Deployment example for Web Application Platform on Kubernetes

This deployment example is not intended for production use. Use it for testing purposes only.

This chapter provides information on how to deploy the product for testing purposes. Keep in mind that the guidelines provided below are not considered the best practices and do not take into account security settings and password encryption. This deployment has been tested with a specific Kubernetes version, CNI, container runtime, and OS. If you use other solutions, use them at your own risk.

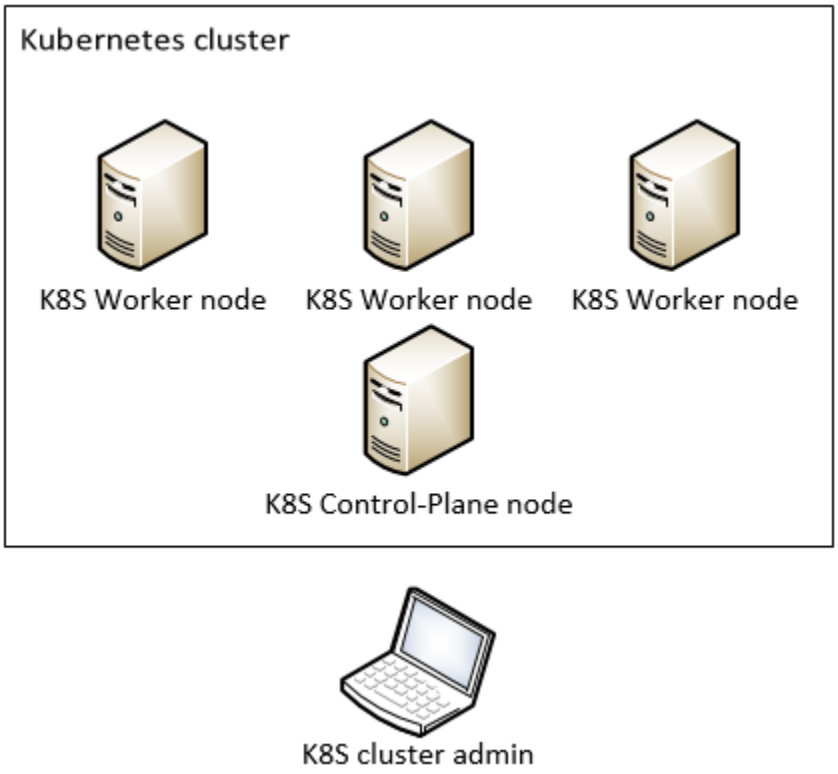
Prerequisites

- Kubernetes cluster (for more information, refer to https://kubernetes.io/):
  - 1 virtual machine used as the control plane node and 3 nodes used as worker nodes
  - Control plane node: 4 CPU cores, 8GB RAM
  - Worker node: 4 CPU cores, 32GB RAM
- Kubernetes version: v1.25.5
- Container runtime: Containerd (for more information, refer to https://containerd.io/)
- CNI: Calico
- OS: Red Hat Enterprise Linux release 8.7 (Ootpa)
- Load Balancer: MetalLB
- IP: 1 IP address with the DNS A record that points to this IP (IP from your network that is not used by your cluster nodes).
- Ingress: ingress-nginx (with sticky session support).

  Note that there are two nginx ingresses: ingress-nginx is maintained by the Kubernetes community, while nginx-ingress is maintained by F5 inc. The free F5 inc. version of nginx-ingress does not support “sticky sessions” which are required. Therefore, make sure you use ingress-nginx.

- A Host with Docker or Podman to build images and push them to the image registry.
- An image registry (repository) such as Harbor, Sonatype Nexus, Gitlab, or Docker registry server to push and pull the images you will build.
- Teamwork Cloud server
- The applications needed to run Web Application Platform services:
  - Zookeeper 3.8.0 (for more information, refer to https://zookeeper.apache.org/)
  - Apache Artemis MQ 2.28.0 (for more information, refer to https://activemq.apache.org/components/artemis/)
  - Keda CRD (deployed with the Simulation web application)

The following figure displays the environment used in this deployment example.



In this environment, the K8S cluster admin host (Linux virtual machine) is used to build images, run the image repository, and manage the Kubernetes cluster.

In the workflow below:

- *admin.example.com* refers to the K8S cluster admin host and should be changed to your K8S cluster admin FQDN.
- *webapp.example.com* refers to the domain name through which Web Application Platform will be accessible.
- All the steps are performed on the K8S cluster admin host.

To deploy the product for testing purposes

---

1. Download the *webapp-charts.zip* file.
2. Install kubectl as described at https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/.
3. Install the helm to the K8S cluster admin host as described at https://helm.sh/docs/intro/install.
4. Execute the following command to copy cluster config files to the K8S cluster admin host:

```
scp -r <user@control_plane_address>:.kube ~
Edit ~/.kube/config
```

Replace the localhost IP address with your control plane IP address or hostname, e.g., *https://127.0.0.1:6443*.

5. Build Docker images:

    a. Prepare the required files and directories.
    b. Download the installation package for version 2022x Refresh2. For more information, refer to Downloading installation files.
    c. Extract the packaged files, and inside your working folder, extract the *Web_App_Platform_{version}_linux_no_install.zip* file. After extracting the files, you should have a folder structure similar to *<your_working_folder>/CATIANoMagicServices/WebAppPlatform/*.

You will find all files mentioned before in *Web_App_Platform_{version}_linux_no_install.zip* or on the software download website (except for Teamwork Cloud keystore and SSL certificate):

- The *<your_working_folder>/CATIANoMagicServices/WebAppPlatform/webapps* directory contains application .war files.
- The *<your_working_folder>/CATIANoMagicServices/WebAppPlatform/shared/conf* directory contains other configuration files.
- The *<your_working_folder>/CATIANoMagicServices/WebAppPlatform/conf* directory contains *server.xml*.

6. Use the same SSL certificate and key that were used for the Teamwork Cloud server. You can copy the key and certificate files from the Teamwork Cloud server.
7. If you use only the .p12 file on the Teamwork Cloud server, execute the following command to extract the certificate and key from the .p12 file:

```
openssl pkcs12 -in INFILE.p12 -out twcloud.crt -nokeys
openssl pkcs12 -in INFILE.p12 -out twcloud.key -nodes –nocerts
```

8. Execute the following command to create the Kubernetes secret:

```
kubectl create secret tls webapp-tls-secret --key twcloud.key --cert twcloud.crt
```

9. Execute the following command to create a directory for each application you are going to use:

```
mkdir -p imagebuild/{admin-console,authentication,artemis,collaborator,document-exporter,oslc,reports,
resources,resource-usage-map,simulation,webapp}
```

10. Copy all files listed below to their own directories (except the *artemis* directory). You will find the Docker files for each service in *dockerfiles.zip*. After placing the Docker files in their appropriate directories, rename them to *Dockerfile* as shown below:

admin-console

- *admin.war*
- *Dockerfile*
- *keystore.p12* (from the Teamwork Cloud server)
- *logback.xml*
- *server.xml*
- *webappplatform.properties*

authentication

- *authentication.properties*
- *authentication.war*
- *Dockerfile*
- *keystore.p12* (from the Teamwork Cloud server)
- *logback.xml*
- *server.xml*
- *teamworkcloud.crt* (from *twcloud.crt* extracted earlier)
- *webappplatform.properties*

collaborator

- *collaborator.war*
- *Dockerfile*
- *keystore.p12* (from the Teamwork Cloud server)
- *logback.xml*
- *server.xml*
- *webappplatform.properties*

document-exporter

- *Dockerfile*
- *document-exporter.war*
- *collaborator.war*
- *keystore.p12* (from the Teamwork Cloud server)
- *logback.xml*
- *server.xml*
- *webappplatform.properties*

oslc

- *collaborator.war*
- *Dockerfile*
- *keystore.p12* (from the Teamwork Cloud server)
- *logback.xml*
- *server.xml*
- *webappplatform.properties*

reports

- *Dockerfile*
- *keystore.p12* (from the Teamwork Cloudd server)
- *logback.xml*
- *reports.war*
- *server.xml*
- *webappplatform.properties*

resource-usage-map

- *Dockerfile*
- *keystore.p12* (from the Teamwork Cloud server)
- *logback.xml*
- *resource-usage-map.war*
- *server.xml*
- *webappplatform.properties*

resources

- *Dockerfile*
- *keystore.p12* (from the Teamwork Cloud server)
- *logback.xml*
- *resources.war*
- *server.xml*
- *webappplatform.properties*

simulation

- *Dockerfile*
- *keystore.p12* (from the Teamwork Cloud server)
- *logback.xml*
- *server.xml*
- *simulation.war*
- *webappplatform.properties*

webapp

- *Dockerfile*
- *keystore.p12* (from the Teamwork Cloud server)
- *logback.xml*
- *server.xml*
- *webapp.war*
- *webappplatform.properties*

11. Execute the following command to start the local Docker registry:

```
docker run -d -p 5000:5000 --restart=always --name registry registry:2
```


In this case, a local Docker registry is run on the K8S cluster admin host. It uses port 5000, so make sure this port is open on your host firewall and accessible to your Kubernetes cluster.

This local Docker registry should be used for testing purposes only and is not considered production-ready. In the production environment, run the image registry with TLS and authentication.

12. Add your repo URL (FQDN of the K8S cluster admin host) to the Docker */etc/docker/daemon.json* file.
13. If the *daemon.json* file does not exist, create it. Assuming there are no other settings in the file, it should have the following content:

```
{
"insecure-registries": ["admin.example.com:5000"]
}
```

14. Execute the following command to restart the Docker service:

```
sudo systemctl restart docker
```

15. If the registry container has started, proceed with building the Artemis image (for more information, refer to https://github.com/apache/activemq-artemis/tree/main/artemis-docker):
    a. Execute the following command:

    ```
    cd imagebuild
    git clone https://github.com/apache/activemq-artemis.git
    cd activemq-artemis/artemis-docker
    ```

    b. Execute the following command to download Artemis binaries:

    ```
    wget https://archive.apache.org/dist/activemq/activemq-artemis/2.28.0/apache-artemis-2.28.0-bin.zip
    ```

    c. Extract the following file:

    ```
    unzip apache-artemis-2.28.0-bin.zip
    ```

    d. Execute the following commands to build the image:

    ```
    ./prepare-docker.sh --from-local-dist --local-dist-path ./apache-artemis-2.20.0
    cd apache-artemis-2.20.0
    docker build -f ./docker/Dockerfile-centos7-11 -t artemis-centos .
    ```

    e. Execute the following command to tag the built image:

    ```
    docker tag artemis-centos admin.example.com:5000/artemis:2.20
    ```

    f. Execute the following command to push the image to your repository:

    ```
    docker push admin.example.com:5000/artemis:2.20
    ```

16. Execute the commands below for each service that you want to build images for inside their own directories, as shown in the examples in steps 17a, 17b, and 17c.

```
docker build -f Dockerfile -t {APP_NAME} .
docker tag {APP_NAME}:{VERSION} {IMAGE_REPO_URL}:5000/{APP_NAME}:{VERSION}
docker push {IMAGE_REPO_URL}/{APP_NAME}:{VERSION}
```


- {IMAGE_REPO_URL} - your image repository URL. It can differ depending on the image registry provider.
- {APP_NAME} - application name.
- {VERSION} - version to be used for the tag.

    a. The following command example builds images for Admin Console:

    ```
    cd imagebuild/admin-console
    docker build -f Dockerfile -t admin .
    ```

    b. The following command example tags the built image:

    ```
    docker tag admin: latest admin.example.com:5000/admin:latest
    ```

    c. The following command example pushes the image to the registry:

```
docker push admin.example.com:5000/admin:latest
```

17. To be able to pull images to your Kubernetes cluster nodes, add your image repository to the Containerd configuration on **all cluster nodes**.
18. Edit the */etc/containerd/config.toml* file by adding the following lines to the "mirrors" part:

```
[plugins."io.containerd.grpc.v1.cri".registry.mirrors."admin.exmple.com:5000"]
              endpoint = ["http://admin.example.com:5000"]

      [plugins."io.containerd.grpc.v1.cri".registry.configs." admin.example.com"]
        [plugins."io.containerd.grpc.v1.cri".registry.configs." admin.example.com".tls]
         insecure_skip_verify = true
```


Keep in mind that alignment is very important, as shown in the example below.

```
    [plugins."io.containerd.grpc.v1.cri".registry]
      [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
        [plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]
          endpoint = ["https://registry-1.docker.io"]

        [plugins."io.containerd.grpc.v1.cri".registry.mirrors."admin.example.com:5000"]
          endpoint = ["http://admin.example.com:5000"]

      [plugins."io.containerd.grpc.v1.cri".registry.configs." admin.example.com"]
        [plugins."io.containerd.grpc.v1.cri".registry.configs." admin.example.com".tls]
         insecure_skip_verify = true
```

19. Execute the following command to restart Containerd services on all cluster nodes:

```
sudo systemctl restart containerd
```

20. Execute the following commands to add dependency repos:

```
helm repo add kedacore https://kedacore.github.io/charts
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update

cd ${webapp-charts_directory}
```

21. Execute the following command to update chart dependencies:

```
helm dependency update
```

22. Execute the command below to install Custom Resource Definitions (CRD). This will create all services and deployments.

```
helm install keda kedacore/keda --namespace keda --create-namespace --wait --version 2.6.0
```


Keda and MetalLB should be added as separate resources.

Since CRDs (being a globally shared resource) are fragile, you have to assume that it is shared across multiple namespaces and groups of users once a CRD is installed. For that reason, installing, modifying, and deleting CRDs is a process that has ramifications for all users and systems of that cluster.

You can check the status of the services and pods with this command:

```
kubectl get all
```

23. Execute the following command to deploy MetalLB:

```
helm install metallb bitnami/metallb --namespace metallb-system --create-namespace --wait
```

24. To check the MetalLB deployment status, execute the following command on the control plane node or the K8S cluster admin host:

```
kubectl get -n metallb-system all
```

You should see an output similar to this:

```
NAME                                  READY   STATUS    RESTARTS   AGE
pod/controller-84d6d4db45-b4sgr   1/1     Running   0          40m
pod/speaker-2wmvm                 1/1     Running   0          40m
pod/speaker-cf89f                 1/1     Running   0          40m
pod/speaker-hd6k7                 1/1     Running   0          40m
pod/speaker-x476h                 1/1     Running   0          40m

NAME                      TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/webhook-service   ClusterIP   [redacted]     <none>        443/TCP   40m

NAME                     DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
daemonset.apps/speaker   4         4         4       4            4           kubernetes.io/os=linux   40m

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/controller   1/1     1            1           40m

NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/controller-84d6d4db45   1         1         1       40m
```

> ⚠ If you see that the MetalLB status is 'Running,' but READY is 0/1, give it some time to start and repeat the command.

25. Create configuration files for MetalLB:

> ⚠ Make sure to complete all the substeps of this step to create and apply both configurations. Otherwise, the product will not work properly.

a. Edit the *metalLB/metallb_ipadresspool.yaml* file as shown below:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: first-pool
  namespace: metallb-system
spec:
  addresses:
  - 192.168.1.1/32
  #- 192.168.10.0/24
  #- 192.168.9.1-192.168.9.5
  #- fc00:f853:0ccd:e799::/124
```

> ⚠
> - Your IT department should give you a reserved IP address or a range of IP addresses (depending on your needs) with a DNS A record that points to this IP address.
> - Configuration files should be formatted in *.yaml*.

b. Edit the *metalLB/l2advertisement.yaml* file as shown below. You can change the metadata name as needed. Keep in mind that the ipAddressPool name should match the specifications.

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: example
  namespace: metallb-system
spec:
  ipAddressPools:
  - first-pool
```

c. Execute the following command to apply the MetalLB configuration:

```
kubectl apply -f metalLB/metallb_ipaddresspool.yaml
```

d. Execute the following command to check the applied configuration:

```
kubectl describe ipaddresspools.metallb.io first-pool -n metallb-system
```

e. Execute the following command to create the MetalLB advertisement:

```
kubectl apply -f metalLB/l2advertisement.yaml
```

f. Execute the following command to check the advertisement configuration:

```
kubectl describe l2advertisements.metallb.io example -n metallb-system
```

26. Find the *Values.yaml* file in the parent helm chart and provide the values of the parameters in the file to enable or disable specific applications or parts of the configuration.
27. Do one of the following:

- Execute this command in the helm parent chart directory to deploy services to the default namespace:

```
helm install webapp .
```

- Execute this command in the helm parent chart directory to create a namespace and deploy services in this namespace:

```
helm install webapp . --namespace=wap --create-namespace --wait
```


This helm chart includes Zookeeper and Ingress-nginx as dependencies. They will be deployed automatically.

28. After all web applications are deployed, execute the following command to check their status:

```
kubectl get all
```


- All pods should be READY:1/1, STATUS:Running.
- The ingress-nginx service should be Type:LoadBalancer.
- EXTERNAL-IP should match the address that you provided in the MetalLB configuration.

If pods do not run, check for problems by executing this command:

```
kubectl describe pod pod_name
```

29. Execute the following command to check Ingress rules:

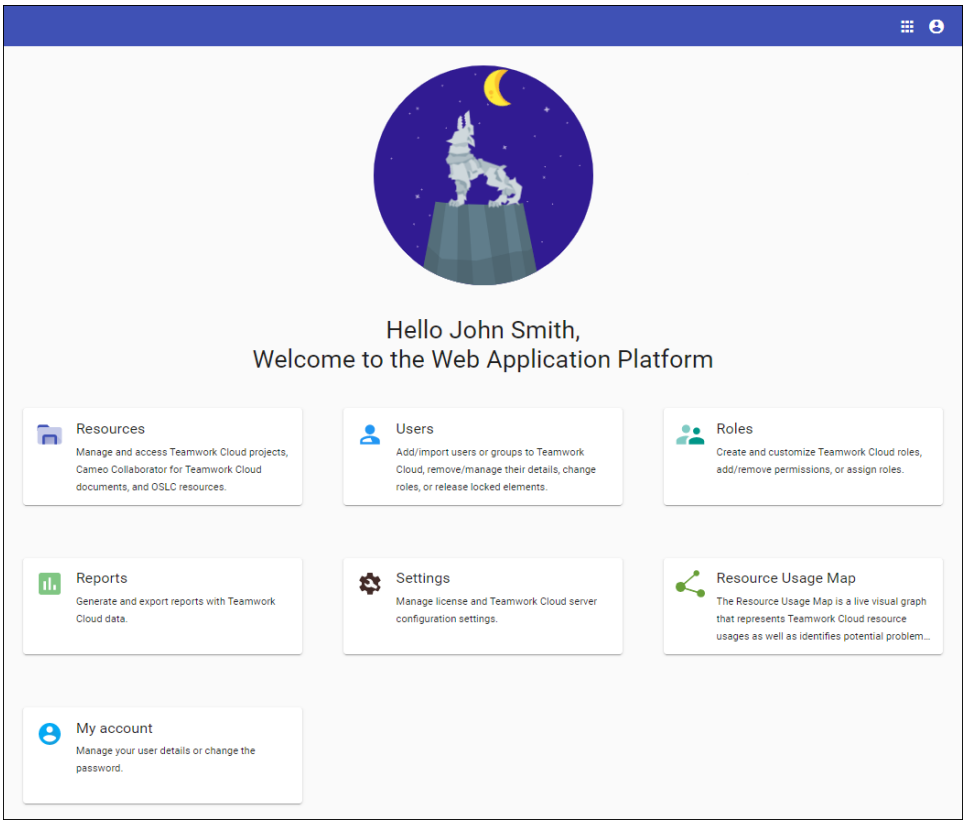
```
kubectl describe ingress
```

You should get an output similar to this:

```
Name:             ingress.resource
Namespace:        default
Address:          **.**.***.***
Default backend:  default-http-backend:80 (<error: endpoints "default-http-backend" not found>)
TLS:
  webapp-tls-secret terminates nm-wap10639.dsone.3ds.com
Rules:
  Host                      Path  Backends
  ----                      ----  --------
  nm-wap10639.dsone.3ds.com
                            /admin               webapp-adminconsole:8443 (10.233.105.39:8443)
                            /authentication      webapp-authentication:8443 (10.233.88.69:8443)
                            /collaborator        webapp-collaborator:8443 (10.233.88.75:8443)
                            /document-exporter   webapp-docexporter:8443 (10.233.105.22:8443)
                            /oslc                webapp-oslc:8443 (10.233.105.23:8443)
                            /reports             webapp-reports:8443 (10.233.73.246:8443)
                            /resources           webapp-resources:8443 (10.233.88.67:8443)
                            /resource-usage-map  webapp-rum:8443 (10.233.105.40:8443)
                            /simulation          webapp-simulation:8443 (10.233.88.123:8443)
                            /webapp              webapp-webapp:8443 (10.233.105.29:8443)
Annotations:                meta.helm.sh/release-name: webapp
                            meta.helm.sh/release-namespace: default
                            nginx.ingress.kubernetes.io/affinity: cookie
                            nginx.ingress.kubernetes.io/affinity-canary-behavior: sticky
                            nginx.ingress.kubernetes.io/affinity-mode: persistent
                            nginx.ingress.kubernetes.io/backend-protocol: https
                            nginx.ingress.kubernetes.io/proxy-body-size: 100m
                            nginx.ingress.kubernetes.io/proxy-connect-timeout: 600
                            nginx.ingress.kubernetes.io/proxy-read-timeout: 600
                            nginx.ingress.kubernetes.io/proxy-send-timeout: 600
                            nginx.ingress.kubernetes.io/send-timeout: 600
                            nginx.ingress.kubernetes.io/session-cookie-name: COOKIE
Events:                     <none>
```

30. Test the product deployment:
    a. In an internet browser, go to *domain_name/webapp* (the DNS A record that points to the Ingress external IP).

> ⚠ If you added a DNS record to bind the domain name to the IP address, you can use the domain name instead of the IP. The browser should show a warning because of the self-signed certificate. Accept it to proceed, and you will be redirected to the Teamwork Cloud Authentication web page.



b. Log in with your credentials, and you will be redirected back to Web Application Platform.