ExternalDNS allows you to automate DNS record management when working with the Ingress or Service resources. If these resources are configured according to ExternalDNS requirements, they will be accessible by their domain names as soon as they are created.

ExternalDNS integrates with the VK Cloud DNS service via a plugin. Below is an example of installing ExternalDNS to a cluster and using this tool with the `Ingress` and `Service` resources.

Preparatory steps

1. Create a DNS zone for ExternalDNS to work with, if not already done.

   In the example below, the `example.com` zone is used.

2. Create a Cloud Containers cluster of the latest version that has an external IP address and is accessible from the Internet.

   Select other cluster parameters at your discretion.

3. Make sure that you can connect to the cluster using `kubectl`.

   To connect, use the cluster configuration file (kubeconfig) downloaded from your VK Cloud personal account.

4. Install Helm version 3.0.0 or higher if the utility is not already installed.

   To install, select a version of Helm that is compatible with the cluster.

5. Set an environment variable that points to kubeconfig for the cluster. This will make it easier to work with the `kubectl` and `helm` utilities when installing ExternalDNS.

   The path to your kubeconfig file may differ from the example below.

   Linux (bash)/macOS (zsh) Windows (PowerShell)

   ```
   export KUBECONFIG="/home/user/.kube/kubernetes-cluster-1234_kubeconfig.yaml"
   ```

   ```
   $Env:KUBECONFIG="C:\Users\user\.kube\kubernetes-cluster-1234_kubeconfig.yaml"
   ```

1. Prepare a user for ExternalDNS

ExternalDNS will use this VK Cloud user's credentials to cooperate with the VK Cloud API and manage DNS resource records.

Prepare the user and get all the necessary credentials:

1. Select an existing user or invite a new user to the project.

   User Requirements:

   - API access must be enabled.
   - One of the following roles must be assigned in order for ExternalDNS to operate resource records within the DNS zone:

     - Network Administrator (a minimum required role).
     - Project Administrator.
     - Superadministrator.
     - Project Owner.

     To work with ExternalDNS, it is recommended to assign a dedicated user with the Network administrator role. This will minimize possible damage if an attacker gains access to this user's credentials.

2. Get the credentials you need to access the VK Cloud API:

   1. Go to your VK Cloud personal account using the credentials of the user assigned to ExternalDNS.

   2. Click the username in the page header and select **Project Settings**.

   3. Go to the **API access** tab and note the values of the following parameters:

      - Project ID (Project ID for OpenStack);
      - User Domain Name;
      - Username;
      - Region Name;
      - Auth URL (Authentication URL).

3. Write down the password for this user: it is also required to access the API.

2. Install ExternalDNS

1. Create a namespace where ExternalDNS will be installed:

   ```
   kubectl create ns external-dns
   ```

2. In this namespace, create a secret that contains the VK Cloud API access credentials obtained during user preparation:

   Linux (bash)/macOS (zsh) Windows (PowerShell)

   ```
   kubectl -n external-dns create secret generic vkcs-auth \
     --from-literal=ProjectID="" \
     --from-literal=UserDomainName="" \
     --from-literal=Username="" \
     --from-literal=RegionName="" \
     --from-literal=AuthURL="" \
     --from-literal=Password=""
   ```

   ```
   kubectl -n external-dns create secret generic vkcs-auth `
     --from-literal=ProjectID="" `
     --from-literal=UserDomainName="" `
     --from-literal=Username="" `
     --from-literal=RegionName="" `
     --from-literal=AuthURL="" `
     --from-literal=Password=""
   ```

3. Add the Bitnami Helm repository:

   ```
   helm repo add bitnami https://charts.bitnami.com/bitnami
   ```

4. Create a file that contains the values needed to install ExternalDNS with Helm:

   external-dns-vkcs-values.yaml

   ```
   policy: upsert-only
   txtPrefix: externaldns-

   provider: webhook

   extraArgs:
     webhook-provider-url: http://localhost:8888

   sidecars:
     - name: vkcs-plugin
       image: registry.infra.mail.ru:5010/external-dns-vkcs-plugin:latest
       imagePullPolicy: Always
       ports:
         - name: http
           containerPort: 8888
       livenessProbe:
         httpGet:
           path: /healthz
           port: http
         initialDelaySeconds: 10
         timeoutSeconds: 5
       readinessProbe:
         httpGet:
           path: /healthz
   ```

```
          port: http
        initialDelaySeconds: 10
        timeoutSeconds: 5
      env:
        - name: OS_AUTH_URL
          valueFrom:
            secretKeyRef:
              name: vkcs-auth
              key: AuthURL
        - name: OS_USERNAME
          valueFrom:
            secretKeyRef:
              name: vkcs-auth
              key: Username
        - name: OS_PASSWORD
          valueFrom:
            secretKeyRef:
              name: vkcs-auth
              key: Password
        - name: OS_PROJECT_ID
          valueFrom:
            secretKeyRef:
              name: vkcs-auth
              key: ProjectID
        - name: OS_DOMAIN_NAME
          valueFrom:
            secretKeyRef:
              name: vkcs-auth
              key: UserDomainName
        - name: OS_REGION_NAME
          valueFrom:
            secretKeyRef:
              name: vkcs-auth
              key: RegionName
        - name: SERVER_HOST
          value: "0.0.0.0"
        - name: SERVER_PORT
          value: "8888"
```

The behavior of the ExternalDNS Helm chart is influenced by many values. The created file contains a minimum set of values that is sufficient to start working with ExternalDNS. The most important values affecting the behavior of ExternalDNS with the VK Cloud DNS are described below. Descriptions of values that do not apply to the VK Cloud plugin (all values except `sidecars[]`) are given in README.md for the chart.

Do not change or delete the required values listed below. This may cause ExternalDNS to work incorrectly.

Description of important values that affect the behavior of ExternalDNS

- (**Required value**) `provider`: the `webhook` value indicates that an external plugin with webhooks support is required to work with DNS.
- (**Required value**) `extraArgs:webhook-provider-url`: the URL to be used to interact with the plugin.
- `policy`: the resource record synchronization policy. The default policy is `upsert-only`: ExternalDNS will only add resource records, not delete them.

  If you want ExternalDNS to remove resource records when deleting Kubernetes resources for which the records were created, use the `sync` policy.
- `txtPrefix`: prefix for TXT records that are created by ExternalDNS.

  ExternalDNS can automatically add both A records and CNAME records for Kubernetes resources.

  ExternalDNS keeps track of which DNS zone resource records it manages by placing service information in TXT records. In particular, with the default settings, it will create a service TXT record with the same name as the record being added: for example, for the A record `example.com`, a corresponding TXT record with the same name `example.com` will be created.

  However, according to RFC 1912, CNAME records cannot coexist with other records with the same name. That is why ExternalDNS is configured to prefix the name of TXT records with the value specified in `txtPrefix`. This allows you to avoid possible collisions when working with CNAME records: for example, for the CNAME record `example.com` a corresponding TXT record with the name `externaldns-example.com` will be created.

  You can specify a prefix different from `externaldns-`, if necessary.

The plugin for ExternalDNS, which provides its integration with the VK Cloud DNS, has many settings that affect the plugin's behavior. The settings are set using environment variables in `sidecars[].env`. In the created file, only the required settings are set. If necessary, you can specify additional settings for the plugin by adding the appropriate environment variables.

Do not modify or delete the required plugin settings listed below. This may cause ExternalDNS to work incorrectly.

Description of values that affect plugin behavior

- (**Required settings**) The settings corresponding to the variables with the `OS_` prefix are used to authenticate the plugin when interacting with the VK Cloud API.

  The values of these variables are stored in the Kubernetes secret that was created earlier.

- (**Required settings**) The settings corresponding to the `SERVER_HOST` and `SERVER_PORT` variables have fixed values and are necessary for the plugin to work correctly.

- DNS zone filtering settings:

  - Filters for DNS zones in which resource records are allowed to be created:

    - `DOMAIN_FILTERS`: a string with a list of domain names separated by commas. For example, `example.com,contoso.com`.
    - `REGEX_DOMAIN_FILTER`: a string with a regular expression (RE2 syntax). For example, `.*.com$`.

    If both filters are configured, the `REGEX_DOMAIN_FILTER` filter takes precedence over `DOMAIN_FILTERS`. By default, no filtering is performed.

  - Filters for DNS zones in which resource records are not allowed to be created:

    - `EXCLUDE_DOMAINS`: a string with a list of domain names separated by commas. For example, `example.org,foo.bar.com`.
    - `REGEX_DOMAIN_FILTER_EXCLUSION`: a string with a regular expression (RE2 syntax). For example, `^stage-.*.com$`.

    If both filters are configured, the `REGEX_DOMAIN_FILTER_EXCLUSION` filter takes precedence over `EXCLUDE_DOMAINS`. By default, no filtering is performed.

  - `SERVER_READ_TIMEOUT`: timeout for reading when the connection to the server is open (in seconds). Default value: `30`.

  - `SERVER_WRITE_TIMEOUT`: timeout for writing when the connection to the server is open (in seconds). Default value: `30`.

  - `LOG_LEVEL`: level of logging of events that occur during plugin operation.

    The `error`, `warn`, `info`, `debug`, and `trace` levels are supported. Default value: `info`.

  - `DRY_RUN`: flag that allows the plugin to be run in "dry run" mode.

    - `false` (default): "dry run" mode is **disabled**. The plugin runs and manipulates resource records in the DNS zone as configured.
    - `true`: "dry run" mode is **enabled**. The plugin runs but does not manipulate resource records in the DNS zone: no resource records will be created or deleted.

5. Install ExternalDNS:

   ```
   helm -n external-dns install external-dns-vkcs bitnami/external-dns -f external-dns-vkcs-values.yaml
   ```

6. Verify that the Helm chart has been successfully deployed:

   ```
   helm -n external-dns list && kubectl -n external-dns get all
   ```

   Example of partial output of the command

   ```
   NAME                    NAMESPACE        ...        ...   STATUS          CHART                   ...
   external-dns-vkcs       external-dns     ...        ...   deployed        external-dns-6.32.1     ...

   NAME                                        READY    STATUS     RESTARTS    AGE
   pod/external-dns-vkcs-NNNNNNNNNN-MMMMM      2/2      Running    0           ...
   ```

```
NAME                          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/external-dns-vkcs     ClusterIP   10.254.169.195          7979/TCP   ...

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/external-dns-vkcs      1/1     1            1           87s

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/external-dns-vkcs-NNNNNNNNNN      1         1         1       ...
```

3. Verify that External DNS is working

Next, several demo applications based on NGINX's Cafe example will be deployed. These applications will be published (made available from the Internet) using `Service` and `Ingress` configured to work with ExternalDNS.

3.1. Publish the application using a service like LoadBalancer

1. Create a manifest for the `tea` application:

   tea-app.yaml

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: tea
     labels:
       app: tea
   spec:
     replicas: 3
     selector:
       matchLabels:
         app: tea
     template:
       metadata:
         labels:
           app: tea
       spec:
         containers:
         - name: tea
           image: nginxdemos/nginx-hello:plain-text
           ports:
           - containerPort: 8080
   ```

2. Apply this manifest to the cluster to deploy the application:

   ```
   kubectl apply -f tea-app.yaml
   ```

3. Verify that the application has been successfully deployed as a `ReplicaSet` of three replicas:

   ```
   kubectl get rs,pod -l app==tea
   ```

   Example of partial output of the command

   ```
   NAME                             DESIRED   CURRENT   READY   AGE
   replicaset.apps/tea-XXXXXXXXX    3         3         3       ...

   NAME                          READY   STATUS    RESTARTS   AGE
   pod/tea-XXXXXXXXX-AAAAA       1/1     Running   0          ...
   pod/tea-XXXXXXXXX-BBBBB       1/1     Running   0          ...
   pod/tea-XXXXXXXXX-CCCCC       1/1     Running   0          ...
   ```

4. Create a `tea-service.yaml` manifest for the Kubernetes service (`Service`).

   This service will be used to publish the deployed application. The application will be accessible by the domain name `tea.example.com`.

   To have ExternalDNS create resource records for the service:

   - The `external-dns.alpha.kubernetes.io/hostname` annotation must be set for the service.
   - The service must have the LoadBalancer type.

   Specify the `external-dns.alpha.kubernetes.io/ttl` annotation if you want to set a non-standard TTL for resource records (default: 86400 seconds, 24 hours).

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    external-dns.alpha.kubernetes.io/hostname: "tea.example.com"
    external-dns.alpha.kubernetes.io/ttl: "3600"
  name: tea-svc
  labels:
    app: tea
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 8080
    protocol: TCP
    name: http
  selector:
    app: tea
```

Here:

- The required `external-dns.alpha.kubernetes.io/hostname` annotation is specified: the domain name to use for the service.
- The optional `external-dns.alpha.kubernetes.io/ttl` annotation is specified: TTL in seconds for the resource record to be created by ExternalDNS.
- The `LoadBalancer` service type is selected. A standard load balancer will be created for such a service. Since the load balancer is created with a public IP address, the application associated with the service will be accessible from the Internet.

  Using the load balancer is charged.

5. Apply this manifest to the cluster to create the service:

   ```
   kubectl apply -f tea-service.yaml
   ```

6. Check the status of the service:

   ```
   kubectl get svc tea-svc
   ```

   Wait until the service is assigned the public IP address of the load balancer. Creating the load balancer can take a long time.

   Example of partial output of the command

   - The balancer creation is in progress:

     ```
     NAME      TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)        AGE
     tea-svc   LoadBalancer   10.254.170.195         80:32314/TCP   ...
     ```

   - The balancer has been successfully created:

     ```
     NAME      TYPE           CLUSTER-IP       EXTERNAL-IP       PORT(S)        AGE
     tea-svc   LoadBalancer   10.254.170.195   203.0.113.111     80:32314/TCP   ...
     ```

7. Verify that ExternalDNS has created the necessary resource records:

   1. Get a list of resource records for the `example.com` zone.
   2. Find the entries created by ExternalDNS in the list:

      - One A record `tea.example.com`.
      - Two TXT records `externaldns-tea.example.com` and `externaldns-a-tea.example.com`.

        These TXT records are service records used by ExternalDNS to track the status of the `tea-svc` A record created for the `tea-svc` service.

        Such records are easily distinguished by the `externaldns-` prefix in their name. Their values have the standard structure of the form `heritage=.../owner=.../resource=...`.

   If you specified a different prefix value when installing ExternalDNS, the names of the service TXT records will be different.

   If the desired resource records are not present, wait a few more minutes. ExternalDNS will start creating resource records after the service has been assigned an IP address. This will take some time.

8. Check that the application is accessible by domain name. To do this, go to `http://tea.example.com` in your browser.

   You should see a page with a response from the application like this:

   ```
   Server address: 10.100.184.219:8080
   Server name: tea-XXXXXXXXX-AAAAA
   Date: 09/Feb/2024:10:09:51 +0000
   URI: /
   Request ID:
   ```

   Successful interaction with the application at this address indicates that ExternalDNS works correctly with a service like `LoadBalancer`.

3.2. Publish the application using Ingress

1. Install the Ingress NGINX addon of the latest version to the cluster.

   Perform a **standard installation**. Do not change any parameters, only edit the addon configuration code:

   1. Make sure that the `service.beta.kubernetes.io/openstack-internal-load-balancer` annotation is set to `false`:

      ```
      controller:
        ...
        service:
          annotations: {"loadbalancer.openstack.org/proxy-protocol": "true", "service.beta.kubernetes.io/openstack-internal-load-balancer": "false"}
          ...
      ```

      This is necessary to create a load balancer with a public IP address for the Ingress controller. Then the application using Ingress will be accessible from the Internet.

   2. Set the `controller.publishService.enabled` field to `true`:

      ```
      controller:
        ...
        publishService:
          enabled: true
          pathOverride: ""
          ...
      ```

      This is necessary to assign a public IP address of the Ingress controller to the Ingress resource. This will allow ExternalDNS to create the correct resource records for Ingress.

   Wait for the addon installation to complete. This process may take a long time.

2. Create a manifest for the `coffee` application:

   coffee-app.yaml

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: coffee
     labels:
       app: coffee
   spec:
     replicas: 2
     selector:
       matchLabels:
         app: coffee
     template:
       metadata:
         labels:
           app: coffee
       spec:
   ```

```
      containers:
      - name: coffee
        image: nginxdemos/nginx-hello:plain-text
        ports:
        - containerPort: 8080
```

3. Create a manifest for the Kubernetes service that the application will use.

   coffee-service.yaml

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: coffee-svc
     labels:
      app: coffee
   spec:
     type: ClusterIP
     ports:
     - port: 80
       targetPort: 8080
       protocol: TCP
       name: http
     selector:
       app: coffee
   ```

   Here, the service type is set to ClusterIP, which is sufficient since the application will be published using Ingress. Such a service is accessible only from within the cluster and does not have a dedicated load balancer, unlike the service created earlier.

4. Apply these manifests to the cluster to create all necessary resources:

   ```
   kubectl apply -f coffee-app.yaml -f coffee-service.yaml
   ```

5. Check that the application has been successfully deployed as a `ReplicaSet` of two replicas along with the corresponding service:

   ```
   kubectl get rs,pod,svc -l app==coffee
   ```

   Example of partial output of the command

   ```
   NAME                                DESIRED   CURRENT   READY   AGE
   replicaset.apps/coffee-YYYYYYYYY    2         2         2       ...

   NAME                          READY   STATUS    RESTARTS   AGE
   pod/coffee-YYYYYYYYY-DDDDD    1/1     Running   0          ...
   pod/coffee-YYYYYYYYY-EEEEE    1/1     Running   0          ...

   NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
   service/coffee-svc   ClusterIP   10.254.243.13           80/TCP    ...
   ```

6. Create the `cafe-ingress.yaml` manifest for the Ingress resource.

   This Ingress resource will be used to publish the deployed application. The application will be available on the `cafe.example.com` domain at the `http://cafe.example.com/coffee` URL.

   For ExternalDNS to create resource records for the service, no additional values need to be specified in the manifest: having an Ingress controller configured in the right way is sufficient. The domain name values will be taken from the `host` fields for the Ingress `spec.rules[]` rules.

   Specify the `external-dns.alpha.kubernetes.io/ttl` annotation if you want to set a non-standard TTL for resource records (default: 86400 seconds, 24 hours).

   ```
   apiVersion: networking.k8s.io/v1
   kind: Ingress
   metadata:
     name: cafe-ingress
     annotations:
       external-dns.alpha.kubernetes.io/ttl: "3600"
   spec:
     ingressClassName: nginx
     rules:
     - host: cafe.example.com
       http:
         paths:
         - path: /coffee
   ```

```
          pathType: Prefix
          backend:
            service:
              name: coffee-svc
              port:
                number: 80
```

Here:

- The optional `external-dns.alpha.kubernetes.io/ttl` annotation is specified: TTL in seconds for the resource records to be created by ExternalDNS.
- The `cafe.example.com` host for the Ingress rule is specified. Since only one host is specified, one resource record will be created for this domain name.
- According to the Ingress rule, the `coffee` application that is behind the `coffee-svc` service will be accessible at the URL `http://cafe.example.com/coffee`.

7. Apply this manifest to the cluster to create the resource:

```
kubectl apply -f cafe-ingress.yaml
```

8. Check the status of Ingress:

```
kubectl get ingress cafe-ingress
```

Wait for the Ingress resource to be assigned a public IP address. This address will be the same as the address of the Ingress controller.

Example of partial output of the command

- Ingress has not yet been assigned an IP address:

```
NAME           CLASS   HOSTS              ADDRESS   PORTS   AGE
cafe-ingress   nginx   cafe.example.com             80      ...
```

- Ingress is assigned an IP address:

```
NAME           CLASS   HOSTS              ADDRESS                PORTS   AGE
cafe-ingress   nginx   cafe.example.com   203.0.113.222.nip.io   80      ...
```

9. Verify that ExternalDNS has created the necessary resource records:

    1. Get a list of resource records for the `example.com` zone.

    2. Find the entries created by ExternalDNS in the list:

        - One CNAME record `cafe.example.com`.

        - Two TXT records `externaldns-cafe.example.com` and `externaldns-cname-cafe.example.com`.

          These TXT records are service records used by ExternalDNS to track the status of the `cafe-ingress` CNAME record created for Ingress.

    If the desired resource records are not present, wait a few more minutes. ExternalDNS will begin creating resource records after the Ingress resource has been assigned an IP address. This will take some time.

10. Verify that the application is accessible by domain name. To do this, go to `http://cafe.example.com/coffee` in your browser.

A page should open with a response from the application like:

```
Server address: 10.100.184.220:8080
Server name: coffee-YYYYYYYYY-DDDDD
Date: 09/Feb/2024:13:07:11 +0000
URI: /coffee
Request ID:
```

Successful interaction with the application at this address indicates that ExternalDNS works correctly with the Ingress resource.

Remove unused resources

1. If you no longer need the Kubernetes resources created for ExternalDNS validation, delete them:

    1. Delete all resources associated with the `tea` application:

        ```
        kubectl delete -f cafe-ingress.yaml -f coffee-service.yaml -f coffee-app.yaml
        ```

        It can take a long time to remove the load balancer associated with the service.

    2. Remove all resources associated with the `coffee` application:

        ```
        kubectl delete -f tea-service.yaml -f tea-app.yaml
        ```

    3. Delete the Ingress NGINX addon.

        It may take a long time to remove the addon and its associated resources.

    4. Delete the resource records created by ExternalDNS.

        This must be done if you did not modify the `external-dns-vkcs-values.yaml` file when installing ExternalDNS: in this case ExternalDNS uses the `upsert-only` policy and does not remove resource records from the DNS zone when removing Kubernetes resources. If you modified this file and selected the `sync` policy, then these records will be deleted automatically.

        List of resource records:

        - A record `tea.example.com`.
        - TXT records `externaldns-tea.example.com` and `externaldns-a-tea.example.com`.
        - CNAME record `cafe.example.com`.
        - TXT records `externaldns-cafe.example.com` and `externaldns-cname-cafe.example.com`.

2. If you no longer need ExternalDNS, delete it:

    1. Remove the Helm chart from ExternalDNS:

        ```
        helm -n external-dns uninstall external-dns-vkcs
        ```

    2. Remove the `external-dns` namespace.

        The `vkcs-auth` secret, which contains the credentials to access the VK Cloud API, will also be removed.

        ```
        kubectl delete ns external-dns
        ```

3. A running Cloud Containers cluster consumes compute resources and is charged. If you no longer need it:

    - stop it to use it later;
    - delete it permanently.

4. Delete the `example.com` DNS zone if you no longer need it.