# FusionAuth

All my homies use SSO for their needs!

FusionAuth is a customer authentication and authorization platform that puts developers in the driver's seat, with control, flexibility and developer ergonomics. We use FusionAuth in our FINoT Platform as our Auth Server.

So am not going to lie, this will be hard, but with patience and hot java everything will end up fine.

## Overview and Requirements

We will have to create a new namespace "fusionauth" and deploy

1. Postgresql
2. ElasticSearch
3. and finally… FusionAuth.

You need to have already installed ingress-nginx. If not, please use this guide NGINX Ingress Controller to deploy that first and then start this deployment procedure.

I have used these guides to make our version. Please have a look if you want to have a clearer picture.

1. https://fusionauth.io/docs/v1/tech/installation-guide/kubernetes/setup/minikube
2. https://fusionauth.io/docs/v1/tech/installation-guide/kubernetes/fusionauth-deployment

Before starting, please create the namespace fusionauth by using the command **kubectl create namespace fusionauth**

## Deploy PostgreSQL

Let's start with PostgreSQL. It is our main Database for FusionAuth. SQL-based, some say better than MySQL and full of capabilities. We are going to use the bitnami chart in order to deploy via helm. Before we do that though, we will have to change some values for our environment. For that reason please download our values.yaml file from our gitlab (https://gitlab.f-in.io/finot/kubernetes/-/blob/master/fusionauth/postgresql/values.yaml)

If for some reason the above file cannot be used, download the values.yaml file from the postgresql bitnami chart

`wget https://github.com/bitnami/charts/raw/master/bitnami/postgresql/values.yaml`

In values.yaml you have to change the `postgresPassword: "futureintelligence"` in the `auth:` section and in every enabled `persistence:` section the `mountPath: /home/fint/storage/fusionauth/postgress` and `storageClass: "standard"` (helm will usually select the default StorageClass but better safe than sorry).

Save it and let's start with helm

Add PostgreSQL chart repository with `helm repo add bitnami https://charts.bitnami.com/bitnami`

and check if it is listed in the available repositories by using `helm repo list`

```
NAME            URL
bitnami         https://charts.bitnami.com/bitnami
```

If it is, let's install the postgresql chart in our cluster using our values.yaml file

`helm install postgres --namespace fusionauth bitnami/postgresql -f ./values.yaml`

Wait patiently and the chart should be installed

```
NAME: postgres
LAST DEPLOYED: Mon Aug  8 16:06:38 2022
NAMESPACE: fusionauth
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: postgresql
CHART VERSION: 11.6.16
APP VERSION: 14.4.0

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

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

    postgres-postgresql.fusionauth.svc.cluster.local - Read/Write
connection

To get the password for "postgres" run:

    export POSTGRES_PASSWORD=$(kubectl get secret --namespace
fusionauth postgres-postgresql -o jsonpath="{.data.postgres-password}"
| base64 -d)

To connect to your database run the following command:

    kubectl run postgres-postgresql-client --rm --tty -i --
restart='Never' --namespace fusionauth --image docker.io/bitnami
/postgresql:14.4.0-debian-11-r21 --env="PGPASSWORD=$POSTGRES_PASSWORD" \
      --command -- psql --host postgres-postgresql -U postgres -d
postgres -p 5432

    > NOTE: If you access the container using bash, make sure that you
execute "/opt/bitnami/scripts/postgresql/entrypoint.sh /bin/bash" in
order to avoid the error "psql: local user with ID 1001} does not exist"

To connect to your database from outside the cluster execute the
following commands:

    kubectl port-forward --namespace fusionauth svc/postgres-postgresql
5432:5432 &
    PGPASSWORD="$POSTGRES_PASSWORD" psql --host 127.0.0.1 -U postgres -
d postgres -p 5432
```

If you check your pods though, you'll see that the pod has a ⚠️ and says "Pending". Is the installation messed up. Not really. As said in the previous guides, we are using a Local StorageClass with no provider. We have to create a PersistentVolume for the PVC that the helm chart has already created.

Before that, please create a folder named “fusionauth” and a subfolder named “postgress” in /home/fint/storage

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: data-postgres-postgresql-0
  labels:
    app: fusion-postgres
    type: local
  finalizers:
    - kubernetes.io/pv-protection
spec:
  capacity:
    storage: 1Gi
  hostPath:
    path: /home/fint/storage/fusionauth/postgress
    type: ''
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  volumeMode: Filesystem
```

And with this, let’s create our PV. As always, either copy-pase and **kubectl apply -f** OR pipe into kubectl apply -f

The PV should now be deployed and bound to the PVC. Let’s check that by using `kubectl get pvc --namespace fusionauth`

```
NAME                                          STATUS
VOLUME                       CAPACITY   ACCESS MODES   STORAGECLASS
AGE
data-postgres-postgresql-0                    Bound    data-postgres-
postgresql-0   1Gi        RWO            standard       18h
```

Give the Pod a minute or two. If it has not started yet correctly, just delete the pod. The pod will be recreated automagically and everything should work fine.

## Check the PostgreSQL deployment

Yes, the pod is up and running, but how do we know if inside the pod PostgreSQL is working? Follow me.

```
kubectl run pg-postgresql-client --rm --tty -i --restart='Never' --namespace fusionauth --image docker.io
/bitnami/postgresql:11.13.0-debian-10-r40 --env="PGPASSWORD=futureintelligence" --command -- psql --host
postgres-postgresql -U postgres -d postgres -p 5432
```

Copy that command to your terminal and run. A new test pod will be created that will allow us to connect to postgresql. You should see something like this

```
kubectl run pg-postgresql-client --rm --tty -i --restart='Never' --
namespace fusionauth --image docker.io/bitnami/postgresql:11.13.0-
debian-10-r40 --env="PGPASSWORD=futureintelligence" --command -- psql --
host postgres-postgresql -U postgres -d postgres -p 5432
If you don't see a command prompt, try pressing enter.

postgres=#
```

See that `postgres=#` ? We are in ladies and gentlemen. Try some PostgreSQL commands. Everything should work as expected.

## Deploy ElasticSearch

Elastic. The pain of my existence. It fails, it tumbles (like metaxenia) but it is the best realtime search and caching app.

Metaxenia and Karnavaloxromata (https://www.youtube.com/watch?v=CSltlAbCMzc )

Again, values.yaml have to change for our environment. Our version is in our gitlab instance (https://gitlab.f-in.io/finot/kubernetes/-/blob/master/fusionauth/elastic/values.yaml).

If this is not applicable to your case, download the elasticsearch example values.yaml file with `curl -O https://raw.githubusercontent.com/elastic/Helm-charts/master/elasticsearch/examples/minikube/values.yaml`

From this change only the storageClassName field in `volumeClaimTemplate`

It should resemble this:

```
---
# Permit co-located instances for solitary minikube virtual machines.
antiAffinity: "soft"

# Shrink default JVM heap.
esJavaOpts: "-Xmx128m -Xms128m"

# Allocate smaller chunks of memory per pod.
resources:
  requests:
    cpu: "100m"
    memory: "512M"
  limits:
    cpu: "1000m"
    memory: "512M"

# Request smaller persistent volumes.
volumeClaimTemplate:
  accessModes: [ "ReadWriteOnce" ]
  storageClassName: "standard"
  resources:
    requests:
      storage: 100M
```

Add the elasticsearch repo in helm with `helm repo add elastic https://helm.elastic.co`

and check that it is there with `helm repo list`

```
helm repo list

NAME                URL
bitnami             https://charts.bitnami.com/bitnami
elastic             https://helm.elastic.co
```

We are ready to deploy the sucker. Use

`helm install elastic --namespace fusionauth elastic/elasticsearch -f values.yaml`

to install the helm chart with our values. The result should be the following.

```
NAME: elastic
LAST DEPLOYED: Mon Aug  8 16:12:06 2022
NAMESPACE: fusionauth
STATUS: deployed
REVISION: 1
NOTES:
1. Watch all cluster members come up.
  $ kubectl get pods --namespace=fusionauth -l app=elasticsearch-master
-w2. Test cluster health using Helm test.
  $ helm --namespace=fusionauth test elastic
```

Once again, due to the StorageClass issue, the pods will be down. We have to create THREE(3) PersistentVolumes for the 3 PVCs that helm created. Before that, please inside the folder /home/fint/storage/fusionauth/elastic, three subfolders named “master0”, “master1”, “master2”.

Let’s now create the **THREE** PersistentVolumes

***data-elastic-master-0***

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: data-elastic-master-0
  labels:
    app: fusion-elastic-master-0
    type: local
  finalizers:
    - kubernetes.io/pv-protection
spec:
  capacity:
    storage: 100Mi
  hostPath:
    path: /home/fint/storage/fusionauth/elastic/master0
    type: ''
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  volumeMode: Filesystem
```

***data-elastic-master-1***

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: data-elastic-master-1
  labels:
    app: fusion-elastic-master-1
    type: local
  finalizers:
    - kubernetes.io/pv-protection
spec:
  capacity:
    storage: 100Mi
  hostPath:
    path: /home/fint/storage/fusionauth/elastic/master1
    type: ''
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  volumeMode: Filesystem
```

***data-elastic-master-2***

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: data-elastic-master-2
  labels:
    app: fusion-elastic-master-2
    type: local
  finalizers:
    - kubernetes.io/pv-protection
spec:
  capacity:
    storage: 100Mi
  hostPath:
    path: /home/fint/storage/fusionauth/elastic/master2
    type: ''
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  volumeMode: Filesystem
```

Apply them all one-by-one with kubectl apply -f

After 2-3 minutes all the PersistentVolumes should be bound to the ElasticSearch PVCs and the Pods should now be running normally. Please check with `kubectl get pvc --namespace fusionauth`

```
NAME                                           STATUS
VOLUME                       CAPACITY   ACCESS MODES   STORAGECLASS
AGE
data-postgres-postgresql-0                     Bound    data-postgres-
postgresql-0   1Gi        RWO            standard       19h
elasticsearch-master-elasticsearch-master-0   Bound    data-elastic-
master-2        100Mi      RWO            standard       19h
elasticsearch-master-elasticsearch-master-1   Bound    data-elastic-
master-1        100Mi      RWO            standard       19h
elasticsearch-master-elasticsearch-master-2   Bound    data-elastic-
master-0        100Mi      RWO            standard       19h
```

Hurray! ElasticSearch is now (hopefully) ready to use. We are close to the finish line, dont give up!

## Deploy FusionAuth

On our K8s installation, we have allowed kubeadm to install kube-dns. This is paramount for the deployment of fusionauth!

Our previously deployed services (postgresql and elasticsearch) are now alive in:

1. postgres-postgresql.fusionauth.svc.cluster.local
2. elasticsearch-master.fusionauth.svc.cluster.local

Well, let’s start.

We have to add the fusionauth chart repo in helm with `helm repo add fusionauth https://fusionauth.github.io/charts`

As per usual, we have to change the values.yaml file to our liking. The ready one is in our gitlab (https://gitlab.f-in.io/finot/kubernetes/-/blob/master/fusionauth/values.yaml). If you want to do it you self download the default values.yaml file with `curl -o values.yaml https://raw.githubusercontent.com/FusionAuth/charts/master/chart/values.yaml`

What we need to change?

**Replicas-Database-Search-Ingress,** GO!

replicaCount to 3

```
# replicaCount -- The number of fusionauth-app instances to run
replicaCount: 3
```

in database change host, user, password, root.user, root.password

```
database:
  # database.protocol -- Should either be postgresql or mysql. Protocol
for jdbc connection to database
  protocol: postgresql
  # database.host -- Hostname or ip of the database instance
  host: "postgres-postgresql.fusionauth.svc.cluster.local"
  # database.host -- Port of the database instance
  port: 5432
  # database.tls -- Configures whether or not to use tls when
connecting to the database
  tls: false
  # database.tlsMode -- If tls is enabled, this configures the mode
  tlsMode: require
  # database.name -- Name of the fusionauth database
  name: fusionauth
  # To use an existing secret, set `existingSecret` to the name of the
secret. We expect at most two keys: `password` is required.
`rootpassword` is only required if `database.root.user` is set.
  # database.existingSecret -- The name of an existing secret that
contains the database passwords
  existingSecret: ""
  # database.user -- Database username for fusionauth to use in normal
operation
  user: "postgres"
  # database.password -- Database password for fusionauth to use in
normal operation - not required if database.existingSecret is configured
  password: "futureintelligence"
  # These credentials are used for bootstrapping the database
  root:
    # database.root.user -- Database username for fusionauth to use
during initial bootstrap - not required if you have manually
bootstrapped your database
    user: "postgres"
    # database.root.password -- Database password for fusionauth to use
during initial bootstrap - not required if database.existingSecret is
configured
    password: "futureintelligence"
```

in search change the host

```
search:
  # search.engine -- Defines backend for fusionauth search
capabilities. Valid values for engine are 'elasticsearch' or 'database'.
  engine: elasticsearch
  # search.engine -- Protocol to use when connecting to elasticsearch.
Ignored when search.engine is NOT elasticsearch
  protocol: http
  # search.host -- Hostname or ip to use when connecting to
elasticsearch. Ignored when search.engine is NOT elasticsearch
  host: "elasticsearch-master.fusionauth.svc.cluster.local"
  # search.port -- Port to use when connecting to elasticsearch.
Ignored when search.engine is NOT elasticsearch
  port: 9200
  # search.user -- Username to use with basic auth when connecting to
elasticsearch. Ignored when search.engine is NOT elasticsearch
  # user: ""
  # search.password -- Password to use with basic auth when connecting
```

```
to elasticsearch. Ignored when search.engine is NOT elasticsearch
  # password: ""
```

in ingress we have to enable it, create the path / to port 9011 and create a host -fusionauth.dev.f-in.io

```
ingress:
  # ingress.enabled -- Enables ingress creation for fusionauth.
  enabled: true
  name: fusionauth-ingress
  # ingress.annotations -- Configure annotations to add to the ingress
object
  annotations:
     kubernetes.io/ingress.class: nginx
    # kubernetes.io/tls-acme: "true"
  paths:
   - path: /
     pathType: Prefix
     service:
       port: 9011
  # ingress.extraPaths -- Define complete path objects, will be
inserted before regular paths. Can be useful for things like ALB
Ingress Controller actions
  #extraPaths: []
  # ingress.hosts -- List of hostnames to configure the ingress with
  hosts:
    - fusionauth.dev.f-in.io
  # ingress.tls -- List of secrets used to configure TLS for the
ingress.
  tls: []
  #  - secretName: chart-example-tls
  #    hosts:
  #      - chart-example.local
```

Done! Save it

Deploy fusionauth with `helm install fusionauth --namespace fusionauth fusionauth/fusionauth -f values.yaml`

*Output*

```
NAME: fusionauth
LAST DEPLOYED: Mon Aug  8 17:50:51 2022
NAMESPACE: fusionauth
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  http://fusionauth.dev.f-in.io/
```

Pods should be up and running buuuut, `http://fusionauth.dev.f-in.io/` is showing a 404 page. What happened? The helm chart of fusionauth has an important fault, it cannot parse the service information to the ingress we just made with the chart file. It is easy to fix though!

Edit the ingress file and change the port from *name:http* to *number:9011*

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: fusionauth
  namespace: fusionauth
status:
  loadBalancer:
    ingress:
      - ip: 10.0.20.31
spec:
  rules:
    - host: fusionauth.dev.f-in.io
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: fusionauth
                port:
                  number: 9011
```

You should be able now to go to `http://fusionauth.dev.f-in.io/` .

Hurray! Congrats! It wasn't easy but you made it! You have fusionauth installed!

## ⚠️ Troubleshooting⚠️

### Problem with Single Mode when first loading FusionAuth

Most probably a configuration error. Check that the PostgreSQL and ElasticSearch pods are up and running normally. If they are, check the fusionauth values.yaml file.

I mean, you certainly know that the default user of PostgreSQL is **postgres** and **not admin** right?