200**: OK. This results in a cache being generated, or a cached representation being delivered to the user.
- **202**: Data not yet available. No cache is generated for this response. Future requests will try again.
- **30X**: Redirect. Redirects are only followed for GET requests, never HEAD requests. A redirect for a HEAD request results in unauthorized behavior for users.
- **404**: Not Found. No cache can be generated for this response, and it is communicated to the user. However, future requests will still try again.
- **401**: Unauthorized. No cache can be generated (GET request), or user does not have authorization (HEAD request). This is communicated to the user.
- **403**: Forbidden. Same behavior as 401.

For more details see the technical API.

Response Headers

In addition to the response code, response headers can be used to give additional hints about the data itself. The following headers can be used:

- **ETag**: Stored on initial GET requests and compared on every HEAD request. If the value does not change, the cached resource is considered up-to-date. The infrastructure will not try to update the cached resource.
- **Content-Type**: Used on initial GET request to signal the data format to the system. This can be used to override a format configured for a URN or to prevent instant3Dhub from trying to guess a type based on a file extension. Known types can be found on https://www.threedy.io/scalability/anydata under *Service Negotiation Key*.
- **Content-Disposition**: Used as a fallback to the Content-Type on GET requests. Defines a filename for the data, of which the file extension is used by the system as a hint for selecting an appropriate loader.

For more details see the technical API.

URN Mapper Configuration

Configuration

System configuration is changed via the provided `values.yaml` file. The `dataGateways` section defines URN mapping rules for each external data gateway the system can connect to. By default no rules are active except the first `defaultGateway` providing the rules for basic shapes provided as part of the installation.

Note: The default `defaultGateway` rule cannot be disabled (even if removed from configuration) and is only part of the configuration as an example:

```
# Set the URN mapping rules per data gateway. See the integration documentation for
# motivation and concepts behind data gateways and URN mappings.
#
# For templates, $(n) is the n-th value in the URN separated after␣
↪urn:namespace:specifier:xxx
# e.g.: urn:x-i3d:shape:sphere ($(1) == sphere)
#
dataGateways:
  defaultGateway:
    - namespace: x-i3d
      specifier: shape
      urlContentType: [ x3d-xml ]
      urlTemplate: http://i3dhub-entrygw:8080/repo/shapes/$(1).x3d
```

To define a new rule we can remove the `defaultGateway` and add a new custom entry:

```
dataGateways:
  customerGateway1:
    - namespace: customer
      specifier: document-uuid
      urlContentType: [ openjt ]
      urlTemplate: https://download.example.com/documents/$(1).jt
```

This will create a new URN in the form of urn:customer:document-uuid:12345 which will then internally resolve to https://download.example.com/documents/12345.jt.

Mapping Logic

The mapper itself applies very simple logic to resolve a URN to a URL:

1. `urn:NAMESPACE:SPECIFIER:` is matched to an incoming URI in order to find the correct mapping rule (if any) to apply.
2. Splitting any arguments (separated by `:`) following the matched prefix and assigning to `$(1)`, `$(2)`, `$(3)`...
3. Replacing any `$(x)` values in the given `urlTemplate` by their values parsed from the URN.

1.4 Signed Input Configuration

1.4.1 Introduction

The system provides a security mechanism to restrict access to system resources to require valid signatures. Signatures are always enabled by default and use internally generated keys if no custom keys are configured.

The system provides the option to require any input URIs to be cryptographically signed. This is disabled by default.

Architecture

Signing keys are stored in the system PostgreSQL database. Any changes to the relevant tables are automatically propagated to the i3dhub-keystore service, which in turn broadcasts changes via an internal queue in RabbitMQ. Any services requiring signing keys listen to these changes and update their keys appropriately.

1.4.2 Custom Keys

Custom keys can be provided by directly inserting these into a running instance of the system. Keys are expected to be in PEM format. There are several methods to provide keys:

Upload

Upload a key directly to the KeyStore, which will then also persist the provided key in PostgreSQL:

```
curl --data-binary @key.pem --header 'Content-Type: application/x-pem-file' i3dhub-
↪keystore:8080/addKeys
```

Insert

> **Warning:** Only the HTTP method will check the validity of the provided key before persisting it to PostgreSQL. Other methods will insert any data without validation.

Insert the key directly into PostgreSQL via a stored procedure:

```
select keys.insert_key('key.pem contents here');
```

Insert the key directly into the correct table:

```
insert into keys.active_keys(key)
values ('key.pem contents here');
```

1.4.3 Signed Input

Note: Custom keys are required when using signed input. Without custom keys the system will not be able to validate the input and will reject all requests.

In order to validate input URIs the system needs to know which part of the URIs is the signature, and which the signed part. A set of regex rules must be provided to extract the relevant parts from inputs.

Each regex must provide at least two capture groups. The first capture group must capture the signed part, the second the signature of the signed part.

Configuration is provided to the system via `values.yaml`:

```
# Enable signed input validation. This list of rules (regexes) is run against resolved
# URLs and will attempt to validate signatures. Only if at least one of the given rules
# passes validation will the input be considered valid.
#
# Each rule must return the signed part as the first capture group and the signatures
# as the second capture group. For example, applying the disabled rule to
# http://example.com/secret.jt?sig=123456 results in the first capture group being
# example.com/secret.jt and the second being 123456. This causes the system to
# attempt to verify 123456 as the signature of example.com/secret.jt.
#
# Default: disabled, no input validation
signedInputRules:
 - https?://([^\\?]+).*sig=([^&]+)
```

1.5 User Authentication Passing

1.5.1 Introduction

The system does not understand the concept of users directly, it is only possible to pass existing credentials along to external systems. Credentials must exist in the form of HTTP headers on any requests the system receives. External systems are then required to perform authorization checks with these headers when 3D data is accessed or downloaded.

Warning: Configuration of authentication forwarding rules is required in order to pass any authentication tokens, no matter the method described here. No headers ever leave the system in the default configuration.

Forwarding Configuration

All configuration takes place in `values.yaml`.

Rules must be configured which define which headers are allowed to be sent to which backend addresses. Preferably this is done directly via URNs:

```
# Set the URN mapping rules per data gateway. See the integration documentation for
# motivation and concepts behind data gateways and URN mappings.
#
# For templates, $(n) is the n-th value in the URN separated after␣
```

(continues on next page)

(continued from previous page)

```
↪urn:namespace:specifier:xxx
# e.g.: urn:x-i3d:shape:sphere ($(1) == sphere)
dataGateways:
  exampleGateway:
    - namespace: customer
      specifier: document-uuid
      urlContentType: [ "openjt" ]
      urlTemplate: https://download.example.com/documents/$(1).jt
      authUrlTemplate: https://auth.example.com/documents/$(1).jt
      forwardCookies: [ "Cookie1" ]
      forwardHeaders: [ "Header1" ]
```

The fields `forwardCookies` and `forwardHeaders` can be set in order to specify which cookies (by name) or which headers (by name) will be forwarded (if present) when accessing any resource to which the URN points.

If all cookies should be forwarded, regardless of name, `Cookies` can be added to `forwardHeaders`.

Alternatively, it is also possible to define a regular expression which defines a set of backends to forward headers to:

```
# Configure options relating to required authorization headers and caching.
auth:
  # Set any authorization header names which must be passed to data backends.
  # These will be copied from incoming client requests and used while downloading
  # data or performing authorization requests against the backends.
  # No client headers or cookies will be forwarded to any backends by default as
  # this could leak client credentials to arbitrary URLs.
  # If dataGateways are defined, forwardHeaders or forwardCookies fields can also be
  # defined there on a per-rule basis. This should be preferred as it is more robust
  # than regex matching URLs.
  # In either case, rules from URN definitions are applied first, after which the
  # regex rules defined here are applied on the URL generated by resolving the URN.
  forwardHeaders:
    - match: "https://server1.*"
      headers: [ "Header1", "Header2" ]
      cookies: [ "Cookie1", "Cookie2" ]
    - match: "https://server2.*"
      headers: [ "Header1", "Header3" ]
      cookies: [ "Cookie1", "Cookie3" ]
```

Caching Configuration

External backends can be exposed to a high load if caching is not configured. As the system does not understand users directly, caching can only work effectively if it is possible to derive a user ID from an authentication token. OpenID-Connect provides this functionality and can be configured if available.

If OpenID-Connect is not available, it is possible to also cache directly by header value, but this may be unstable if values frequently change on a request by request basis.

```
# Configure options relating to required authorization headers and caching.
auth:
  caching:
    # Address of the used openid provider. For google this would be
    # https://accounts.google.com
```

(continues on next page)

(continued from previous page)

```
    # .well-known/openid-configuration is appended to this URL in order to
    # discover relevant endpoints.
    oidcProviderEndpoint:

    # Name of the header which contains the OIDC token. If a provider endpoint
    # is given, the service will attempt to validate this token as an
    # OpenID-Connect JWT token and use the contained user ID to cache
    # authorization responses. Without a provider endpoint, the token
    # value will be used to cache authorization responses.
    oidcHeader:

    # Duration for which to cache an authorization result. Only successful
    # authorizations are cached. Must be set to 0s if no header is given.
    # Otherwise responses will be cached for the given duration and used for
    # every user.
    duration: 0s
```

Custom Client Header Configuration

There are two use-cases based on how authentication tokens are aquired:

- Applications always have SSO authentication tokens set by cookies
- Applications must perform authentication flow and set headers themselves

For the second case, the backend must be configured to allow these to be set in CORS scenarios:

```
# Configure options relating to required authorization headers and caching.
auth:
  # Sets a list of headers to include in any Access-Control-Allow-Headers preflight
  # responses. This is required when setting headers manually via the webvis-API.
  # For example, if clients are required to attach credentials not in cookie form,
  # the specfic headers must be allowed here.
  clientHeaders:
    - "Custom-Authorization"
    - "X-Requested-With"
```

1.6 Autoscaling

1.6.1 Introduction

Note: Autoscaling is currently only supported for transcoder pods.

The system utilizes a message queue (RabbitMQ) internally to schedule transcoder (data conversion) tasks to pods. Autoscaling can use the length of one of these queues as a scaling metric.

We recommend using Keda to configure autoscaling based on these metrics. Keda can easily be installed via Helm or directly with kubectl as described in the official documentation.

1.6.2 Configuration

Currently a queue is created for each data format supported by instant3Dhub. We will need to create a Keda scaling configuration for each data format we want to allow to scale. The pod we scale will be the same for every format, as every transcoder pod is able to handle every format in the default configuration.

A list of available queues to scale can be found under the RabbitMQ management interface of the instant3Dhub installation under `/rabbitmq/#/queues` (default login is `guest` `guest`).

Note: Only queues starting with name `TranscoderService.` can be configured for autoscaling. Others are not yet supported.

Below is a configuration for scaling `i3dhub-transcodersvc` pods based on the JT queue. Pods will be scaled to a minimum of at least one and a maximum of 10.

Important: The correct namespace must be set in both the connection string of `i3dhub-keda-rabbitmq-secret` as well as the `TriggerAuthentication` and the `ScaledObject`.

```
apiVersion: v1
kind: Secret
metadata:
  name: i3dhub-keda-rabbitmq-secret
data:
  # base64 of amqp://guest:guest@i3dhub-rabbitmq.NAMESPACE.svc.cluster.local:5672
  host:␣
↪YW1xcDovL2d1ZXN0Omd1ZXN0QGkzZGh1Yi1yYWJiaXRtcS5OQU1FU1BBQ0Uuc3ZjLmNsdXN0ZXIubG9jYWw6NTY3Mg==
---
apiVersion: keda.sh/v1alpha1
kind: TriggerAuthentication
metadata:
  name: i3dhub-keda-trigger-auth-rabbitmq-conn
  namespace: NAMESPACE
spec:
  secretTargetRef:
    - parameter: host
      name: i3dhub-keda-rabbitmq-secret
      key: host
---
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: rabbitmq-scaledobject
  namespace: NAMESPACE
spec:
  scaleTargetRef:
    name: i3dhub-transcodersvc
  pollingInterval: 10  # Optional. Default: 30 seconds
  cooldownPeriod:  300 # Optional. Default: 300 seconds
  minReplicaCount: 1   # Optional. Default: 0
  maxReplicaCount: 10  # Optional. Default: 100
  triggers:
```

(continues on next page)

(continued from previous page)

```
  - type: rabbitmq
    metadata:
      protocol: auto
      queueName: TranscoderService.l3dGen.openjt.l3d
      mode: QueueLength
      value: "40"
    authenticationRef:
      name: i3dhub-keda-trigger-auth-rabbitmq-conn
```

1.7 License Server Usage Export

For license fee calculation the license server records usage statistics. The license server provides an interface to view and export this usage.

If the license server endpoint is opened via browser an easy to understand UI can be used to export and view the usage.

For functional operation the license server endpoint only needs to be exposed to the pods of instant3Dhub. However network access to the license server must be opened in order for administration to get the usage export.

Alternatively the license server endpoint can be accessed via curl:

```
curl 'http://your.license.server:8200/usage/export' > export.zip
```

To export the usage for a specific time interval, define `start` and `end` as unix timestamps.

For example, the usage between 01.07.2022 and 31.07.2022 can be exported with the following timestamps: `start=1656626400` and `end=1659218400`

```
curl "http://your.license.server:8200/usage/export?start=1656626400&end=1659218400
```