Automating ESS Deployment

Understand your ESS configuration files and how you can automate ESS deployment(s).

The `.element-enterprise-server` Directory

> Config examples included on this page may not up-to-date and are solely provided for demonstration purposes. It is highly recommended to run the version of the installer you wish to install to generate and configure config files that work with that version.

> Once these config files have been created by the installer, you should refer to the up-to-date config examples available in the installation documentation to understand how each config option can be modified.

When you first run the installer binary, it will create a directory in your home folder, `~/.element-enterprise-server`. This is where you'll find everything the installer uses / generates as part of the installation including your configuration, the installer itself and logs.

As you run through the GUI, it will output config files within `~/.element-enterprise-server/config` that will be used when you deploy. This is the best way to get started, before any automation effort, you should run through the installer and get a working config that suits your requirements.

This will generate the config files, which can then be modified as needed, for your automation efforts, then in order to understand how deployments could be automated, you should understand what config is stored where.

The `cluster.yml` Config File

The Cluster YAML configuration file is populated with information used by all aspects of the installer. To start you'll find `apiVersion:`, `kind:` and `metadata` which are used by the installer itself to identify the version of your configuration file. In cases where you switch to a new version of the installer, it will then upgrade this config in-line with the latest versions requirements.

Config Example

```
apiVersion: ess.element.io/v1alpha1
kind: InstallerSettings
metadata:
  annotations:
    k8s.element.io/version: 2023-07.09-gui
  name: first-element-cluster
```

The configuration information is then stored in the `spec:` section, for instance you'll see; your Postgres in cluster information; DNS Resolvers; EMS Token; etc. See the example below:

```
spec:
  connectivity:
    dockerhub: {}
  install:
    certManager:
      adminEmail: admin@example.com
    emsImageStore:
      password: examplesubscriptionpassword
      username: examplesubscriptionusername
    microk8s:
      dnsResolvers:
      - 8.8.8.8
      - 8.8.4.4
      postgresInCluster:
        hostPath: /data/postgres
        passwordsSeed: examplepasswordsseed
```

The `deployment.yml` Config File

The Deployment YAML configuration file is populated with the bulk of the configuration for you're deployment. As above, you'll find `apiVersion:`, `kind:` and `metadata` which are used by the installer itself to identify the version of your configuration file. In cases where you switch to a new version of the installer, it will then upgrade this config in-line with the latest versions requirements.

Config Example

```
apiVersion: matrix.element.io/v1alpha1
kind: ElementDeployment
metadata:
```

```
  name: first-element-deployment
  namespace: element-onprem
```

The configuration is again found within the `spec:` section of this file, which itself has two main sections:

- `components:` which contains the set configuration for each individual component i.e. Element Web or Synapse
- `global:` which contains configuration required by all components i.e. the root FQDN and Certificate Authority information

`components:`

First each component has a named section, such as `elementWeb`, `integrator`, `synapseAdmin`, or in this example `synapse`:

```
    synapse:
```

Within each component, there are two sections to organise the configuration:

- `config:` which is configuration of the component itself i.e. whether Synapse registration is Open / Closed

Config Example

```
    config:
      acceptInvites: manual
      adminPasswordSecretKey: adminPassword
      externalAppservices:
        configMaps: []
        files: {}
      federation:
        certificateAutoritiesSecretKeys: []
        clientMinimumTlsVersion: '1.2'
        trustedKeyServers: []
      log:
        rootLevel: Info
      macaroonSecretKey: macaroon
      maxMauUsers: 250
      media:
        maxUploadSize: 100M
        volume:
```

```
      size: 50Gi
  postgresql:
    passwordSecretKey: postgresPassword
    port: 5432
    sslMode: require
  registration: closed
  registrationSharedSecretSecretKey: registrationSharedSecret
  security:
    defaultRoomEncryption: not_set
  signingKeySecretKey: signingKey
  telemetry:
    enabled: true
    passwordSecretKey: telemetryPassword
    room: '#element-telemetry'
  urlPreview:
    config:
      acceptLanguage:
      - en
  workers: []
```

- `k8s:` which is configuration of the pod itself in k8s i.e. CPU and Memory resource limits or FQDN

Config Example

```
k8s:
  common:
    annotations: {}
  haproxy:
    workloads:
      annotations: {}
      resources:
        limits:
          memory: 200Mi
        requests:
          cpu: 1
          memory: 100Mi
      securityContext:
        fsGroup: 10001
```

```
      runAsUser: 10001
ingress:
  annotations: {}
  fqdn: synapse.example.com
  services: {}
  tls:
    certmanager:
      issuer: letsencrypt
    mode: certmanager
redis:
  workloads:
    annotations: {}
    resources:
      limits:
        memory: 50Mi
      requests:
        cpu: 200m
        memory: 50Mi
    securityContext:
      fsGroup: 10002
      runAsUser: 10002
synapse:
  common:
    annotations: {}
  monitoring:
    serviceMonitor:
      deploy: auto
  storage: {}
  workloads:
    annotations: {}
    resources:
      limits:
        memory: 4Gi
      requests:
        cpu: 1
        memory: 2Gi
    securityContext:
      fsGroup: 10991
```

```
            runAsUser: 10991
        secretName: synapse
```

`global:`

The `global:` section works just like `component:` above, split into two sections `config:` and `k8s:`. It will set the default settings for all new components, you can see an example below:

Config Example

```
global:
  config:
    adminAllowIps:
    - 0.0.0.0/0
    - ::/0
    certificateAuthoritySecretKey: ca.pem
    domainName: example.com
    genericSharedSecretSecretKey: genericSharedSecret
    supportDnsFederationDelegation: false
    verifyTls: true
  k8s:
    common:
      annotations: {}
    ingresses:
      annotations: {}
      services:
        type: ClusterIP
      tls:
        certmanager:
          issuer: letsencrypt
        mode: certmanager
    monitoring:
      serviceMonitor:
        deploy: auto
    workloads:
      annotations: {}
      hostAliases: []
```

```
        replicas: 2
        securityContext:
          forceUidGid: auto
          setSecComp: auto
    secretName: global
```

The `secrets.yml` Config File

The Secrets YAML configuration file is populated, as expected, the secrets used for your configuration. It consists of multiple entries, separated by lines of `---` each following the below format:

Config Example

```
apiVersion: v1
data:
  genericSharedSecret: Q1BoVmNIaEIzWUR6VVZjZXpkMXhuQnNubHhLVVlM
kind: Secret
metadata:
  name: global
  namespace: element-onprem
```

The main section of interest for automation purposes, is the `data:` section, here you will find a dictionary of secrets, in the above you can see a `genericSharedSecret` and it's value opposite.

The `legacy` Directory

The `legacy` directory stores configuration for specific components not yet updated to the new format within the `component:` section of the `deployment.yml`. Work is steadily progressing on updating these legacy components to the new format, however in the meantime, you will find a folder for each legacy component here.

> As integrations are upgraded to the new format this example (IRC) may become outdated, however the process remains identical for any integrations still using the legacy format. Make sure to check via the installer if the integration you are looking for is configured in this way.

Within each components folder, you will see a `.yml` file, which is where the configuration of that component is stored. For instance, if you setup the IRC Bridge, it will create `~/.element-enterprise-server/config/legacy/ircbridge` with `bridge.yml` inside. You can use the Integrations and Add-Ons chapter of our documentation for guidance on how these files are configured. Using the IRC Bridge example, you would have a `bridge.yml` like so:

Config Example

```
key_file: passkey.pem
bridged_irc_servers:
- postgres_fqdn: ircbridge-postgres
  postgres_user: ircbridge
  postgres_db: ircbridge
  postgres_password: postgres_password
  admins:
  - "@user:example.com"
  logging_level: debug
  enable_presence: true
  drop_matrix_messages_after_seconds: 0
  bot_username: "ircbridgebot"
  provisioning_room_limit: 50
  rmau_limit: 100
  users_prefix: "irc_"
  alias_prefix: "irc_"
  address: irc.example.com
  parameters:
    name: "Example IRC"
    port: 6697
    ssl: true
    botConfig:
      enabled: true
      nick: "MatrixBot"
      username: "matrixbot"
      password: "some_password"
    dynamicChannels:
      enabled: true
    mappings:
      "#welcome":
        roomIds: ["!MLdeIFVsWCgrPkcYkL:example.com"]
    ircClients:
      allowNickChanges: true
```

There is also another important folder in `legacy`. The `certs` directory, here you will need to add any CA.pem file and certificates for the FQDN of any legacy components. As part of any automation, you will need to ensure these files are correct per setup and named correctly, the certificates in this directory should be named using the

fully qualified domain name (.key and .crt).

Automating your deployment

Once you have a set of working configuration, you should make a backup of your `~/.element-enterprise-server/config` directory. Through whatever form of automation you choose, automate the modification of your `cluster.yml`, `deployment.yml`, `secrets.yml` and any legacy `*.ymls` to adjust set values as needed.

For instance, perhaps you need 6 identical homeservers each with their own domain name, you would need to edit the `fqdn` of each component and the `domainName` in `deployment.yml`. You'd then have 6 config directories, each differing in domain, ready to be used by an installer binary.

On each of the 6 hosts, create the `~/.element-enterprise-server` directory and copy that hosts specific config to `~/.element-enterprise-server/config`. Copy the installer binary to the host, ensuring it's executable.

Running the installer unattended

Once host system is setup, you can add `unattended` when running the binary to run the installer unattended. It will pickup the configuration and start the deployment installation without needing to use the GUI to get it started.

```
./element-enterprise-graphical-installer-YYYY-MM.VERSION-gui.bin unattended
```

Revision #2
Created 2025-06-04 09:18:27 UTC by Kieran Mitchell Lane
Updated 2025-06-04 09:34:22 UTC by Kieran Mitchell Lane