Document Purpose

This is the InfluxDB Clustered Getting Started Guide. Its purpose is to guide you through the process of deploying an InfluxDB Cluster. The process consists of 3 parts:

- Set up the prerequisites
- Configure and install InfluxDB
- Use the cluster to confirm that the cluster is correctly set up.

Document conventions

Throughout the document we'll use the string `<package-version>` as a placeholder for the package version mentioned inside the example-customer.yaml file. e.g

```
Unset
us-docker.pkg.dev/influxdb2-artifacts/clustered/influxdb:xxxxxxxx-yyyyyy
                                                        ^^^^^^^^^^^^^^^^
                                                        package-version
```

For simplicity the document will always reference the filename `myinfluxdb.yml` and the namespace `influxdb` but you're free to pick whatever name suits you.

Part 1 - Setting up the prerequisites

The following are the prerequisites needed for InfluxDB clustered.

- Kubernetes cluster - Version 1.25 or higher
- Object storage (S3 or compatible) to store the InfluxDB parquet files
- Postgres compatible database (AWS Aurora or hosted Postgres, etc) - for the InfluxDB catalog (supported versions: 13.8 to 14.6)
- OAuth 2.0 provider
    - Requires support of Device Authorization Flow
    - Tested and supported: Azure Active Directory, Keycloak, Auth0
- TLS certificate (for ingress to the cluster)

Cluster sizing recommendation

For a medium-size workload (ref: https://www.influxdata.com/resources/influxdb-3-0-vs-oss/) we have configured our cluster on AWS as follows:

- S3 for object store (size is determined by how much data you write)
- Aurora Postgresql - serverless v2 scaling configuration (2-64 ACUs)
- EC2 instances - primarily m6i.2xlarge (8 CPU, 32GB RAM)
    - 3 m6i.2xlarge instances for ingesters/routers (with min 2Gi local storage)

- 3 m6i.2xlarge instances for queriers
- 1 m6i.2xlarge instance for compactor
- 1 t3.large for K8s control plane

Your sizing may need to be different, based on your environment and your workload, but this is a reasonable starting size for your initial testing.

Kubernetes setup

- You must be running v1.25 or higher of Kubernetes.
- You will need to be able to create two namespaces - influxdb and kubit.
- An Ingress controller must be installed in the cluster (currently we support Nginx but others may work) and a mechanism to obtain a valid TLS certificate (e.g. certmanager or provide the certificate PEM manually out of band).
- Access to the Influx container registry must be available from your Kubernetes cluster (or if running in an air-gapped environment, a local container registry to which you will need to copy the Influx images)

Note: It is strongly recommended that you run the Postgres-compatible database (which will house the InfluxDB Catalog) and the Object Store (which will house the InfluxDB parquet files) in a separate namespace from InfluxDB or external to Kubernetes entirely. Running the Catalog database and Object Store in a separate namespace/outside of Kubernetes makes management of the InfluxDB instance easier and prevents the potential for accidental data loss.

Local storage setup

The ingester pods need some local storage (for the Write-Ahead-Log). We recommend a minimum of 2Gi.

OAuth2 Server setup

InfluxDB requires access to an OAuth2 authentication service to authenticate user access. The OAuth2 service requires support of Device Authorization Flow. Internally we have tested with Azure Active Directory, Keycloak, and Auth0, but the software should work with any OAuth2 provider. To access the OAuth2 server, we will need the configured client ID, the JWKS endpoint, the device authorization endpoint, and the token endpoint.

Client Software setup

On the system/laptop from where you'll be configuring the cluster you'll need to have the following installed:

- Kubectl v1.27
- Yq
- jq
- Crane

Part 2 - Configuring and installing InfluxDB

Make sure you have the following items provided by InfluxData:

- A tar-ball that contains:
    - Clustered: Getting Started Runbook.pdf (This document)
    - example-customer.yml
        - Where you will define the configuration for the InfluxDB Clustered product, including information about the prerequisites
        - Note that throughout this document we refer to myinfluxdb.yml, this is a copy of the example-customer.yml that you edit as outlined below.
    - app-instance-schema.json
        - This defines the schema for example-customer.yml
- Authenticated docker config file - influxdb-docker-config.json

The InfluxDB software is in a container registry, and the provided influxdb-docker-config.json file gives you access to the collection of container images that are required to install InfluxDB Clustered.

Configuration data

When you are ready to install InfluxDB, you will need to have the following information available to configure InfluxDB 3.0:

1. The hostname via which K8s will expose the InfluxDB API endpoints
2. A postgresql style DSN that points at your postgresql compatible database
3. Object store
    a. Endpoint URL
    b. Access key
    c. Bucket name
    d. Region (required for S3, may not be required for other object stores)
4. Local storage (ingester pods)
    a. Storage class
    b. Storage size
5. OAuth2
    a. Client ID
    b. JWKS endpoint
    c. Device authorization endpoint
    d. Token endpoint

Deploy InfluxDB

InfluxDB is deployed in one kubernetes namespace. Throughout the installation procedure, this namespace is referred to as the "target" namespace.

You're free to pick any name for the namespace but for simplicity the rest of this document will assume that everything is deployed in the `influxdb` namespace.

The InfluxDB installation/upgrade and re-configuration process is driven by editing and applying a Kubernetes Custom Resource called `AppInstance`.

The Kubernetes Custom Resource is defined in a YAML file (you can use example-customer.yml as a starting point) that contains key information, such as:

1. the name of the target namespace
2. the version of the InfluxDB package
3. the reference to the container registry pull secrets
4. the hostname on which the API will be exposed
5. parameters to connect to external prerequisites, such as authn tools, object store, …

We'll now go through a step-by-step guide that will show you how to install InfluxDB on your kubernetes cluster. Start by copying the example-customer.yml file so that you can edit it as you go through each step.

```
Unset
cp example-customer.yml myinfluxdb.yml
```

Edit myinfluxdb.yml in your favorite text editor.

The VSCode editor is a good choice since it has a simple way to associate a JSON Schema which can help with autocompletion and validation ensuring the best experience in editing your influxdb parameters. We've provided app-instance-schema.json for this use case.

Step 1: Create a namespace for InfluxDB

Before installing InfluxDB the target namespace must exist. Your friendly admins may provide you with a namespace already, or otherwise you can create one with the following command:

```
Unset
kubectl create namespace influxdb
```

If you chose a different namespace name than `influxdb`, you will need to change `.metadata.namespace` in myinfluxdb.yml to the namespace name that you created.

Step 2: Install kubecfg kubit operator

Install the kubecfg kubit operator which is maintained by InfluxData. The operator simplifies the installation and management of the InfluxDB Clustered package. More specifically, we use it to manage the application of the jsonnet templates that we are using to install our product, and management of product updates. The official page contains instructions, but TL;DR:

```
Unset
kubectl apply -k
'https://github.com/kubecfg/kubit//kustomize/global?ref=v0.0.9'
```

Step 3: Configure access to container registry

The provided influxdb-docker-config.json gives access to a collection of container images that are required to run InfluxDB clustered. Your Kubernetes cluster will need access to the container registry to pull down and install InfluxDB.

There are two main scenarios:

1. You have a kubernetes cluster that can pull from the InfluxData container registry.
2. You run in an air-gapped environment where you can only access a private container registry.

In both cases you need a valid container registry secret file, let's first make sure the one you're provided actually works:

```
Unset
<install crane>
mkdir /tmp/influxdbsecret
cp influxdb-docker-config.json /tmp/influxdbsecret/config.json
DOCKER_CONFIG=/tmp/influxdbsecret \
  crane manifest \
  us-docker.pkg.dev/influxdb2-artifacts/clustered/influxdb:<package-version>
```

If your docker config is valid and you're able to connect to the container registry, that command will succeed and you should see the JSON manifest of that docker image, which looks like this.

```
Unset
{"schemaVersion":2,"config":{"mediaType":"application/vnd.kubecfg.bundle.config
.v1+json","digest":"sha256:6900d2f248e678176c68f3768e7e48958bb96a59232070ff31b3
b018cf299aa7","size":8598},"layers":[{"mediaType":"application/vnd.kubecfg.bund
le.tar+gzip","digest":"sha256:7c1d62e76287035a9b22b2c155f328fae9beff2c6aa7a09a2
dd2697539f41d98","size":404059}],"annotations":{"org.opencontainers.image.creat
ed":"1970-01-01T00:00:00Z","org.opencontainers.image.revision":"unknown","org.o
pencontainers.image.source":"kubecfg pack"}}
```

If there's a problem with the docker config, then you will not be able to retrieve the manifest, and the command will error. For example:

```
Unset
Error: fetching manifest
us-docker.pkg.dev/influxdb2-artifacts/clustered/influxdb:<package-version>: GET
https://us-docker.pkg.dev/v2/token?scope=repository%3Ainfluxdb2-artifacts%2Fclu
stered%2Finfluxdb%3Apull&service=: DENIED: Permission
"artifactregistry.repositories.downloadArtifacts" denied on resource
"projects/influxdb2-artifacts/locations/us/repositories/clustered" (or it may
not exist)
```

Public registry (non air-gapped)

To pull from the InfluxData registry, you need to create a Kubernetes secret in the target namespace.

```
Unset
kubectl create secret docker-registry gar-docker-secret \
  --from-file=.dockerconfigjson=influxdb-docker-config.json \
  -n influxdb
```

You should see "secret/gar-docker-secret created" in response to the above command.

By default this secret is named "gar-docker-secret". If you need to change the name of this secret you can, but you must also change the value of the imagePullSecret field in the AppInstance custom resource to match.

Private registry (air-gapped)

If your kubernetes cluster doesn't have the ability to download container images from our container registry in the public internet, you will need to:

1. copy the images from our registry to your registry
2. configure your AppInstance with a reference to your registry name
3. provide credentials to your own registry

The list of images that you need to copy is included in the package metadata.
You can obtain it with any standard OCI image inspection tool. For example:

```
Unset
DOCKER_CONFIG=/tmp/influxdbsecret \
crane config \
 us-docker.pkg.dev/influxdb2-artifacts/clustered/influxdb:<package-version> \
  | jq -r '.metadata["oci.image.list"].images[]' \
 > /tmp/images.txt
```

This command will produce a list of image names, like

```
Unset
us-docker.pkg.dev/influxdb2-artifacts/idpe/idpe-cd-ioxauth@sha256:5f015a7f28a81
6df706b66d59cb9d6f087d24614f485610619f0e3a808a73864
us-docker.pkg.dev/influxdb2-artifacts/iox/iox@sha256:b59d80add235f29b806badf741
0239a3176bc77cf2dc335a1b07ab68615b870c
...
```

You can then copy the images to your private registry, for example, with the crane command:

```
Unset
</tmp/images.txt  xargs -I% crane cp % myregistry.mystuff.io/%
```

To tell InfluxDB where your registry is, set the `.spec.package.spec.images.registryOverride` field in myinfluxdb.yml, for example:

```
Unset
apiVersion: kubecfg.dev/v1alpha1
kind: AppInstance
...
spec:
  package:
    spec:
      images:
        registryOverride:  myregistry.mystuff.io
```

Step 4: Cluster Setup

Either ingress will need to be provided, or the Nginx Ingress Controller will need to be installed to use our defined Ingress. If using our Ingress, a valid TLS Certificate is needed and should be added to the cluster as a secret. That secret name will then be provided to the config in the next step ("Configure ingress" section).

```
Unset
kubectl create secret tls ingress-tls \
 -n influxdb \
 --cert=path/to/cert/file \
 --key=path/to/key/file
```

Step 5: Modify configuration to point to prereqs

You must configure the hostname via which k8s will expose the API endpoints. In the rest of this document we shall refer to the API endpoint as `INFLUXDB_URL`

Examples: https://influxdb.vpn.my.co

Configure ingress

```
Unset
apiVersion: kubecfg.dev/v1alpha1
kind: AppInstance
...
spec:
  package:
    spec:
...
      ingress:
        hosts:
          - influxdb.vpn.my.co
        tlsSecretName: ingress-tls
```

The `tlsSecretName` parameter is optional. You may want to use it if you already have a TLS certificate for your DNS name.

If instead you want to automatically create an ACME certificate (e.g. with Letsencrypt), please refer to the certmanager documentation for more details on how to annotate the `Ingress`

resource produced by the InfluxDB installer operator. The operator allows you to add annotations (e.g. with `kubectl annotate`) and will preserve them as it operates on resources. It's your responsibility to install and configure certmanager, if you wish to use it.

You can provide multiple hostnames. The ingress layer will accept requests incoming for all the listed hostnames. This can be useful if you want to have distinct paths for your internal and external traffic (e.g. to save on networking costs).

It's your responsibility to configure DNS. Popular options are: a) manually managing DNS records b) external-dns (often provided by your friendly kubernetes admins).

Configure the Object Store

Update the configuration to point to your Object Store (S3 or equivalent).

```
Unset
apiVersion: kubecfg.dev/v1alpha1
kind: AppInstance
...
spec:
  package:
    spec:
      objectStore:
        # URL for S3 Compatible object store
        endpoint: <S3 url>

        # Set to true to allow communication over HTTP (instead of HTTPS)
        allowHttp: "false"

        # S3 Access Key
        # This can also be provided as a valueFrom: secretKeyRef:
        accessKey:
          value: <your access key>

        # S3 Secret Key
        # This can also be provided as a valueFrom: secretKeyRef:
        secretKey:
          value: <your secret>

        # Bucket that the parquet files will be stored in
        bucket: <bucket name>

        # This value is required for AWS S3, it may or may not be required for
other providers.
```

```
        region: <region>
```

Configure the Catalog

Update the configuration to point to your Postgres DB.

```
Unset
apiVersion: kubecfg.dev/v1alpha1
kind: AppInstance
...
spec:
  package:
    spec:
      catalog:
        # A postgresql style DSN that points at a postgresql compatible
database.
        # eg:
postgres://[user[:password]@][netloc][:port][/dbname][?param1=value1&...]
        dsn:
          valueFrom:
            secretKeyRef:
              name: <your secret name here>
              key: <the key in the secret that contains the dsn>
```

If your postgres instance runs without SSL, you must pass the `sslmode=disable` parameter in the Data Source Name (DSN) parameter, e.g.
`postgres://foo:bar@baz:5432/influxdb?sslmode=disable`

Configure local storage for Ingesters

The ingesters need some local storage (for the Write Ahead Log).

```
Unset
apiVersion: kubecfg.dev/v1alpha1
kind: AppInstance
...
spec:
```

```
  package:
    spec:
      ingesterStorage:
        # Set the storage class. This will differ based on the K8s environment
and desired storage characteristics.
        storageClassName: <storage-class>
        # Set the storage size (minimum 2Gi recommended)
        storage: <storage-size>
```

Configure OAuth2

Update Kubernetes configuration file to point to OAuth provider

To configure your OAuth provider, two fields need to be updated in myinfluxdb.yml: `identityProvider` should be set to the name of your identity provider. Note that if you are using Azure Active Directory, this must be set to “azure”. (This is because for Azure AD, we use the OID instead of the subject for authentication. For all other OAuth providers, we use the subject.)

`jwksEndpoint` should be set to the JWKS endpoint value that was obtained in Configuring Identity Provider.

This is what the updates to myinfluxdb.yml look for the Keycloak example:

```
Unset
apiVersion: kubecfg.dev/v1alpha1
kind: AppInstance
...
spec:
  package:
    spec:
      admin:
        # Note for Azure Active Directory it must be exactly "azure"
        identityProvider: keycloak
        # The JWKS endpoint provided by the Identity Provider
        jwksEndpoint: |-
          {KeycloakDomain}/realms/{KeycloakRealm}/protocol/openid-connect/certs
```

Adding users

Finally, you will need to add all the users you wish to have access to use influxctl. Update the `spec.package.spec.admin.users` field with a list of these users. See Adding or removing users for more details.

Configure the size of your cluster

By default, the cluster is configured with 3 ingesters, 1 compactor and 1 querier. This is a reasonable starting point for your testing. We highly recommend 3 ingesters to ensure redundancy on the write path. We also highly recommend a single compactor, as it is more efficient to vertically scale the compactor (assign it more CPU and memory) rather than run multiple compactors. The number of queriers is determined by how many parallel queries you are likely to have, and how long they will take to execute. Once you have your cluster up and running, we will be happy to work with you to recommend appropriate settings for these parameters, based on your anticipated workload.

This is where the updates to myinfluxdb.yml would go, if you want to change the default configuration.

```
Unset
apiVersion: kubecfg.dev/v1alpha1
kind: AppInstance
...
spec:
  package:
    spec:
      # Uncomment the following block to tune the various pods for their
cpu/memory/replicas based on workload needs.
      # Only uncomment the specific resources you want to change, anything
uncommented will use the package default.
      # resources:
      #   # The ingester handles data being written
      #   ingester:
      #     requests:
      #       cpu: <cpu amount>
      #       memory: <ram amount>
      #       replicas: <num replicas> # The default for ingesters is 3 to
increase availability

      #   # The compactor reorganizes old data to improve query and storage
efficiency.
```

```
#   compactor:
#     requests:
#       cpu: <cpu amount>
#       memory: <ram amount>
#       replicas: <num replicas> # the default is 1

#   # The querier handles querying data.
#   querier:
#     requests:
#       cpu: <cpu amount>
#       memory: <ram amount>
#       replicas: <num replicas> # the default is 1

#   # The router performs some api routing.
#   router:
#     requests:
#       cpu: <cpu amount>
#       memory: <ram amount>
#       replicas: <num replicas> # the default is 1
```

Step 6: Deploy an InfluxDB cluster

Unset

```
kubectl apply -f myinfluxdb.yml -n influxdb
```

If you want to check on the status of the deployment, you can use the following command

Unset

```
kubectl get -f myinfluxdb.yml  -o yaml | yq -P .status.conditions
```

The status field contains two main useful pieces of information:

- conditions: summary of the what's going on
- lastLogs: verbose logs of the various stages (template expansion and apply)

For example, a common error is caused by bad container registry credentials:

```
Unset
$ kubectl get -f myinfluxdb.yml -o yaml | yq -P .status.conditions
- lastTransitionTime: "2023-08-18T12:53:54Z"
  message: ""
  observedGeneration: null
  reason: Failed
  status: "False"
  type: Reconcilier
- lastTransitionTime: "2023-08-18T12:53:54Z"
  message: |
    Cannot launch installation job: OCI error: Authentication failure:
{"errors":[{"code":"UNAUTHORIZED","message":"authentication failed"}]}
  observedGeneration: null
  reason: Failed
  status: "False"
  type: Ready
```

After deploying the cluster, you should expect to see a collection of pods like this (the end of the names will be slightly different):

```
Unset
$ kubectl get pods -n influxdb

NAMESPACE     NAME                                  READY   STATUS
RESTARTS       AGE
influxdb      minio-0                               2/2     Running     2
(101s ago)   114s
influxdb      catalog-db-0                          2/2     Running     0
114s
influxdb      keycloak-b89bc7b77-zpt2r              1/1     Running     0
114s
influxdb      debug-service-548749c554-m4sxk        1/1     Running     0
91s
influxdb      token-gen-56a2e859-zlvnw              0/1     Completed   0
91s
influxdb      database-management-579bfb9fcb-dw5sv  1/1     Running     0
91s
influxdb      database-management-579bfb9fcb-22qgm  1/1     Running     0
91s
influxdb      authz-59f456795b-qt52p                1/1     Running     0
91s
influxdb      account-df457db78-j9z6f               1/1     Running     0
91s
```

```
influxdb      authz-59f456795b-ldvmt                  1/1     Running     0
91s
influxdb      account-df457db78-8ds4f                 1/1     Running     0
91s
influxdb      token-management-754d966555-fmkbk       1/1     Running     0
90s
influxdb      token-management-754d966555-rbvtv       1/1     Running     0
90s
influxdb      global-gc-7db9b7cb4-ml6wd               1/1     Running     0
91s
influxdb      iox-shared-compactor-0                  1/1     Running     1
(62s ago)    91s
influxdb      iox-shared-ingester-0                   1/1     Running     1
(62s ago)    91s
influxdb      iox-shared-ingester-1                   1/1     Running     1
(62s ago)    91s
influxdb      iox-shared-ingester-2                   1/1     Running     1
(62s ago)    91s
influxdb      global-router-86cf6b869b-56skm          3/3     Running     1
(62s ago)    90s
influxdb      iox-shared-querier-7f5998b9b-fpt62      4/4     Running     1
(62s ago)    90s
influxdb      kubit-apply-influxdb-g6qpx              0/1     Completed   0
8s
```

Part 3: Interacting with InfluxDB

Influxctl

Influxctl is the command line tool for administering the InfluxDB product. Download the latest version of influxctl from influxdata.com/downloads or from the InfluxData Package archive.

Next, create a configuration file, `config.toml`, for influxctl to interact with the cluster. The following example uses a cluster running on localhost and port 7080 with Keycloak as the OAuth2 provider. The host and port is based on the Ingress setup chosen. The client_id, device_url, and token_url fields should be set to the corresponding values obtained in Configuring Identity Provider.

```
Unset
[[profile]]
name = "default"
product = "clustered"
host = "localhost" # The hostname/IP of the InfluxDB 3.0 Ingress
port = "7080"      # The port of the InfluxDB 3.0 Ingress
                   # (will typically be 80 when using a hostname)

[profile.auth.oauth2]
client_id = "client"
scopes = [""]
token_url = "http://localhost:8080/realms/realm/protocol/openid-connect/token"
device_url =
"http://localhost:8080/realms/realm/protocol/openid-connect/auth/device"
```

Create a database

The first step in using InfluxDB is to create a database. On the first command run with influxctl, you will be prompted to authenticate with their OAuth2 provider. Note that your user id must have been added to myinfluxdb.yml in order for you to be authorized to use the InfluxDB cluster. Once authenticated, the command will create the requested database.

```
Unset
influxctl --config config.toml database create mytestdatabase
```

Create a token

Use influxctl to create a token. Ensure you save the token as this is the only time you will ever see the token string itself.

```
Unset
influxctl --config config.toml token create \
    --read-database mytestdatabase \
    --write-database mytestdatabase \
    mytoken
```

Write to InfluxDB

Use the token created above with your favorite /write api tool. If you don't have a line protocol file to use, you can start by using our sample file at https://raw.githubusercontent.com/influxdata/influxdb2-sample-data/master/air-sensor-data/air-sensor-data.lp

```
Unset
influx write --token "$TOKEN" \
    --host "$INFLUXDB_URL" \
    --org testing \
    --bucket mytestdatabase \
    --file <your-line-protocol.lp>
```

Query from InfluxDB

Use the token created above with your favorite GRPC query api tool

```
Unset
influxdb_iox query --token "$TOKEN" \
    --host "$INFLUX_URL" \
    mytestdatabase  <SQL Query>
```

If you used our sample file, the following command will show you the last 15 minutes of sample data:

```
Unset
influxdb_iox query --token "$TOKEN" \
--host "$INFLUX_URL" \
mytestdatabase 'SELECT * FROM airSensors WHERE time >= now() - 15m'
```

Appendix

Updating your InfluxDB Cluster

Updating your InfluxDB cluster is as simple as re-applying your app-instance with a new package version. Note that if the new version of the package has changes to the AppInstance

schema, those changes will need to be made at the same time that the new package is deployed.

Redeploying your cluster safely

The word safely here means being able to redeploy your cluster while still being able to use the tokens you've created, and being able to write/query to the database you've previously created.

All of the important state in Influxdb 3.0 lives in the Catalog (the Postgres equivalent database) and the Object Store (the S3 compatible store). These should be treated with the utmost care.

If a full redeploy of your cluster needs to happen, the namespace containing the Influxdb instance can be deleted ***as long as your Catalog and Object Store are not in this namespace***. Then, the influxdb AppInstance can be redeployed. It is possible the operator may need to be removed and reinstalled. In that case, deleting the namespace that the operator is deployed into and redeploying is acceptable.

Backing up your data

The Catalog and Object store contain all of the important state for Influxdb 3.0. They should be the primary focus of backups. Following the industry standard best practices for your chosen Catalog implementation and Object Store implementation should provide sufficient backups. In our Cloud products, we do daily backups of our Catalog, in addition to automatic snapshots, and we preserve our Object Store files for 100 days after they have been soft-deleted.

Recovering your data

After recovering the catalog and object store, you will need to update the dsn in myinfluxdb.yml and re-apply.

Configuring Identity Provider

Keycloak

To configure Keycloak as your identity provider, you must have a Keycloak realm with users added and a Keycloak client with device flow enabled. You also need to configure your Clustered config file and your influxctl profile to reference the Keycloak realm/client.

Create a New Keycloak Client

1. In the Keycloak Admin Console, navigate to Clients from the left-hand-nav and click "Create Client".

2. On the General Settings step, set the “Client type” to “OpenID Connect” and enter a “Client ID”, then click “Next”. This Client ID will be used later when configuring Clustered.
3. On the Capability config step, enable the “OAuth 2.0 Device Authorization Grant” Authentication flow, then click “Next”.
4. On the Login settings step, don’t change anything and just click “Save”.

Configure InfluxDB Clustered to Use Keycloak

Navigate to `{Keycloak Domain}/realms/{Keycloak Realm}/.well-known/openid-configuration` to retrieve a json of the OpenID configuration values for your Keycloak realm.

Copy the following values from the response:

- **"jwks_uri"**
  - example:
  - `{Keycloak Domain}/realms/{Keycloak Realm}/protocol/openid-connect/certs`
- **"device_authorization_endpoint"**
  - example:
  - `{Keycloak Domain}/realms/{Keycloak Realm}/protocol/openid-connect/auth/device`
- **"token_endpoint"**
  - example:
  - `{Keycloak Domain}/realms/{Keycloak Realm}/protocol/openid-connect/token`

In your configuration yaml for clustered (for example, myinfluxdb.yml - see Step 5), set your `identityProvider` to `keycloak` (all lowercase) and your `jwksEndpoint` to the *jwks_uri* you copied.

Example:

```
Unset
...
  spec:
    admin:
      identityProvider: keycloak
      jwksEndpoint: |-
        {Keycloak Domain}/realms/{Keycloak Realm}/protocol/openid-connect/certs
```

In your configuration TOML for the influxctl CLI (see Step 7), set the `client_id` to the Client ID that you specified in step 2 of the previous section; your `device_url` to your *device_authorization_endpoint*; and your `token_url` to your *token_endpoint*.

Example:

```
Unset
[profile.auth.oauth2]
client_id = “your_client_id”
device_url =
"http://localhost:8080/realms/your_realm/protocol/openid-connect/auth/device"
token_url =
"http://localhost:8080/realms/your_realm/protocol/openid-connect/token"
```

Azure

To configure Azure as your identity provider, you need an existing Azure Active Directory (aka Microsoft Entra ID) tenant, with your users added. Copy your **Azure AD Tenant ID**.

Retrieve a json of the OpenID configuration values for your Azure AD Tenant by navigating to the openid configuration endpoint. For Azure, you must use the values from the **v2** endpoint.

```
https://login.microsoftonline.com/{AZURE_TENANT_ID}/v2.0/.well-known/openid-configuration
```

Configure the Azure AD Portal

In the Azure Portal, select “App Registrations” (left-hand nav), click “New Registration” (top nav), and enter a name for a new application to handle authentication requests. Hit “register application.” Copy the **Application (Client) ID** for this newly registered app.

Within your newly registered application, click the “Authentication” menu on the left-hand side of the screen. Scroll down to “Advanced Settings” and set “Allow public client flows” to “yes.” This will enable use of the “device code” flow for logging in to InfluxDB Clustered.

Configure InfluxDB Clustered to Use Azure AD

Retrieve the json of the OpenID configuration values for your Azure AD Tenant by navigating to:

```
https://login.microsoftonline.com/{AZURE_TENANT_ID}/v2.0/.well-known/openid-configuration
```

Copy the following values from the response.

- **“jwks_uri”**
  - `https://login.microsoftonline.com/{AZURE_TENANT_ID}/discovery/v2.0/keys`
- **"device_authorization_endpoint"**
  - `https://login.microsoftonline.com/{AZURE_TENANT_ID}/oauth2/v2.0/devicecode`

- **“token_endpoint”**
  - `https://login.microsoftonline.com/{AZURE_TENANT_ID}/oauth2/v2.0/token`

In your configuration YAML for clustered (for example, myinfluxdb.yml - see Step 5), set your `identityProvider` to `azure` (all lowercase) and your `jwksEndpoint` to the *jwks_uri* you copied.

Example:

```
Unset
...
  spec:
    admin:
      identityProvider: azure
      jwksEndpoint: |-
        https://login.microsoftonline.com/{Azure Tenant ID}/discovery/v2.0/keys
```

In your configuration TOML for the influxctl CLI (see Step 7), set the `client_id` to your *Application (Client) ID*; your scopes to `["{your application (client) id}/.default"]` ; your `device_url` to the v2.0 *device_authorization_endpoint*; and your `token_url` to the v2.0 *token_endpoint*;

Example:

```
Unset
[profile.auth.oauth2]
client_id = “ff5c3415-17b1-4220-ac78-5631a6891b9d”
scopes = ["ff5c3415-17b1-4220-ac78-5631a6891b9d/.default"]
device_url =
"https://ff5c3415-17b1-4220-ac78-5631a6891b9d/oauth2/v2.0/devicecode"
token_url =
"https://login.microsoftonline.com/ff5c3415-17b1-4220-ac78-5631a6891b9d/oauth2/
v2.0/token"
```

Adding or removing users

Finding User ID

When specifying users in `myinfluxdb.yml`, you must provide the id that your identity provider uses to identify the user. For Azure, this is the OID in any JWTs for that user. For all other providers, it is the Subject in the JWT.

Keycloak

For Keycloak, you can find the user ID either through the admin console or the REST API. In the admin console, navigate to your realm, then to users, and then to the user you wish to find the ID for. The ID should be listed in the *Details* tab. With the REST api, you can make a get request to the following url to fetch the ID for a specific user.

```
Unset
https://{keycloak-domain}:{keycloak-port}/auth/admin/realms/{realm-name}/users?username=testUser
```

Azure

For Azure AD, the unique user id is the Microsoft ObjectId (OID). You can download a list of the OIDs for all users in your Tenant by navigating to the “Users” page, selecting the checkbox for the users you wish to download (click the top checkbox to select all users), and clicking the “Download Users” button at the top of the screen. In the downloaded CSV file, the users’ OIDs are reflected in the `id` field.

Auth0

For Keycloak, you can find the user ID through the web UI. Log in to your tenant and navigate to User Management. From there, search for the user. Once found, the user ID can be found at the top of the page under the user's name.

Applying Changes

To add or remove users, update the users list in myinfluxdb.yml file. The users list is found at spec.package.spec.admin.users. After updating the list, re-apply `myinfluxdb.yml`. See Step 6: Deploy an InfluxDB cluster for details on how to apply myinfluxdb.yml. Once the `myinfluxdb.yml` has been applied, it will take a couple minutes for the updates to roll out. When complete, any new users will have been added and any removed users will have been deleted.

Custom CA

See `example-customer.yml` section on "Custom CA"