Kubernetes Installations - Quick Start

For testing and evaluation purposes - Element cannot guarantee production readiness with these sample configurations.

Requires Helm installed locally

Deploying PostgreSQL to Kubernetes

If you do not have a database present, it is possible to deploy PostgreSQL to your Kubernetes cluster.

This is great for testing and *can* work great in a production environment, but only for those with a high degree of comfort with PostgreSQL as well as the tradeoffs involved with k8s-managed databases.

There are many different ways to do this depending on your organization's preferences - as long as it can create an instance / database with the required locale and encoding it will work just fine. For a simple non-production deployment, we will demonstrate deployment of the bitnami/postgresql into your cluster using Helm.

You can add the `bitnami` repo with a few commands:

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update
helm search repo
bitnami/postgresql

                              ~/Desktop
NAME                    CHART VERSION   APP VERSION
DESCRIPTION
bitnami/postgresql      12.5.7          15.3.0          PostgreSQL (Postgres) is an open
source object-...
bitnami/postgresql-ha   11.7.5          15.3.0          This PostgreSQL cluster solution
includes the P...
```

Next, you'll need to create a `values.yaml` file to configure your PostgreSQL instance. This example is enough to get started, but please consult the chart's README and values.yaml for a list of full parameters and options.

```
auth:
  # This is the necessary configuration you will need for the Installer, minus the hostname
  database: "synapse"

  username: "synapse"
  password: "PleaseChangeMe!"
```

```
primary:
  initdb:
    # This ensures that the initial database will be created with the proper collation
settings
    args: "--lc-collate=C --lc-ctype=C"

  persistence:
    enabled: true
    # Set this value if you need to use a non-default StorageClass for your database's PVC
    # storageClass: ""
    size: 20Gi



  # Optional - resource requests / requirements
  # These are sufficient for a 10 - 20 user server
  resources:
    requests:
      cpu: 500m
      memory: 512Mi
    limits:
      memory: 2Gi
```

This example `values.yaml` file is enough to get you started for testing purposes, but things such as TLS configuration, backups, HA and maintenance tasks are outside of the scope of the installer and this document.

Next, pick a namespace to deploy it to - this can be the same as the Installer's target namespace if you desire. For this example we'll use the `postgresql` namespace.

Then it's just a single Helm command to install:

```
# format:
# helm install --create-namespace -n <namespace> <helm-release-name> <repo/chart> -f <values
file> (-f <additional values file>)

helm install --create-namespace -n postgresql postgresql bitnami/postgresql -f values.yaml
```

Which should output something like this when it is successful:

```
-- snip --

PostgreSQL can be accessed via port 5432 on the following DNS names from within your cluster:
```

```
    postgresql.postgresql.svc.cluster.local - Read/Write connection
-- snip --
```

This is telling us that `postgresql.postgresql.svc.cluster.local` will be our hostname for PostgreSQL connections, which is the remaining bit of configuration required for the Installer in addition to the database/username/password set in `values.yaml`. This will differ depending on what namespace you deploy to, so be sure to check everything over.

If needed, this output can be re-displayed with `helm get notes -n <namespace> <release name>`, which for this example would be `helm get notes -n postgresql postgresql`)

Deploying ingress-nginx controller

Similar to the PostgreSQL quick start example, this requires Helm

The kubernetes/ingress-nginx chart is an easy way to get a cluster outfitted with Ingress capabilities.

In an environment where LoadBalancer services are handled transparently, such as in a simple test k3s environment with `svclb` enabled there's a minimal amount of configuration.

This example `values.yaml` file will create an IngressClass named `nginx` that will be used by default for any `Ingress` objects in the cluster.

```
controller:
  ingressClassResource:
    name: nginx
    default: true
    enabled: true
```

However, depending on your cloud provider / vendor (i.e. AWS ALB, Google Cloud Load Balancing etc) the configuration for this can vary widely. There are several example configurations for many cloud providers in the chart's README

You can see what your resulting HTTP / HTTPS IP address for this ingress controller by examining the service it creates - for example, in my test environment I have an installed release of the `ingress-nginx` chart called `k3s` under the `ingress-nginx` namespace, so I can run the following:

```
# format:
# kubectl get service -n <namespace> <release-name>-ingress-nginx-controller
$ kubectl get service -n ingress-nginx k3s-ingress-nginx-controller

NAME                                 TYPE           CLUSTER-IP      EXTERNAL-
IP                                             PORT(S)                      AGE
```

```
k3s-ingress-nginx-controller            LoadBalancer   10.43.254.210
192.168.1.129                                          80:30634/TCP,443:31500/TCP   79d
```

The value of `EXTERNAL-IP` will be the address that you'll need your DNS to point to (either locally via /etc/hosts or LAN / WAN DNS configuration) to access your installer-provisioned services.

Revision #3
Created 14 June 2023 18:49:24 by Rhea Danzey
Updated 28 December 2023 11:51:06 by Rhea Danzey