OCI Installation

Table of Content

1. Create Cluster

1. Create a new cluster in the **Kubernetes Clusters (OKE)**.

2. Wait until the cluster is in a **Active** state.

3. Install **kubectl***(command line utility)* on your machine and create namespace in you cluster for your client where **hie** is a name of namespace.

```
kubectl create ns hie
```

4. Install **helm***(command line utility)* on your machine and install **HA proxy** to you cluster. **HA proxy** provides load balancing and mapping of the contextpath to HIE-Engine services.

```
helm repo add haproxy-ingress https://haproxy-ingress.github.io/charts

helm install haproxy-ingress haproxy-ingress/haproxy-ingress \
-n haproxy-ingress \
--create-namespace \
--set rbac.create=true \
--set controller.ingressClass=haproxy \
--set controller.ingressClassResource.enabled=true \
--set controller.stats.enabled=true \
--set controller.kind=DaemonSet \
--set controller.daemonset.useHostPort=true

kubectl --namespace haproxy-ingress get services haproxy-ingress -o wide
```

2. Service Account

Create **Service Account** to access the OCI cluster.

1. Create **kubenode-sa** service account within the **hie** namespace.

```
kubectl -n hie create serviceaccount kubenode-sa
kubectl -n hie create rolebinding kubenode-sa-admin-rb --clusterrole=cluster-admin --serviceaccount=hie:kubenode-sa
```

2. Create **secret.yaml** file to access the OCI cluster.

```
tee secret.yaml > /dev/null <<EOT
apiVersion: v1
kind: Secret
metadata:
  name: kubenode-sa-token
  annotations:
    kubernetes.io/service-account.name: kubenode-sa
type: kubernetes.io/service-account-token
EOT
```

3. Apply **secret.yaml** file to access the OCI cluster.

```
kubectl -n hie apply -f secret.yaml
```

4. Display content of the **secret.yaml** file.

```
kubectl -n hie describe secrets kubenode-sa-token
```

   The following is the example of described **secret.yaml** file.

```
Name:         kubenode-sa-token
Namespace:    hie
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: kubenode-sa
              kubernetes.io/service-account.uid: 7ef46ac7...

Type:  kubernetes.io/service-account-token

Data
```

```
====
ca.crt:     1285 bytes
namespace:  3 bytes
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6ImFGQ1J4aF9xRnhFNnZ...
```

Where value of the **token** should be used in the **token** attribute of the **kubernetes** element within the **Kube Node** config.xml configuration file.

5. **Optional** step to set token to the **$TOKEN** local system variable.

```
TOKEN=`kubectl -n hie get secret kubenode-sa-token -o jsonpath='{.data.token}' | base64 --decode`
echo $TOKEN
```

6. **Optional** step to set access to the OCI cluster for **kubectl** using **token**.

```
kubectl config set-credentials kubenode-sa --token="<token>"
kubectl config set-cluster oci --server=144.24.177.92:6443 # --insecure-skip-tls-verify
kubectl config set-context oci --cluster=oci --user=kubenode-sa
kubectl config use-context oci
```

3. Create S3 Storage

1. Go to **Object Storage and Archive Storage/Buckets** in OCI cloud and create new one with name of your client.

2. Go to **profile/My profile** and click on **Customer secret keys > Generate Secret Key**, the string which is generated copy immediately - this is your **secretKey**. In line which is created is **Access key** and this is your **accessKey**.

3. Configure following part of config for **Kube Node** with correct **region** As a aesSecret password you can choose anything you want. It is used just for cypher/decipher entities in s3.

```
<s3storage url="https://frzcfiay9tsa.compat.objectstorage.eu-frankfurt-1.oraclecloud.com"
        accessKey="your_access_key"
        secretKey="your_secret_key"
        region="eu-frankfurt-1"
        aesSecret="some_possword"/>
```

4. In root element of the *config.xml* for **Kube Node** configure client name same as the name of the bucket.

```
<config clientId="exampleclient" ...
```

4. Kube Node Installation, configuration, and run

1. Download **Kube Node** Linux installation from the Configurator Studio from the following path.

   <Hub>/Releases/install/kubenode.tar.gz

2. Copy and unpack downloaded folder to target system.
3. Configure config.xml configuration file based on the following example.

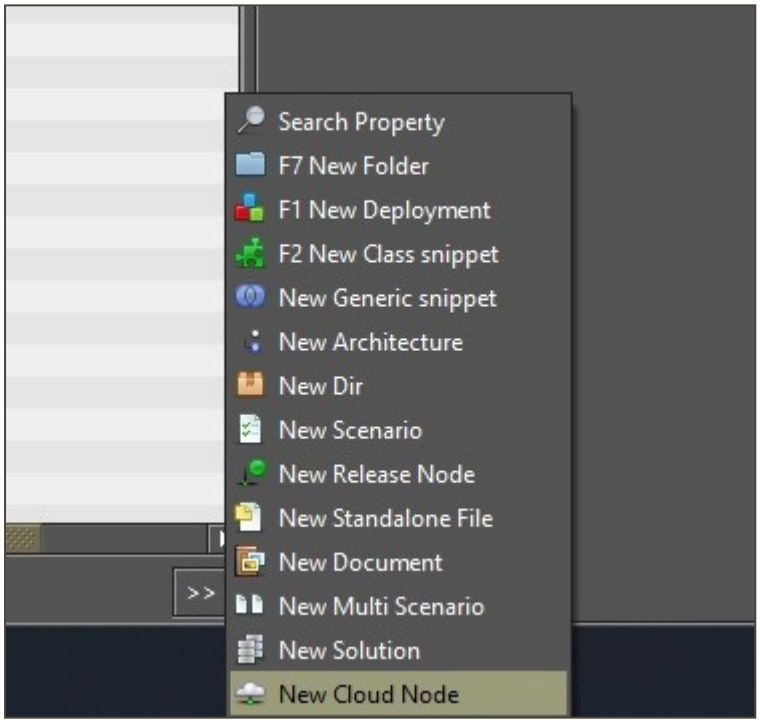
```
<config clientId="engineId">
   <upgrade
         nodeRoot="./node/temp/upgrade"/>
   <server
         port="8023"
         keystore="keystore.jks"
         ksPass="password"
         crPass="password"
         useTLS="false"
   />
   <access
         user="accessUser"
         password="accessPassword"
   />
   <storage
         root="data"
   />
   <kubernetes
         url="https://kubernetes.example.com:6443/"
         token="eyJhbGciOiJSUzI1NiIsImtpZCI6ImFGQ1J4aF9xRnhFNnZ..."
         namespace="hie"
         debug="false"
   />
   <dockerRegistry
         url=""
         user="dockerRegistryUser"
         password="dockerRegistryPassword"
   />
   <s3storage
         url="http://s3.example.com:8090/"
         accessKey=""
         secretKey=""
         region="US_EAST_1"
         aesSecret="aesSecret"
   />
   <console root="data" />
</config>
```

- Note, that **Kube Node** can be installed in the **Cloud**, but cannot be installed in **Kubernetes**.
- dockerregistry serves as storage for images of releases.
- s3storage serves as storage for deployments configurations and configuration dirs. sluzi na ulozenie konfiguracie deploymentov a dirs

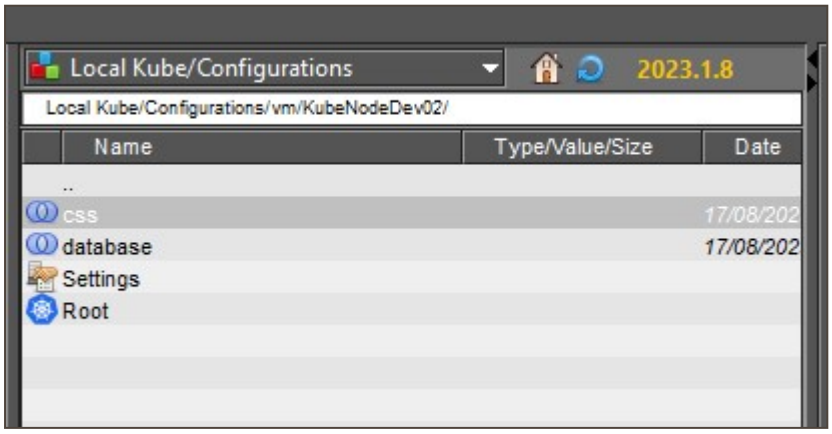
4. To start the **Kube Node** enter ./kubenode.sh start. To stop the **Kube Node** enter ./kubenode.sh stop.

5. Cloud Node

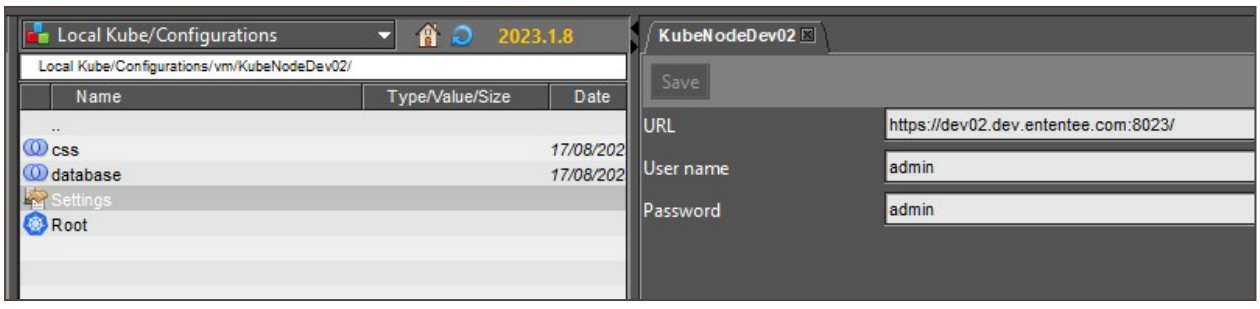
There is a new **Cloud Node** component of the Configurator Studio where you can create and configure **Cloud Node** with the similar purpose as the **Node** component.



Cloud Node consists of **Root** and **Settings** components and you can also create a **Generic Snippets** there.



Settings component allows you to configure connection to the system where the **Kube Node** is installed.

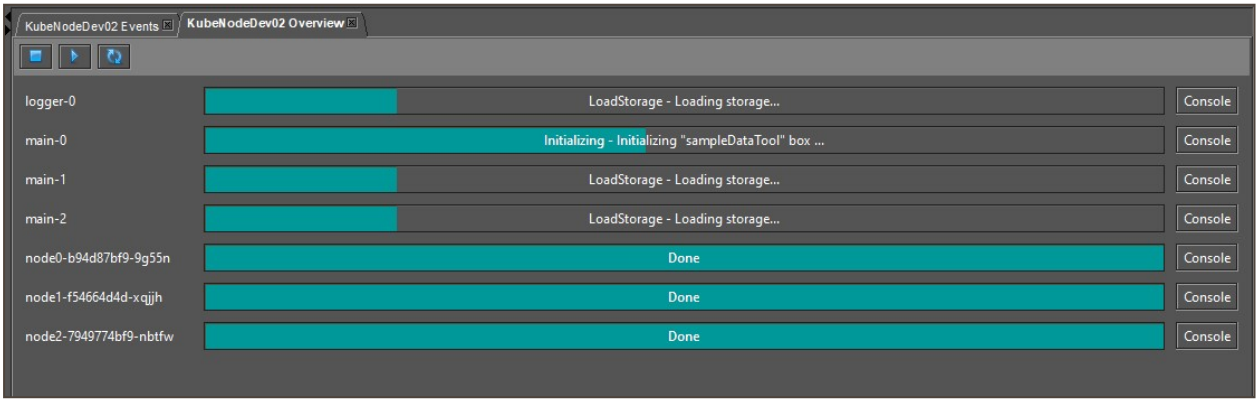


5.1 Root

A new **Root** element that contains following sub-components.

- **Overview** where you can check the status of running HIE-Engines, access the Console of an individual systems (pods) and Start, Stop, or Restart all systems (pods).

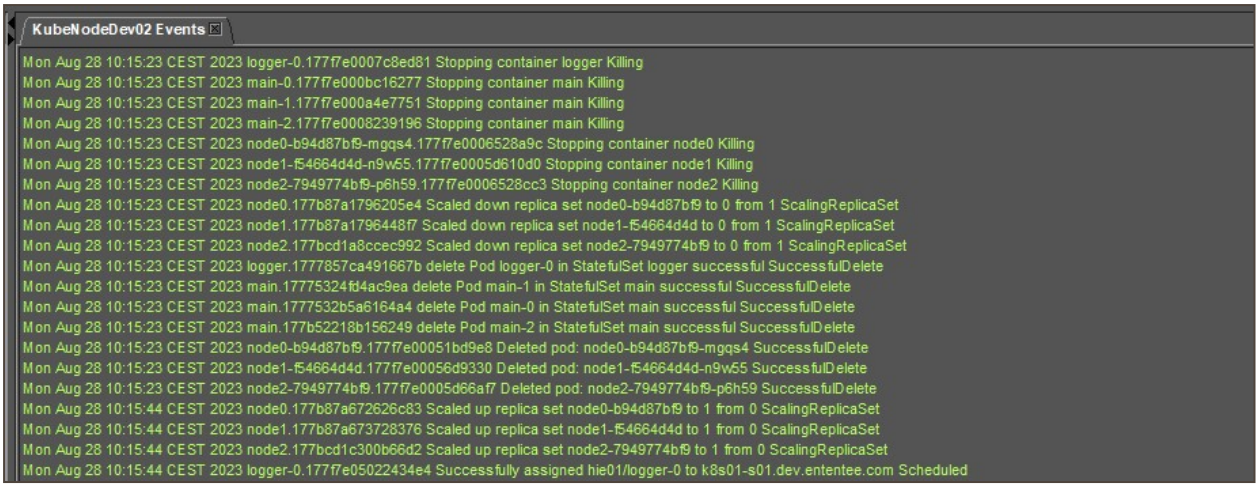
Following is the example when all systems were restarted.



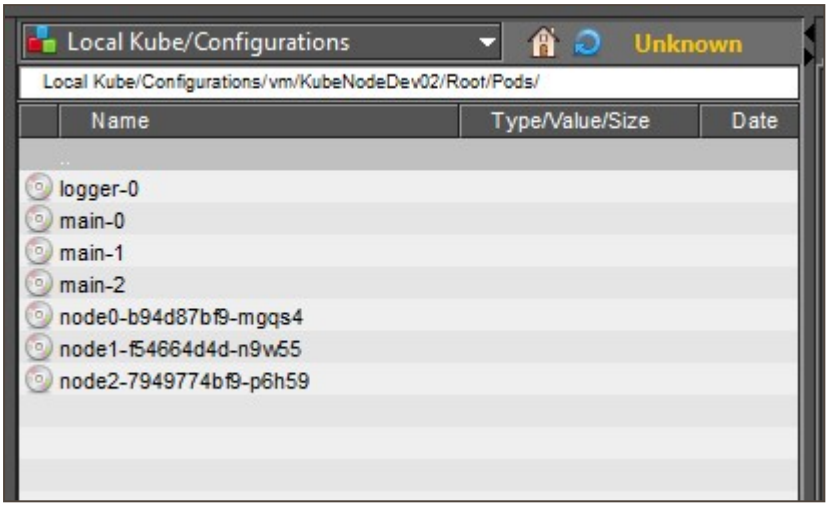
When all systems are stopped from the **Overview**, the value in the replica attribute is set to 0 on all systems. When all systems are started from the **Overview**, the value in the replica attribute is set back to original value on all systems. Both previously described actions are performed when all systems are restarted from the **Overview**. Currently, the restart is not the **rolling restart**.

- **Console** of the Kube Node.

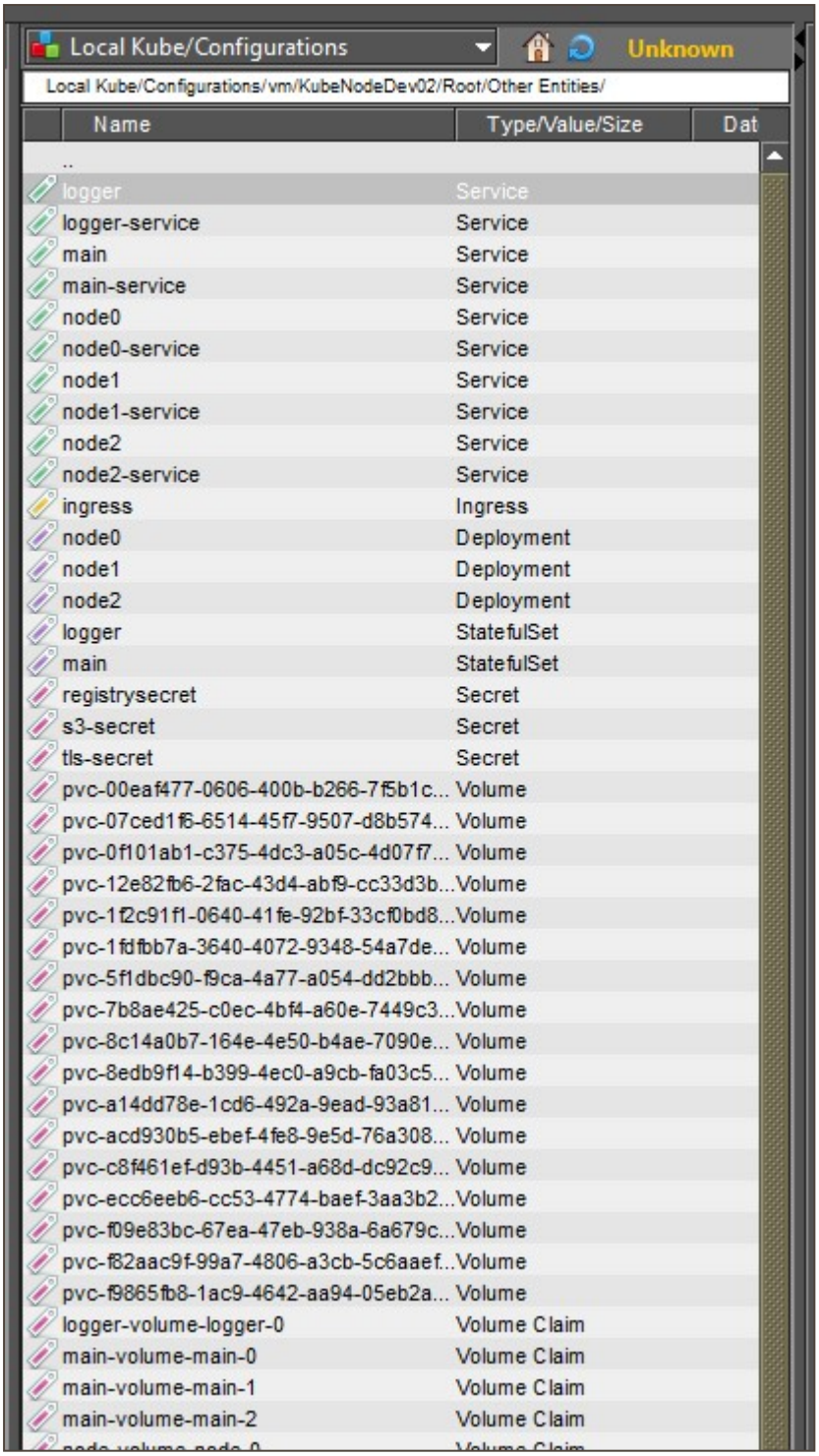
- **Events** from kubernetes displaying general activities on HIE-engine systems (pods).



- **Pods** where you can access individual pods (HIE-engine systems) and where you can access Console, File System, or Terminal of each individual pod.

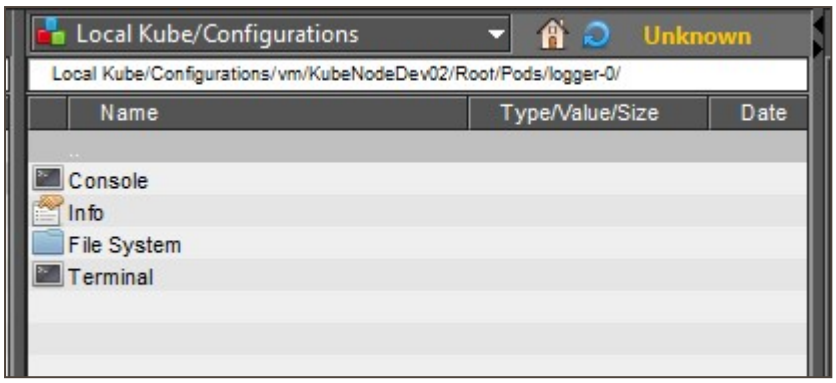


- **Other Entities**

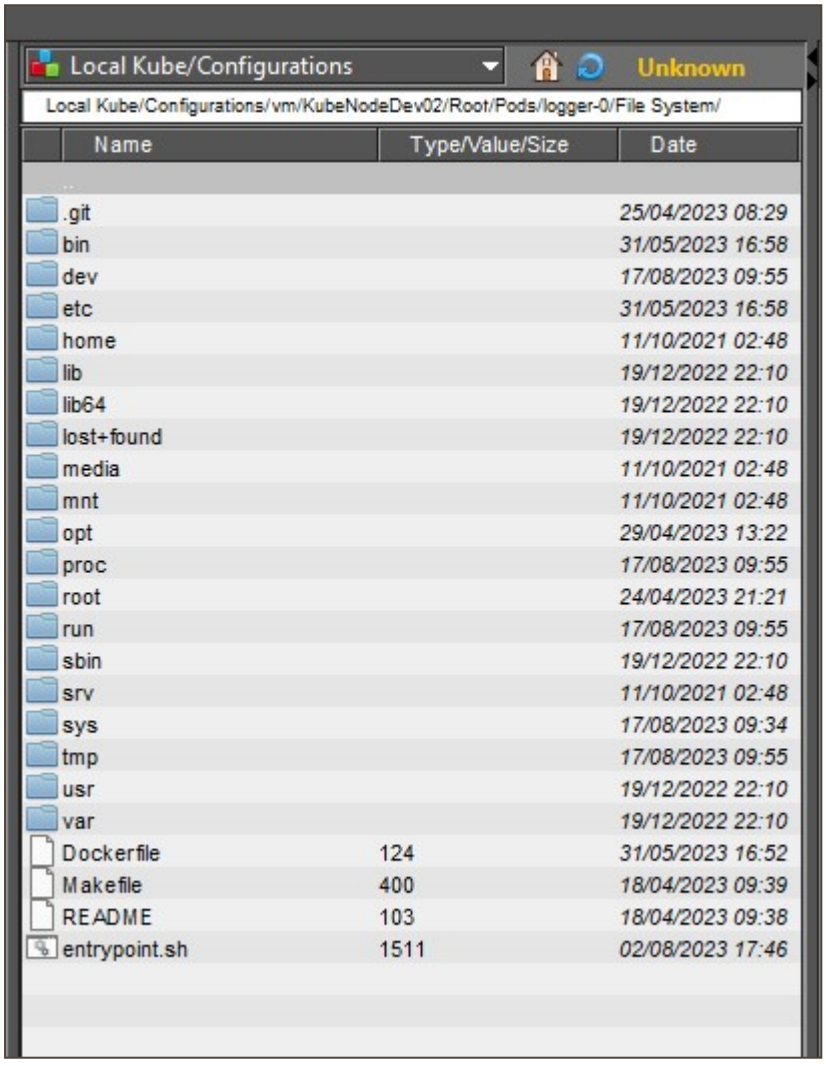


5.2 Pod

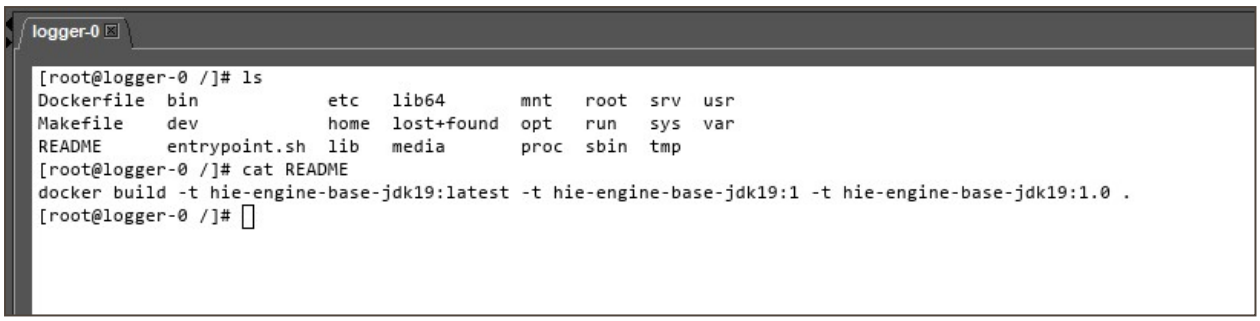
Each pod contains following components



Where **File System** allows you to access the file system of each pod (HIE-Engine).



And **Terminal** allows you to issue basic commands against the operating system. For example ls, cat <filename>.


logger-0
[root@logger-0 /]# ls
Dockerfile bin   etc   lib64   mnt   root srv usr
Makefile   dev   home lost+found opt   run   sys var
README   entrypoint.sh lib   media   proc sbin tmp
[root@logger-0 /]# cat README
docker build -t hie-engine-base-jdk19:latest -t hie-engine-base-jdk19:1 -t hie-engine-base-jdk19:1.0 .
[root@logger-0 /]#

6. Solution component

There is a new **Solution** component of the Configurator Studio where you can configure whole Cloud Solution.

It consists of the following components.

- **Cloud Installation** you can install the solution to Kube Node from. It contains automatically generated kubernetes.yaml configuration file.
- **solution.xml** configuration file where you can configure the **ingress** and **deployments** elements. Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster and provides load balancing.
  The domain element needs to be configured with the name instead of IP address.

  The id attribute of the deployment element contains the name of a real deployment where currently **logger/main/node/tunnel** deployments are supported.

  The replicas specifies number of systems that will be started with a given deployment scaling the service and providing High Availability of the service.

  Note, the meaning of the replicas element in the node element is number of nodes providing the service (population is distributed on) you need to configure nodeInHa="yes" to run two replicas for the same MPI node. Also note, there is no need to specify configuration related to **Other Masters** as Configurator Studio does it for you. Additionally, all systems are automatically connected to the **Central Logger** represented by **logger** deployment.

  You can also configure cpu and memory but it is not recommended to specify them ad in case you need to scale the system configure it using number of **replicas**.

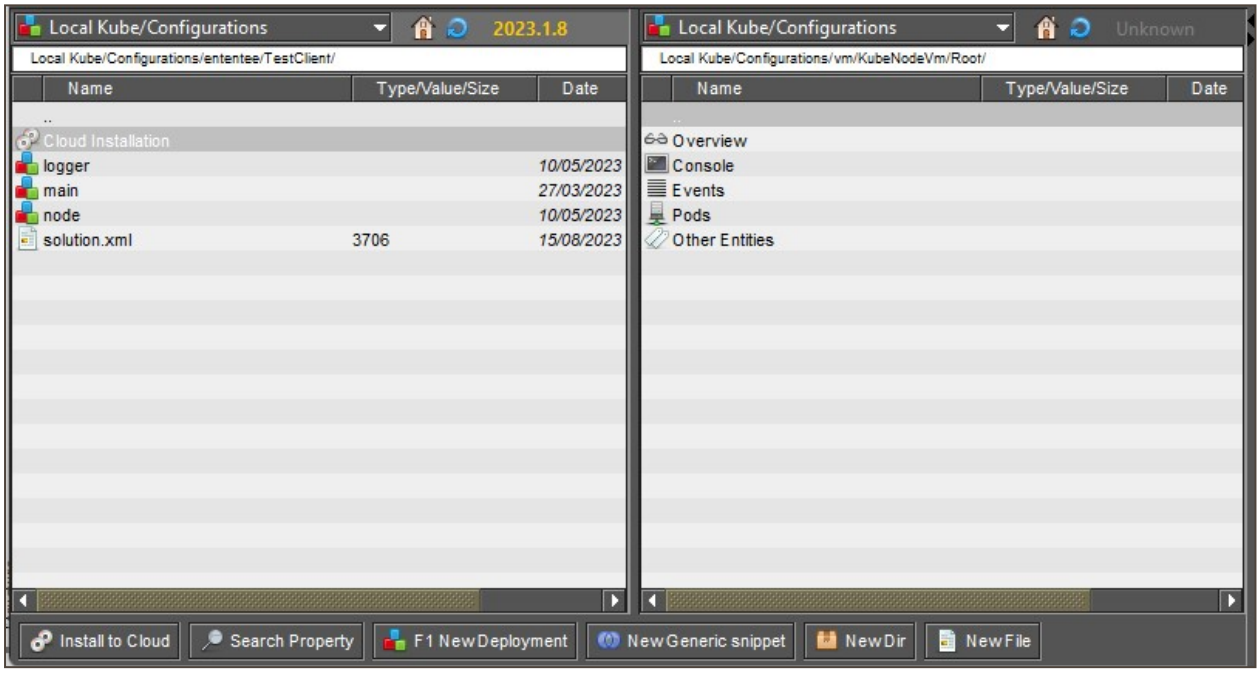
```
<root>
  <ingress>
    <!-- only for TLS on ingress -->
    <domain>echo.hie01.k8s01.dev.ententee.com</domain>
    <certificate>
      MIIDXzCCAk...
    </certificate>
    <key>
      MIIEuwIBADANB...
    </key>
    <paths>
      <path uri="/main" engineId="main" port="7211"/>
      <path uri="/central" engineId="logger" port="7211"/>
    </paths>
  </ingress>
  <deployments>
    <deployment id="logger"
          cpu="0.3"
          memory="2"/>
    <deployment id="node"
```

```
                cpu="0.3"
                replicas="2"
                memory="2"/>
        <deployment id="main"
                replicas="2"
                cpu="0.3"
                memory="2"/>
    </deployments>
</root>
```

- Deployments, generic snippets, and directories used in the solution.

7. Cloud Installation

To install the solution select **Cloud Installation** on first pane, enter the **Root** element on second pane, and click the **Install to Cloud** button.



Then two installation possibilities are available.

- **Update** where only systems that needs to be updated are updated. In this case the **NO DOWNTIME** strategy will be used and services will always be available as replicas are updated one by one. Load balancer redirects requests to other replicas and a given one is updated and restarted.
  Additionally, when one of the replicas ius stopped, a new one is automatically started.

  Note, the systems are restarted one by one, and it might happen, that for a limited short period a new updated MPI Master can communicate with an old not-updated MPI Node.

- **Reinstall** where all systems are stopped, updated and started.