

# OCI Installation

# Table of Content

1. Create Cluster.....	3
2. Service Account .....	4
3. Create S3 Storage.....	6
4. Kube Node Installation, configuration, and run .....	7
5. Cloud Node .....	9
5.1 Root.....	10
5.2 Pod .....	13
6. Solution component.....	16
7. Cloud Installation .....	18

# 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>
```

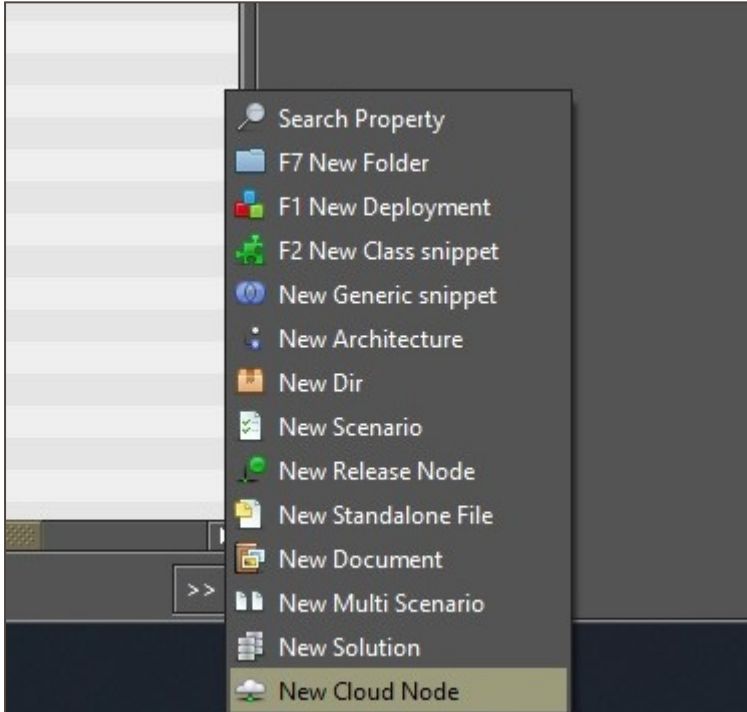
- Note, that **Kube Node** can be installed in the **Cloud**, but cannot be installed in **Kubernetes**.
- dockerregistry serves as storage for images of releases.
- s3storage serves as storage for deployments configurations and configuration dirs. sluzi na ulozenie konfiguracie deploymentov a dirs

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

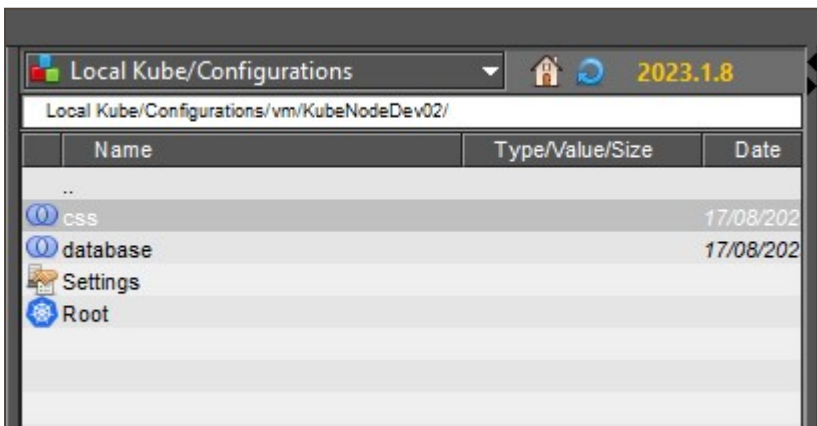


## 5. Cloud Node

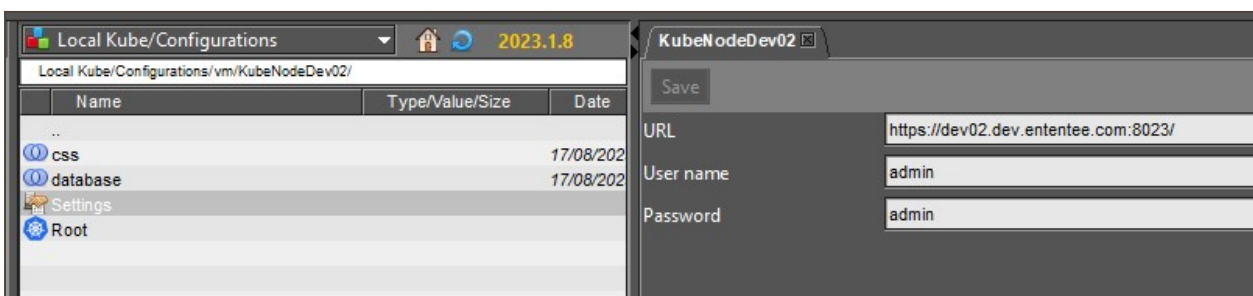
There is a new **Cloud Node** component of the Configurator Studio where you can create and configure **Cloud Node** with the similar purpose as the **Node** component.



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



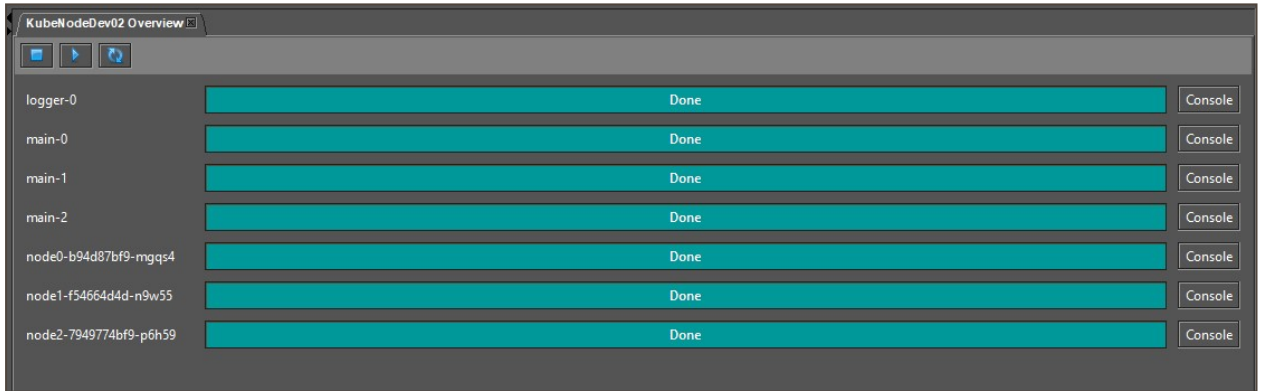
**Settings** component allows you to configure connection to the system where the **Kube Node** is installed.



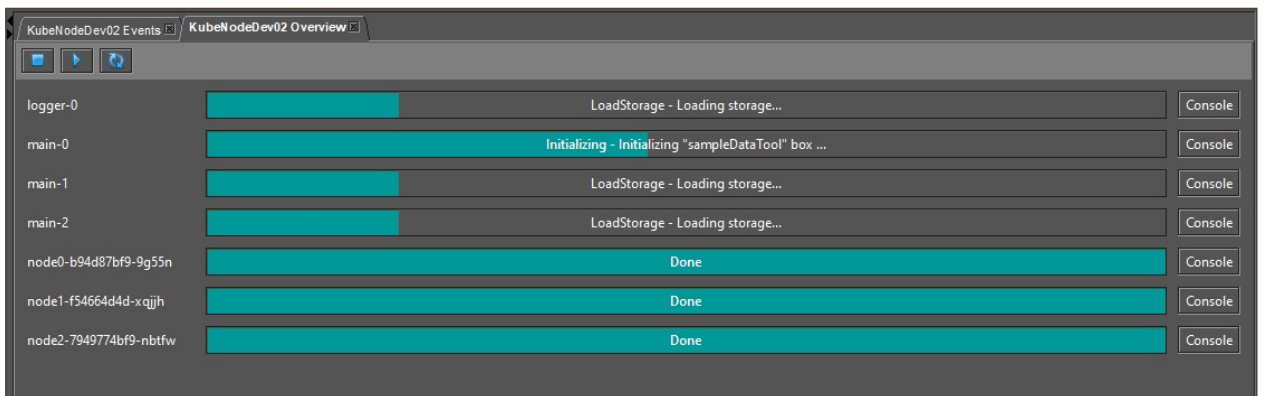
## 5.1 Root

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.

```
KubeNodeDev02 Console [x]
  terminationGracePeriodSeconds: 60
-----
applying patch
OK
Deploying yaml part
-----
# #####
# NODE2 service
# #####
apiVersion: v1
kind: Service
metadata:
  name: node2-service
spec:
  ports:
    - name: http
      port: 2004
      targetPort: 2004
    - name: enginetestsinterface
      port: 2221
      targetPort: 2221
    - name: webportaltunnel
      port: 7213
      targetPort: 7213
    - name: adminportaltunnel
      port: 7223
      targetPort: 7223
    - name: webportal
      port: 7211
      targetPort: 7211
    - name: adminwebportal
      port: 7221
      targetPort: 7221
    - name: node
      port: 8242
      targetPort: 8242
  selector:
    app: node2
-----
applying patch
OK
Deploying yaml part
-----
# #####
# NODE2 - control service
# #####
apiVersion: v1
kind: Service
metadata:
  name: node2
```

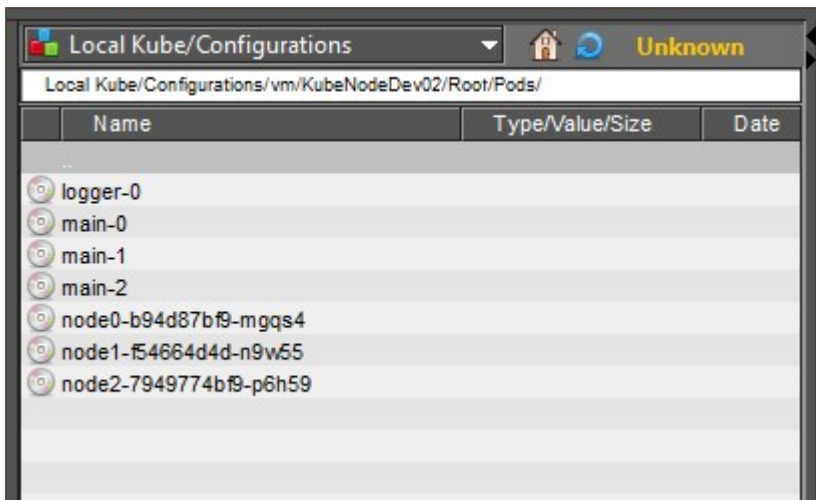
- **Events** from kubernetes displaying general activities on HIE-engine systems (pods).

```

KubeNodeDev02 Events
Mon Aug 28 10:15:23 CEST 2023 logger-0.1777e0007c8ed81 Stopping container logger Killing
Mon Aug 28 10:15:23 CEST 2023 main-0.1777e000bc16277 Stopping container main Killing
Mon Aug 28 10:15:23 CEST 2023 main-1.1777e000a4e7751 Stopping container main Killing
Mon Aug 28 10:15:23 CEST 2023 main-2.1777e0008239196 Stopping container main Killing
Mon Aug 28 10:15:23 CEST 2023 node0-b94d87b9-mgqs4.1777e0006528a9c Stopping container node0 Killing
Mon Aug 28 10:15:23 CEST 2023 node1-f54664d4d-n9w55.1777e0005d610d0 Stopping container node1 Killing
Mon Aug 28 10:15:23 CEST 2023 node2-7949774b9-p6h59.1777e0006528cc3 Stopping container node2 Killing
Mon Aug 28 10:15:23 CEST 2023 node0.1777b87a1796205e4 Scaled down replica set node0-b94d87b9 to 0 from 1 ScalingReplicaSet
Mon Aug 28 10:15:23 CEST 2023 node1.1777b87a1796448f7 Scaled down replica set node1-f54664d4d to 0 from 1 ScalingReplicaSet
Mon Aug 28 10:15:23 CEST 2023 node2.1777bcd1a8ccec992 Scaled down replica set node2-7949774b9 to 0 from 1 ScalingReplicaSet
Mon Aug 28 10:15:23 CEST 2023 logger.1777857ca491667b delete Pod logger-0 in StatefulSet logger successful SuccessfulDelete
Mon Aug 28 10:15:23 CEST 2023 main.1777532af4ac9ea delete Pod main-1 in StatefulSet main successful SuccessfulDelete
Mon Aug 28 10:15:23 CEST 2023 main.1777532b5a6164a4 delete Pod main-0 in StatefulSet main successful SuccessfulDelete
Mon Aug 28 10:15:23 CEST 2023 main.1777b52218b156249 delete Pod main-2 in StatefulSet main successful SuccessfulDelete
Mon Aug 28 10:15:23 CEST 2023 node0-b94d87b9.1777e00051bd9e8 Deleted pod: node0-b94d87b9-mgqs4 SuccessfulDelete
Mon Aug 28 10:15:23 CEST 2023 node1-f54664d4d.1777e00056d9330 Deleted pod: node1-f54664d4d-n9w55 SuccessfulDelete
Mon Aug 28 10:15:23 CEST 2023 node2-7949774b9.1777e0005d66af7 Deleted pod: node2-7949774b9-p6h59 SuccessfulDelete
Mon Aug 28 10:15:44 CEST 2023 node0.1777b87a672626c83 Scaled up replica set node0-b94d87b9 to 1 from 0 ScalingReplicaSet
Mon Aug 28 10:15:44 CEST 2023 node1.1777b87a673728376 Scaled up replica set node1-f54664d4d to 1 from 0 ScalingReplicaSet
Mon Aug 28 10:15:44 CEST 2023 node2.1777bcd1c300b66d2 Scaled up replica set node2-7949774b9 to 1 from 0 ScalingReplicaSet
Mon Aug 28 10:15:44 CEST 2023 logger-0.1777e05022434e4 Successfully assigned hie01/logger-0 to k8s01-s01.dev.ententee.com Scheduled

```

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

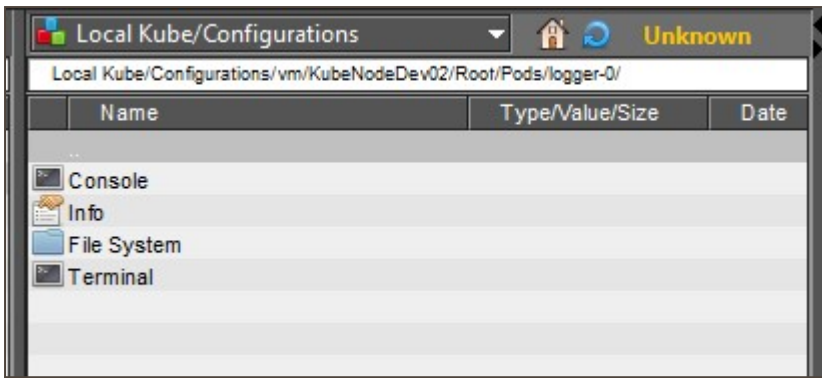


- **Other Entities**

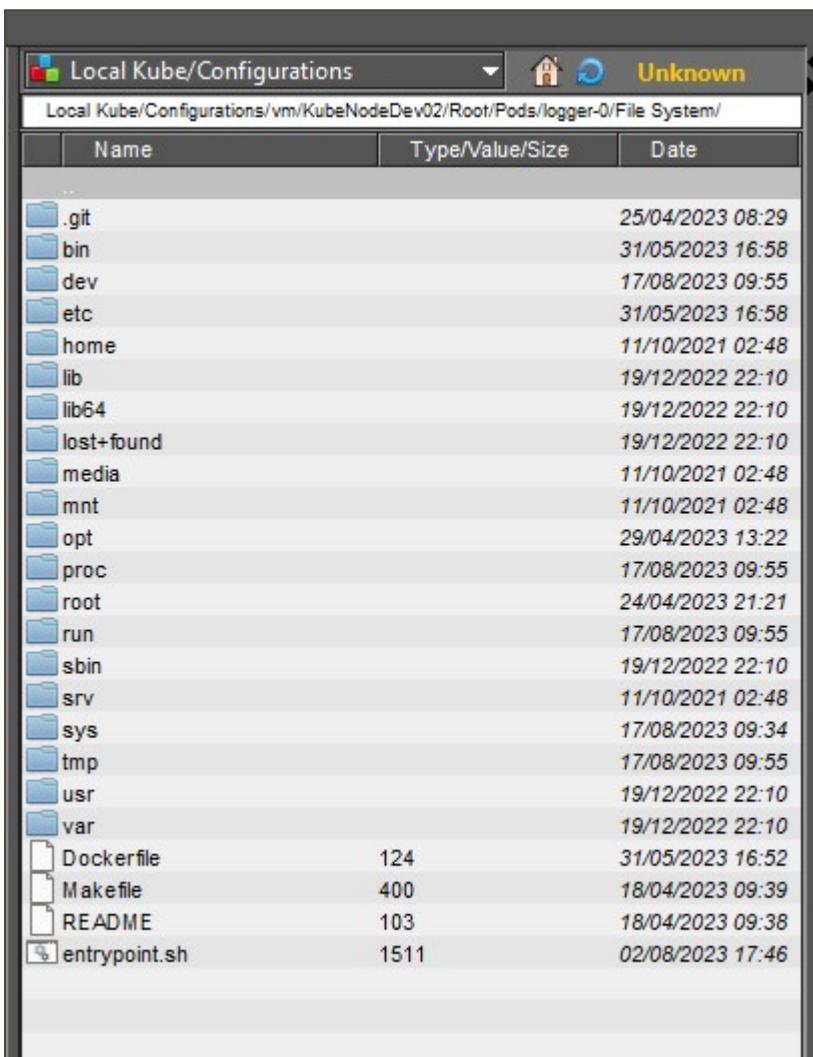
Name	Type/Value/Size	Dat
..		
logger	Service	
logger-service	Service	
main	Service	
main-service	Service	
node0	Service	
node0-service	Service	
node1	Service	
node1-service	Service	
node2	Service	
node2-service	Service	
ingress	Ingress	
node0	Deployment	
node1	Deployment	
node2	Deployment	
logger	StatefulSet	
main	StatefulSet	
registrysecret	Secret	
s3-secret	Secret	
tls-secret	Secret	
pvc-00eaf477-0606-400b-b266-7f5b1c...	Volume	
pvc-07ced1f6-6514-45f7-9507-d8b574...	Volume	
pvc-0f101ab1-c375-4dc3-a05c-4d07f7...	Volume	
pvc-12e82fb6-2fac-43d4-abf9-cc33d3b...	Volume	
pvc-12c91f1-0640-41fe-92bf-33cf0bd8...	Volume	
pvc-1fd1bb7a-3640-4072-9348-54a7de...	Volume	
pvc-5f1dbc90-f9ca-4a77-a054-dd2bbb...	Volume	
pvc-7b8ae425-c0ec-4bf4-a60e-7449c3...	Volume	
pvc-8c14a0b7-164e-4e50-b4ae-7090e...	Volume	
pvc-8edb9f14-b399-4ec0-a9cb-fa03c5...	Volume	
pvc-a14dd78e-1cd6-492a-9ead-93a81...	Volume	
pvc-acd930b5-ebef-4fe8-9e5d-76a308...	Volume	
pvc-c8f461ef-d93b-4451-a68d-dc92c9...	Volume	
pvc-ecc6eeb6-cc53-4774-baef-3aa3b2...	Volume	
pvc-f09e83bc-67ea-47eb-938a-6a679c...	Volume	
pvc-f82aac9f-99a7-4806-a3cb-5c6aaef...	Volume	
pvc-f9865fb8-1ac9-4642-aa94-05eb2a...	Volume	
logger-volume-logger-0	Volume Claim	
main-volume-main-0	Volume Claim	
main-volume-main-1	Volume Claim	
main-volume-main-2	Volume Claim	
node-volume-node-0	Volume Claim	

## 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 [x]
[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 as in case you need to scale the system configure it using number of **replicas**.

```
<root>
  <ingress>
    <!-- only for TLS on ingress -->
    <domain>echo.hic01.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"
```

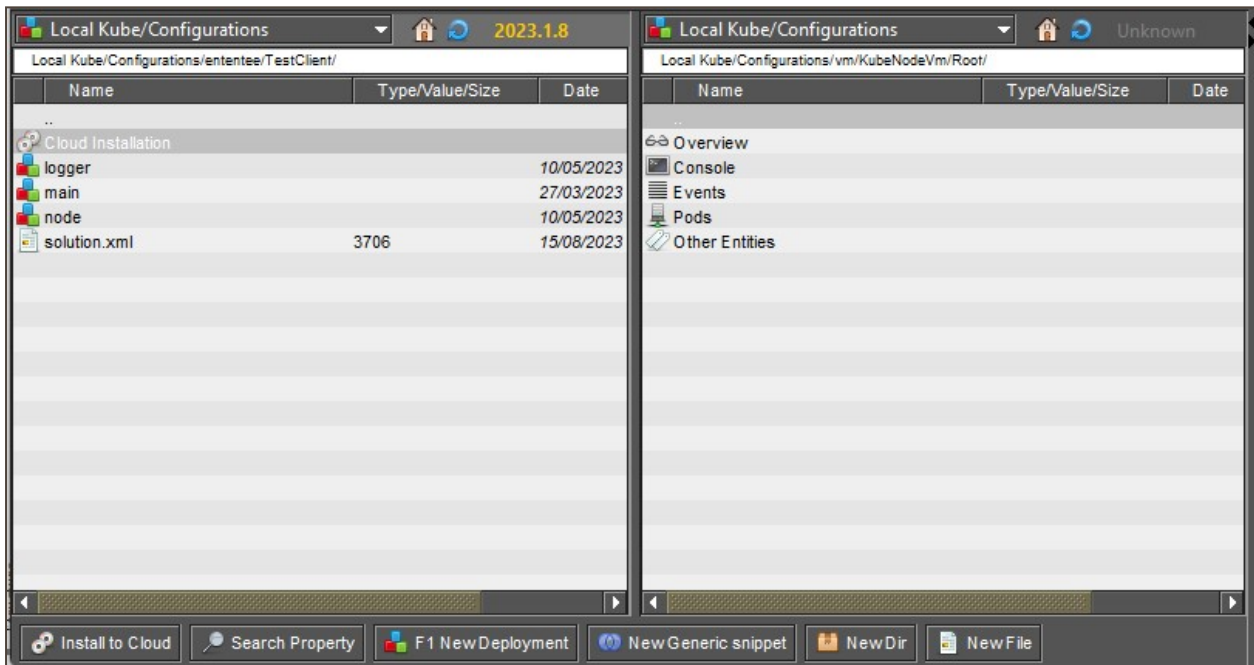


```
    cpu="0.3"
    replicas="2"
    memory="2"/>
  <deployment id="main"
    replicas="2"
    cpu="0.3"
    memory="2"/>
</deployments>
</root>
```

- 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.

