

How to run a Webserver on Standalone Deployments

This guide does not come with support by Element. It is not part of the Element Server Suite (ESS) product. Use at your own risk. Remember you are responsible of maintaining this software stack yourself.

Some config options require a web content to be served. For example:

- Changing Element Web appearance with custom background pictures.
- Providing a HomePage for display in Element Web.
- Providing a Guide PDF from your server in an airgapped environment.

One way to provide this content is to run a web server in the same `microk8s` Kubernetes Cluster as the Element Enterprise Suite.

You should first consider using an existing webserver before installing and maintaining an additional webserver for these requirements.

The following guide describes the steps to setup the Bitnami Apache helm chart in the Standalone `microk8s` cluster setup by Element Server Suite.

Requirements:

- a DNS entry pages.BASEDOMAIN.
- a Certificate (private key + certificate) for pages.BASEDOMAIN
- an installed standalone Element Server Suite setup
- access to the server on the command line

Results:

- a web server that runs in the `microk8s` cluster
- a directory `/var/www/apache-content` to place and modify web content like homepage, backgrounds and guides.

This guide is applicable to the Single Node deployment of Element Server Suite but can be used for guidance on how to host a webserver in other Kubernetes Clusters as well.

You can use any webserver that you like, in this example we will use the Bitnami Apache chart.

We need helm version 3. You can follow [this Guide](#) or ask `microk8s` to install `helm3`.

Installing Prerequisites

Enabling Helm3 with microk8s

```
$ microk8s enable helm3
Infer repository core for addon helm3
Enabling Helm 3
Fetching helm version v3.8.0.
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload  Total   Spent    Left     Speed
100 12.9M  100 12.9M    0     0  17.4M    0 --:--:-- --:--:-- --:--:-- 17.4M
Helm 3 is enabled
```

Let's check if it is working

```
$ microk8s.helm3 version
version.BuildInfo{Version:"v3.8.0", GitCommit:"d14138609b01886f544b2025f5000351c9eb092e",
GitTreeState:"clean", GoVersion:"go1.17.5"}
```

Create and Alias for helm

```
echo alias helm=microk8s.helm3 >> ~/.bashrc
source ~/.bashrc
```

Enable the Bitnami Helm Chart repository

Add the bitnami repository

```
helm repo add bitnami https://charts.bitnami.com/bitnami
```

Update the repo information

```
helm repo update
```

Preparation and Configuration

Prepare the Web-Server Content

Create a directory to supply content:

```
sudo mkdir /var/www/apache-content
```

Create a homepage `home.html`, i.e.:

```
<h2 style="text-align:center"><br />
Welcome to the Element Chat Server.</h2>

<p style="text-align:center">You can find a <a href="https://static.element.io/pdfs/element-user-
guide.pdf">Getting Started Guide here</a></p>

<p style="text-align:center">Powered by&nbsp;<a href="https://matrix.org/">Matrix</a>, provided by <a
href="http://element.io">Element</a>.</p>

<p style="text-align:center"><a href="https://element.BASEDOMAIN/#/directory">Explore rooms</a></p>

<p style="text-align:center"><strong><span style="font-size:20px"><span style="color:#c0392b">Create a
Key Backup & Passphrase now!<br />
(see Getting Started Guite p. 5)</span></span></strong></p>
```

Put your content into the apache-content directory:

```
cp /tmp/background.jpg /apache-content/
cp /tmp/home.html ~element/apache-content/
```

There are multiple ways to provide this content to the apache pod. The bitnami helm chart user ConfigMaps, Physical Volumes or a Git Repository.

ConfigMaps are a good choice for smaller amounts of data. There is a hard limit of 1MiB on ConfigMaps. So if all your data is not more that 1MiB, the config map is a good choice for you.

Physical Volumes are a good choice for larger amounts of data. There are several choices for backing storage available. In the context of the standalone deployments of ESS a Physical Hostpath is the most practical. HostPath is not a good solution for mutli node k8s clusters, unless you pin a pod to a certain node. Pinning the pod to a single node would put the workload at risk, should that node go down.

Git Repository is a favourite as it versions the content and you track and revert to earlier states easily. The bitnami apache helm chart is built in a way that updates in regular intervals to your latest changes.

We are selecting the Physical Volume option to serve content in this case. Our instance of Microk8s comes with the Hostpath storage addon enabled.

Define the physical volume:

```
cat <<EOF>pv-volume.yaml
apiVersion: v1
```

```
kind: PersistentVolume
metadata:
  name: apache-content-pv
  labels:
    type: local
spec:
  storageClassName: microk8s-hostpath
  persistentVolumeReclaimPolicy: Retain
  capacity:
    storage: 100Mi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/var/www/apache-content"
EOF
```

Apply to the cluster

```
kubectl apply -f pv-volume.yaml
```

Next we need a Physical Volume Claim:

```
cat <<EOF>pv-claim.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: apache-content-pvc
spec:
  volumeName: apache-content-pv
  storageClassName: microk8s-hostpath
  accessModes: [ReadWriteOnce]
  resources: { requests: { storage: 100Mi } }
EOF
```

Apply to the cluster to create the pvc

```
kubectl apply -f pv-claim.yaml
```

Configure the Helm Chart

We need to add configurations to adjust the apache deployment to our needs. The K8s service should be switched to ClusterIP. The Single Node deployment includes an Ingress configuration through nginx that we can use to route traffic to this webserver. The name of the ingressClass is "public". We will need to provide a hostname. This name needs to be resolvable through DNS. This could be done through the wildcard entry for *.**BASEDOMAIN** that you might already have. You will need a certificate and certificate private key to secure this connection through TLS.

The full list of configuration options of this chart is explained in [the bitnami repository here](#)

Create a file called apache-values.yml in the home directory of your element user directory.

Remember to replace **BASEDOMAIN** with the correct value for your deployment.

```
cat <<EOF>apache-values.yml
service:
  type: ClusterIP
ingress:
  enabled: true
  ingressClassName: "public"
  hostname: pages.BASEDOMAIN
htdocsPVC: apache-content-pvc
EOF
```

Deployment

Deploy the Apache Helm Chart

Now we are ready to deploy the apache helm chart

```
helm install myhomepage -f apache-values.yml oci://registry-1.docker.io/bitnamicharts/apache
```

Manage the deployment

List the deployed helm charts:

```
$ helm list
NAME          NAMESPACE    REVISION    UPDATED                               STATUS    CHART          APP VERSION
myhomepage    default      1           2023-09-06 14:46:33.352124975 +0000 UTC    deployed  apache-10.1.0  2.4.57
```

Get more details:

```
$ helm status myhomepage  
NAME: myhomepage  
LAST DEPLOYED: Wed Sep  6 14:46:33 2023  
NAMESPACE: default  
STATUS: deployed  
REVISION: 1  
TEST SUITE: None  
NOTES:  
CHART NAME: apache  
CHART VERSION: 10.1.0  
APP VERSION: 2.4.57
```

**** Please be patient while the chart is being deployed ****

1. Get the Apache URL by running:

You should be able to access your new Apache installation through:
- <http://pages.lutz-gui.sales-demos.element.io>

If you need to update the deployment, modify the required `apache-values.yaml` and run :

```
helm upgrade myhomepage -f apache-values.yaml oci://registry-1.docker.io/bitnamicharts/apache
```

If you don't want the deployment any more, you can remove it.

```
helm uninstall myhomepage
```

Secure the deployment with certificates

If you are in a connected environment, you can rely on `cert-manager` to create certificates and secrets for you.

Cert-manager with letsencrypt

If you have `cert-manager` enabled. You will just need to add the right **annotations to the ingress** of your deployment. Modify your `apache-values.yaml` and add these lines to the ingress block :

```
tls: true  
annotations:  
  cert-manager.io/cluster-issuer: letsencrypt  
  kubernetes.io/ingress.class: public
```

You will need to upgrade your deployment to reflect these changes:

```
helm upgrade myhomepage -f apache-values.yaml oci://registry-1.docker.io/bitnamicharts/apache
```

Custom Certificates

There are situations in which you want custom certificates instead. These can be used by modifying your `apache-values.yaml`. Add the following lines to the ingress block in the `apache-values.yaml`. Take care to get the indentation right. Replace the ... with your data.

```
tls: true
extraTls:
- hosts:
  - pages.lutz-gui.sales-demos.element.io
  secretName: "pages.lutz-gui.sales-demos.element.io-tls"
secrets:
- name: pages.lutz-gui.sales-demos.element.io-tls
  key: |-
    -----BEGIN RSA PRIVATE KEY-----
    ...
    -----END RSA PRIVATE KEY-----
certificate: |-
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
```

You will need to upgrade your deployment to reflect these changes:

```
helm upgrade myhomepage -f apache-values.yaml oci://registry-1.docker.io/bitnamicharts/apache
```

Tips and Tricks

You can make your life easier by using bash completing and an alias for `kubectl`. You will need to have the bash-completion package installed as a prerequisite.

For all users on the system:

```
kubectl completion bash | sudo tee /etc/bash_completion.d/kubectl > /dev/null
```

Set an alias for `kubectl` for your user:

```
echo 'alias k=kubectl' >> ~/.bashrc
```

Enable auto-completion for your alias

```
echo 'complete -o default -F __start_kubectl k' >> ~/.bashrc
```

After reloading your Shell, you can now enjoy auto completion for your k (kubectl) commands.

Revision #6

Created 1 May 2024 15:30:01 by Kieran Mitchell Lane

Updated 6 November 2024 13:21:02 by Kieran Mitchell Lane