# **OpenShift Deployment Guide: 3scale API Management with Red Hat SSO**

This document provides a step-by-step guide for deploying and configuring 3scale API Management and integrating it with Red Hat Single Sign-On (RH-SSO) on an OpenShift cluster. This setup is typical for a Microservices as a Service (MaaS) environment where API security and management are critical.

## **Prerequisites**

Before proceeding, ensure you have the following:

* **OpenShift Cluster:** An active OpenShift cluster where you have cluster-admin or sufficient permissions to create projects, apply operators, and manage resources.
* oc **CLI:** The OpenShift Command Line Interface (CLI) installed and configured to connect to your cluster.
* git**:** Git installed for cloning repositories.
* podman**:** Podman installed for managing container images (used for the 3scale CMS).
* **Repository Access:** Access to the Git repository containing the maas project (e.g., scale, 3scale, redhat-sso components).

## **1. 3scale Operator Installation and Configuration**

This section covers the deployment of the 3scale API Management operator and initial configuration steps.

1. Clone the maas repository:  
   Navigate to your desired directory and clone the repository containing the deployment artifacts.  
   git clone <repository-url>/maas  
   cd ./maas
2. Install the 3scale Operator:  
   Apply the Kustomize overlay for the 3scale operator in "fast" mode. This will install the operator and its Custom Resource Definitions (CRDs).  
   oc apply -k ./components/3scale/operator/overlays/fast
3. Switch to the 3scale project:  
   Create and switch to the OpenShift project (namespace) where 3scale will be deployed. If the project already exists, this command will simply switch to it.  
   oc project 3scale
4. Verify 3scale Operator Pods:  
   Ensure that the 3scale operator pods are running successfully. It might take a few moments for them to become Running.  
   oc get pods -n 3scale
5. Create a Secret for 1LM Metrics (Apicast Policy):  
   This secret stores custom Apicast policies and Lua scripts for metrics collection (e.g., for One Last Mile - 1LM). Make sure you are in the correct directory (./3scale/config) before running this.  
   cd ./3scale/config  
   oc create secret generic 1lm-metrics \  
    -n 3scale \  
    --from-file=./apicast-policy.json \  
    --from-file=./custom\_metrics.lua \  
    --from-file=./init.lua \  
    --from-file=./1lm.lua \  
    --from-file=./portal\_client.lua \  
    --from-file=./response.lua \  
    && oc label secret 1lm-metrics apimanager.apps.3scale.net/watched-by-apimanager -n 3scale  
   * **Explanation:**
     + oc create secret generic 1lm-metrics: Creates a generic secret named 1lm-metrics.
     + -n 3scale: Specifies the 3scale namespace.
     + --from-file=...: Adds each specified file as a key-value pair in the secret, where the filename is the key and the file content is the value.
     + oc label secret ...: Labels the secret so that the 3scale API Manager recognizes it and can use its contents.
6. **Return to the root directory:**  
   cd ../../
7. Apply 3scale Cluster Instance and 1LM Metrics Policy:  
   These commands apply the main 3scale instance configuration and the 1LM metrics policy, configuring 3scale services and their integration.  
   # IMPORTANT: Before applying, modify ./components/3scale/cluster/instance.yaml  
   # Change the 'wildcardDomain' to your OpenShift cluster's wildcard domain.  
   # Example: wildcardDomain: apps.cluster-cbq4n.cbq4n.sandbox906.opentlc.com  
     
   oc apply -f ./components/3scale/cluster/instance.yaml  
   oc apply -f ./components/3scale/cluster/11m-metrics-policy.yaml  
   * **Note on** wildcardDomain**:** The instance.yaml file typically contains a placeholder for wildcardDomain. This must be updated to match the actual wildcard domain of your OpenShift cluster (e.g., apps.your-cluster-name.your-domain.com). This domain is crucial for 3scale's routes and service access.
8. Update the Storage Class of PVCs (if necessary):  
   This command is a placeholder to remind you to review and potentially update the storage class used by the Persistent Volume Claims (PVCs) in the 3scale namespace. If your OpenShift cluster uses a specific default storage class, this step might not be strictly necessary, but it's good practice to verify.  
   # This command lists resources, but does not modify storage classes.  
   # To actually update storage classes, you would need to edit each PVC.  
   for resource in $(oc api-resources --verbs=list --namespaced -o name --sort-by name | grep -v "events"); do \  
    oc get $resource -n 3scale --ignore-not-found -o custom-columns=KIND:.kind,NAME:.metadata.name; \  
   done > output2.txt  
   # Example of how you would patch a PVC to change its storage class (DO NOT RUN WITHOUT CAUTION):  
   # oc patch pvc <pvc-name> -p '{"spec":{"storageClassName":"<new-storage-class>"}}' -n 3scale
9. Retrieve 3scale Admin Password:  
   The 3scale admin password is stored in a secret. Use this command to decode and display it.  
   oc get secret system-seed -n 3scale --template='{{.data.ADMIN\_PASSWORD}}' | base64 -d; echo

## **2. Red Hat Single Sign-On (RH-SSO) Operator Installation and Configuration**

This section details the deployment of the RH-SSO operator (which also installs the OpenShift Elastic Search operator as a dependency) and the configuration of Keycloak for user authentication.

1. Install the RH-SSO Operator:  
   Apply the Kustomize overlay for the RH-SSO operator, which also brings in the OpenShift Elastic Search operator.  
   oc apply -k ./components/redhat-sso/operator/overlays/stable
2. Verify RH-SSO Operator Pods:  
   Ensure the RH-SSO operator pods are running successfully.  
   oc get pods -n rh-sso
3. Switch to the rh-sso project:  
   Create and switch to the OpenShift project (namespace) for RH-SSO.  
   oc project rh-sso
4. Update the Storage Class of the keycloak-postgresql-claim PVC (if necessary):  
   Similar to 3scale, review and potentially update the storage class for the Keycloak PostgreSQL PVC. This often requires manual editing or patching of the PVC.  
   # This is a reminder. To actually update, you would do:  
   # oc patch pvc keycloak-postgresql-claim -p '{"spec":{"storageClassName":"<new-storage-class>"}}' -n rh-sso
5. Create Keycloak PostgreSQL Database Secret:  
   This secret stores the credentials for the Keycloak PostgreSQL database. A random UUID is used for the password, though the example shows 0. It is strongly recommended to use a strong, randomly generated password instead of 0.  
   uuidgen | xargs -I{} oc create secret generic keycloak-db-secret \  
    --from-literal=POSTGRES\_USERNAME=keycloak \  
    --from-literal=POSTGRES\_PASSWORD={} \  
    --from-literal=POSTGRES\_DATABASE=keycloak \  
    --from-literal=POSTGRES\_EXTERNAL\_ADDRESS=rh-sso-postgresql.rh-sso.svc.cluster.local \  
    --from-literal=POSTGRES\_EXTERNAL\_PORT=5432 \  
    --from-literal=POSTGRES\_HOST=rh-sso-postgresql.rh-sso.svc.cluster.local \  
    --from-literal=POSTGRES\_SUPERUSER=true \  
    -n rh-sso  
   * **Note:** The uuidgen command generates a unique ID, making the password more secure than a hardcoded 0.
6. Apply PostgreSQL Database and Service for Keycloak:  
   These commands deploy the PostgreSQL database instance and expose it as a service for Keycloak.  
   oc apply -f ./components/redhat-sso/cluster/postgres-db.yaml -n rh-sso  
   oc apply -f ./components/redhat-sso/cluster/postgres-db-service.yaml -n rh-sso
7. Apply Keycloak Instance:  
   This command deploys the Keycloak instance itself.  
   oc apply -f ./components/redhat-sso/cluster/keycloak-v2.yaml -n rh-sso
8. Apply RH-SSO Realm Configuration:  
   This command applies a Keycloak realm configuration, often defining users, clients, and identity providers for your RH-SSO instance.  
   oc apply -f ./components/redhat-sso/cluster/03\_realm.yaml -n rh-sso
9. Retrieve RH-SSO Admin Username and Password:  
   The admin credentials for the Keycloak realm are stored in a secret. Use these commands to decode and display them.  
   oc get secret credential-rh-sso -n rh-sso --template='{{.data.ADMIN\_USERNAME}}' | base64 -d; echo  
   oc get secret credential-rh-sso -n rh-sso --template='{{.data.ADMIN\_PASSWORD}}' | base64 -d; echo

## **3. RH-SSO Integration with 3scale (SSL/Truststore)**

This section details how to configure Keycloak to trust external certificates (e.g., for an LDAP server) and how to handle SSL verification issues when 3scale attempts to connect to RH-SSO.

* **Reference:** The commands refer to the Red Hat documentation at https://access.redhat.com/solutions/6623671.

1. Patch Keycloak for Trusted Certificates:  
   This command patches the Keycloak custom resource to include a trustedCerts configuration, allowing it to load certificates from a specified ConfigMap.  
   oc patch keycloak keycloak --type-merge -p '{"spec": {"trustedCerts" : [{"configMapName": "ldap-ca-cm", "certificateAlias": "ldap-ca"}]}}'  
   * **Explanation:** This adds a trustedCerts array to the spec of the keycloak instance named keycloak. It tells Keycloak to look for a ConfigMap named ldap-ca-cm and use the certificate within it, aliased as ldap-ca.
2. Extract CA Certificate from ConfigMap:  
   This command extracts the certificate authority (CA) certificate from the ldap-ca-cm ConfigMap and saves it locally as ca.crt.  
   oc get configmap ldap-ca-cm -o jsonpath='{.data.ca\.crt}' > ca.crt
3. Copy CA Certificate into Keycloak Pod:  
   Copy the extracted ca.crt file into the running Keycloak pod. This is often necessary for Keycloak to access the certificate before it's imported into its truststore.  
   oc cp ./ca.crt keycloak-0:/home/jboss -c keycloak # in the keycloak pod  
   * **Note:** keycloak-0 assumes your Keycloak pod is named keycloak-0. Adjust if your pod naming differs. -c keycloak specifies the container within the pod if there are multiple.
4. Import Certificate into Keycloak Truststore:  
   Execute this command inside the Keycloak pod to import the ca.crt into Keycloak's Java truststore (truststore.jks). This makes the certificate trusted by the Java applications running within Keycloak.  
   # Run this command INSIDE the keycloak-0 pod:  
   # oc rsh keycloak-0  
   keytool -importcert -file ca.crt -alias ldap-root-ca -keystore truststore.jks -storepass citicert -noprompt  
   # exit  
   * citicert**:** This is the password for the truststore. Ensure this matches what's configured for Keycloak if it's not the default.
5. Copy Truststore Locally:  
   Copy the updated truststore.jks from the Keycloak pod back to your local machine.  
   oc cp keycloak-0:/home/jboss/truststore.jks ./truststore.jks -c keycloak
6. Create SSO Truststore Secret:  
   Create an OpenShift secret from the local truststore.jks file. This secret can then be mounted into other applications (like 3scale components) that need to trust the same CA certificate.  
   oc create secret generic sso-truststore-secret --from-file=truststore.jks -n rh-sso
7. Address SSL Verification Errors for 3scale Login:  
   If you encounter "Authentication error: the auth provider does not have a valid SSL certificate" when 3scale tries to log in using RH-SSO, it often means 3scale is not trusting the RH-SSO's certificate. The following commands disable SSL verification for specific 3scale deployments.
   * **Security Warning:** Disabling SSL verification (SSL\_VERIFY=false) is generally **NOT RECOMMENDED FOR PRODUCTION ENVIRONMENTS** as it makes your application vulnerable to man-in-the-middle attacks. This is often used as a temporary workaround during initial setup. The proper solution is to ensure 3scale truststore contains the RH-SSO certificate.

# Navigate to the correct directory if needed  
# cd ./3scale/config/ # (assuming inject-sso-cert.sh is here)  
# inject-sso-cert.sh # (If this script handles certificate injection, execute it)  
# cd ../../ # Return to root  
  
# Verify Keycloak realm URL (optional, for debugging SSL)  
curl -v https://keycloak-rh-sso.apps.mlab-ctigtdc02d.ecs.dyn.nsroot.net/auth/realms/maas  
  
# Disable SSL verification for Zync and System App deployments  
oc set env deployment/zync ZYNC\_AUTHENTICATION\_SSL\_VERIFY=false --overwrite  
oc set env deployment/system-app OAUTH\_SSL\_VERIFY=false --overwrite

## **4. 3scale Developer Portal (CMS) Configuration**

This section describes how to manage the 3scale Developer Portal's content management system (CMS) using a podman containerized tool.

* Reference Documentation: For detailed 3scale-cms CLI usage, refer to:  
  https://github.com/FwMotion/3scale-cms/blob/main/docs/cli-usage.adoc

1. Pull the 3scale-cms Docker Image:  
   Download the 3scale-cms CLI tool container image from your internal repository.  
   podman pull docker-icg-dev-local.artifactrepository.citigroup.net/isg-helix-docker-shared-175383/cti-cti-ake-gcs-ilab-175383/3scale-cms:latest
2. Set Environment Variables:  
   Define ACCESS\_TOKEN (your 3scale access token) and ADMIN\_HOST (the 3scale admin portal route).  
   export ACCESS\_TOKEN=<your\_3scale\_access\_token>  
   export ADMIN\_HOST=$(oc get route -n 3scale | grep maas-admin | awk '{print $2}')  
   * <your\_3scale\_access\_token>**:** Replace this with a valid 3scale access token generated from your 3scale admin portal. This token grants the CMS tool permissions to interact with your 3scale instance.
3. Define cms Alias:  
   Create a convenient alias for running the 3scale-cms tool using podman. This mounts a local directory (~/maas/cms) into the container for file synchronization.  
   alias cms='podman run --userns=keep-id:uid=$(id -u) -it --rm -v ~/maas/cms:/cms:Z docker-icg-dev-local.artifactrepository.citigroup.net/isg-helix-docker-shared-175383/cti-cti-ake-gcs-ilab-175383/3scale-cms:latest'  
   * **Note:** id -u dynamically gets your current user ID, ensuring correct permissions for the mounted volume.
4. Use cms Commands:  
   Now you can use the cms alias to interact with your 3scale Developer Portal.
   * **Get Info:**  
     cms -k --access-token=${ACCESS\_TOKEN} https://${ADMIN\_HOST} info  
     + -k: Allows insecure connections (e.g., if your 3scale admin route uses self-signed certs).
   * **Download CMS Content:**  
     cms -k --access-token=${ACCESS\_TOKEN} https://${ADMIN\_HOST} download  
     + This will download all CMS content (pages, partials, layouts, etc.) from your 3scale Developer Portal into the ~/maas/cms directory.
   * **Upload CMS Content (Incremental):**  
     cms -k --access-token=${ACCESS\_TOKEN} https://${ADMIN\_HOST}/ upload -u  
     + This uploads only the changed files from your local ~/maas/cms directory to the 3scale Developer Portal.
   * Upload CMS Content (Clean and Copy from Backup/Local):  
     This command uploads all files from your local ~/maas/cms directory, deleting any existing files on the portal that are not present locally. This is useful for syncing from a known good state or a backup. It also specifically ensures that 1\_main\_layout.html.liquid is uploaded.  
     # Ensure all files from the github cms folder are in your local ~/maas/cms folder  
     cms -k --access-token=${ACCESS\_TOKEN} https://${ADMIN\_HOST}/ upload --delete-missing --layout=/1\_main\_layout.html.liquid

## **5. Post-Installation / Additional Steps**

This section outlines further configuration and automation steps, which may require more detailed documentation or custom scripting.

1. LDAP Setup in RH-SSO:  
   This typically involves configuring a new User Federation provider within Keycloak's admin console to connect to an external LDAP server for user authentication and synchronization.
   * *(Specific commands for this are not provided in your input, but would involve Keycloak's UI or Keycloak API calls.)*
2. RH-SSO Configuration on 3scale Developer Portal:  
   After setting up LDAP in RH-SSO, you'll need to configure 3scale to use RH-SSO as its authentication provider for the Developer Portal. This involves:
   * Creating an OpenID Connect (OIDC) client in Keycloak for 3scale.
   * Configuring 3scale's "SSO Integrations" section in its admin portal with the Keycloak client ID, client secret, and OIDC discovery URL.
   * *(Specific commands for this are not provided in your input, but involve 3scale Admin UI configuration.)*
3. Automate Service Creation, Backend Creation, Plan Creation, and Publishing:  
   For a fully automated MaaS environment, you would typically use 3scale's API (ActiveDocs/Swagger, or direct API calls) to programmatically:
   * Create new API Backends.
   * Create new API Services associated with those Backends.
   * Define Application Plans and Usage Limits.
   * Publish APIs to the Developer Portal.
   * *(This often involves custom scripts (e.g., Python, Ruby, Node.js) interacting with the 3scale Admin API.)*

This documentation provides a comprehensive overview of the commands and steps for setting up 3scale and RH-SSO on OpenShift. Remember to adapt paths, names, and configurations to your specific environment.