Appendices

- Preparing Element Server Suite PoC
- How to run a Webserver on Standalone Deployments
- Notifications, MDM & Push Gateway
- Verifying ESS releases against Cosign
- ESS CRDs support in ArgoCD
- Synapse database troubleshooting
- Auditbot troubleshooting

Preparing Element Server Suite PoC

Please reach out our Element Sales Team if you want to run a Proof of Concept for Element Server Suite.

Note This guide is for running Proof of Concepts. We don't aim to show every feature here, we want to get you up and running most quickly. This guide is focusing on connected standalone installations currently. There are scenarios currently not covered by this guide. Installing into airgapped / disconnected environments, or testing our Cloud Based offering.

A Proof-of-Concept is done in preparation of a subscription sale with the goal of demonstrating the required capabilities.

Create an account on element.io

Please create an account on element.io. We will enable this account as part of the PoC process and grant you access to the Element Server Suite software packages.

Communication via matrix room

The account team will create a matrix room to improve communication and invite you. To do this We will need your Matrix ID (MXID) to invite you.

If you don't already have a MXID, you can create one here by signing up. This will create an account on matrix.org, you can authenticate via several identity providers.

When you have a MXID, we recommend adding it to your EMS Account via `Your Account`, `Account`. You should then send this to the account team so they can add you to the room. You could use the Element Web Client that you used to create the account or install one of the Element Mobile apps from the App or Playstore.

PoC preparation

Element Server Suite can be installed in a Kubernetes Cluster or as a standalone installation on top of an Operating System (RHEL 8/9 or Ubuntu 20.04/22.04). Most Proof-of-Concept installations will select the Standalone Installation on top of a VM which we recommend for speed and ease of operation.

Click here for an overview of the Element Server Suite. Here is the link detailing the single node installation.

Preparation of the VM and Ports

Please set up a VM with **8 vCPUs** and **32GB RAM** and **100 GB Storage**. If this sounds like a lot of resources to you, the requirements do in fact vary and could be scaled down later if required. Install Ubuntu 20.04 LTS or RHEL8. Update the system to the latest available patches and create a user to be used for maintaining the Element Server Suite. See our documentation for this step here.

You will need to be able to reach the VM on Ports 80, 443 and 8443.

DNS Names and Certificates

You need to select a base domain for the Server. This can differ from the base domain of the matrix IDs but is often the same. Read more about this in the section on Matrix IDs and Well Known delegation below.

You have chosen eng.acme.com. The following DNS entries must be prepared and point to the external IP of the VM.

This results in the following hostnames for you :

- eng.acme.com (base domain - might already exist)
- matrix.eng.acme.com (the synapse homeserver)
- element.eng.acme.com (element web)
- admin.eng.acme.com (admin dashboard)
- integrator.eng.acme.com (integration manager)
- hookshot.eng.acme.com (Our integrations)

Optional for Monitoring and Integrations :

- grafana.eng.acme.com (Our Grafana server)

Optional for Video Chat with Jitsi :

- jitsi.eng.acme.com (Our VoIP platform)
- coturn.eng.acme.com (Our TURN server)

Optional for Video Chat with Element Call :

- call.acme.com (Element Call)
- sfu.acme.com (Selective Forwarding Unit)

Opitonal for Element X support :

- sliding-sync.acme.com

Optional for the Admin / Audit functionality :

- roomadmin.eng.acme.com
- audit.eng.acme.com

We require certificates for all these hostnames including the base domain to enable SSL/TLS encryption. The quick and easy way is to use the embedded letsencrypt. This is only available if you are in a connected environment. You can provide and use your own certificates.

Matrix IDs & Well Know delegation

Matrix IDs have the following format :

@USER:SERVER

In our example case the matrix server is matrix.eng.acme.com. If a user Tom Maier has a username **tmaier** in your LDAP, this would lead to an MXID **@tmaier:matrix.eng.acme.com**. This is often not desired as we like to keep the MXIDs short. It is more elegant to drop the "matrix" host name from the MXIDs. Tom's MXID would look like this **@tmaier:eng.acme.com** .

In order to be able to offer matrix IDs with the base domain, we recommend setting up a reverse proxy on eng.acme.com, which forwards https://eng.acme.com/.well-known/matrix/ to the matrix/synapse server on https://matrix.eng.acme.com/.well-known/matrix . Or you shorten the hostname part of your MXIDs even more to acme.com, this would require you to put the reverse proxy onto acme.com.

The configuration on your Apache WebServer should be similar to this :

```
ProxyPass /.well-known/matrix/ https://matrix.eng.acme.com/.well-known/matrix/
ProxyPassReverse /.well-known/matrix/ https://matrix.eng.acme.com/.well-known/matrix/
ProxyPreserveHost On
```

More about well-known and MXIDs can be found in our Upstream Documentation here and here. Further configurations can be made using the well-known mechanism. An example is documented here.

Authentication and Postgres DB

The quickest setup is using local authentication and users only. This is what we recommend in a Proof-of-Concept situation. User accounts are created in the local Postgresql DB (recommended only up to 300 users) through our Admin UI or through API scripts for automation in this case. We support many mechanisms for AUthentication like LDAP, SAML2 and OIDC. We recommend to configure these as a 2nd step only if required.

You have the option to use an internal or external Postgres DB. We do recommend to use the internal Postgres DB for Proof-of-Concept installations. The internal Postgres DB is only available when you are opting for the Standalone Installation on top of an Operating System. You will need an external Postgres DB when installing into an existing Kubernetes cluster.

Checklist before starting the installation

Please prepare the above items before starting the installation. Make sure you have :

- created and communicated your MXID to the Element Sales Team
- registered an account on element.io
- created and prepared your vm / machine with enough resources
- created DNS entries
- decided on letsencrypt / created host certificates for your hostnames
- installed the reverse proxy on the webserver of your MXID URL e.g. eng.acme.com or acme.com

Don't hesitate to reach out to your Element Sales Team for support. We are here to guide you.

How to run a Webserver on Standalone Deployments

This guide is 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 Kubernetes Cluster as the Element Enterprise Suite.

Please consider other options before installing and maintaining just another webserver for this.

Consider to use an existing web server 1st.

The following guide describes the steps to setup the Bitnami Apache helm chart in the Standalone Microk8s cluster setup by Element Server Suite..

You need:

- 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

You get:

- a web server that runs in the mircok8s cluster
- a directory /var/www/apache-content to place and modify web content like homepage, backgrounds and guides.

You can deploy a Webserver to the same Kubernetes cluster that Element Server Suite is using. 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 user the Bitnami Apache chart.

We need helm version 3. You can follow this Guide or ask microk8s to install helm3.

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
```

Prepare the Web-Server Content

Create a directory to supply content :

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

Put your content e.g. a homepage 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.yaml
service:
  type: ClusterIP
ingress:
  enabled: true
```

```
  ingressClassName: "public"
  hostname: pages.BASEDOMAIN
htdocsPVC: apache-content-pvc
EOF
```

Deploy the Apache Helm Chart

Now we are ready to deploy the apache helm chart

```
helm install myhomepage -f apache-values.yaml 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 you 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 aias 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.

Notifications, MDM & Push Gateway

The stock Android and iOS Apps will use an Element owned Push Gateway to send Notification via the Apple or Google Notifiction Services.

The URL of our push gateway is **https://matrix.org/_matrix/push/v1/notify**

The apps will on startup register with the Google or Apple Notification Services (APNs) and request a push_notification_client_identifier. If notifications need sending, the homeserver will use the configured Push Gateway to sent notification through the APNs.

What is a Notification?

A notification will not contain sensitive content. This is what notificatons actually look like :

```
▿ 5 elements
  ▿ 0 : 2 elements
    ▿ key : AnyHashable("unread_count")
      - value : "unread_count"
    - value : 1
  ▿ 1 : 2 elements
    ▿ key : AnyHashable("pusher_notification_client_identifier")
      - value : "pusher_notification_client_identifier"
    - value : ad0bd22bb90fabde45429b3b79cdbba12bd86f3dafb80ea22d2b1343995d8418
  ▿ 2 : 2 elements
    ▿ key : AnyHashable("aps")
      - value : "aps"
    ▿ value : 2 elements
      ▿ 0 : 2 elements
        - key : alert
        ▿ value : 2 elements
          ▿ 0 : 2 elements
            - key : loc-key
            - value : Notification
          ▿ 1 : 2 elements
            - key : loc-args
            - value : 0 elements
      ▿ 1 : 2 elements
```

```
      - key : mutable-content
      - value : 1
  ▿ 3 : 2 elements
    ▿ key : AnyHashable("room_id")
      - value : "room_id"
    - value : !vkibNVqwhZVOaNskRU:matrix.org
  ▿ 4 : 2 elements
    ▿ key : AnyHashable("event_id")
      - value : "event_id"
    - value : $0cTr40iZmOd3Aj0c65e_7F6NNVF_BwzEFpyXuMEp29g
```

We recommend that you use the stock Element Apps from PlayStore or Applestore together with the Push Gateway that we as Element host.

Mobile Device Management (MDM)

You can use Mobile Device Management to configure and roll out Mobil Applications. To be able to configure mobile apps this way, the app needs to implement certain interfaces in a standard way. This is called AppConfig.

The Android Element App does not support AppConfig currently. You will need to rebuild the apk to include changes like a different homeserver or a diffrent pusherURL.

The iOS Element App got enabled for AppConfig in version 1.11.2. this allows the change of the following parameters and keys without the need to recompile the app.

- im.vector.app.serverConfigDefaultHomeserverUrlString
- im.vector.app.clientPermalinkBaseUrl
- im.vector.app.serverConfigSygnalAPIUrlString

If you employ a Mobile Device Management solution like e.g. VmWare Workspace One, you will need to configure your iOS Element app with these keys as documented here in section **Publish and update Managed AppConfig for your app in Workspace ONE**.

Depending on the brand of MDM you are using, you can create the required keys manually, or enable these setting with an XML file. The XML file might look like this :

```
<managedAppConfiguration>
    <version>1</version>
    <bundleId>im.vector.app</bundleId>
    <dict>
        <string keyName="im.vector.app.serverConfigDefaultHomeserverUrlString">
            <defaultValue>
```

```
            <value>https://matrix.BASEDOMAIN</value>
        </defaultValue>
    </string>
    <string keyName="im.vector.app.clientPermalinkBaseUrl">
        <defaultValue>
            <value>https://messenger.BASEDOMAIN</value>
        </defaultValue>
    </string>
  </dict>
</managedAppConfiguration>
```

Using your own Push Gateway (Sygnal)

Some organization still feel uncomfortable with using our Push Gateway. You are able to use your own push gateway (e.g. Sygnal) if you want.

You can install Sygnal as an integration with the Element Server Suite.

During the App Upload process a private key is created. We as Element Company retain and use that key on our Push infrastructure. This is why you can not use the stock Element Apps, but will need to upload your own version of the Element App. This will give you access to your own private notification key that is bound to the app you uploaded.

You will need to configure your Sygnal with the private key of your Element App.

You will need to set the "im.vector.app.serverConfigSygnalAPIUrlString" for the iOS App or the equilivant in the Android App Source code.

Verifying ESS releases against Cosign

Cosign ESS Verification Key

ESS does not use Cosign transaction log to be able to support airgapped deployment. We are instead relying on a public key that you can ask if you need to run image verification in your cluster.

The ESS Cosign public key is the following one :

```
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE1Lc+7BqkqD+0XYft05CeXto/Ga1Y
DKNk3o48PIJ2JMrq3mzw13/m5rzlGjdgJCs6yctf4+UdACZx5WSiIWTFbQ==
-----END PUBLIC KEY-----
```

Verifying manually

To verify a container against ESS Keys, you will have to run the following command :

- Operator : `cosign verify registry.element.io/ess-operator:<version> --key cosign.pub`
- Updater : `cosign verify registry.element.io/ess-updater:<version> --key cosign.pub`

If you are running in an airgapped environment, then you will need to append `--insecure-ignore-tlog=true` to the above commands

Verifying automatically

You will have to setup and configure your SIGStore Admission Policy to use ESS Public Key.

ESS CRDs support in ArgoCD

ArgoCD can support getting the ESS CRDs Status as resource health using Custom Health Checks

You need to configure the following under the configmap `argocd-cm` of argocd :

```
data:
  resource.customizations: |
    matrix.element.io/*:
      health.lua: |
        hs = {}
        if obj.status ~= nil then
          if obj.status.conditions ~= nil then
            for i, condition in ipairs(obj.status.conditions) do
              if condition.type == "Failure" and condition.status == "True" then
                hs.status = "Degraded"
                hs.message = condition.message
                return hs
              end
              if condition.type == "Running" and condition.status == "True" and condition.reason ~= "Successful"
then
                hs.status = "Progressing"
                hs.message = condition.message
                return hs
              end
              if condition.type == "Available" and condition.status == "True" then
                hs.status = "Healthy"
                hs.message = condition.message
                return hs
              end
              if condition.type == "Available" and condition.status == "False" then
                hs.status = "Degraded"
                hs.message = condition.message
                return hs
              end
              if condition.type == "Successful" and condition.status == "True" then
                hs.status = "Healthy"
                hs.message = condition.message
                return hs
```

```
        end
      end
    end
  end

  hs.status = "Progressing"
  hs.message = "Waiting for the CR to start to converge..."
  return hs
EOT
```

Synapse database troubleshooting

Room Retention policy enabled causes Synapse database to consume a lot of disk space

1. Run the following command against synapse postgres database : `\d+`. On an installer-managed postgresql, you can access psql command using : `kubectl exec -it -n element-onprem synapse-postgres-0 -- bash -c 'psql "dbname=$POSTGRES_DB user=$POSTGRES_USER password=$POSTGRES_PASSWORD"'`
2. Check the space taken by the table `state_groups_state`. For example, here it's consuming 540 GB : `public | state_groups_state | table | synapse_user | permanent | 244 GB |`
3. If you have Room retention policy enabled, there's a bug which causes some state groups to be orphaned, and as a consequence they are not cleaned up from the database automatically.
4. Follow the instruction from the page synapse-find-unreferenced-state-groups. The tool is available for download in the following link rust-synapse-find-unreferenced-state-groups.
5. On Standalone and Installer-managed postgresql database, you can use the following script to do it automatically. It's going to involve a downtime because Synapse is stopped before cleaning up orphaned state groups. **Please make sure that you have appropriate disk space before running the script because the script generates a backup of the database before cleaning up the tables.**. If you need to restore, the command will be `kubectl exec -it pods/synapse-postgres-0 -n element-onprem -- psql "dbname=synapse user=synapse_user password=$POSTGRES_PASSWORD" < /path/to/dump.sql` :

```
#!/bin/sh
set -e

echo "Stopping Operator..."
kubectl scale deploy/element-operator-controller-manager -n operator-onprem --replicas=0
echo "Stopping Synapse..."
kubectl delete synapse/first-element-deployment -n element-onprem

while kubectl get statefulsets -n element-onprem --no-headers -o custom-columns=":metadata.labels" | grep -q "matrix-server"; do
  echo "Waiting for synapse StatefulSets to be deleted..."
  sleep 2
done

echo "Forwarding postgresql port..."
kubectl port-forward pods/synapse-postgres-0 -n element-onprem   15432:5432 &
port_forward_pid=$!
sleep 1s
```

```
POSTGRES_PASSWORD=`echo $(kubectl get secrets/first-element-deployment-synapse-secrets -n element-onprem -o yaml | grep postgresPassword | cut -d ':' -f2) | base64 -d`
POSTGRES_USER=synapse_user
POSTGRES_DB=synapse

echo "Find unreferenced state groups..."
./rust-synapse-find-unreferenced-state-groups -p postgres://$POSTGRES_USER:$POSTGRES_PASSWORD@localhost:15432/$POSTGRES_DB -o ./sgs.txt
kill -9 $port_forward_pid

echo "Copy unreferenced state groups list to postgres pod..."
kubectl cp ./sgs.txt -n element-onprem synapse-postgres-0:/tmp/sgs.txt

echo "Backing up postgres database..."
kubectl exec -it pods/synapse-postgres-0 -n element-onprem -- bash -c 'pg_dump "dbname=$POSTGRES_DB user=$POSTGRES_USER password=$POSTGRES_PASSWORD"' > backup-unreferenced-state-groups-$(date '+%Y-%m-%d-%H:%M:%S').sql

echo "Cleanup postgres database..."
kubectl exec -it pods/synapse-postgres-0 -n element-onprem -- psql "dbname=$POSTGRES_DB user=$POSTGRES_USER password=$POSTGRES_PASSWORD" -c "CREATE TEMPORARY TABLE unreffed(id BIGINT PRIMARY KEY); COPY unreffed FROM '/tmp/sgs.txt' WITH (FORMAT 'csv'); DELETE FROM state_groups_state WHERE state_group IN (SELECT id FROM unreffed); DELETE FROM state_group_edges WHERE state_group IN (SELECT id FROM unreffed); DELETE FROM state_groups WHERE id IN (SELECT id FROM unreffed);"
echo "Starting Operator..."
kubectl scale deploy/element-operator-controller-manager -n operator-onprem --replicas=1
```

Running the script should look like this :

```
 bash cleanup-sgs.sh
Stopping Operator...
deployment.apps/element-operator-controller-manager scaled
Stopping Synapse...
synapse.matrix.element.io "first-element-deployment" deleted
Waiting for synapse StatefulSets to be deleted...
Waiting for synapse StatefulSets to be deleted...
Waiting for synapse StatefulSets to be deleted...
```

```
Waiting for synapse StatefulSets to be deleted...
Waiting for synapse StatefulSets to be deleted...
Waiting for synapse StatefulSets to be deleted...
Waiting for synapse StatefulSets to be deleted...
Forwarding postgresql port...
Forwarding from 127.0.0.1:15432 -> 5432
Forwarding from [::1]:15432 -> 5432
Find unreferenced state groups...
Handling connection for 15432
  [0s] 741 rows retrieved
Fetched 725 state groups from DB
Total state groups: 725
Found 2 unreferenced groups
Copy unreferenced state groups list to postgres pod...
Defaulted container "postgres" out of: postgres, postgres-init-password, postgres-exporter
cleanup-sgs.sh: line 28: 428645 Killed                  kubectl port-forward pods/synapse-postgres-0 -n element-
onprem 15432:5432
Backing up postgres database...
Cleanup postgres database...
Defaulted container "postgres" out of: postgres, postgres-init-password, postgres-exporter
DELETE 2
Starting Operator...
deployment.apps/element-operator-controller-manager scaled
```

Auditbot troubleshooting

Auditbot Viewing Error - Bad MAC

This is a symptom that Auditbot Secure Storage got corrupted. It can happen if you try to change the passphrase of Auditbot through the UI for example.

> This procedure will make rooms history unable to decrypt in auditbot UI. The rooms history will still be available in audit logs generated by auditbot, in the S3 or file storage.

To resolve you will need to reset the 4S passphrase of auditbot.

1. Stop the operator and edit the auditbot `pipe`:

```
kubectl scale deploy/element-operator-controller-manager -n operator-onprem --replicas=0
kubectl edit statefulsets.apps first-element-deployment-auditbot-pipe -n element-onprem
```

2. Add the following under `env`:

```
- name: AUDIT_FORCE_NEW_SSSS
  value: "true"
```

Wait for the pipe to restart, check its logs, and check that you can log in through the Admin Console.

3. Edit the Statefulset again:

```
kubectl edit statefulsets.apps first-element-deployment-auditbot-pipe -n element-onprem
```

4. Remove the env variable you added:

```
- name: AUDIT_FORCE_NEW_SSSS
  value: "true"
```

5. Restart the operator :

```
kubectl scale deploy/element-operator-controller-manager -n operator-onprem --replicas=1
```

This will restart auditbot and it normal functionality should be restored.