Element On-Premise Documentation

How to Install a POC Environment

Overview

Our Element Enterprise PoC Installer can handle the installation of Element Proof of Concept (POC) environments. Our standard POC environment is a single node server with microk8s running that we deploy our Element Enterprise Operator to, resulting in a fully functioning Synapse server with Element Web that can be used to conduct a POC. On-premise production deployments use the same installer and operator, but are intended to be deployed into a full kubernetes environment.

POC installations are not intended to be run for production purposes. You should plan on having a different installation for your production environment. The settings that you use with the installer will carry over for your production install, but your rooms and spaces will not.

To get started with a POC installation, there are several things that need to be considered and this guide will work through them:

- Hostnames/DNS
- Machine Size
- Operating System
- Users
- Network Specifics
- Postgresql Database
- TURN Server
- SSL Certificates
- Extra configuration items

Once these areas have been covered, you'll be able to install a POC environment!

Hostnames/DNS

You will need hostnames for the following pieces of infrastructure:

- Element Server (Required)
- Synapse Server (Required)

- Dimension Server (Required if you plan to use hookshot)
- Hookshot Server (Required if you need jira, gitlab, or github integrations)

These hostnames must resolve to the appropriate IP addresses. If you have a proper DNS server with records for these hostnames in place, then you will be good to go.

`/etc/hosts` may be used as an alternative to proper DNS in a POC scenario only. In this case, you will need entries similar to:

```
192.168.122.39 element.local element
192.168.122.39 synapse.local synapse
192.168.122.39 dimension.local dimension
192.168.122.39 hookshot.local hookshot
192.168.122.39 local
```

In the absence of proper DNS, for this to work in microk8s, you will also need to add the following to your parameters.yml: *(This was added in installer version 2022-05.03. If you have an installer prior to this and need this functionality, please update.)*

```
host_aliases:
  - ip: "192.168.122.39"
    hostnames:
      - "element.local"
      - "synapse.local"
      - "hookshot.local"
      - "dimension.local"
      - "local"
```

Machine Size

For running a proof of concept with our installer, we support only the x86_64 architecture and recommend the following minimums:

- No federation: 4 vCPUs/CPUS and 16GB RAM
- Federation: 8 vCPUs/CPUS and 32GB RAM

Operating System

To get started, we have tested on Ubuntu 20.04 and Red Hat Enterprise Linux 8.5 and suggest that you start there as well. For x86_64, you can grab an Ubuntu iso here:

https://releases.ubuntu.com/20.04.3/ubuntu-20.04.3-live-server-amd64.iso

or you can get Red Hat Enterprise Linux 8 with a Developer Subscription

https://developers.redhat.com/content-gateway/file/rhel-8.5-aarch64-dvd.iso

Note that future references in this document to `EL` reference Enterprise Linux.

Ubuntu Specific Directions

Make sure to select docker as a package option. Do set up ssh.

Once you log in, please run:

```
sudo apt-get update
sudo apt-get upgrade
```

EL Specific directions

Make sure to select "Container Management" in the "Additional Software" section.

Once you log in, please run:

```
sudo yum update -y
sudo yum install podman-docker python39-pip -y
sudo yum install
https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm -y
sudo alternatives --set python3 /usr/bin/python3.9
```

Add the following lines to `/etc/security/limits.conf`:

```
*               soft    nofile  100000
*               hard    nofile  100000
```

Further Pre-requisites

You should have the installer unpacked in a directory on your server. We will refer to this as the installer directory. You will also need to create a configuration directory that we will call the config directory. Both the `parameters.yml` and `secrets.yml` file live in the config directory.

To create the configuration directory, run the following:

```
mkdir ~/.element-onpremise-config
```

Please run the following commands to create the `/mnt/data` directory and install the `python3-signedjson` and `pwgen` packages which will be used during the configuration of the installer.

The `/mnt/data` directory should have at least 50 GB of space.

```
sudo mkdir /mnt/data
sudo mkdir /mnt/data/synapse-media
```

If you will be letting the installer install the Postgres database for you, also do:

```
sudo mkdir /mnt/data/synapse-postgres
```

Ubuntu:

```
sudo apt-get install python3-signedjson pwgen -y
```

EL:

```
sudo yum install make gcc python39-devel pwgen -y
```

EL: (as a normal user)

```
pip3 install signedjson --user
```

Network Specifics

Element Enterprise On-Premise needs to bind and serve content over:

- Port 80 TCP
- Port 443 TCP

microk8s needs to bind and serve content over:

- Port 16443 TCP
- Port 10250 TCP

- Port 10255 TCP
- Port 25000 TCP
- Port 12379 TCP
- Port 10257 TCP
- Port 10259 TCP
- Port 19001 TCP

For more information, see https://microk8s.io/docs/ports.

In a default Ubuntu installation, these ports are allowed through the firewall. You will need to ensure that these ports are passed through your firewall.

For EL, you need to explicitly open the above ports and enabling masquerading:

```
sudo firewall-cmd --add-service={http,https} --permanent
sudo firewall-cmd --add-port=16443/tcp --add-port=10250/tcp --add-port=10255/tcp --add-port=25000/tcp --add-port=12379/tcp --add-port=10257/tcp --add-port=10259/tcp --add-port=19001/tcp --permanent
sudo firewall-cmd --add-masquerade --permanent
sudo firewall-cmd --reload
```

Further, you need to make sure that your host is able to access the following hosts on the internet:

- api.snapcraft.io
- *.snapcraftcontent.com
- gitlab.matrix.org
- gitlab-registry.matrix.org
- pypi.org
- docker.io
- *.docker.com
- get.helm.sh

Further, you will also need to make sure that your host can access your distributions' package repositories. As these hostnames can vary, it is beyond the scope of this documentation to enumerate them.

Network Proxies

We also cover the case where you need to use a proxy to access the internet. Please make sure that the following host variables are set:

Ubuntu Specific Directions

If your company's proxy is `http://corporate.proxy:3128`, you would edit `/etc/environment` and add the following lines:

```
HTTPS_PROXY=http://corporate.proxy:3128
HTTP_PROXY=http://corporate.proxy:3128
https_proxy=http://corporate.proxy:3128
http_proxy=http://corporate.proxy:3128
NO_PROXY=10.1.0.0/16,10.152.183.0/24,127.0.0.1
no_proxy=10.1.0.0/16,10.152.183.0/24,127.0.0.1
```

The IP Ranges specified to `NO_PROXY` and `no_proxy` are specific to the microk8s cluster and prevent microk8s traffic from going over the proxy.

EL Specific Directions

Using the same example of having a company proxy at `http://corporate.proxy:3128`, you would edit `/etc/profile.d/http_proxy.sh` and add the following lines:

```
export HTTP_PROXY=http://corporate.proxy:3128
export HTTPS_PROXY=http://corporate.proxy:3128
export http_proxy=http://corporate.proxy:3128
export https_proxy=http://corporate.proxy:3128
export NO_PROXY=10.1.0.0/16,10.152.183.0/24,127.0.0.1
export no_proxy=10.1.0.0/16,10.152.183.0/24,127.0.0.1
```

The IP Ranges specified to `NO_PROXY` and `no_proxy` are specific to the microk8s cluster and prevent microk8s traffic from going over the proxy.

In Conclusion

You will need to log out and back in for the environment variables to be re-read after setting them. If you already have microk8s running, you will need to issue:

```
microk8s.stop
microk8s.start
```

to have it reload the new environment variables.

If you need to use an authenticated proxy, then the URL schema for both EL and Ubuntu is as follows:

`protocol://user:password@host:port`

So if your proxy is `corporate.proxy` and listens on port 3128 without SSL and requires a username of `bob` and a password of `inmye1em3nt` then your url would be formatted:

`http://bob:inmye1em3nt@corporate.proxy:3128`

For further help with proxies, we suggest that you contact your proxy administrator or operating system vendor.

Users

The installer requires that you run it as a non-root user who has sudo permissions. Please make sure that you have a user who can use `sudo`. If you wanted to make a user called `element-demo` that can use `sudo`, the following commands (run as root) would achieve that:

On Ubuntu:

```
useradd element-demo
gpasswd -a element-demo sudo
```

On EL:

```
useradd element-demo
gpasswd -a element-demo wheel
```

Unpacking the Installer

Please make sure that you unpack `element-enterprise-installer` onto your POC system. The directory that it unpacks into will be referenced in this document as the installer directory.

Postgresql Database

The installation requires that you have a postgresql database with a locale of C and UTF8 encoding set up. See https://github.com/matrix-org/synapse/blob/develop/docs/postgres.md#set-up-database for further details.

If you have this already, please make note of the database name, user, and password as you will need these to begin the installation.

If you do not already have a database, then the PoC installer will set up PostgreSQL on your behalf.

TURN Server

For installations in which you desire to use video conferencing functionality, you will need to have a TURN server installed and available for Element to use.

If you do not have an existing TURN server, we recommend installing `coturn`. Instructions on how to do that are available here: https://github.com/matrix-org/synapse/blob/master/docs/turn-howto.md (Note: On EL, you can do `yum install coturn -y`.)

Under "Synapse Setup" in the above instructions, you'll see what to change on the config. With the installer, you can create a file called `synapse/turn.yml` in your config directory and put the following in it:

```
turn_uris: [ "turn:turn.matrix.org?transport=udp", "turn:turn.matrix.org?transport=tcp" ]
turn_shared_secret: "n0t4ctuAllymatr1Xd0TorgSshar3d5ecret4obvIousreAsons"
turn_user_lifetime: 86400000
turn_allow_guests: True
```

based on how you installed the TURN server. This will allow the installer to configure synapse to use your TURN server.

A few notes on TURN servers:

- The TURN server has to be directly accessible by end-users. Normally this means a public IP, however if all the end-users are going to be on a VPN/private network then they just need to be able to access the private IP of the TURN server.
- The only reason to have TURN on a private network is if the private network disallows user <-> user traffic and only allows user <-> TURN server traffic. If user <-> user is allowed within the private network then a TURN server isn't needed.

SSL Certificates

For SSL Certificates, you have three options:

- Signed PEM encoded certificates from an internet recognized authority.
- Signed PEM encoded certificates from an internal to your company authority.
- LetsEncrypt
- Self-signed certificates

In the case of Signed certificates or LetsEncrypt, your hostnames must be accessible on the internet.

In the case of self-signed certificates, these are acceptable for a PoC environment, but will not be supported in a production environment as the security risk would be too high. Configuring mobile clients and federation will not be possible with self-signed certificates.

You will need to configure certificates for the following names:

- fqdn.tld
- element.fqdn.tld
- synapse.fqdn.tld
- dimension.fqdn.tld
- hookshot.fqdn.tld

Using our example hosts, this would mean that we need certificates for:

- local
- element.local
- synapse.local
- dimension.local
- hookshot.local

Certificates without LetsEncrypt

If you have certificates for all of the aforementioned host names, then you can simply place the `.crt` and `.key` files in the certs directory under the installer directory. Certificates in the certs directory must take the form of `fqdn.cert` and `fqdn.key`.

Self-signed certificates with mkcert

The following instructions will enable you to use a tool called mkcert to generate self-signed certificates. Element nor Canonical ship this tool and so these directions are provided as one example of how to get self-signed certificates.

Ubuntu:

```
sudo apt-get install wget libnss3-tools
```

EL:

```
sudo yum install wget nss-tools -y
```

Both EL and Ubuntu:

```
wget
https://github.com/FiloSottile/mkcert/releases/download/v1.4.3/mkcert-v1.4.3-linux-amd64
sudo mv mkcert-v1.4.3-linux-amd64 /usr/bin/mkcert
sudo chmod +x /usr/bin/mkcert
```

Once you have mkcert executable, you can run:

```
mkcert -install
The local CA is now installed in the system trust store! ⚡
```

Now, you can verify the CA Root by doing:

```
mkcert -CAROOT
/home/element-demo/.local/share/mkcert
```

Your output may not be exactly the same, but it should be similar. Once we've done this, we need to generate self-signed certificates for our hostnames. The following is an example of how to do it for `element.local`. You will need to do this for all of the aforementioned hostnames, including the `fqdn.tld`.

The run for the element fqdn looks like this:

```
mkcert element.local element 192.168.122.39 127.0.0.1

Created a new certificate valid for the following names
- "element.local"
- "element"
- "192.168.122.39"
- "127.0.0.1"

The certificate is at "./element.local+3.pem" and the key at
"./element.local+3-key.pem"

It will expire on 1 May 2024
```

Once you have self-signed certificates, you need to copy them into the certs directory under the config directory. Certificates in the certs directory must take the form of `fqdn.crt` and `fqdn.key`.

Using our above example, these are the commands we would need to run from the installer directory: (We ran `mkcert` in that directory as well.)

```
mkdir ~/.element-onpremise-config/certs
cp element.local+3.pem  ~/.element-onpremise-config/certs/element.local.crt
cp element.local+3-key.pem  ~/.element-onpremise-config/certs/element.local.key
cp synapse.local+3.pem  ~/.element-onpremise-config/certs/synapse.local.crt
cp synapse.local+3-key.pem  ~/.element-onpremise-config/certs/synapse.local.key
cp dimension.local+3.pem  ~/.element-onpremise-config/certs/dimension.local.crt
```

```
cp dimension.local+3-key.pem  ~/.element-onpremise-config/certs/dimension.local.key
cp hookshot.local+3.pem  ~/.element-onpremise-config/certs/hookshot.local.crt
cp hookshot.local+3-key.pem  ~/.element-onpremise-config/certs/hookshot.local.key
cp local+2.pem  ~/.element-onpremise-config/certs/local.crt
cp local+2-key.pem  ~/.element-onpremise-config/certs/local.key
```

Certificates with LetsEncrypt

Our installer also supports using LetsEncrypt to build certificates for your host names and automatically install them into your environment. If your hosts are internet accessible, this is the easiest method and only requires an admin email address to provide to LetsEncrypt.

parameters.yml

Now it is time to set `parameters.yml`. A sample has been provided and to get started, it is easiest to do:

```
cp config-sample/parameters.yml.sample ~/.element-onpremise-config/parameters.yml
```

Using the example hostnames of `element.local` and `synapse.local` (not resolvable on the internet), we would set the following parameters first in `parameters.yml`:

```
domain_name: local
element_fqdn: element.local
synapse_fqdn: synapse.local
```

Next, we need to set the variables related to Postgres. If you do not have an existing Postgres server, do not make any changes. If you have an existing Postgres server, set the following:

```
postgres_create_in_cluster: false
postgres_fqdn: `Postgres Server`
postgres_user: `Postgres User`
postgres_db: `Postgres Database for Element`
```

The next item in the configuration is the microk8s DNS resolvers. By default, the installer will use Google's publicly available DNS servers. If you have defined your hosts on a non-publicly available DNS server, then you should use your DNS servers instead of the publicly available Google DNS servers. Let's assume that your local dns servers are 192.168.122.253 and 192.168.122.252. To use those servers, you would need to add this line:

```
microk8s_dns_resolvers: "192.168.122.253,192.168.122.252"
```

The next section pertains to certmanager. If you are using your own certificates, please leave these items both blank, as such:

```
certmanager_issuer:
certmanager_admin_email:
```

If you have chosen to use letsencrypt, please specify “letsencrypt” for the certmanager_issue and an actual email address for who should manage the certificates for certmanager_admin_email:

```
certmanager_issuer: 'letsencrypt'
certmanager_admin_email: 'admin@mydomain.com'
```

secrets.yml

Now we move on to configuring `secrets.yml`. You will need the following items here:

- A Macaroon key
- Your postgres password for the user specified in parameters.yml
- A Registration Shared Secret
- A signing Key
- An EMS Image Store username and token, which will have been provided to you by Element.

To build a `secrets.yml` with the macaroon key, the registration shared secret, the generic shared secret, and the signing key already filled in, please run:

```
sh build_secrets.sh
mv secrets.yml  ~/.element-onpremise-config/
```

If you are using your own Postgres server, you will need to uncomment and fill in the `postgres_passwd`. If you are letting the installer install Postgres for you, then you will need to set a random password. You can generate a random password with:

```
pwgen 32 1
```

and then insert that value in the `postgres_passwd` field, making sure that you uncomment the line.

Do not forget to also set the values for `ems_image_store_username` and `ems_image_store_token`, which will both be provided by Element.

If you have a paid docker hub account, you can specify your username and password to avoid being throttled in the `dockerhub_username` and `dockerhub_token` fields. This is optional.

Extra Configuration Items

It is possible to configure anything in Synapse's homeserver.yaml or Element’s config.json.

To do so, you need to create json or yaml files in the appropriate directory under the config directory. These files will be merged to the target configuration file.

Samples are available in `config-sample` under the installer directory.

To configure synapse:

- Create a directory `synapse` at the root of the config directory : `mkdir ~/.element-onpremise-config/synapse`
- Copy the configurations extensions you want to setup from `config-sample/synapse` to `~/.element-onpremise-config/synapse`.
- Edit the values in the file accordingly to your configuration

To configure element:

- Create a directory `element` at the root of the installer directory : `mkdir ~/.element-onpremise-config/element`
- Copy the configurations extensions you want to setup from `config-sample/element` to `~/.element-onpremise-config/element`.
- Edit the values in the file accordingly to your configuration

For specifics on configuring permalinks for Element, please see Setting up Permalinks With the Installer

For specifics on setting up Delegated Authentication, please see Setting up Delegated Authentication With the Installer

For specifics on setting up Group Sync, please see Setting up Group Sync with the Installer

For specifics on setting up the Integration Manager, please see Setting Up the Integration Manager With the Installer

For specifics on setting up GitLab, GitHub, and JIRA integrations, please see Setting up GitLab, GitHub, and JIRA Integrations With the Installer

For specifics on pointing your installation at an existing Jitsi instance, please see Setting up Jitsi With the Installer

Installation

Let’s review! Have you considered:

- Hostnames/DNS
- Operating System
- Users
- Network Specifics
- Postgresql Database
- TURN Server
- SSL Certificates
- Extra configuration items

Once you have the above sections taken care of and your `parameters.yml` and `secrets.yml` files are in order, you are ready to begin the actual installation.

From the installer directory, run: (Note: You can replace `~/.element-onpremise-config` with whatever you have specified for your config directory.)

```
bash install.sh  ~/.element-onpremise-config
```

The first run should go for a little while and then exit, instructing you to log out and back in.

Please log out and back in and re-run the installer from the installer directory again:

```
bash install.sh  ~/.element-onpremise-config
```

Once this has finished, you can run:

```
kubectl get pods -n element-onprem
```

And you should get similar output to:

```
NAME                                    READY   STATUS    RESTARTS   AGE
app-element-web-c5bd87777-rqr6s         1/1     Running   1          29m
server-well-known-8c6bd8447-wddtm       1/1     Running   1          29m
postgres-0                              1/1     Running   1          40m
instance-synapse-main-0                 1/1     Running   2          29m
```

```
instance-synapse-haproxy-5b4b55fc9c-hnlmp   1/1     Running   0          20m
```

At this time, you should also be able to browse to: `https://fqdn` and create a test account with Element on your new homeserver. Using our example values, I am able to go to `https://element.local/` and register an account, sign in and begin testing the proof of concept!

How to Install a Production Environment

Our Element Enterprise Production Installer can handle the installation of Element Enterprise into your production k8s environment.

To get started with a production installation, there are several things that need to be considered and this guide will work through them:

- Hostnames/DNS
- Resource Requirements
- k8s Environments
- Postgresql Database
- TURN Server
- SSL Certificates
- Extra configuation items

Once these areas have been covered, you'll be able to install a production environment!

Hostnames/DNS

You will need hostnames for the following pieces of infrastructure:

- Element Server
- Synapse Server
- Dimension Server
- Hookshot Server

These hostnames must resolve to the appropriate IP addresses. You should have a proper DNS server to serve these records in a production environment.

In the event that you do not have a prop

Resource Requirements

For running running in production, we support only the x86_64 architecture and recommend the following minimums:

- No federation: 4 vCPUs/CPUS and 16GB RAM
- Federation: 8 vCPUs/CPUS and 32GB RAM

Unpacking the Installer

Please make sure that you unpack `element-enterprise-installer` onto a system that has access to your k8s environment. The directory that it unpacks into will be referenced in this document as the installer directory.

You will also need to create a directory for holding the configurations for the installer. This will be referenced as the config directory going forward.

```
mkdir  ~/.element-onpremise-config
```

k8s Environments

Element Enterprise Installer allows you to either deploy directly into a kubernetes environment or to render a set of manifests for a future deployment in a kubernetes environment.

To configure your kubernetes environment for a direct deployment, you need to :

- Configure a kubectl context able to connect to your kubernetes instance
- Copy `k8s.yml.sample` to `k8s.yml` in your config directory. Edit `k8s.yml` with the following values :
- `provider_storage_class_name`: The storage class to use when creating PVCs.
- `ingress_annotations`: The annotations to add to the ingresses created by the operator.
- `tls_managed_externally`: Should be true if you don't expect the operator to manage the certificates of your kubernetes deployment. In this case, you will be able to skip the *_Certificates_* chapter of the `CONFIGURE.md` file.
- `operator_namespace`: The namespace to create to deploy the operator.
- `element_namespace`: The namespace to create to deploy the element resources.
- `k8s_auth_context`: The value of the context used in kubectl. If you want to use cert-manager for your tls certificates, it needs to be already installed in the targeted k8s cluster.

An example k8s.yml file would look like:

```
provider_storage_class_name:  gp8-delete  # select an available storage class
```

```
ingress_annotations:  ## below are expected annotations for an aws deployment
    kubernetes.io/ingress.class:  alb
    alb.ingress.kubernetes.io/scheme:  internet-facing
    alb.ingress.kubernetes.io/group.name:  global
    alb.ingress.kubernetes.io/target-type:  ip
    alb.ingress.kubernetes.io/ip-address-type:  ipv4
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTP":  80},{"HTTPS":  443}]'
synapse_ingress_annotations:   # below are required annotations if using the NGINX ingress
controller
    nginx.ingress.kubernetes.io/proxy-body-size:  "50m"
tls_managed_externally:  true  # true if the certificates are managed externaly to k8s
security_context_force_uid_gid:  true  # true to enable pod runAsUser and fsGroup in security
context.  false if it should not be used,  in the case of openshift for example.
security_context_set_seccomp:  true  # true to enable RuntimeDefault pod seccomp.  false if it
should not be used,  in the case of openshift for example.
operator_namespace:  <namespace to create to deploy the operator>
element_namespace:  <namespace to create to deploy the element resources>
k8s_auth_context:  <the k8s auth context>
out_dir:  # Absolute path to the directory where to render manifests,  if render mode is used
# operator_manager_limits:   # Can be used to defined upper limits if the default one are not
large enough for your operator deployment
#   cpu:  "2"
#   memory:  8Gi
```

If you do not want to deploy directly to kubernetes, but wish to render manifests instead, set all of the above mentioned variables except for `k8s_auth_context` and define a value for the parameter `out_dir`, which specifies where to write the kubernetes manifests. Further, when you go to run the installer, you need to invoke it as such:

```
bash install.sh  ~/.element-onpremise-config --target render
```

Using the above syntax, you will have a set of manifests written out to `out_dir` that you can then deploy into your kubernetes environment.

N.B. You will need to set your ingress controller's upload size to be at least 50 Mb to match synapse's default upload size if you wish to be able to have users upload files up to 50 Mb in size. Instructions for doing this with nginx are included in the `parameters.yml` section below.

Postgresql Database

The installation requires that you have a postgresql database with a locale of C and UTF8 encoding set up. See https://matrix-org.github.io/synapse/latest/postgres.html#set-up-database for further details.

Please make note of the database hostname, database name, user, and password as you will need these to begin the installation.

TURN Server

For installations in which you desire to use video conferencing functionality, you will need to have a TURN server installed and available for Element to use.

If you do not have an existing TURN server, we recommend installing `coturn` outside of your k8s environment. `coturn` must open a lot of ports to work and this can be problematic for k8s environments. Instructions on how to do that are available here: https://github.com/matrix-org/synapse/blob/master/docs/turn-howto.md (Note: On EL, you can do `yum install coturn -y`.)

Under "Synapse Setup" in the above instructions, you'll see what to change on the config. With the installer, you can create a file called `synapse/turn.yml` in your config directory and put the following in it:

```
turn_uris: [ "turn:turn.matrix.org?transport=udp", "turn:turn.matrix.org?transport=tcp" ]
turn_shared_secret: "n0t4ctuAllymatr1Xd0TorgSshar3d5ecret4obvIousreAsons"
turn_user_lifetime: 86400000
turn_allow_guests: True
```

based on how you installed the TURN server. This will allow the installer to configure synapse to use your TURN server.

A few notes on TURN servers:

- The TURN server has to be directly accessible by end-users. Normally this means a public IP, however if all the end-users are going to be on a VPN/private network then they just need to be able to access the private IP of the TURN server.
- The only reason to have TURN on a private network is if the private network disallows user <-> user traffic and only allows user <-> TURN server traffic. If user <-> user is allowed within the private network then a TURN server isn't needed.

SSL Certificates

For SSL Certificates, you have three options:

- Signed certificates from an internet recognized authority.
- LetsEncrypt
- Signed certificates from an internal to your company authority.

In the case of Internet Recognized Signed certificates or LetsEncrypt, your hostnames must be accessible on the internet.

Certificates without LetsEncrypt

If you have certificates for all of the aforementioned host names, then you can simply place the `.crt` and `.key` files in the `certs` directory under the config directory. Certificates in the `certs` directory must take the form of `fqdn.cert` and `fqdn.key`.

Certificates with LetsEncrypt

Our installer also supports using LetsEncrypt to build certificates for your host names and automatically install them into your environment. If your hosts are internet accessible, this is the easiest method and only requires an admin email address to provide to LetsEncrypt.

parameters.yml

Now it is time to set `parameters.yml`. A sample has been provided and to get started, it is easiest to do:

```
cp parameters.yml.sample  ~/.element-onpremise-config/parameters.yml
```

Using the example hostnames of `element.local` and `synapse.local` (not resolvable on the internet), we would set the following parameters first in `parameters.yml`:

```
domain_name: local
element_fqdn: element.local
synapse_fqdn: synapse.local
```

Next, we need to set the variables related to Postgres. For your Postgres server, please set the following:

```
postgres_fqdn: `Postgres Server`
postgres_user: `Postgres User`
```

```
postgres_db: `Postgres Database for Element`
```

The next item in the configuration is the microk8s DNS resolvers. By default, the installer will use Google's publicly available DNS servers. If you have defined your hosts on a non-publicly available DNS server, then you should use your DNS servers instead of the publicly available Google DNS servers. Let's assume that your local dns servers are 192.168.122.253 and 192.168.122.252. To use those servers, you would need to add this line:

```
microk8s_dns_resolvers: "192.168.122.253,192.168.122.252"
```

The next section pertains to certmanager. If you are not using LetsEncrypt, please leave these items both blank, as such:

```
certmanager_issuer:
certmanager_admin_email:
```

If you have chosen to use LetsEncrypt, please specify “letsencrypt” for the certmanager_issue and an actual email address for who should manage the certificates for certmanager_admin_email:

```
certmanager_issuer: 'letsencrypt'
certmanager_admin_email: 'admin@mydomain.com'
```

If you are using nginx as your ingress controller and wish to send files up to 50 Mb in size, please add these two lines to parameters.yml:

```
synapse_ingress_annotations:
  nginx.ingress.kubernetes.io/proxy-body-size: "50m"
```

secrets.yml

Now we move on to configuring `secrets.yml`. You will need the following items here:

- A Macaroon key
- Your postgres password for the user specified in parameters.yml
- A Registration Shared Secret
- A signing Key
- An EMS Image Store username and token, which will have been provided to you by Element.

To build a `secrets.yml` with the macaroon key, the registration shared secret, the generic shared secret, and the signing key already filled in, please run:

```
sh build_secrets.sh
mv secrets.yml  ~/.element-onpremise-config/
```

You will need to uncomment and set your `postgres_password` field to the proper password for your database.

Do not forget to also set the values for `ems_image_store_username` and `ems_image_store_token`, which will both be provided by Element.

If you have a paid docker hub account, you can specify your username and password to avoid being throttled in the `dockerhub_username` and `dockerhub_token` fields. This is optional.

Extra Configuration Items

It is possible to configure anything in Synapse's homeserver.yaml or Element’s config.json.

To do so, you need to create json or yaml files in the appropriate directory under the config directory. These files will be merged to the target configuration file.

Samples are available in `config-sample` under the installer directory.

To configure synapse:

- Create a directory `synapse` at the root of the config directory : `mkdir  ~/.element-onpremise-config/synapse`
- Copy the configurations extensions you want to setup from `config-sample/synapse` to `~/.element-onpremise-config/synapse`.
- Edit the values in the file accordingly to your configuration

To configure element:

- Create a directory `element` at the root of the config directory : `mkdir  ~/.element-onpremise-config/element`
- Copy the configurations extensions you want to setup from `config-sample/element` to `~/.element-onpremise-config/element`.
- Edit the values in the file accordingly to your configuration

For specifics on configuring permalinks for Element, please see Setting up Permalinks.

For specifics on setting up Delegated Authentication, please see Setting up Delegated Authentication With the Installer

For specifics on setting up Group Sync, please see Setting up Group Sync

For specifics on setting up the Integration Manager, please see Setting Up the Integration Manager With the Installer

For specifics on setting up GitLab, GitHub, and JIRA integrations, please see Setting up GitLab, GitHub, and JIRA Integrations With the Installer

For specifics on pointing your installation at an existing Jitsi instance, please see Setting up Jitsi With the Installer

Installation

Let’s review! Have you considered:

- Hostnames/DNS
- k8s Environments
- Postgresql Database
- TURN Server
- SSL Certificates
- Extra configuration items

Once you have the above sections taken care of and your `parameters.yml` and `secrets.yml` files are in order, you are ready to begin the actual installation.

From the installer directory, run:

```
bash install.sh  ~/.element-onpremise-config
```

The first run should go for a little while and then exit, instructing you to log out and back in.

Please log out and back in and re-run the installer from the installer directory again:

```
bash install.sh  ~/.element-onpremise-config
```

Setting up Permalinks With the Installer

Element Extra Configurations

- Copy sample file from `config-sample/element/permalinks.json` in the installer directory to `~/.element-onpremise-config/element`
- Edit the file :

```
{
    "permalinkPrefix": "https://<element fqdn>"
}
```

- Restart the install script

Setting up Jitsi With the Installer

By default, our installer will give you an instance of element-web configured to use the `meet.element.io` Jitsi server. If you would like to specify your own Jitsi server for your element-web instance to use, please follow these directions.

Element Extra Configurations

- Create a file called `jitsi.json` in the `~/.element-onpremise-config/element` directory.
- Edit the file :

```
{
      "jitsi": {
            "preferredDomain": "your.jitsi.example.org"
      }
}
```

replacing `your.jitsi.example.org` with the hostname of your Jitsi server.

- Restart the install script

Setting up Delegated Authentication With the Installer

On Element Enterprise

- Depending on your provider, copy the sample file in the installer root directory from `config-sample/synapse/` to `~/.element-onpremise-config/synapse`
- Edit the file for the provider you are setting up. You have at least 3 parameters to edit :
  - The IdP metadata url
  - The name and description of your synapse server, which your provider would display to inform the users to which app they are logging in
- Disable the local synapse user database and password workflows by creating a file `~/.element-onpremise-config/synapse/disable-local.yml` and putting the following in it:

```
password_config:
   localdb_enabled: false
   enabled: false
```

- Run the installer to configure SAML provisioning

On the provider

Azure ADFS

Keycloak

Other SAML providers can be configured for use with Element Enterprise. Please contact Element for further information in the event that you are not using one of the above providers.

Azure ADFS

- With an account with enough rights, go to : Enterprise Applications Portal
- Click on `New Application`
- Click on `Create your own application` on the top left corner
- Choose a name for it, and select `Integrate any other application you don't find in the gallery`
- Click on "Create"
- Select `Set up single sign on`
- Select `SAML`
- `Edit` on `Basic SAML Configuration`
- In `Identifier`, add the following URL : `https://<synapse fqdn>/_synapse/client/saml2/metadata.xml`
- Remove the default URL
- In `Reply URL`, add the following URL : `https://<synapse fqdn>/_synapse/client/saml2/authn_response`
- Click on `Save`
- `Edit` on `Attributes & Claims`
- Remove all defaults additional claims
- Click on `Add new claim` to add the following claims. The UID will be used as the MXID, the value here is mostly a suggestion :
  - Name: `uid`, Transformation : `ExtractMailPrefix`, Parameter 1 : `user.userprincipalname`
  - Name: `email`, Source attribute : `user.mail`
  - Name: `displayName`, Source attribute : `user.displayname`
- Click on `Save`
- In `Users and Groups`, add groups and users which may have access to element

Keycloak

- In `Configure` > `Clients`, add a new client. Enter `https://<synapse fqdn>/_synapse/client/saml2/metadata.xml` as its Client ID
- In `Mappers`, add the 3 following mappers :
  - Name: `uid` : User attribute : `username`
  - Name: `email`, User attribute : `email`
  - Name: `displayName`, Javascript mapper : `user.FirstName + " " + user.lastName`

Setting up Group Sync with the Installer

What is Group Sync?

Group Sync allows you to use the ACLs from your identity infrastructure in order to set up permissions on Spaces and Rooms in the Element Ecosystem. Please note that the initial version we are providing only supports a single node, non-federated configuration.

General settings

- Copy sample file from `config-sample/groupsync/gsync.yml` in the installer directory to `~/.element-onpremise-config/groupsync`
- Edit the file with the following values :
  - `group_power_levels` : A list of groups that'll determine people's Matrix power levels. This affects only the space that the Group belongs to – doesn't leak up or down. For MSGraph source, groups should be identified by their ids. On LDAP, they should be identified by their names.
  - `provisioner.dn_default_prefix`: Display names starting with this prefix will get corrected according to the names we found for their users in LDAP. Optional. Useful if you're using an OIDC provider that doesn't give you users' display names.
  - `provisioner.default_rooms` : Optional. A list of rooms that'll get automatically created in in managed space. The ID is required to enable GPS to track whether they were already created or not. You can change it, but it'll cause new rooms to be generated.
  - `provisioner.whitelisted_users` : Optional. A list of userid patterns that will not get kicked from rooms even if they don't belong to them according to LDAP. This is useful for things like the auditbot. Patterns listed here will be wrapped in ^ and $ before matching.
  - `verify_tls` : Optional. If doing a POC with self-signed certs, set this to 0. The default value is 1.

Configuring the source

LDAP Servers

- You should create a ldap account with read access to the OUs containing the users
- This account should use password authentication
- To use LDAP source, copy the file `config-sample/groupsync/ldap.yml` in the installer directory to `~/.element-onpremise-config/groupsync` and edit the following variables :
  - `ldap_check_interval_seconds`: The interval check in seconds
  - `ldap_uri`: The LDAP Uri to connect to the ldap server
  - `ldap_base`: The LDAP base used to build the space hierarchy. This OU will become the root space. Every OU below this base will be a child-space.
  - `ldap_bind_dn`: The user bind dn to use to read the space hierarchy.
  - `ldap_bind_password`: The user password
  - `ldap_attrs_uid`: The attribute to use to map to users mxids
  - `ldap_attrs_name`: The attribute to use to map to spaces names
- Restart the install script

MS Graph (Azure AD)

- You need to create an `App registration`. You'll need the `Tenant ID` of the organization, the `Application (client ID)` and a secret generated from `Certificates & secrets` on the app.
- For the bridge to be able to operate correctly, navigate to API permissions and ensure it has access to Group.Read.All, GroupMember.Read.All and User.Read.All
- Remember to grant the admin consent for those.
- To use MSGraph source, copy the file `config-sample/groupsync/msgraph.yml` in the installer directory to `~/.element-onpremise-config/groupsync` and edit the following variables :
  - `msgraph_tenant_id`: This is the "Tenant ID" from your Azure Active Directory Overview
  - `msgraph_client_id`: Register your app in "App registrations". This will be its "Application (client) ID"
  - `msgraph_client_secret` : Go to "Certificates & secrets", and click on "New client secret". This will be the "Value" of the created secret (not the "Secret ID").
- Restart the install script

Setting Up the Integration Manager With the Installer

Known Issues

The Dimension Integration Manager ships with a number of integrations that do not work in an on-premise environment. The following integrations are known to work with proper internet connectivity:

- Jitsi Widget
- Hookshot Frontend

Please note that we recognise this situation is less than ideal. We will be working to improve the situation around integrations in the near future.

On the hosting machine

- Copy sample file from `config-sample/dimension.yml` in the installer directory to `~/.element-onpremise-config/dimension`
- Edit the file with the following values :
  - `dimension_fqdn`: The access address to dimension. It should match something like `dimension.<fqdn.tld>`
  - `admins`: List of mxids with admin access to dimension
  - `widget_blocklist`: CIDRs listed here will be blocked from becoming widgets.
  - `postgres_fqdn`: PostgreSQL server fqdn or ip
  - `postgres_user`: PostgreSQL username
  - `postgres_db`: PostgreSQL dimension database
  - `postgres_password`: PostgreSQL dimension password
  - `bot_data_size`: The size of the space allocated to bot data.
  - `bot_data_path`: The path on the hosting machine to the space allocated to bot data
  - `postgres_create_in_cluster`: OPTIONAL. If doing a POC and using the same PostgreSQL server as Synapse, set to `true`
  - `verify_tls`: OPTIONAL. If doing a POC with self-signed certs, set this to `0`. The default is `1`.
- Restart the install script

On element

- Copy sample file from `config-sample/element/dimension.json` in the installer directory to `~/.element-onpremise-config/element`
- Edit the file to replace `< dimension_fqdn >` to your dimension instance fqdn.
- Restart the install script

Enabling BigBlueButton

To enable BigBlueButton integration into Element through Dimension, you should set the following variables.

- `bbb_api_base_url`: The full base URL of the API of your BigBlueButton instance
- `bbb_shared_secret`: The "shared secret" of your BigBlueButton instance. This is used to authenticate to the API above.
- `bbb_widget_name`: The title for BigBlueButton widgets that are generated by Dimension.
- `bbb_widget_title`: The subtitle for BigBlueButton widgets that are generated by Dimension.
- `bbb_widget_avatar_mxc`: The avatar for BigBlueButton widgets that are generated by Dimension. Usually this doen't need to be changed, however if your homeserver is not able to reach t2bot.io then you should specify your own here.

Setting up GitLab, GitHub, and JIRA Integrations With the Installer

In Element Enterprise On-Premise, our GitLab, GitHub, and JIRA integrations are provided by the hookshot package. This documentation explains how to configure the installer to install hookshot and then how to interact with hookshot once installed.

Configuring Hookshot with the Installer

- Copy sample file from `config-sample/hookshot/hookshot.yml` in the installer directory to `~/.element-onpremise-config/hookshot`
- Edit the file with the following values :
  - `logging_level` : The logging level
  - `hookshot_fqdn` : The adress of hookshot webhook fqdn. It should match something like `hookshot.<fqdn.tld>`
  - `passkey` : The name of the local key file. It can be generated using openssl - `openssl genrsa -out key.pem 4096`
  - `provisioning_secret` : The provisioning secret used with integration managers. Necessary for integration with dimension.
  - `bot_name` : The name of hookshot bot
  - `bot_avatar` : An `mxc://` uri to the hookshot bot avatar image.
  - `verify_tls` : Optional. If doing a POC with self-signed certificates, set this to 0. Defaults to 1.
- Restart the install script

Enabling GitHub Integration

On GitHub

- This bridge requires a GitHub App. You will need to create one.
- On the callback URL, set the following one : `https://<hookshot_fqdn>/oauth` and enable `Request user authorization (OAuth) during installation`
- On the webhook URL, set the following one : `https://<hookshot_fqdn>/`
- For the webhook secret, you can generate one using `pwgen 32 1` to generate one for example. Keep it somewhere safe, you'll need to to configure the bridge.
- Set the following permissions for the webhook :
  - Repository
    - Actions (read)
    - Contents (read)
    - Discussions (read & write)
    - Issues (read & write)
    - Metadata
    - Projects (read & write)
    - Pull requests (read & write)
  - Organisation
    - Team Discussions (read & write)

On the installation

- Copy sample file from `config-sample/hookshot/github.yml` in the installer directory to `~/.element-onpremise-config/hookshot`
- Edit the file with the following values :
  - `github_auth_id` : The AppID given in your github app page
  - `github_key_file` : The key file received via the `Generate a private key` button under `Private keys` section of the github app page.
  - `github_webhook_secret` : The webhook secret configured in the app.
  - `github_oauth_client_id` : The OAuth ClientID of the github app page.
  - `github_oauth_client_secret` : The OAuth Client Secret of the github app page.
  - `github_oauth_default_options` A mapping to enable special oauth options.
- Restart the install script

In Element's room

- As an administrator of the room, invite the hookshot bot
- Start a private conversation with the bot
- Type `github login`
- Follow the link to connect the bot to the configured app
- If you have setup Dimension, you can use the integration manager to add a bridge to github

Enabling GitLab integration

On GitLab

- Add a webhook under the group or the repository you are targeting
- On the webhook URL, set the following one : `https://<hookshot_fqdn>/`
- For the webhook secret, you can generate one using `pwgen 32 1` to generate one for example. Keep it somewhere safe, you'll need to to configure the bridge.
- You should add the events you wish to trigger on. Hookshot currently supports:
  - Push events
  - Tag events
  - Issues events
  - Merge request events
  - Releases events

On the installation

- Copy sample file from `config-sample/hookshot/gitlab.yml` in the installer directory to `~/.element-onpremise-config/hookshot`
- Edit the file with the following values :
  - `gitlab_instances`: A mapping of the GitLab servers
    - `git.example.org`: Replace with name of the GitLab server
      - `url`: Replace with URL of the GitLab server
  - `gitlab_webhook_secret`: The secret configured in the webhook.

In Element's room

- As an administrator of the room, invite the hookshot bot
- As an administrator of the room, run the command `/devtools`
- Choose `Send Custom Event`
- Switch the button `Event` to `State Event` by clicking on it
- In `Event Type`, enter `uk.half-shot.matrix-hookshot.gitlab.repository`
- In `State key`, enter a random value, for example generated after `pwgen 32 1`
- In `Event Content`, enter :

```
{
    "instance": "<your instance name in gitlab.yml>",
    "path": "<username-or-group/repo>"
}
```

Enabling JIRA integration

On JIRA

- This should be done for all JIRA organisations you wish to bridge. The steps may differ for SaaS and on-prem, but you need to go to the webhooks configuration page under Settings > System. It should point to `https://<hookshot_fqdn>/`
- For the webhook secret, you can generate one using `pwgen 32 1` to generate one for example. Keep it somewhere safe, you'll need to to configure the bridge.

Enable OAuth

The JIRA service currently only supports atlassian.com (JIRA SaaS) when handling user authentication. Support for on-prem deployments is hoping to land soon.

- You'll first need to head to https://developer.atlassian.com/console/myapps/create-3lo-app/ to create a "OAuth 2.0 (3LO)" integration.
- Once named and created, you will need to:
- Enable the User REST, JIRA Platform REST and User Identity APIs under Permissions.
- Use rotating tokens under Authorisation.
- Set a callback url. This will be the public URL to hookshot with a path of /jira/oauth.
- Copy the client ID and Secret from Settings

On the installation

- Copy sample file from `config-sample/hookshot/jira.yml` in the installer directory to `~/.element-onpremise-config/hookshot`
- Edit the file with the following values :
  - `jira_webhook_secret`: The webhook secret configured
  - `jira_oauth_client_id`: If Oauth is enabled, it should point to the ClientID in Jira's App page. Else, you can keep it empty.
  - `jira_oauth_client_secret`: If Oauth is enabled, it should point to the Client secret in Jira's App page. Else, you can keep it empty.

In Element's room

- As an administrator of the room, invite the hookshot bot
- If you have setup Dimension, you can use the integration manager to add a bridge to JIRA

Enabling generic webhooks integration

On the installation

- Copy sample file from `config-sample/hookshot/generic.yml` in the installer directory to `~/.element-onpremise-config/hookshot`
- Edit the file with the following values :
  - `generic_enabled`: `true` to enable it
  - `generic_allow_js_transformation_functions`: `true` if you want to enable javascript transformations
  - `generic_user_id_prefix`: Choose a prefix for the users generated by hookshot for webhooks you'll create

In Element's room

- As an administrator of the room, invite the hookshot bot
- Type `!hookshot webhook <name of the webhook>`
- The bot will answer with a URL that you can set up as a webhook.
- Please ensure that the `Content-Type` is set to the type matching what the webhook sends
- If you have setup Dimension, you can use the integration manager to add a bridge to a new webhook

Troubleshooting

Introduction to Troubleshooting

Troubleshooting the Element Installer comes down to knowing a little bit about kubernetes and how to check the status of the various resources. This guide will walk you through some of the initial steps that you'll want to take when things are going wrong.

install.sh problems

Sometimes there will be problems when running the ansible-playbook portion of the installer. When this happens, you can increase the verbosity of ansible logging by editing `.ansible.rc` in the installer directory and setting:

```
export ANSIBLE_DEBUG=true
export ANSIBLE_VERBOSITY=4
```

and re-running the installer. This will generate quite verbose output, but that typically will help pinpoint what the actual problem with the installer is.

Problems post-installation

Checking Pod Status and Getting Logs

- In general, a well-functioning Element stack has at it's minimum the following containers (or pods in kubernetes language) running:

```
[user@element2 ~]$ kubectl get pods -n element-onprem
NAME                                      READY   STATUS    RESTARTS      AGE
instance-synapse-main-0                   1/1     Running   4 (27h ago)   6d21h
postgres-0                                1/1     Running   2 (27h ago)   6d21h
app-element-web-688489b777-v7l2m          1/1     Running   6 (27h ago)   6d22h
server-well-known-55bdb6b66-m8px6         1/1     Running   2 (27h ago)   6d21h
```

```
instance-synapse-haproxy-554bd57975-z2ppv   1/1     Running   3 (27h ago)   6d21h
```

The above `kubectl get pods -n element-onprem` is the first place to start. You'll notice in the above, all of the pods are in the `Running` status and this indicates that all should be well. If the state is anything other than "Running" or "Creating", then you'll want to grab logs for those pods. To grab the logs for a pod, run:

```
kubectl logs -n element-onprem <pod name>
```

replacing `<pod name>` with the actual pod name. If we wanted to get the logs from synapse, the specific syntax would be:

```
kubectl logs -n element-onprem instance-synapse-main-0
```

and this would generate logs similar to:

```
 2022-05-03 17:46:33,333 - synapse.util.caches.lrucache - 154 - INFO -
LruCache._expire_old_entries-2887 - Dropped 0 items from caches
2022-05-03 17:46:33,375 - synapse.storage.databases.main.metrics - 471 - INFO -
generate_user_daily_visits-289 - Calling _generate_user_daily_visits
2022-05-03 17:46:58,424 - synapse.metrics._gc - 118 - INFO - sentinel - Collecting
gc 1
2022-05-03 17:47:03,334 - synapse.util.caches.lrucache - 154 - INFO -
LruCache._expire_old_entries-2888 - Dropped 0 items from caches
2022-05-03 17:47:33,333 - synapse.util.caches.lrucache - 154 - INFO -
LruCache._expire_old_entries-2889 - Dropped 0 items from caches
2022-05-03 17:48:03,333 - synapse.util.caches.lrucache - 154 - INFO -
LruCache._expire_old_entries-2890 - Dropped 0 items from caches
```

- Again, for every pod not in the `Running` or `Creating` status, you'll want to use the above procedure to get the logs for Element to look at.
- If you don't have any pods in the `element-onprem` namespace as indicated by running the above command, then you should run:

```
[user@element2 ~]$ kubectl get pods -A
NAMESPACE            NAME                                        READY   STATUS
RESTARTS       AGE
container-registry   registry-5f697bb7df-dbzpq                   1/1     Running
6 (27h ago)    6d22h
kube-system          dashboard-metrics-scraper-69d9497b54-hdrdq  1/1     Running
6 (27h ago)    6d22h
kube-system          hostpath-provisioner-7764447d7c-jckkc       1/1     Running
11 (17h ago)   6d22h
element-onprem       instance-synapse-main-0                     1/1     Running
```

```
4 (27h ago)    6d22h
element-onprem       postgres-0                                  1/1     Running
2 (27h ago)    6d22h
element-onprem       app-element-web-688489b777-v7l2m            1/1     Running
6 (27h ago)    6d22h
element-onprem       server-well-known-55bdb6b66-m8px6           1/1     Running
2 (27h ago)    6d21h
kube-system          calico-kube-controllers-6966456d6b-x4scn    1/1     Running
6 (27h ago)    6d22h
element-onprem       instance-synapse-haproxy-554bd57975-z2ppv   1/1     Running
3 (27h ago)    6d21h
kube-system          calico-node-l28tp                           1/1     Running
6 (27h ago)    6d22h
kube-system          coredns-64c6478b6c-h5jp4                    1/1     Running
6 (27h ago)    6d22h
ingress              nginx-ingress-microk8s-controller-n6wmk     1/1     Running
6 (27h ago)    6d22h
operator-onprem      osdk-controller-manager-5f9d86f765-t2kn9    2/2     Running
9 (17h ago)    6d22h
kube-system          metrics-server-679c5f986d-msfc5             1/1     Running
6 (27h ago)    6d22h
kube-system          kubernetes-dashboard-585bdb5648-vrn42       1/1     Running
10 (17h ago)   6d22h
```

- This is the output from a healthy system, but if you have any of these pods not in the `Running` or `Creating` state, then please gather logs using the following syntax:

```
kubectl logs -n <namespace> <pod name>
```

- So to gather logs for the kubernetes ingress, you would run:

```
kubectl logs -n ingress nginx-ingress-microk8s-controller-n6wmk
```

  and you would see logs similar to:

```
I0502 14:15:08.467258       6 leaderelection.go:248] attempting to acquire leader
lease ingress/ingress-controller-leader...
I0502 14:15:08.467587       6 controller.go:155] "Configuration changes detected,
backend reload required"
I0502 14:15:08.481539       6 leaderelection.go:258] successfully acquired lease
ingress/ingress-controller-leader
I0502 14:15:08.481656       6 status.go:84] "New leader elected" identity="nginx-
```

```
ingress-microk8s-controller-n6wmk"
I0502 14:15:08.515623       6 controller.go:172] "Backend successfully reloaded"
I0502 14:15:08.515681       6 controller.go:183] "Initial sync, sleeping for 1
second"
I0502 14:15:08.515705       6 event.go:282] Event(v1.ObjectReference{Kind:"Pod",
Namespace:"ingress", Name:"nginx-ingress-microk8s-controller-n6wmk", UID:"548d9478-
094e-4a19-ba61-284b60152b85", APIVersion:"v1", ResourceVersion:"524688",
FieldPath:""}): type: 'Normal' reason: 'RELOAD' NGINX reload triggered due to a
change in configuration
```

Again, for all pods not in the `Running` or `Creating` state, please use the above method to get log data to send to Element.

Other Commands of Interest

Some other commands that may yield some interesting data while troubleshooting are:

- **Show all persistent volumes and persistent volume claims for the `element-onprem` namespace:**

  ```
  kubectl get pv -n element-onprem
  ```

  This will give you output similar to:

  ```
  NAME                                       CAPACITY   ACCESS MODES   RECLAIM
  POLICY   STATUS   CLAIM                              STORAGECLASS        REASON
  AGE
  pvc-9fc3bc29-2e5d-4b88-a9cd-a4c855352404   20Gi       RWX
  Delete           Bound    container-registry/registry-claim   microk8s-
  hostpath            55d
  synapse-media                              50Gi       RWO
  Delete           Bound    element-onprem/synapse-media        microk8s-
  hostpath            7d
  postgres                                   5Gi        RWO
  Delete           Bound    element-onprem/postgres             microk8s-
  hostpath            7d
  ```

- **Show the synapse configuration:**
  For installers prior to 2022-05.06, use:

  ```
  kubectl describe cm -n element-onprem instance-synapse-shared
  ```

  and this will return output similar to:

```
send_federation: True
start_pushers: True
turn_allow_guests: true
turn_shared_secret: n0t4ctuAllymatr1Xd0TorgSshar3d5ecret4obvIousreAsons
turn_uris:
- turns: turn.matrix.org?transport=udp
- turns: turn.matrix.org?transport=tcp
turn_user_lifetime: 86400000
```

For the 2022-05.06 installer and later, use:

```
kubectl -n element-onprem get secret synapse-secrets -o yaml 2>&1 | grep shared.yaml
| awk -F 'shared.yaml: ' '{print $2}' - | base64 -d
```

and you will get output similar to the above.

- **Show the Element Web configuration:**

```
kubectl describe cm -n element-onprem app-element-web
```

and this will return output similar to:

```
config.json:
----
{
    "default_server_config": {
        "m.homeserver": {
            "base_url": "https://synapse2.local",
            "server_name": "local"
        }
  },
  "dummy_end": "placeholder",
  "integrations_jitsi_widget_url":
"https://dimension.element2.local/widgets/jitsi",
  "integrations_rest_url": "https://dimension.element2.local/api/v1/scalar",
  "integrations_ui_url": "https://dimension.element2.local/element",
  "integrations_widgets_urls": [
      "https://dimension.element2.local/widgets"
  ]
}
```

- **Show the nginx configuration for Element Web: (If using nginx as your ingress controller in production or using the PoC installer.)**

```
kubectl describe cm -n element-onprem app-element-web-nginx
```

and this will return output similar to:

```
  server {
      listen       8080;

      add_header X-Frame-Options SAMEORIGIN;
      add_header X-Content-Type-Options nosniff;
      add_header X-XSS-Protection "1; mode=block";
      add_header Content-Security-Policy "frame-ancestors 'self'";
      add_header X-Robots-Tag "noindex, nofollow, noarchive, noimageindex";

      location / {
          root   /usr/share/nginx/html;
          index  index.html index.htm;

          charset utf-8;
      }
  }
```

- **Check list of active kubernetes events:**

```
kubectl get events -A
```

You will see a list of events or the message `No resources found`.

- Show the state of services in the `element-onprem` namespace:

```
kubectl get services -n element-onprem
```

This should return output similar to:

```
NAME                              TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)                   AGE
postgres                          ClusterIP   10.152.183.47    <none>
5432/TCP                  6d23h
app-element-web                   ClusterIP   10.152.183.60    <none>
80/TCP                    6d23h
server-well-known                 ClusterIP   10.152.183.185   <none>
80/TCP                    6d23h
instance-synapse-main-headless    ClusterIP   None             <none>
80/TCP                    6d23h
instance-synapse-main-0           ClusterIP   10.152.183.105   <none>
```

```
80/TCP,9093/TCP,9001/TCP   6d23h
instance-synapse-haproxy         ClusterIP   10.152.183.78    <none>
80/TCP                     6d23h
```

- **Show the status of the stateful sets in the `element-onprem` namespace:**

```
kubectl get sts -n element-onprem
```

This should return output similar to:

```
NAME                   READY   AGE
postgres               1/1     6d23h
instance-synapse-main  1/1     6d23h
```

- **Show deployments in the `element-onprem` namespace:**

```
kubectl get deploy -n element-onprem
```

This will return output similar to:

```
NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
app-element-web            1/1     1            1           6d23h
server-well-known          1/1     1            1           6d23h
instance-synapse-haproxy   1/1     1            1           6d23h
```

- **Show the status of all namespaces:**

```
kubectl get namespaces
```

which will return output similar to:

```
NAME                STATUS   AGE
kube-system         Active   20d
kube-public         Active   20d
kube-node-lease     Active   20d
default             Active   20d
ingress             Active   6d23h
container-registry  Active   6d23h
operator-onprem     Active   6d23h
element-onprem      Active   6d23h
```

- **Destroy the micro8ks setup**
  If you wish to start over, you can reset the microk8s setup by doing:

```
microk8s.reset --destroy-storage
```

WARNING: This will destroy all of your microk8s containers and storage. Use with caution.

Migrating From 0.6.1 to 1.0

Introduction

With the release of the 1.0 installer, we've made some changes that we think will greatly improve your experience with the installer, but which also require some changes to your 0.6.1 environment. This document will walk through the following changes:

- New configuration directory structure.
- Switching to the EMS Image Store registry.

New Configuration Directory Structure

Our on-premise installer operates on the premise that to get new updates for your environment, you download the latest version of the installer and run the installer again. Prior to the 1.0 release, this also required copying configuration files from one installer directory to the new installer directory. In 1.0, we are introducing a new configuration directory structure that will make it much simpler to download the latest installer and run it, without having to worry about moving configuration files around.

To move configuration from 0.6.1 to 1.0, please do the following:

```
mkdir ~/.element-onpremise-config
```

Now, we need to copy files from the 0.6.1 installer directory into this new configuration directory. For this example, I will assume that you have the 0.6.1 installer unpacked into `~/element-enterprise-installer-0.6.1`. Given this assumption, here is what you would need to do:

```
cd ~/element-enterprise-installer-0.6.1
cp parameters.yml ~/.element-onpremise-config/
cp secrets.yml ~/.element-onpremise-config/
cp -R certs ~/.element-onpremise-config/
cp -R extra-config/* ~/.element-onpremise-config/
```

Now you have migrated to the new configuration directory structure. Going forward, your configuration will stay in this directory.

Switching to the EMS Image Store Registry

In 1.0, we've also moved our container images to a new registry known as the EMS image store. With this move, you now only have one username and token to keep up with as opposed to the two credentials you had to manage pre-1.0.

To make the switch to the EMS Image Store Registry, you will need to contact Element and ask for an EMS Image Store user name and token. Once you have gotten these, you will need to edit `secrets.yml` in the configuration directory and remove these two lines:

```
registry_username: "myuser-2022"
registry_token: "mytoken"
```

and replace them with:

```
ems_image_store_username: "username_from_element"
ems_image_store_token: "token_from_element"
```

Further, if you've set the following lines in groupsync.yml, dimension.yml, and/or hookshot.yml, you will need to remove them as they are no longer used:

```
ems_bridges_registry_username: <ems bridges registry token name>
ems_bridges_registry_password: <ems bridges registry token password>
```

Wrapping Up

Once you have taken care of addressing these changes, you'll be able to run the 1.0 installer. Going forward, the syntax for running the installer will look like:

```
bash install.sh  ~/.element-onpremise-config
```

where the first parameter passed to install.sh is the config directory that contains your configurations.