

The case of the impossible additional users

Abstract and Motivation

Working in a engagement for a [OSD non CCS in GCP](#) I found myself caught in a strange situation. While helping the dedicated engineer working with the cluster he posed to me the issue: "I need to create new users in the cluster, but there's no IDP yet defined by the client", meaning from [all the many possible IDPs managed Openshift](#) can use the only one in place was htpasswd and the one user possible was already taken, the one he was operating with. Yes, htpasswd IDP in a managed Openshift can only have one user defined.


The motivation for this blog is exactly that. Extra users were needed temporarily and if possible with logins the same as their future logins would be. The future IDP coming was LDAP.

At the time I offered him, in my humble opinion, what would be two of the most quick (and almost dirty) solutions from the top of my head, I said "ok, create a good old user that will authenticate with a good old kubernetes client certificate and/or setup a temporary LDAP server and use the LDAP IDP".

What seemed to me at the time a fast and easy solution was not in fact so automatic to pull off the hat, so I decided to put together all the scripting, code, actions, steps taken to make it not only possible, but also simple in this blog + code post.

Authenticate with a user certificate

This solution is part of [Red Hat's solution knowledge base](#) and can also be found in [Kubernetes official documentation](#). With this, all you need to do is follow those instructions or clone the blog repo and run the [create-cert-user](#) script ([yq](#) is needed to run it). You'll need to be logged as administrator into the cluster you are currently targeting.

 The knowledge base procedure considers your kube config first cluster entry is the cluster you want when creating the new context. If you work with many clusters at once this may not be true for you, please check your kube config with

```
oc config get-clusters
```

```
# from repo base directory
cd redhat/idp/cert+ldap/cert
./create-cert-user.sh newuser
```

In case you don't want to clone the repo here's the script:

```
cfg="$HOME/.kube/config"
export usr=$1
export cluster=`oc config get-contexts | grep '^*' | awk '{print $3}'`
export ctx="$usr@$cluster"
current_ctx=`oc config current-context`
```

```

openssl req -new -newkey rsa:2048 -nodes -keyout $usr.key -out $usr.csr -
subj "/CN=$usr"
oc create user $usr

cat <<EOF | oc apply -f -
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: $usr
spec:
  request: `cat $usr.csr | base64 -w 0`
  signerName: kubernetes.io/kube-apiserver-client
  expirationSeconds: 31536000
  usages:
    - client auth
EOF

sleep 1
oc adm certificate approve $usr
sleep 3
oc get csr $usr -o jsonpath='{.status.certificate}' | base64 -d > $usr.crt
oc config set-credentials $usr --client-certificate=$usr.crt --client-
key=$usr.key --embed-certs=true
oc config set-context "$ctx" --namespace=default --cluster=$cluster --
user=$usr
oc config use-context "$ctx"
oc whoami
oc config use-context "$current_ctx"

# extract the context recently created to share with created user owner
cat <<EOF | yq > $usr.cfg
apiVersion: v1
kind: Config
preferences: {}
current-context: $ctx
clusters:
-
`yq '.clusters[] | select(.name == strenv(cluster))' $cfg | sed -e 's/^/
/g'`
users:
-
`yq '.users [] | select(.name == strenv(usr))' $cfg | sed -e 's/^/ /g'`
contexts:
-
`yq '.contexts [] | select(.name == strenv(ctx))' $cfg | sed -e 's/^/ /g'`
EOF

```

👉 The file named '*newuser.cfg*' from the above script should be a valid kube config file that can be shared with the newly created user.

👉 Remember to bind a role to the new user.

! Attention to the fact that the users' private keys will be lying around and embedded in the kube config !

Authenticate with a LDAP deployed in the cluster

With LDAP as the second quick and almost dirty solution to rapidly create users for the cluster I needed a simple, no-thrills LDAP server to run and I found [Glauth](#) which is tested, neat and has a different number of backend choices for the user database including simple text config in TOML, which we're using below.

In the repo you'll find a [deployment manifest](#) with everything, but first you need to generate the key and certificate for the secure port of the LDAP server and update the 'ConfigMap' in the deployment file with both. If you generate the certificate and key with different names than the example below bare in mind [the IDP configuration step](#) will need to be modified accordingly.

```
openssl genrsa -out glaauth.key 2048
openssl req -new -x509 -sha256 -key glaauth.key -out glaauth.crt -subj
"/CN=glaauth" -days 3650 -addext
"subjectAltName=DNS:glaauth.glaauth.svc.cluster.local, DNS:glaauth.ddns.net,
DNS:glaauth.duckdns.org, DNS:glaauth.mscaastro.net"
```

If you are planning on [exposing the service to the outside of the cluster](#) you need to add the DNS names you intend using to 'subjectAltName' when generating the certificate. Also, if for any extraordinary reason you are not using the default domain name of a kubernetes cluster, you also need to adjust 'glaauth.glaauth.svc.cluster.local' to your needs.

If you have [yq](#) installed you can run the [create-cert-and-deploy](#) script from the repo to accomplish all of the above in one go, see it below

```
# from repo base directory
cd redhat/idp/cert+ldap/ldap
./create-cert-and-deploy.sh
```

```
dns_names="glaauth.glaauth.svc.cluster.local glaauth.ddns.net
glaauth.duckdns.org glaauth.mscaastro.net"
subj_alt_name="subjectAltName=DNS:`echo $dns_names | sed -e 's/ /,/DNS:/g'`"

openssl genrsa -out glaauth.key 2048
openssl req -new -x509 -sha256 -key glaauth.key -out glaauth.crt -subj
"/CN=glaauth" -days 3650 -addext "$subj_alt_name"
yq 'with(select(.kind=="ConfigMap" and .metadata.name=="glaauth");
.data."glaauth.key" = load_str("glaauth.key") | .data."glaauth.crt" =
load_str("glaauth.crt"))' deployment.yaml | oc apply -f -
```

If you did not use the above script, of course you need to run `oc apply -f deployment.yaml` before moving on.

Let's use a image container with ldapsearch and test it from inside the cluster:

```
oc debug --image=emeraldsquad/ldapsearch
ldapsearch -v -LLL -H ldap://glauth.glauth.svc.cluster.local -D
cn=redhat,ou=csa,dc=latam,dc=redhat -w redhat -x -b "dc=latam,dc=redhat"
cn=hi
```

The above search should return information about user named 'hi'. If you want to test the secure port 636, export `LDAPTLS_REQCERT=never` and redo the search with `ldaps://`.

Configure an LDAP IDP

Configure using OCM WEB UI

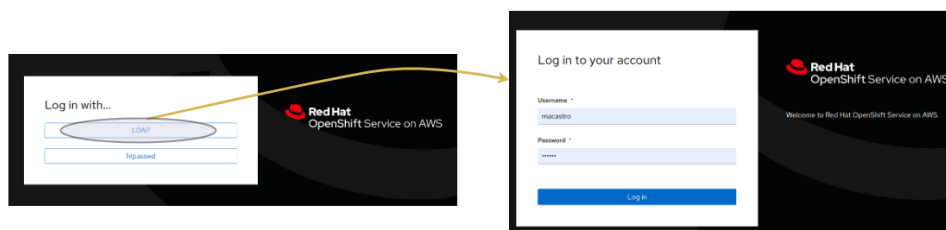
Now, it's all about adding a new LDAP IDP using the [OCM web UI](#) like below

The screenshot shows the 'Edit identity provider: LDAP' configuration page in the Red Hat Hybrid Cloud Console. The page includes the following fields and annotations:

- Name:** LDAP. Annotation: 'ldap (default port 389) or ldaps (default port 636) if using tls.'
- Mapping method:** claim. Annotation: 'Specifies how new identities are mapped to users when they log in. Claim is recommended in most cases.'
- LDAP URL:** ldap://192.168.10.180/dc=latam,dc=redhat?cn. Annotation: 'An RFC 2255 URL which specifies the LDAP search parameters to use.'
- Bind DN:** cn=macastro,ou=csa,dc=latam,dc=redhat. Annotation: 'DN to bind with during the search phase.'
- Bind password:** redhat. Annotation: 'Password to bind with during the search phase.'

A callout box provides a sample `ldapsearch` command line:

```
> ldapsearch -LLL \
-H ldap://192.168.10.180 \
-D cn=macastro,ou=csa,dc=latam,dc=redhat \
-w redhat -x \
-b "dc=latam,dc=redhat" \
"(&(objectClass=*)(cn=macastro))"
```



LDAP URL: `ldaps://glauth.glauth.svc.cluster.local/dc=latam,dc=redhat?cn`

👉 and make sure to add `glauth.crt`

The list of attributes whose values should be used as the email address.

mail

[+ Add more](#)

▼ [Hide advanced options](#)

CA file [Browse...](#)

Hide

```
-----BEGIN CERTIFICATE-----
MIIFcTCCA1mgAwIBAgIUe9INDls/D58j0F22bT6v0Jw6f5EwDQYJKoZIhvcNAQEL
BQAwHjELMAKGA1UEBhMCQlIxZDZANBgNVBAMMBmVsYXV0aDAeFw0yMjA2MjAxOTA3
MDlaFw0zMzA2MTcxOTA3MDlaMB4xCzAJBgNVBAYTAKJSMQ8wDQYDVQQDDAZnbGF1
dGwggIiMA0GCSqGSIb3DQEBAQUAA4ICDwAwggIKAoICAQDzLN+3FdWQNVs25S0g
Q6v5VaG0m3rqYvYRYB9H7nHT7cYXW/Hj0r0vZdtzLzfjKpJW1jwoEppDq3mrfl
76//GIgRvrh5AK+bFAIc5q688n4q1XPEgPCpeqU1s7H+DshrK9jMVyA5TSBAs2jm
/o3kUB0mZLzjD9FwhcNg483E0u7tSuiPR7G0V9qkxvu4wdJ0MT5ubktvKDN0272n
z91DXk20HTwskhSkMhQ4MaxSIBL4fUaNRXwvtQ2RFmIsdSwIgJ5gJGkEbD4t5dzw
vZvFEK2Yfv1S6PbBTHMhPL7nZVP0Xo28Hkg6aUnKctP65S2kczmPMfiy/C8t+PZu
o2GgUEE6GEj6enFLQ8LYaT6HsLbQ0eq/Uc1SKvTLWdmXTKQwusx1qL5ugaoraJ/a
Zf9ExNqq3wTHcjueZfz3qQ0J6SVCqvU+Jh+ytG0QtQYEiKgjfabhNuF3+zA26P/y
EmJEi1VwZVh7Fs4wFz1vY3mDQqUnUswCCx4iW1a7BfUkhBzX1G0Z94/Jf7XvgAF1f
K6JJ9noSiom2XF2JdpxgC4b5Bpxpcb8970aBZfv0E20r+1W05YjElasxW2RadAdh
5N5X2rJayyAet4pzVrh6V/nPH4ZsFbqQ6HJmcCGQjiYm1SEZ1v25Zwey7XELw6es
a30UAATY240FYv4Ka2spCcFb5QIDAQABo4GmMI6jMB0GA1UdDgQWBBQvL3IZv+Ju
+yqvYL396aJWMM1XFTAFBgNVHSMGDAWgBQvL3IZv+Ju+yqvYL396aJWMM1XFTAP
BgNVHRMBAf8EBTADAQH/MFAGA1UdEQRJMEeCE2dsYXV0aC5tc2Nhc3Ryby5uZXSC
-----
```

PEM encoded certificate bundle to use to validate server certificates for the configured URL.

☐ Insecure

[Save](#) [Cancel](#)

Configure using the terminal and oc CLI

The other way you can configure is following the [OpenShift documentation for adding a new LDAP IDP](#) while using the CLI or using the [script provided in the repo](#) and below

```
# from repo base directory
cd redhat/idp/cert+ldap/cert/ldap
./add-ldap-idp.sh
```

```
type="private"
ldap_host="glauth.glauth.svc.cluster.local"
bind_dn="cn=redhat,ou=csa,dc=latam,dc=redhat"
bind_pwd="redhat" # change the passwd according to the bindDN used
crt_file=glauth.crt
ldap_searchbase="dc=latam,dc=redhat"
ldap_filter="cn"
ldap_url="ldaps://$ldap_host/$ldap_searchbase?$ldap_filter"
secret_name="glauth-bind-passwd"
```

```

config_map_name="glauth-cert-$type"
export patch_file="/tmp/ldap-idp-$type.yaml"
export idp_name="ldap-$type"

oc delete secret $secret_name -n openshift-config --ignore-not-found
oc delete configmap $config_map_name -n openshift-config --ignore-not-found
oc create secret generic $secret_name --from-
literal=bindPassword="$bind_pwd" -n openshift-config
oc create configmap $config_map_name --from-file=ca.crt=$crt_file -n
openshift-config

cat <<EOF > $patch_file
- ldap:
  attributes:
    email:
      - mail
    id:
      - dn
    name:
      - cn
    preferredUsername:
      - uid
  bindDN: $bind_dn
  bindPassword:
    name: $secret_name
  ca:
    name: $config_map_name
  insecure: false
  url: $ldap_url
  mappingMethod: claim
  name: $idp_name
  type: LDAP
EOF

echo -n "removing $idp_name... "
oc get oauth cluster -o yaml | yq 'del(.spec.identityProviders[] |
select(.name == strenv(idp_name)))' | oc apply -f -
echo -n "adding $idp_name... "
oc get oauth cluster -o yaml | yq '.spec.identityProviders +=
load(strenv(patch_file))' | oc apply -f -
rm $patch_file

```

👉 Above script adds a 'type' suffix to the ldap idp config in Openshift oauth. If you plan to [expose it](#) and want to test the exposed version, add a new one pointing to the outside version, change the suffix and the 'ldap_host'.

Adding new users

If you looked at the [configmap](#) you easily spotted how you'd add more users. Just go about repeating the block below and change the fields, mandatorily 'name'. The password is the sha256 of the clear text.

```
[[users]]
  name = "hi"
  givenname="Hi"
  sn="HowAreYou"
  mail = "hi@redhat.com"
  uidnumber = 5003
  primarygroup = 5500
  passsha256 = "2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824" # hello
  [[users.capabilities]]
    action = "search"
    object = "*"

```

Exposing your internal LDAP

You can expose your recently added LDAP server by running

```
oc project glauth
oc expose deployment glauth --name='glauth-public-exposed' --target-
port='3636' --port='636' --type='LoadBalancer'
```

That will add a load balancer to the cluster. If you need it to be deployed with a private IP add the [custom annotation for your cloud provider](#)

```
apiVersion: v1
kind: Service
metadata:
  name: glauth-private-exposed
  namespace: glauth
  annotations:
    # GCP
    # networking.gke.io/load-balancer-type: "Internal"
    # AWS
    # service.beta.kubernetes.io/aws-load-balancer-internal: "true"
    # AZR
    # service.beta.kubernetes.io/azure-load-balancer-internal: "true"
spec:
  type: LoadBalancer
  ports:
    - name: sldap
      port: 636
      targetPort: 3636
      protocol: TCP
  selector:
    app: glauth
```

Once you know the load balancer IP `oc get svc glauth-public-exposed -n glauth -o jsonpath='{.status.loadBalancer.ingress[0].hostname}'` you can add an entry to your DNS server pointing to it, but remember that the name you use needs to be in the certificate DNS names list of the `glauth.crt` certificate file created before.

About sharing the load balancer for other Services in the cluster

Yes, this is the case where an ingress won't do because we're talking about ports different than 80 and 443 and managed Openshift does not provide this easily, the best, fastest, simplest way I could find to get this going is using the NGINX Ingress Controller, which can be deployed using the operators hub and requires some configuration tricks to work, but that is a story for a different blog post...