

# **Integrating OIDC with Kubernetes using Active Directory with secure LDAP**



Created By: Scott Surovich  
Date: October 2018  
Version 1.0

## Table of Contents

<b>Overview .....</b>	<b>3</b>
<b>Requirements .....</b>	<b>3</b>
<b>Why Tremolo Security? .....</b>	<b>4</b>
<b>Openunison Components .....</b>	<b>5</b>
Create a MariaDB Instance Using Docker .....	5
<b>Deploying Openunison.....</b>	<b>7</b>
Creating base Directories and Retrieving your LDAPS Certificate .....	7
<b>Integrating Openunison with Kuberenetes .....</b>	<b>8</b>
Creating Your input.props File .....	8
Deploy Openunison using your Certificate and Props File .....	8
Obtaining OIDC Information for the Kubernetes API Manifest .....	9
Adding the Certificate and Options to Your API Server .....	10
<b>Initial Log-In to Openunison.....</b>	<b>11</b>
<b>Using your Token .....</b>	<b>14</b>
Adding the Openunison Certificate to Your Client .....	15
Updating your Client Config File .....	16
<b>Appendix .....</b>	<b>18</b>

## Overview

When most people start to learn Kubernetes they select some installation method like Kubeadm, CDK, OCP, Origin, PKS, Apprenda (RIP), or the excellent “Kubernetes the hard way” tutorial. While some of the mentioned installations do include an authentication add on, most do not – they simply deploy a vanilla upstream Kubernetes with either a single admin config file or, potentially, basic authentication using a CSV on the Master node(s).

Once you start to get deeper into Kubernetes you will quickly see the limitations of the authentication methods that are “easy” to deploy.

There are multiple methods that you can select for authentication, I do not plan to cover each method in this paper, but you can read about the options at the Kubernetes site:

<https://kubernetes.io/docs/reference/access-authn-authz/authentication/>

For this paper I will focus on using an OIDC provider for authentication. While there are various options to implement OIDC in your Kubernetes infrastructure, one of the most complete solutions is by Tremolo Security called Openunison.

I will use OpenUnison’s Active Directory Branch which is free to deploy and use. (They also offer a supported model, based on a per user basis.)

**Standard disclaimer:** I am not affiliated with Tremolo Security. I decided to create this paper to assist others who may want to add an OIDC provider to their Kubernetes installation. While these procedures have been tested multiple times on different installations, I take no responsibility for any lose of data or issues that may be caused by your installation.

## Requirements

The example installation will use a Kubernetes 1.12 installation using kubeadm. Creating your cluster is out of the scope of this document, so I will assume you have a running K8’s cluster created with kubeadm for the remainder of this document.

**NOTE:** At this time, any Kubernetes installation that use an external cert provider like EasyRSA (Canonical and Rancher), are not compatible with the newest release of Openunison. You can use a previous build, but they are working on an update to add compatibility. I will add the steps to the document once they have been added to the project.

To continue, you will need the following:

- A Linux workstation (I will use CentOS 7) with:
  - Docker installed

- kubectl installed, using the config file for your cluster
- A working K8's cluster with a NGINX Ingress Controller
- A Database server – I will use MariaDB for our example
- An Active Directory Domain with AD LDS (ADFS Also Works, but I will use LDS for this doc)
  - A User account with read access to the Domain
  - Your AD target information, including:
    - Base DN
    - Search DN
- Your Active Directory Certificate (Steps are provided in this doc to retrieve your certificate if you do not have a copy already)
- Internet Access
  - I will not cover an offline installation in this document

## Why Tremolo Security?

Over the past year I have looked at many methods to integrate Active Directory or LDAP into my home lab and while there are many solutions out there including Keycloak and Dex, I found that most I'm missing some component(s). Some solutions offered integration but they did not offer an easy method to:

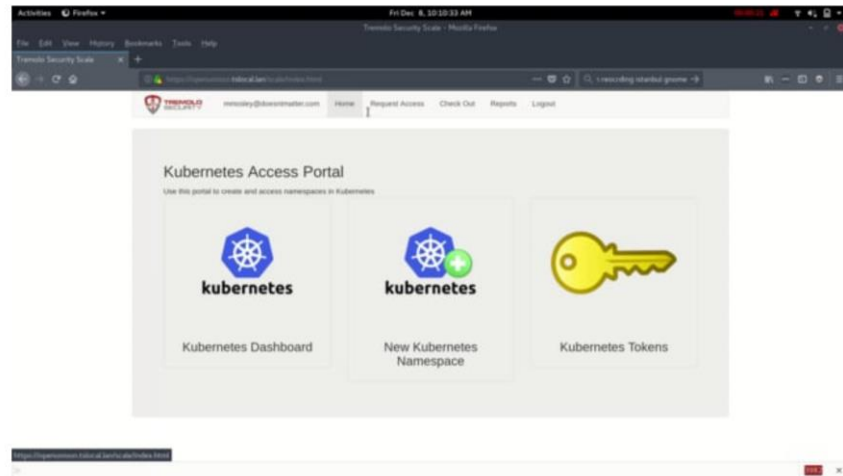
- Obtain your JWT (JSON web token)
- Integrate the Kubernetes Dashboard

Many solutions included links to other projects to accomplish the above tasks, but it required adding a product from a public Github project, so any support of a final installation may require engaging multiple vendors.

Openunison includes all of the desired feature and more. You can read more about the feature on their Github page - <https://github.com/TremoloSecurity/openunison-gs-kubernetes/tree/activedirectory>

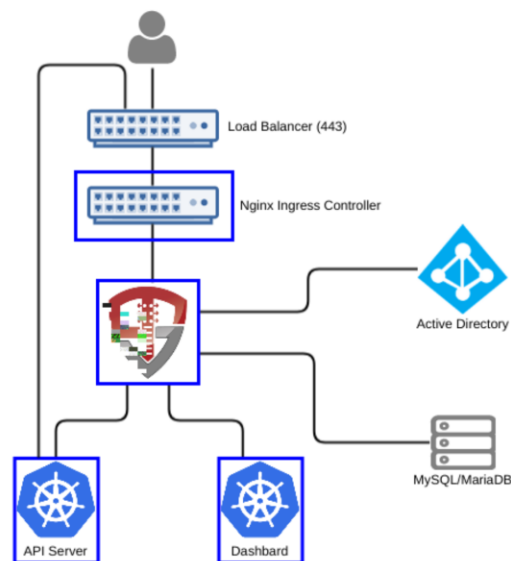
The above link is a direct link to the Github project I want to use, the quick start Active Directory branch.

Once installed and configured, your users will have access to a webpage that looks like the below image, allowing easy access to their kubectl command and the Kubernetes dashboard:



## Openunison Components

For this demonstration I will not install a load balancer, using a simple Ingress rule to target the Openunison deployment on our Kubernetes cluster. In a production environment you may have a load balancer fronting a wildcard DNS for your cluster(s). Since there are multiple options to accomplish the load balancing, I will not cover it in this document.



## Create a MariaDB Instance Using Docker

**If you have an existing database you can target, you may skip this section.**

For the ease of testing, I will deploy a MariaDB container image and simply use a local mount on the server as our DB storage. While this is not ideal, it will work as a quick example for deploying Openunison. In a production environment you would want to use a production ready database solution.

Pull the mariadb image from Docker Hub.

***sudo docker pull mariadb***

This will pull the latest mariadb container and you should receive a successful pull as shown below.

```
latest: Pulling from library/mariadb
124c757242f8: Pull complete
9d866f8bde2a: Pull complete
fa3f2f277e67: Pull complete
398d32b153e8: Pull complete
afde35469481: Pull complete
31f2ae82b3e3: Pull complete
3eeaf7e45ea6: Pull complete
716982328e17: Pull complete
34ce605c9036: Pull complete
4502ed9073c0: Pull complete
2afafbdf5a96: Pull complete
7997117735b2: Pull complete
b4fd3d5963fe: Pull complete
b8ecbec33e2f: Pull complete
0053e3b37e07: Pull complete
Digest: sha256:0d4bad4ea58ec976f3de2b5e16f4ed9513da58f21a2bb0df9fdd8422526dd5e5
Status: Downloaded newer image for mariadb:latest
```

Create a directory in the root called /docker/mariadb

***sudo mkdir -p /docker/mariadb***

Now I can run our mariadb container. The following command will run the container in detached mode and use the local file system on the Linux host for storage.

***sudo docker run -d -p 3306:3306 --name mariadb -v /docker/mariadb:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=mypassword -d mariadb:latest***

You can set your own mysql root password – but be sure to write it down, you will need it for the Openunison configuration and to create the blank database.

To verify your DB is running you can run a docker ps command. Since I know the command worked from the execution, I usually check the directory I used for any new data to confirm the database started up and created any initial files. To check your database, execute:

***sudo ls -la /docker/mariadb***

You should see data in your directory.

```
total 122944
drwxr-xr-x 4 999 root 4096 Oct 11 14:40 .
drwxr-xr-x 3 root root 4096 Oct 11 14:35 ..
-rw-rw---- 1 999 docker 16384 Oct 11 14:40 aria_log.00000001
-rw-rw---- 1 999 docker 52 Oct 11 14:40 aria_log_control
-rw-rw---- 1 999 docker 976 Oct 11 14:40 ib_buffer_pool
-rw-rw---- 1 999 docker 12582912 Oct 11 14:40 ibdata1
-rw-rw---- 1 999 docker 50331648 Oct 11 14:40 ib_logfile0
-rw-rw---- 1 999 docker 50331648 Oct 11 14:39 ib_logfile1
-rw-rw---- 1 999 docker 12582912 Oct 11 14:40 ibtmp1
-rw-rw---- 1 999 docker 0 Oct 11 14:39 multi-master.info
drwx----- 2 999 docker 4096 Oct 11 14:39 mysql
drwx----- 2 999 docker 4096 Oct 11 14:39 performance_schema
-rw-rw---- 1 999 docker 24576 Oct 11 14:40 tc.log
```

Now I need to create an empty database called unison. Execute docker exec into the container to launch the mysql cli:

```
sudo docker exec -it mariadb mysql -uroot -pmypassword
```

This will provide you with an interactive mysql cli, I simply need to create a new database, so enter:

```
create database unison;
```

Press enter and you should receive a success, as shown below.

```
Query OK, 1 row affected (0.001 sec)
MariaDB [(none)]>
```

Type **exit** to quit the interactive session.

## Deploying Openunison

### Creating base Directories and Retrieving your LDAPS Certificate

Before I begin, verify that you are in your home folder and create a new folder called Openunison and change to that directory. Then create two subdirectories under the new Openunison directory, one called certs and another called props.

As I mentioned in section 2, I need the AD certificate for your LDAP server. If you not using a secure connection to your LDAP server then you can skip this requirement, however, using a non-encrypted connection to LDAP is not recommended.

You can retrieve the certificate using OpenSSL if you do not have a copy of the cert file. On a workstation run the following OpenSSL command to create a file called trusted-adldaps.pem. Note: Change the address to the IP / Hostname of your AD LDS Server.

```
openssl s_client -showcerts -connect ad2.example.com:636 </dev/null 2>/dev/null|openssl x509 -outform PEM >/home/<username>/openunison/certs/trusted-adldaps.pem
```

You now have the certificate for your AD LDS server in the current directory. I will assume this is in the home folder for root.

## Integrating Openunison with Kuberenetes

### Creating Your input.props File

Openunison will store its configuration information in a Kubernetes secret. Using your favorite editor, create/edit a file in your /home/<username>/openunison/props directory called **input.props**. You are not required to use an SMTP server but you need to have some values in the variables. A detailed explanation of each variable is in the appendix.

```
OU_HOST=k8sou.tremolo.lan
K8S_DASHBOARD_HOST=k8sdb.tremolo.lan
K8S_URL=https://k8s-installer-master.tremolo.lan:6443
OU_COOKIE_DOMAIN=tremolo.lan
OU_HIBERNATE_DIALECT=org.hibernate.dialect.MySQL5InnoDBDialect
OU_JDBC_DRIVER=com.mysql.jdbc.Driver
OU_JDBC_URL=jdbc:mysql://dbs.tremolo.lan:3308/unison
OU_JDBC_USER=root
OU_JDBC_PASSWORD=start123
OU_JDBC_VALIDATION=SELECT 1
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=donotreply@domain.com
SMTP_PASSWORD=xxxx
SMTP_FROM=donotreply@domain.com
SMTP_TLS=true
AD_BASE_DN=cn=users,dc=ent2k12,dc=domain,dc=com
AD_HOST=192.168.2.75
AD_PORT=636
AD_BIND_DN=cn=Administrator,cn=users,dc=ent2k12,dc=domain,dc=com
AD_BIND_PASSWORD=password
AD_CON_TYPE=ldaps
SRV_DNS=false
OU_CERT_OU=k8s
OU_CERT_O=Tremolo Security
OU_CERT_L=Alexandria
OU_CERT_ST=Virginia
OU_CERT_C=US
unisonKeystorePassword=start123
```

Example input.props File

### Deploy Openunison using your Certificate and Props File

Next, I need to execute the following command which will create the deployment for Openunison.

```
curl https://raw.githubusercontent.com/TremoloSecurity/kubernetes-artifact-
deployment/master/src/main/bash/deploy_openunison.sh | bash -s
/home/<username>/openunison/certs /home/<username>/openunison/props
```



```
https://raw.githubusercontent.com/TremoloSecurity/openunison-qs-kubernetes/activedirectory/src/main/yaml/artifact-deployment.yaml
```

The command above will download a small shell script that will pull in the files from the two directories you created earlier (Your AD Certificate and your props file).

As the deployment is executed, you will see the output from the script on your screen.

```
namespace/openunison-deploy created
configmap/extracerts created
secret/input created
clusterrolebinding.rbac.authorization.k8s.io/artifact-deployment created
job.batch/artifact-deployment created
```

NAME	READY	STATUS	RESTARTS	AGE
artifact-deployment-jzmnr	0/1	Pending	0	0s
artifact-deployment-jzmnr	0/1	Pending	0	0s
artifact-deployment-jzmnr	0/1	ContainerCreating	0	0s
artifact-deployment-jzmnr	1/1	Running	0	4s
artifact-deployment-jzmnr	0/1	Completed	0	15s

If you see that the pods have completed, you can press control-c to exit the script.

At this point, let's verify you have the two namespaces that were created by the deployment, you should see an openunison-deploy and an openunison namespace.

***kubectl get ns***

NAME	STATUS	AGE
default	Active	8h
ingress-nginx	Active	8h
kube-public	Active	8h
kube-system	Active	8h
openunison	Active	74m
openunison-deploy	Active	74m

If you are missing the openunison namespace, your deployment did not complete successfully and you will need to delete the openunison-deploy namespace before deploying again.

## Obtaining OIDC Information for the Kubernetes API Manifest

The next step is to integrate your new OIDC provider with your cluster. Openunison includes a command that will provide the required information you will need for your API server. Execute the following the command:

***kubectl describe configmap api-server-config -n openunison***

This will output the API options you need to add on your API server(s) and the certificate you will need to trust, which will be added to your API server(s) as **/etc/kubernetes/pki/ou-ca.pem**

```
Data
====
oidc-api-server-flags:
----
```

```
--oidc-issuer-url=https://unison.example.com/auth/idp/k8sIdp
--oidc-client-id=kubernetes
--oidc-username-claim=sub
--oidc-groups-claim=groups
--oidc-ca-file=/etc/kubernetes/pki/ou-ca.pem
ou-ca.pem-base64-encoded:
-----
-----BEGIN CERTIFICATE-----
MIID+TCCAuGgAwIBAgIGAWZv6BPdMA0GCSqGSIb3DQEBCwUAMIGBMQswCQYDVQQG
EwJVUzERMA8GA1UECBMIVmlyZ2luaXlzEzARBgNVBACTCkFsZXhhbmRyaXxGTAX
BgNVBAoTEFRyZW1vbG8gU2VjdXJpdHkxDDAKBgNVBAcTA2s4czEhMB8GA1UEAxMY
dW5pc29uLmh5cGVyLXZwbGFuZXQuY29tMB4XDTE4MTAxNDAwMDkzMloXDTI4MTAx
MTAwMDkzMlowgYExCzAJBgNVBAYTAiVMTREwDwYDVQQIEWhWaXJnaW5pYTETMBEG
A1UEBxMKQWxleGFuZHIjYpYTEZMBcGA1UEChMQVHJlbW9sbyBTZW51cmI0eTEEMMAoG
A1UECXMdazhMSEwHwYDVQQDEh1bmlzb24uaHlwZXItbnBsYW5ldC5jb20wggei
MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCtd0oNY2Ls+qQSmgLYwf3RCi6P
DH++BoQWUmHdhx5nkZ/qvchQwxewlwFxuoYByeuSi/4Sgv5Jksw3GYkRUHXS2x
uO1daRBpO8nT8Kmf4MdkJ1FehNOiYuUpAagIEKq7HxfFOUsUavbpmJC9MI63PhEA
WfsmgScvyQa8+EHZrS4Mc0mkJ6oh/vCP62szHhmfWDlqpJiRMt04XS6wl7bLARdS
14Okp1OUPBhO4nl0OqojzP+psTZHgguanVABTTWqZlaFHOzK20cVH7Jj7wJajKZd
6YXemL4TkMcD604RNVMR2+f1ZRxEJrriFlkSevMiRtgDI3kt5cfl6oQZQXAgMB
AAGjdTBzMA8GA1UdEIB/wQFMAMBAf8wDgYDVR0PAQH/BAQDAgIEMB1GA1UdJQEB
/wQIMAYGBFUdJQAwPAYDVR0RBduwM4IYdW5pc29uLmh5cGVyLXZwbGFuZXQuY29t
Ghdr23edfi5oeXBlici12cGxhbmV0LmNvbTANBgkqhkiG9w0BAQsFAAOCAQEAbfP
Y+FkaOGGKNZa7X+4N9y4g3YWRcckRX9LHkYYdqQpapC6zEe0Ky/d1UyBMRDcl1lw
e5jN8L567oAHdQf7IA+ELkWUZL/+5afJrLpA24Ya3jgbMG7VOW9A5ggDQOWJRRJ
fep5mqJclFwMB5kyqn+FLGt3gO9idhJc4O3hvopPyR+8aoeo4S2nwhQbeg+Wm6RG
ZubsPmjMejhSvIXkmPx81PQcucZHiU2Z6FQ6eXy1q0uJZld2TVf1NRGdxxwo7kW
4GyoYUqDMSmXPtjvXmpM2x8YbNTGc4IR3VfPLdbpqHDOVcifyUxI0b6Ti1j97IOS
JLLF3KgLCuNT/w0wCw==
-----END CERTIFICATE-----
```

## Adding the Certificate and Options to Your API Server

Using the information from your output, SSH into your API server and create a file called **/etc/kubernetes/pki/ou-ca.pem** with the certificate information from the output (You will need to copy and paste the information from the Begin Certificate to the End Certificate – DO not forget to include the Begin and End in your certificate.) If you have multiple API servers, you will need this file on each server.

Next, edit your manifest file to include the OIDC parameters. This file is located on each API server in **/etc/kubernetes/manifest/kube-apiserver.yaml**

You can add this information anywhere in the options for the API server. Below is an example:

```
containers:
- command:
  - kube-apiserver
  - --oidc-issuer-url=https://unison.example.com/auth/idp/k8sIdp
  - --oidc-client-id=kubernetes
  - --oidc-username-claim=sub
  - --oidc-groups-claim=groups
  - --oidc-ca-file=/etc/kubernetes/pki/ou-ca.pem
  - --authorization-mode=Node,RBAC
```

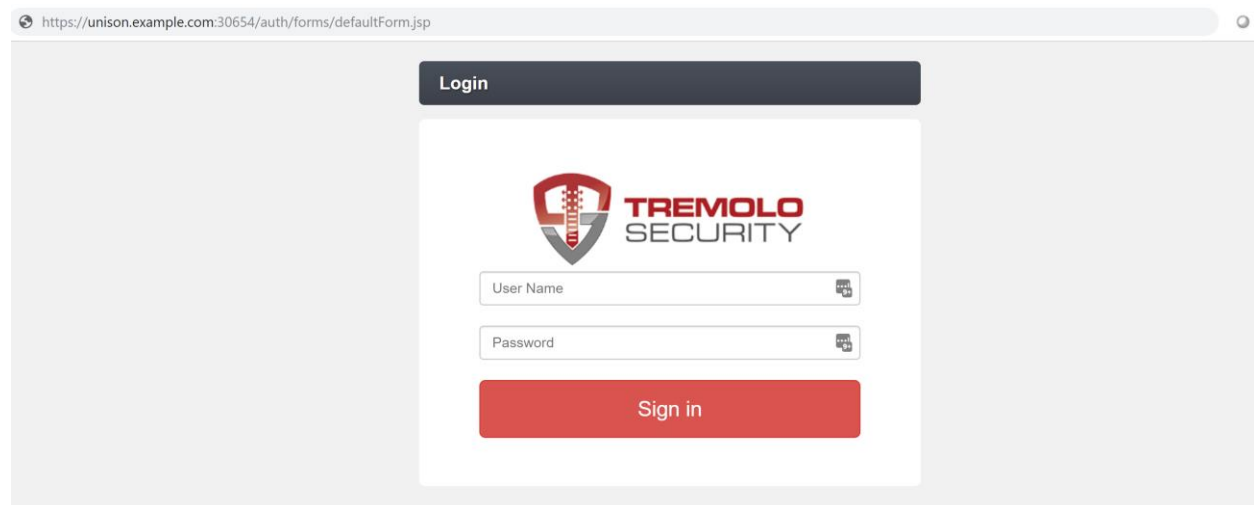
Now you can either restart the API service or reboot the API server for the changes to take effect.

**Note:** If you are using NodePort for your ingress you will need to add the port of your NodePort to the `oidc-issuer-url` option.

## Initial Log-In to Openunison

Now, you should be able to access your OIDC login page using the URL you added to your props file.

If you are using a NodePort for your Ingress you will need to add that to your URL, as shown in our example screen below.



You can use an account from your LDAP server for the initial login. The account you decide to use will be created with limited permissions.

Once created, you will need to edit the account using the steps provided below to add administrator privileges to the account.

To add administrator permissions to the account, you will need to edit the database directly. For our example I will edit the database using the MySQL CLI.

If you used our example for your database, you will need to connect to the MariaDB container. On the machine running the MariaDB container execute the following command to get CLI access.

***sudo docker exec -it mariadb mysql -uroot -pmypassword***

We need to use our unison database, to tell MariaDB to use the unison database type the bolded text below.

```
MariaDB [(none)]> use unison;
```

This will return:

```
Reading table information for completion of table and column names
```

```
You can turn off this feature to get a quicker startup with -A
```

```
Database changed
```

```
MariaDB [unison]>
```

Notice your prompt has changed to contain the database name [unison].

Next, execute the following command to edit the user account.

```
insert into userGroups (userId,groupId) values (2,1);
```

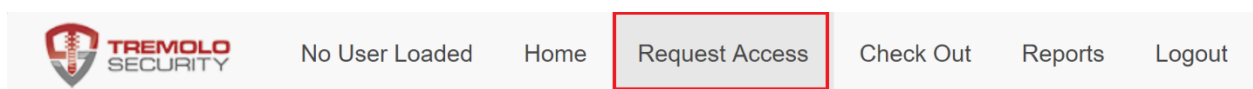
You should get a success return:

```
Query OK, 1 row affected (0.002 sec)
```

Now exit your interactive container connection by typing exit and press enter.

At this point, your account is now an Administrator but you still need to edit your account in Openunison. Log back into Openunison with the same account and follow the steps below to complete your access.

1. Click on "Request Access" in the title bar



2. Click on "Kubernetes Administration"



3. Click "Add To Cart" next to "Cluster Administrator"

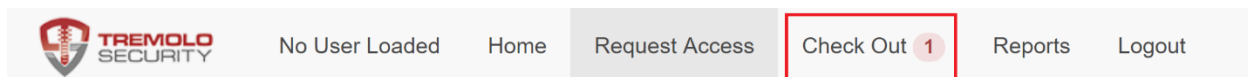


**Request Access to Applications**

- ▼ Kubernetes Enterprise
  - Audit Reports
  - Kubernetes Namespace Administrators
  - Kubernetes Administration**
  - Kubernetes Namespace Viewers
  - OpenUnison Administrators and Approvers

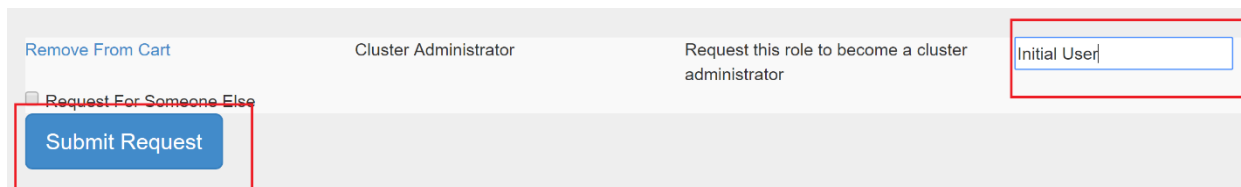
**Add To Cart** Cluster Administrator Request this role to become a cluster administrator

4. Next to "Check Out" in the title bar you'll see a red 1, click on "Check Out"



TREMOLO SECURITY No User Loaded Home Request Access **Check Out 1** Reports Logout

5. For "Supply Reason", give a reason like "Initial user" and click "Submit Request"



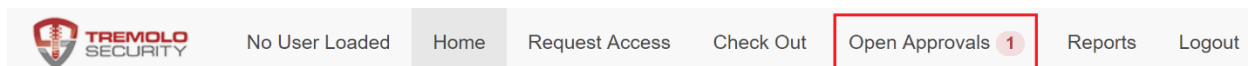
Remove From Cart Cluster Administrator Request this role to become a cluster administrator

Initial User

☐ Request For Someone Else

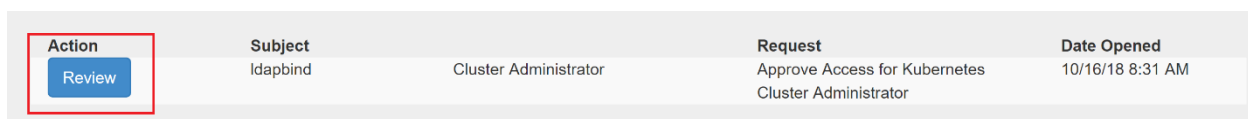
**Submit Request**

6. Since you are the only approver refresh OpenUnison, you will see a red 1 next to "Open Approvals". Click on "Open Approvals"



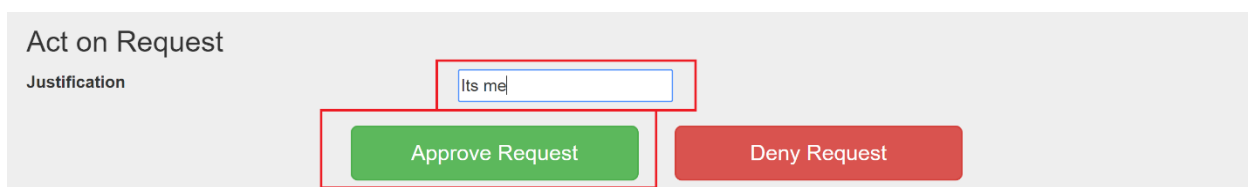
TREMOLO SECURITY No User Loaded Home Request Access Check Out **Open Approvals 1** Reports Logout

7. Click "Review" next to your email address



Action	Subject	Request	Date Opened
<b>Review</b>	Idapbind Cluster Administrator	Approve Access for Kubernetes Cluster Administrator	10/16/18 8:31 AM

8. Specify any justification for the "Justification" and click "Approve"



**Act on Request**

Justification

Its me

**Approve Request** **Deny Request**

9. Click on "Confirm Approval"



Act on Request

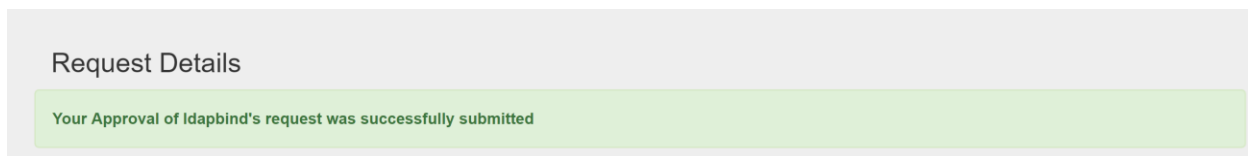
Justification

Its me

Confirm Approval

Cancel Approval

After clicking on Confirm Approval you should receive a successful submission confirmation:

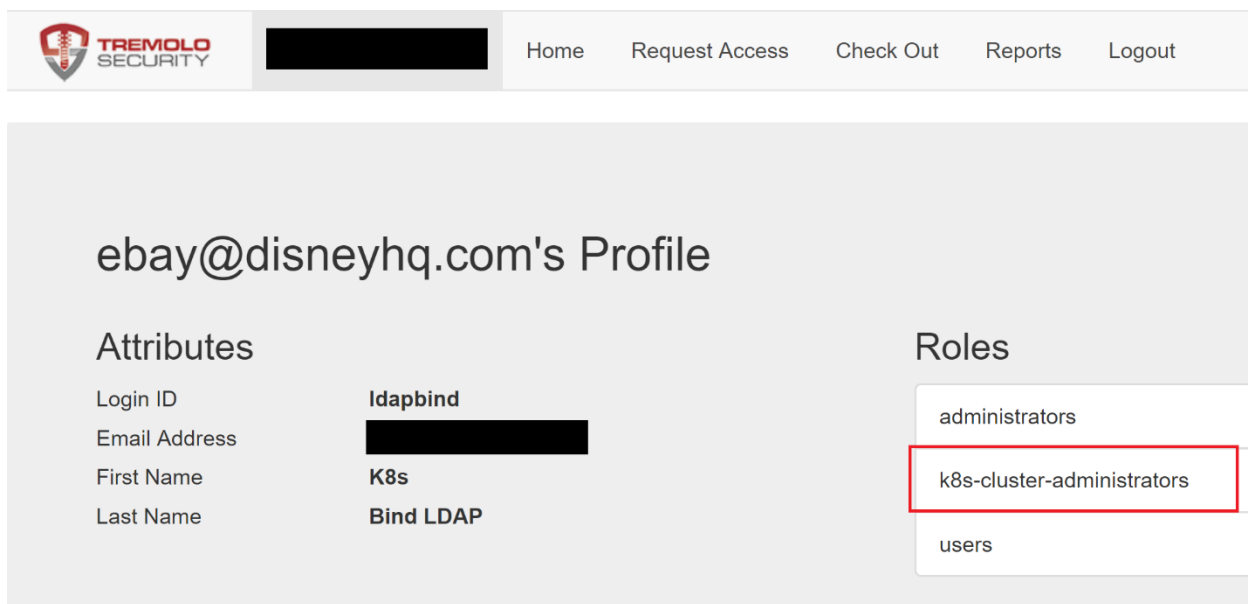


Request Details

Your Approval of Idapbind's request was successfully submitted

At this point you will be provisioned to the k8s-cluster-administrators group in the database that has a RoleBinding to the cluster-admin Role. **Logout of OpenUnison and log back in.** Just refreshing your browser will not work, you must logout completely and log back in.

If you click on your email address in the upper left, you'll see that you have the Role k8s-cluster-administrators.



TREMOLO SECURITY

Home Request Access Check Out Reports Logout

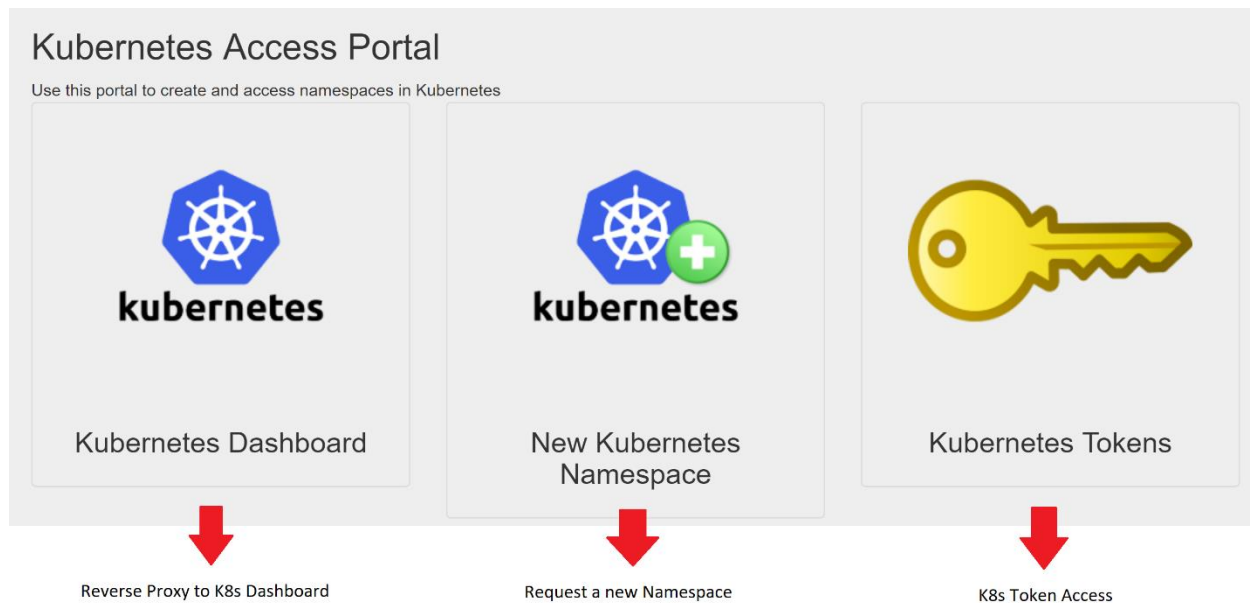
## ebay@disneyhq.com's Profile

Attributes		Roles
Login ID	Idapbind	administrators
Email Address	[redacted]	k8s-cluster-administrators
First Name	K8s	users
Last Name	Bind LDAP	

Congratulations, you now have an OIDC provider to handle authentication to your Kubernetes cluster using your LDAP directory.

## Using your Token

Now that you have Openunison running and you have approved your access, your main screen will show three options:



This document will not detail the new namespace request form, it may be added in the future if people find the initial document useful.

The first option is straight forward, its your link to the K8s dashboard using Openunison's reverse proxy. Again, out of scope for this initial document.

The third option is what I want to focus on, your tokens. Click the icon and you will be taken to a page that has your token information on it.

### Adding the Openunison Certificate to Your Client

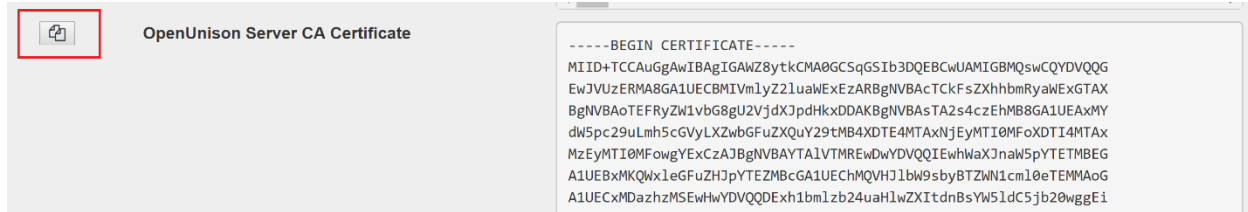
I will update this document for custom certificates, including your own internal CA signed certs, but for the initial release I am only covering using the web site certificate that is generated by the Openunison installation.

If you try to issue an API command at this point you will receive an error similar to the one shown below.

**Unable to connect to the server: Get https://unison.example.com:30654/auth/idp/k8sidp/.well-known/openid-configuration: x509: certificate signed by unknown authority**

To fix this, we need to trust the certificate for the Openunison server.

Log into your Openunison page and click on the token's icon and click the copy icon for the Openunison certificate.



On your client, create a new file called `ou-cert.crt` and paste the data in your clipboard into the file and save your file.

Next, you will need to add this certificate to your trusted CA store. This will vary based on your client base OS but for this example we will use CentOS 7.

Copy your certificate to `/etc/pki/tls/certs/`

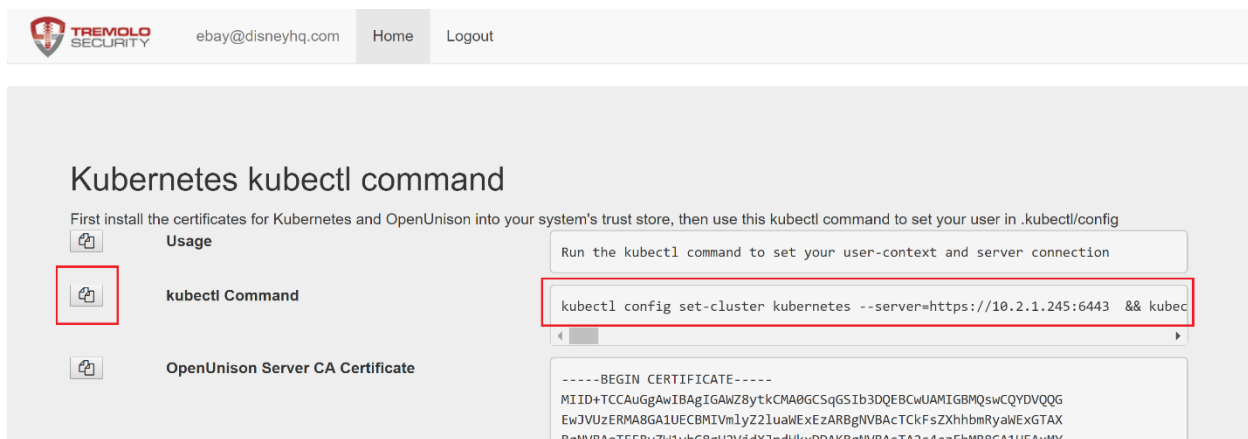
```
cp ou-cert.crt /etc/pki/tls/certs/
```

Then update the CA store by executing:

```
update-ca-trust
```

### Updating your Client Config File

The second section has a small button that will copy the command line to your clipboard so you can paste it into your kubectl client.



Click the copy button and paste it into your client machine and press enter.

You will receive a reply that the information was added to your config file and the context was set to the new context.

```
Cluster "kubernetes" set.
Context "kubernetes" created.
User "ldapbind" set.
Switched to context "kubernetes".
```



You will still have any original contexts in your config, to verify check your config contexts by using the config command in your client:

**kubectI config get-contexts**

CURRENT	NAME	CLUSTER	AUTHINFO	NAMESPACE
*	kubernetes	kubernetes	ldapbind	
	kubernetes-admin@kubernetes	kubernetes	kubernetes-admin	

Now you have a fully configured client configured to use OIDC with Kubernetes and secure LDAP.

## Appendix

Props properties:

Property	Description
OU_HOST	The host name for OpenUnison. This is what user's will put into their browser to login to Kubernetes
K8S_DASHBOARD_HOST	The host name for the dashboard. This is what users will put into the browser to access to the dashboard. <b>NOTE:</b> OU_HOST and K8S_DASHBOARD_HOST <b>MUST</b> share the same DNS suffix. Both OU_HOST and K8S_DASHBOARD_HOST <b>MUST</b> point to OpenUnison
K8S_URL	The URL for the Kubernetes API server
OU_COOKIE_DOMAIN	The DNS Domain for cookies generated by OpenUnison. This domain <b>MUST</b> contain both OU_HOST and K8S_DASHBOARD_HOST
OU_HIBERNATE_DIALECT	Hibernate dialect for accessing the database. Unless customizing for a different database do not change
OU_JDBC_DRIVER	JDBC driver for accessing the database. Unless customizing for a different database do not change
OU_JDBC_URL	The URL for accessing the database
OU_JDBC_USER	The user for accessing the database
OU_JDBC_PASSWORD	The password for accessing the database
OU_JDBC_VALIDATION	A query for validating database connections/ Unless customizing for a different database do not change
SMTP_HOST	Host for an email server to send notifications
SMTP_PORT	Port for an email server to send notifications
SMTP_USER	Username for accessing the SMTP server (may be blank)
SMTP_PASSWORD	Password for accessing the SMTP server (may be blank)
SMTP_FROM	The email address that messages from OpenUnison are addressed from
SMTP_TLS	true or false, depending if SMTP should use start tls

Property	Description
AD_BASE_DN	The search base for Active Directory
AD_HOST	The host name for a domain controller or VIP. If using SRV records to determine hosts, this should be the fully qualified domain name of the domain
AD_PORT	The port to communicate with Active Directory
AD_BIND_DN	The full distinguished name (DN) of a read-only service account for working with Active Directory
AD_BIND_PASSWORD	The password for the AD_BIND_DN
AD_CON_TYPE	ldaps for secure, ldap for plain text
SRV_DNS	If true, OpenUnison will lookup domain controllers by the domain's SRV DNS record
OU_CERT_OU	The OU attribute for the forward facing certificate
OU_CERT_O	The O attribute for the forward facing certificate
OU_CERT_L	The L attribute for the forward facing certificate
OU_CERT_ST	The ST attribute for the forward facing certificate
OU_CERT_C	The C attribute for the forward facing certificate
unisonKeystorePassword	The password for OpenUnison's keystore