imc
part of Scheer

LMS Installation Guide (k8s)

imc Learning Suite

im-c.com

LMS Installation Guide (k8s)

Author: Malte Zieseniß

Document	Description
Version	1.1
Status (Draft / Review / Finalisation)	Final
Contact person(s)	Malte Zieseniß

History	Status	Who
27.09.2024	Draft	Malte Zieseniß
02.10.2024	Review	Roman Muth
02.10.2024	Final	Roman Muth
16.10.2024	Validation/Test	Raffael Willems

Content

imc
information multimedia communication AG
Headquarters Saarbrücken
Scheer Tower, Uni-Campus Nord
D-66123 Saarbrücken
T. +49 681 9476-0 | Fax -530
info@im-c.com
im-c.com

1 Introduction

This document outlines the steps required to set up the LMS software within an existing Kubernetes environment. The structure will be the following:

- **Initial Requirements**: We'll begin by outlining the prerequisites for installation, including what is provided by IMC.
- **Deployment Steps**: This section will guide you through configuring the necessary YAML files and other settings, such as SSL certificates and ingress rules.
- **Service Deployment**: You'll then learn how to deploy the LMS services within your cluster.
- **Testing and Validation**: After deployment, testing and validation steps are crucial to ensure the installation is successful.
- **Post-Installation**: Finally, we'll cover the actions required after the installation is complete.

The document is aimed at IT professionals who are commissioned to install the system.

2 Prerequisites

2.1 System Requirements Whitepaper

To begin the installation of the LMS software, several prerequisites must be met. These are detailed in the provided *System Requirements* whitepaper, which includes comprehensive information on hardware and Kubernetes cluster specifications, required software versions, database configuration, and more.

2.2 Additional Requirements

Before deploying the system, certain preliminary requirements must be addressed:

- **Database Setup:** The database is not included in the deployment and must be installed and configured separately. Supported databases include Oracle, Microsoft SQL Server, or PostgreSQL. Refer to the “Database Preparation” documentation for detailed setup instructions.
- **Storage Configuration:** Adequate storage must be configured for the system. Content storage should be on a remote shared storage system, such as NFS, EFS, or DFS.
- **Ingress Configuration:** An ingress is needed to expose HTTP and HTTPS routes from outside the cluster to the services within it. This requires deploying an ingress controller with SSL support, such as ingress-nginx. There are several Ingress controllers available for you to choose from.

This is an example of a minimal ingress resource:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx-example
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

2.3 Necessary Tools and Configurations

To install software on a Kubernetes cluster, a combination of tools and configurations is needed. Here is a breakdown:

Essential Tools:

- **Kubernetes CLI (kubectl)**: This is the primary tool for interacting with Kubernetes clusters. It allows you to create, deploy, and manage resources like pods, deployments, and services.
- **Container Runtime**: A container runtime (like Docker, containerd, or CRI-O) is necessary to execute containers within pods.
- **Package Manager**: A package manager (like Helm, Kustomize, or Ansible) can simplify the management of complex Kubernetes applications.

Configurations:

- **YAML Files**: Kubernetes objects are defined using YAML files. These files describe the desired state of the cluster, such as pods, deployments, services, and namespaces.
- **Namespace**: A namespace is a logical way to organize resources within a Kubernetes cluster. It helps isolate different applications.
- **Storage**: If your application requires persistent storage, you will need to configure storage classes and persistent volumes.
- **Networking:** Kubernetes provides a built-in network for communication between pods. You may need to configure additional network policies or services for specific use cases.

2.4 Helpful Tools

Additional Considerations:

- **Security**: Follow security best practices, such as implementing Role-Based Access Control (RBAC) to manage permissions and configuring network policies to control and restrict communication between resources.
- **Monitoring and Logging**: Use monitoring tools to track the health of your applications and set up logging to facilitate troubleshooting and diagnostics.
- **CI/CD Integration**: Integrate Kubernetes deployments into your CI/CD pipeline to enable automated testing, continuous integration, and seamless deployment.

2.5 Delivery

The LMS software is provided to you in a zipped download package with the following naming convention:

LS_%CUSTOMER%%REVISION%.zip

- **%CUSTOMER%** will be replaced by the customer's name.
- **%REVISION%** will represent the patch level of the delivery.

The package contains two zip files:

1. **kubernetes-%CUSTOMER%_%REVISION%.zip**: This file contains a config folder with default configuration files to set up your system. It may also include additional folders with the prefix "config" for different environments (e.g., ref, stage, prod).
2. **data.zip**: This file includes all the binary files needed by LMS, such as images and videos.

Additionally, the package includes a **deployment.yaml** file, which defines the desired state of a Kubernetes deployment. This YAML file is used to create, update, or delete deployments in a Kubernetes cluster. It specifies key details such as the number of replicas, pod specifications, labels, and other settings needed for the deployment.

This is an example of a basic YAML deployment file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
      - name: example-container
        image: example-image
        ports:
        - containerPort: 8080
```

3 Deployment Steps

3.1 Preparing the YAML Files

Within the provided config folder in the package, you create an additional directory to store persistent customer configurations (such as database passwords and other sensitive information). This new directory needs to be named "*config-override*". Any configurations placed in this folder will take precedence over those in the default config folder.
To configure your database connection, create a new YAML file named *application.yml* in the persistent config folder. You should enter your database connection details as illustrated below. The first example demonstrates a PostgreSQL configuration, while the second is for Microsoft SQL Server:

```
imc:
  database:
    username:
    password:
    usevault: false
    dbname: db
    host: 10.7.222.5
    url: "jdbc:postgresql://${imc.database.host}:5432/
         ${imc.database.dbname}?defaultRowFetchSize=500"

imc:
  database:
    host: srv.database.windows.net
    url: "jdbc:sqlserver://srv.database.windows.net;DatabaseName=db"
    username:
    password:
    usevault: false
```

Another YAML file that needs to be configured is the one that defines the connection to your storage. For example, you might name this file nfs-claims.yaml. In Kubernetes, storage is managed through API resources called **PersistentVolume (PV)** and **PersistentVolumeClaim (PVC)**.

A **PersistentVolume (PV)** represents a provisioned storage resource in the cluster, such as an NFS (Network File System). It defines the underlying storage implementation.

A **PersistentVolumeClaim (PVC)**, on the other hand, is a request for storage by a user, similar to how Pods request node resources. While Pods consume CPU and memory from nodes, PVCs consume storage from PVs.

The following is an example YAML file for setting up NFS-based storage:

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-data-%CUSTOMER%prod-nfs
spec:
  capacity:
    storage: 250Gi # Set to max available data of share
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy:
  mountOptions:
    - soft
    - nfsvers=4.1
  nfs:
    path: /%PATH_TO_SHARE%/data # path to share
    server: # IP/FQDN of NFS-Server
    readOnly: false

---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-data-%CUSTOMER%prod-nfs
spec:
  storageClassName: ""
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 250Gi
  volumeName: pv-data-%CUSTOMER%prod-nfs
  volumeMode: Filesystem
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-config-%CUSTOMER%prod-nfs
spec:
  capacity:
    storage: 10Gi # Set to max available data of share
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy:
  mountOptions:
    - soft
    - nfsvers=4.1
  nfs:
    path: /%PATH_TO_SHARE%/config # path to share
    server: # IP/FQDN of NFS-Server
    readOnly: false

---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-config-%CUSTOMER%prod-nfs
spec:
  storageClassName: ""
```

```
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  volumeName: pv-config-%CUSTOMER%prod-nfs
  volumeMode: Filesystem
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-solr-%CUSTOMER%prod-nfs
spec:
  capacity:
    storage: 250Gi # Set to max available data of share
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy:
  mountOptions:
    - soft
    - nfsvers=4.1
  nfs:
    path: /%PATH_TO_SHARE%/solr # path to share
    server: # IP/FQDN of NFS-Server
    readOnly: false

---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-solr-%CUSTOMER%prod-nfs
spec:
  storageClassName: ""
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  volumeName: pv-solr-%CUSTOMER%prod-nfs
  volumeMode: Filesystem
```

As outlined in the YAML file, three specific directories must be set up on the NFS: **config**, **data**, and **solr**. The **config** and **data** directories are already included in the delivered package, while the **solr** directory, used for the Apache Solr search engine, needs to be created manually. Additionally, within the **solr** directory, you must create a subdirectory called **solrhome**.

The paths to these directories are defined in the YAML file under the spec: section, specifically within the **`nfs:`** block. Here, you will also specify the IP address or Fully Qualified Domain Name (FQDN) of your NFS server.

The most important YAML file for the LMS installation is *deployment.yaml*, which is part of the provided package and comes preconfigured. However, some manual adjustments are necessary to ensure it works correctly in your environment. This file contains the specifications for all LMS services that will be deployed.

As an example, we look at the config service:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    kompose.cmd: kompose convert
    kompose.version: 1.19.0 (f63a961c)
  creationTimestamp: null
  labels:
    io.kompose.service: config
  name: config
spec:
  replicas: 1
  selector:
      matchLabels:
        io.kompose.service: config
  strategy:
    type: Recreate
  template:
    metadata:
      annotations:
        kompose.cmd: kompose convert
        kompose.version: 1.19.0 (f63a961c)
      creationTimestamp: null
      labels:
        io.kompose.service: config
    spec:
      volumes:
      - name: configfileshare
        persistentVolumeClaim:
          claimName: pvc-config-%CUSTOMER%prod-nfs
      containers:
      - env:
        - name: SPRING_PROFILES_ACTIVE
          value: native
        - name: SPRING_CLOUD_CONFIG_SERVER_NATIVE_SEARCHLOCATIONS
          value:  file:///config/,file:///config/config-override/
        - name: JAVA_OPTS
          value: -Dlog4j2.formatMsgNoLookups=true -
Xlog:gc=info,gc+init=info:file=/home/imc/learning-suite/instance/logs/gc-
%t.log:utc,u,pid,tags:filecount=10,filesize=100M
        image: 933991266655.dkr.ecr.eu-central-
1.amazonaws.com/customer/%CUSTOMER%/imc-ms-config-
server:%CUSTOMER%_%REVISION%
        imagePullPolicy: Always
        name: config
        ports:
        - containerPort: 8888
        resources:
          limits:
            memory: "512Mi"
          requests:
            memory: "512Mi"
        volumeMounts:
        - name: configfileshare
          mountPath: /config
      hostname: config
      imagePullSecrets:
```

```
      - name: %CUSTOMER%cred
      restartPolicy: Always
status: {}
---
apiVersion: v1
kind: Service
metadata:
  annotations:
    kompose.cmd: kompose convert
    kompose.version: 1.19.0 (f63a961c)
  creationTimestamp: null
  labels:
    io.kompose.service: config
  name: config
spec:
  ports:
  - name: "8888"
    port: 8888
    targetPort: 8888
  sessionAffinity: ClientIP
  sessionAffinityConfig:
    clientIP:
      timeoutSeconds: 3600
  selector:
    io.kompose.service: config
status:
  loadBalancer: {}
---
```

For this service, ensure that the claimName: matches the name defined under metadata: in the *nfs-claims.yaml* file. In this example, it should be **`pvc-config-%CUSTOMER%prod-nfs`**. Additionally, the persistent folder you created in your config directory is referenced under the **`SPRING_CLOUD_CONFIG_SERVER_NATIVE_SEARCHLOCATIONS`** environment variable. If you didn't use the default name config-override, be sure to update this variable with the correct folder name.

There are two other services where you need to set the correct claimName::

- For the **ils** service, the standard value is **`pvc-data-%CUSTOMER%prod-nfs`**.
- For the **solr** service, the standard value is **`pvc-solr-%CUSTOMER%prod-nfs`.**

Another YAML file you can configure at this point is *ingress.yml*. The ingress resource defines traffic routing rules and exposes HTTP and HTTPS routes from outside the cluster. It can be set up to provide externally accessible URLs for services, along with load balancing and SSL/TLS configuration.
The following YAML file shows how an ingress resource can look like:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: %CUSTOMER%
  namespace: %CUSTOMER%prod
  annotations:
    kubernetes.io/ingress.class: "ingress"
    cert-manager.io/cluster-issuer: "letsencrypt"
    nginx.ingress.kubernetes.io/affinity: cookie
    nginx.ingress.kubernetes.io/affinity-mode: persistent
    nginx.ingress.kubernetes.io/rewrite-target: /$1
    nginx.ingress.kubernetes.io/proxy-body-size: "3200m"
    nginx.ingress.kubernetes.io/proxy-connect-timeout: "3600"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "3600"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "3600"
    nginx.ingress.kubernetes.io/configuration-snippet: |
        if ($request_uri ~* \.(css|gif|jpe?g|png|woff|svg)) {
        expires 240m;
        add_header Cache-Control "public";
        }
    nginx.ingress.kubernetes.io/server-snippet: |
        gzip on;
        gzip_disable "msie6";
        gzip_vary on;
        gzip_proxied any;
        gzip_comp_level 6;
        gzip_buffers 16 8k;
        gzip_min_length 256;
        gzip_types
            application/atom+xml
            application/geo+json
            application/javascript
            application/x-javascript
            application/json
            application/ld+json
            application/manifest+json
            application/rdf+xml
            application/rss+xml
            application/xhtml+xml
            application/xml
            font/eot
            font/otf
            font/ttf
            image/svg+xml
            text/css
            text/javascript
            text/plain
            text/xml;
spec:
  tls:
```

```
  - hosts:
      - %CUSTOMERURL%
    secretName: %CUSTOMER%-tls
  rules:
  - host: %CUSTOMERURL%
    http:
      paths:
      - path: /(.*)
        pathType: Prefix
        backend:
          service:
            name: gateway
            port:
              number: 8080
```

For **hosts:** and **host:** you should set your system URL and **secretName:** will refer to your SSL certificate secrets. Don't forget to enter the chosen **namespace:** of your system.

3.2 Configuration

Before deploying the LMS services in your cluster, additional configuration is required. To allow your system to pull the images for the various LMS services from the IMC repository, you need to create a secret. We provide the following YAML file to help you create this secret in your cluster:

```
apiVersion: v1
kind: Secret
metadata:
  name: %CUSTOMER%cred
  namespace: %CUSTOMER%prod
data:
  .dockercfg: >-
    %SECRET%
type: kubernetes.io/dockercfg
```

In the file, the placeholder **%SECRET%** needs to be replaced with your current secret value. To make sure that the secret is renewed regularly, the following YAML file will also be provided to you. It is called **%CUSTOMER%-prod-token-recreation-CronJob.yaml:**

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: tokencreator
automountServiceAccountToken: true
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tokencreator-admin
subjects:
  - kind: ServiceAccount
    name: tokencreator
```

```
    namespace: %CUSTOMER%prod
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: admin
---
apiVersion: batch/v1
kind: CronJob
metadata:
  name: aws-registry-credential-cron
spec:
  schedule: "0 */4 * * *"
  successfulJobsHistoryLimit: 2
  failedJobsHistoryLimit: 2
  jobTemplate:
    spec:
      backoffLimit: 4
      template:
        spec:
          serviceAccountName: tokencreator
          terminationGracePeriodSeconds: 0
          restartPolicy: Never
          containers:
          - name: kubectl
            env:
            - name: AWS_ACCESS_KEY_ID
              value: AKIA5S5RXXFPZEYYP5ST
            - name: AWS_SECRET_ACCESS_KEY
              value: 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
            - name: AWS_DEFAULT_REGION
              value: eu-central-1
            imagePullPolicy: IfNotPresent
            image: 933991266655.dkr.ecr.eu-central-
1.amazonaws.com/ecmt/cronhelper:1.0.0
            command:
            - "/bin/sh"
            - "-c"
            - |
              AWS_ACCOUNT=933991266655
              AWS_REGION=eu-central-1
              SECRET_NAME=%CUSTOMER%cred

DOCKER_REGISTRY_SERVER=https://${AWS_ACCOUNT}.dkr.ecr.${AWS_REGION}.amazona
ws.com
              DOCKER_USER=AWS
              DOCKER_PASSWORD=`aws ecr get-login --region ${AWS_REGION} --
registry-ids ${AWS_ACCOUNT} | cut -d' ' -f6`
              kubectl delete secret $SECRET_NAME
              kubectl create secret docker-registry $SECRET_NAME --docker-
server=$DOCKER_REGISTRY_SERVER --docker-username=$DOCKER_USER --docker-
password=$DOCKER_PASSWORD --docker-email=no@email.local
              echo 'cronjob done'
          imagePullSecrets:
          - name: %CUSTOMER%cred
```

When you deploy this YAML file in your cluster, it will create a CronJob named aws-registry-credential-cron, which runs every four hours to automatically renew the secret.

Additionally, you can create SSL certificate secrets using the following command:

```
kubectl create secret tls %CUSTOMER%-tls --namespace %CUSTOMER%prod
--key "%PATH_TO_FILES% \private.key" --cert "%PATH_TO_FILES% \certificate.crt"
```

You need to use the secret name **`%CUSTOMER%-tls`** in your ingress YAML file to setup HTTPS access for your URL.

3.3 Deploying LMS Services and Components

Before deploying the LMS services in your Kubernetes cluster, ensure that your database and storage are properly set up, and that all YAML file configurations are complete. Once everything is prepared, you can deploy the files step by step in your cluster using the following command:

```
kubectl apply -f "%PATH_TO_FILES%\YAML" --namespace=%CUSTOMER%prod
```

Ideally, you should start with the *nfs-claims.yaml* so that your storage space is connected. Then you can deploy your *ingress.yml* and *deployment.yaml*.

4 Testing & Validation

Once the LMS system is successfully deployed and configured, you can access your URL and log in to perform testing and validation.

After logging in, click the magnifying glass icon in the top right corner to initiate a search. Type “Licences” and select it from the results. This will take you to a new screen where you’ll see several icons on the left side. Click the lowest icon, labeled "System info." A pop-up window will display the system information, including the version and revision number, allowing you to verify that the correct version was installed.

For another test, search for “Catalogues.” On the resulting screen, click the lowest icon on the left side again to select “Update search index.” This will trigger the indexing process and confirm that the Apache Solr search engine is functioning properly, as your catalogues will be indexed.

5 After the Installation

After the installation is complete, it's important to return any modified files from the delivery package to IMC. This ensures that all changes are included in the next delivery and won't be lost or require reconfiguration.
Typically, only the *deployment.yaml* file will have changes, and this file should be submitted in a support ticket to IMC.