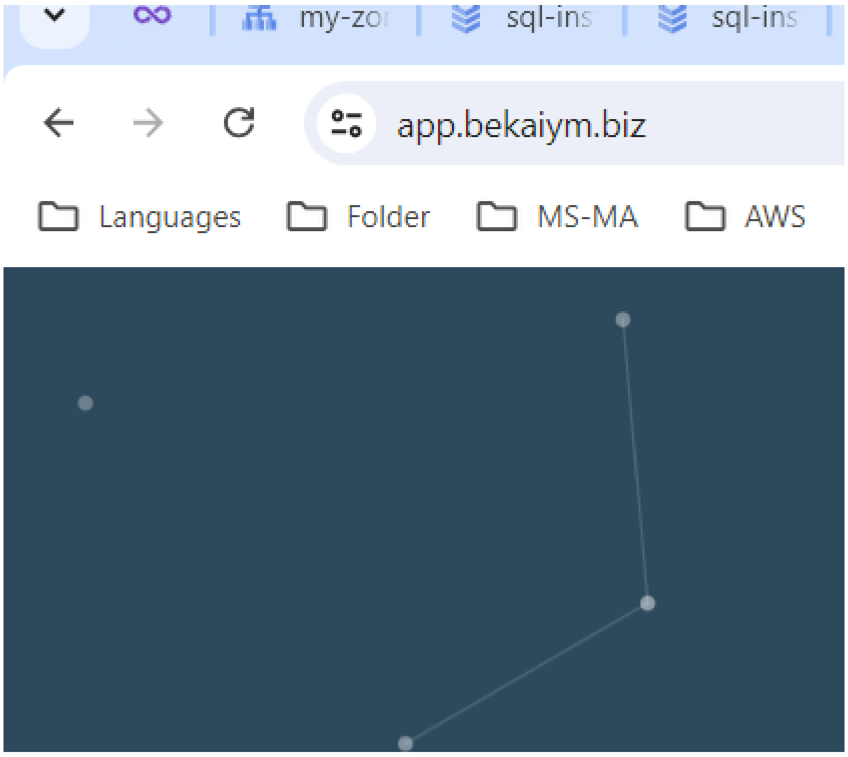
Done by Bekaiym Egemkulova

1. Create RDS or CloudSQL instance, connect to your database and create tables inside postgres database
2. Create a secret with your hostname, username, password and database name.
3. Clone this repos locally:
https://github.com/AntTechLabs/awesome_cats_backend.git
https://github.com/AntTechLabs/awesome_cats_frontend.git
4. Write Dockerfile for frontend and backend images
5. Push your images to private repos in ECR or GCR
6. Write yaml files for backend and frontend deployments with 2 replicas, show database credentials as env variable
7. Create clusterIP services for your deployments
8. Install nginx controller and create ingress to access your application with load balancer.
9. Configure your domain name with Cloud DNS or Route53 and ExternalDNS, access awesome cats application from web browser with your domain name
10. Get certificate to your domain name with cert-manager

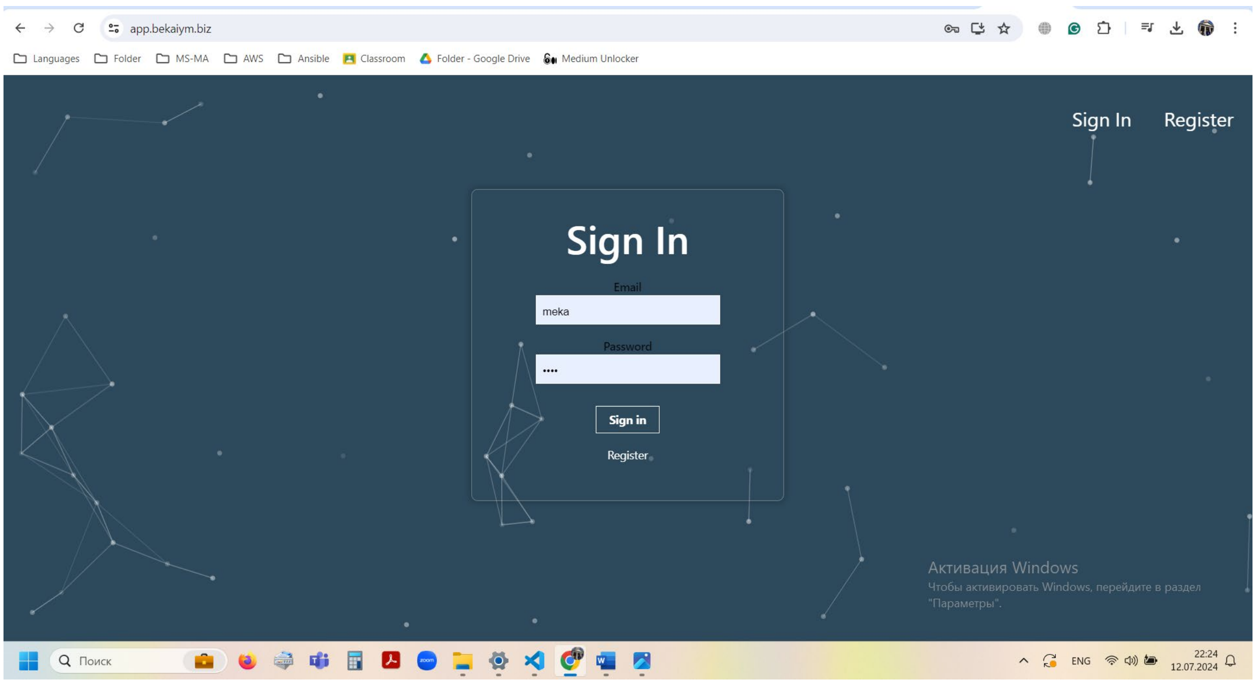
== && == Answers == && == Bekaiym Egemkulova

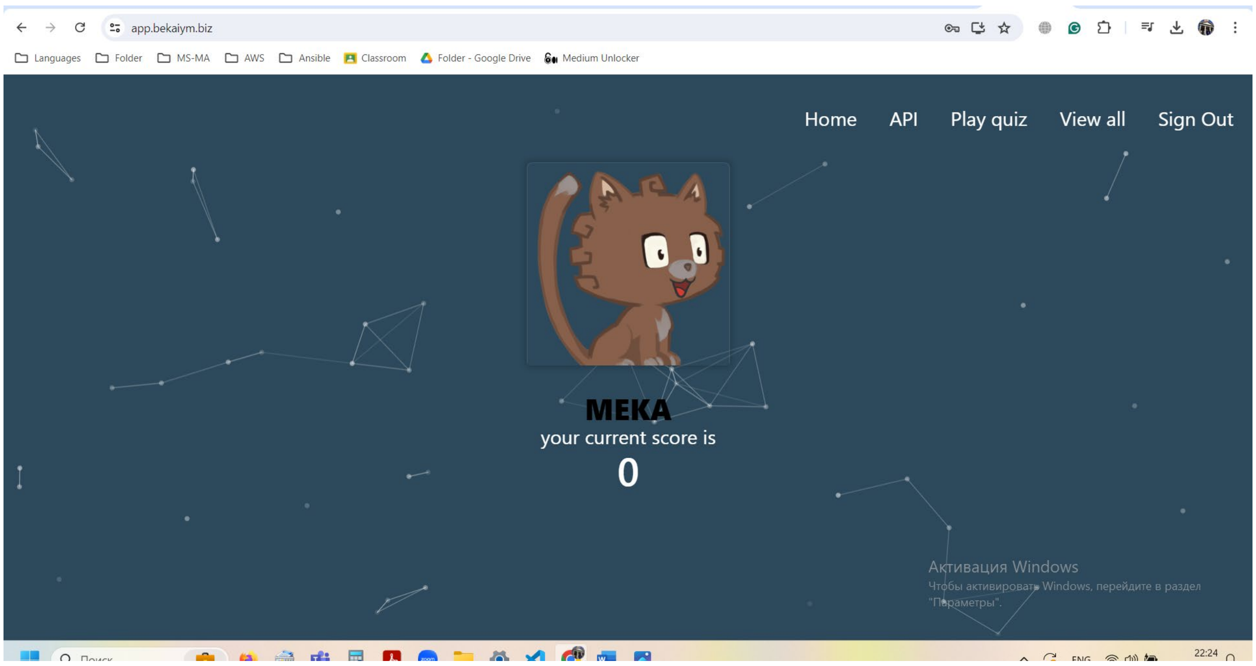
https://app.bekaiym.biz/

My website: app.bekaiym.biz



Done by Bekaiym Egemkulova



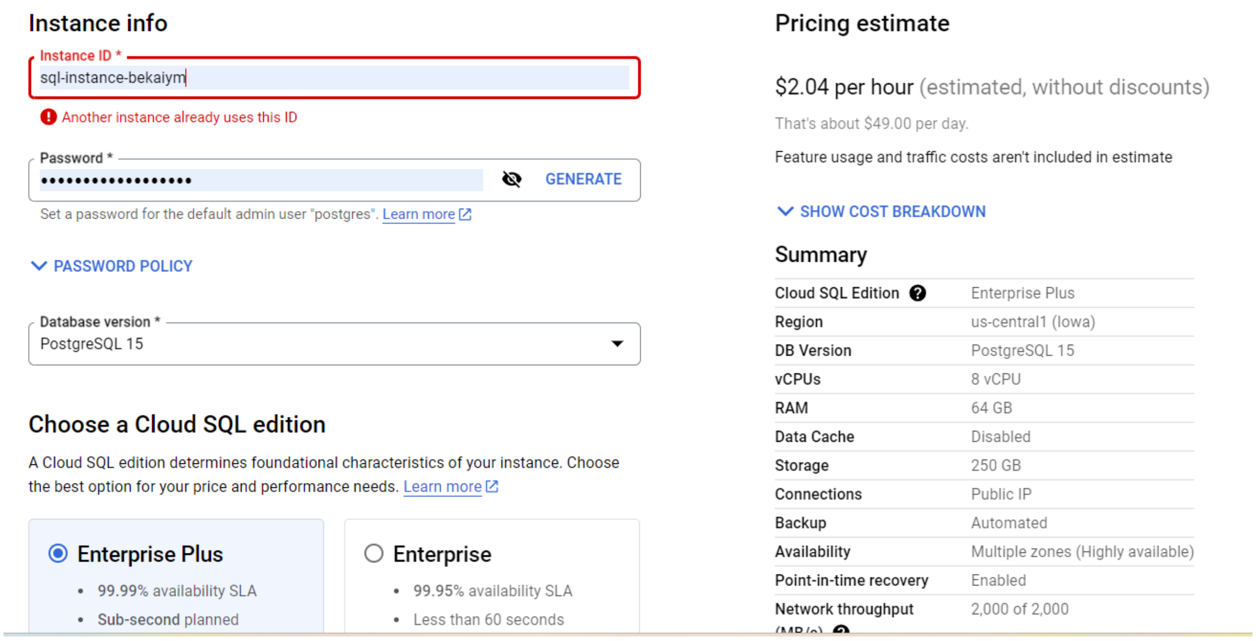


Guidance:

$ git clone https://github.com/AntTechLabs/awesome_cats_backend.git
$ git clone https://github.com/AntTechLabs/awesome_cats_frontend.git

1) First, we need manually go the Google Console.
Search for Cloud SQL (I chose the PostgreSQL) and create an Instance.
We need to save the details (Instance ID, Password) we insert during the creation process.

Done by Bekaiym Egemkulova



Instance info

Instance ID *: sql-instance-bekaiym

Another instance already uses this ID

Password *: ••••••••••••••••• GENERATE

Set a password for the default admin user "postgres". Learn more

PASSWORD POLICY

Database version *: PostgreSQL 15

Choose a Cloud SQL edition

A Cloud SQL edition determines foundational characteristics of your instance. Choose the best option for your price and performance needs. Learn more

[✓ Enterprise Plus]
- 99.99% availability SLA
- Sub-second planned

Enterprise
- 99.95% availability SLA
- Less than 60 seconds

Pricing estimate

$2.04 per hour (estimated, without discounts)

That's about $49.00 per day.

Feature usage and traffic costs aren't included in estimate

SHOW COST BREAKDOWN

Summary

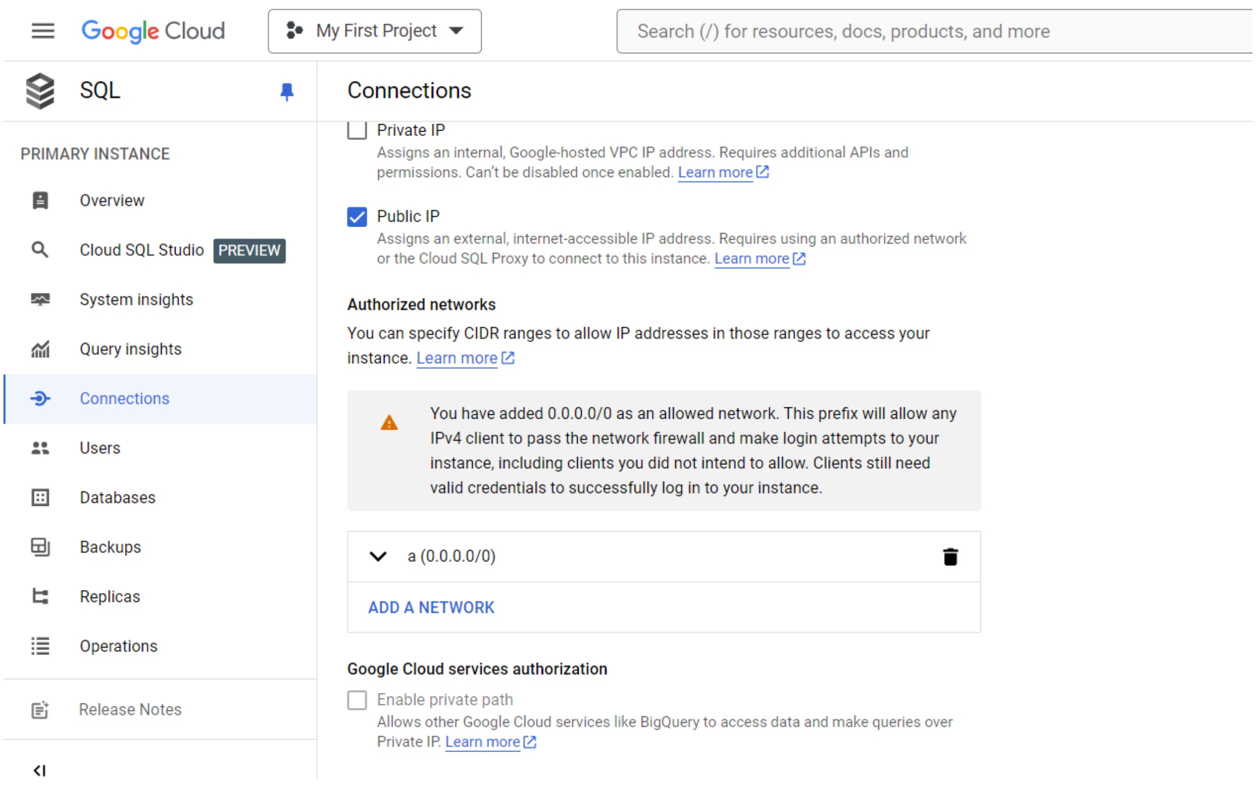
Cloud SQL Edition	Enterprise Plus
Region	us-central1 (Iowa)
DB Version	PostgreSQL 15
vCPUs	8 vCPU
RAM	64 GB
Data Cache	Disabled
Storage	250 GB
Connections	Public IP
Backup	Automated
Availability	Multiple zones (Highly available)
Point-in-time recovery	Enabled
Network throughput (MB/s)	2,000 of 2,000

Choosing different features for the PostgreSQL can decrease/increase the Pricing Estimate.

After this, we create it and wait till it is created. We remember the generated IP address of the instance. Be sure to create it in the same project and cluster where your whole app will be.

We go inside of it and create a Database/ OR choose the default database called postgres. We need also to remember the name of the database. I named it “awesome_cats_db”.
Next, we go to the Users section on the Left side of the Console.
We see our who is the User and remember it.
Next, we go to the Connections section and Choose Networking. We need to scroll down and press “Add Network”. You can name it as you want and we can set it to 0.0.0.0/0. (However, it is not a best practice).

Done by Bekaiym Egemkulova



☐ Private IP
Assigns an internal, Google-hosted VPC IP address. Requires additional APIs and permissions. Can't be disabled once enabled. Learn more

☑ Public IP
Assigns an external, internet-accessible IP address. Requires using an authorized network or the Cloud SQL Proxy to connect to this instance. Learn more

Authorized networks

You can specify CIDR ranges to allow IP addresses in those ranges to access your instance. Learn more

You have added 0.0.0.0/0 as an allowed network. This prefix will allow any IPv4 client to pass the network firewall and make login attempts to your instance, including clients you did not intend to allow. Clients still need valid credentials to successfully log in to your instance.

a (0.0.0.0/0)

ADD A NETWORK

Google Cloud services authorization

☐ Enable private path
Allows other Google Cloud services like BigQuery to access data and make queries over Private IP. Learn more

Next, we need to install Postgres through the Official Website (https://www.postgresql.org/download/). Please be careful and choose the version of the Postgres that you choose in GCP CloudSQL. I chose Postgres 15 in GCP, and I downloaded Postgres exe file of 15th version. To check, if it is installed successfully, please run:

```
$ psql --version
psql (PostgreSQL) 15.7
```

Next, we need to connect to via CLI. These are my codes for my project:

```
$ gcloud sql connect sql-instance-bekaiym --user=postgres
postgres=> \c awesome_cats_db;
awesome_cats_db=> \dt;
```

Then, we need to create a table inside of our Database:

```
awesome_cats_db=> CREATE TABLE login (
awesome_cats_db(>   id serial PRIMARY KEY,
awesome_cats_db(>   email text UNIQUE NOT NULL,
awesome_cats_db(>   hash VARCHAR(100) NOT NULL
awesome_cats_db(> );

awesome_cats_db=> CREATE TABLE users (
awesome_cats_db(>   id serial PRIMARY KEY,
awesome_cats_db(>   name  VARCHAR(100),
awesome_cats_db(>   email text UNIQUE NOT NULL,
awesome_cats_db(>   score BIGINT DEFAULT 0,
awesome_cats_db(>   joined TIMESTAMP NOT NULL
awesome_cats_db(>   id serial PRIMARY KEY,
awesome_cats_db(>   name  VARCHAR(100),
awesome_cats_db(>   email text UNIQUE NOT NULL,
awesome_cats_db(>  score BIGINT DEFAULT 0,
awesome_cats_db(>  joined TIMESTAMP NOT NULL
awesome_cats_db(>  name  VARCHAR(100),
awesome_cats_db(>  email text UNIQUE NOT NULL,
awesome_cats_db(>  score BIGINT DEFAULT 0,
awesome_cats_db(>  joined TIMESTAMP NOT NULL
awesome_cats_db(> );
```

Then, you can check if it exists and quit.

2) Next, we need to create a secret for the postgres. These are my codes:

```
kubectl create secret generic db-secret \
--from-literal=PGHOST='34.23.212.10' \
--from-literal=PGUSER='postgres' \
--from-literal=PGDATABASE='awesome_cats_db' \
--from-literal=PGPASSWORD='**********'
```

The env names of these credentials can be found in awesome_cats_backend folder, .env.demo file.

```
$ kubectl get secrets
NAME                    TYPE          DATA   AGE
postgres-secret              Opaque          8     23h
```

In the folder, awesome_cats_backend, we need to create a Dockerfile:

```
FROM node:14
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["node", "server.js"]
```

We run the commands:

```
$ docker build -t gcr.io/intrepid-nova-426815-g6/awesome-cats-backend:v2 .
$ docker push gcr.io/intrepid-nova-426815-g6/awesome-cats-backend:v2
```

Comments:
- we use the official Node.js version 14 image
- we need to copy package*.json files from the backend folder in our computer to the directory in the container.
- npm install ->installs and reads the package.json and package-lock.json files and installs into the container.
-listens to port 3000 of our backend because it is node app.

Next, we create two files:
backend-deployment.yaml

```
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: awesome-cats-backend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: awesome-cats-backend
  template:
    metadata:
      labels:
        app: awesome-cats-backend
    spec:
      containers:
        - name: backend
          image: gcr.io/intrepid-nova-426815-g6/awesome-cats-backend:v2
          ports:
            - containerPort: 3000
          env:
            - name: PGHOST
              valueFrom:
                secretKeyRef:
                  name: postgres-secret
                  key: PGHOST
            - name: PGUSER
              valueFrom:
                secretKeyRef:
                  name: postgres-secret
                  key: PGUSER
            - name: PGPASSWORD
              valueFrom:
                secretKeyRef:
                  name: postgres-secret
                  key: PGPASSWORD
            - name: PGDATABASE
              valueFrom:
                secretKeyRef:
                  name: postgres-secret
                  key: PGDATABASE
```

Here, we used our ENV from postgres-secret and docker image for backend. Port 3000 because it is the Node app.

backend-service.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  selector:
    app: awesome-cats-backend
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
  type: ClusterIP
```

Run:
$ kubectl apply -f backend-deployment.yaml
$ kubectl apply -f backend-service.yaml:

Please, note that our selector (`awesome-cats-backend`) matches the labels of the Deployment pods. ClusterIP is used to make sure that it is accessible only within cluster. To check if the backend works properly, you can set it to LoadBalancer, (run kubectl get svc backend-service), it shows Loadbalancer's ExternalIP and check it from Browser. If it says "it is working", then everything is good and you can set it back to ClusterIP.

NOW, go to the awesome_cats_frontend folder and and create the following files:
Dockerfile:

```
# Build
FROM node:14 AS builder
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build


# Prod
FROM nginx:alpine
COPY --from=builder /app/build /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

We run the commands:
$ docker build -t gcr.io/intrepid-nova-426815-g6/awesome-cats-frontend:v2 .
$ docker push gcr.io/intrepid-nova-426815-g6/awesome-cats-frontend:v2

Here, we do this:
Build stage: we use NODE to install dependencies and build the project as before.
Production stage: we use NGINX. It is an example of the multi-stage build in Dockerfile (--from=builder). We make the final image lighter and more secure by separating these stages. It listens to port 80.

frontend-deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: awesome-cats-frontend
```

```
spec:
  replicas: 2
  selector:
    matchLabels:
      app: awesome-cats-frontend
  template:
    metadata:
      labels:
        app: awesome-cats-frontend
    spec:
      containers:
        - name: frontend
          image: gcr.io/intrepid-nova-426815-g6/awesome-cats-frontend:v2
          ports:
            - containerPort: 80
```

Here, we use Dockerfile's image for frontend. We use port 80. We set Labels "awesome_cats_frontend".

frontend-service.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  selector:
    app: awesome-cats-frontend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: ClusterIP
```

Run:
$ kubectl apply -f frontend-deployment.yaml
$ kubectl apply -f frontend-service.yaml:

Please, note that our selector (`awesome-cats-frontend`) matches the labels of the Deployment pods. ClusterIP is used to make sure that it is accessible only within cluster. To check if the frontend works properly, you can set it to LoadBalancer, (run kubectl get svc fronted-service), it shows Loadbalancer's ExternalIP and check it from Browser. If it shows the app graphically, then everything is good, and you can set it back to ClusterIP.

Now, we get out from these folders and do the following in another folder:
NOTE: I downloaded the from the NAMECHEAP website my domain's name keys:

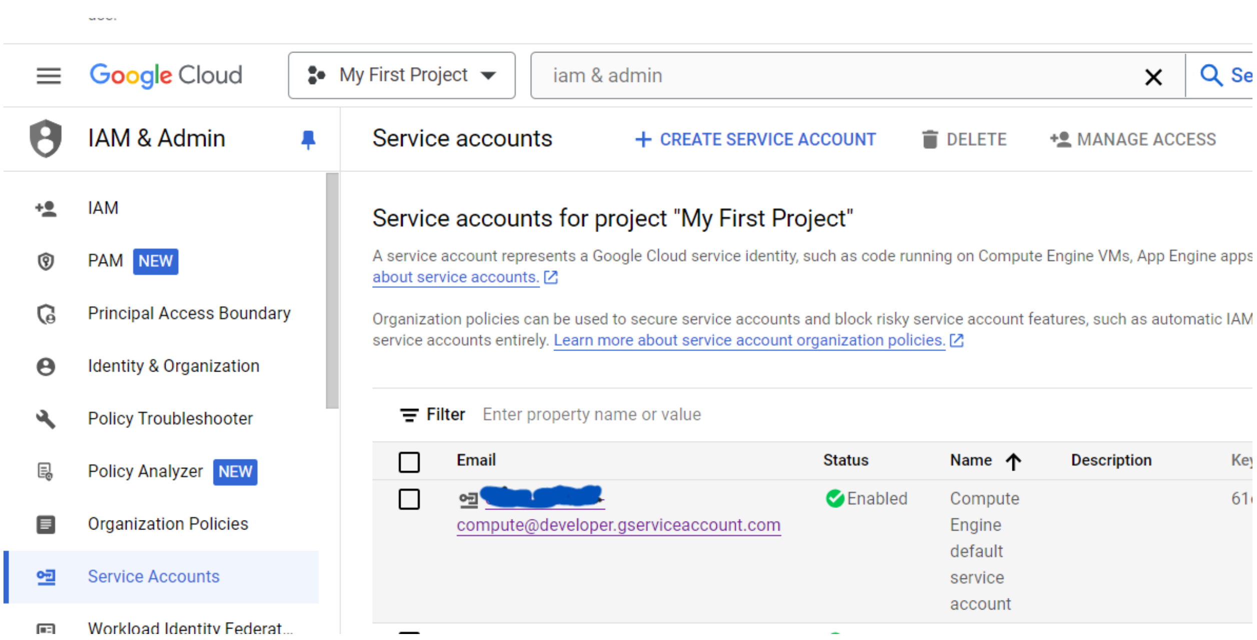
Done by Bekaiym Egemkulova

bekaiym.biz.csr bekaiym.biz.key bekaiym_biz.ca-bundle bekaiym_biz.crt bekaiym_biz.p7b
From them, I created certificates and secrets in the folder certificates.

secret.yaml:

```
apiVersion: v1
data:
  tls.crt: *** -> base64 encoded
  tls.key: *** -> base64 encoded
kind: Secret
metadata:
  creationTimestamp: null
  name: bekaiym-biz-secret
  namespace: default
type: kubernetes.io/tls
```

You can find json file needed for the Service role for your Cert-manager, if you do it manually via Kubernetes yaml manifests and not via HELM charts. But if you do via HELM charts, it handles everything.



Service accounts for project "My First Project"

A service account represents a Google Cloud service identity, such as code running on Compute Engine VMs, App Engine apps
about service accounts.

Organization policies can be used to secure service accounts and block risky service account features, such as automatic IAM
service accounts entirely. Learn more about service account organization policies.

Email	Status	Name ↑	Description	Key
[redacted] compute@developer.gserviceaccount.com	Enabled	Compute Engine default service account		61

certificate.yaml:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: bekaiym-biz-cert
spec:
  secretName: bekaiym-biz-secret #
  dnsNames:
    - app.bekaiym.biz    #
  issuerRef:
    name: letsencrypt-prod
    kind: ClusterIssuer
```

Done by Bekaiym Egemkulova

cluster-issuer.yaml:

```yaml
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    email: beccaagem@gmail.com  #
    server: https://acme-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
    - http01:
        ingress:
          class: nginx
```

```
$ kubectl apply -f certificates/
```

```
$ kubectl get certificates
NAME              READY  SECRET             AGE
bekaiym-biz-cert    True   bekaiym-biz-secret  7h48m
bekaiym-biz-secret  True   bekaiym-biz-secret  7h46m
```

```
$ kubectl get orders
NAME                            STATE  AGE
bekaiym-biz-cert-n8bpd-792741381  valid  7h48m
```

```
$ kubectl get clusterissuer
NAME             READY  AGE
letsencrypt-prod  True   7h51m
```

Ingress.yaml:

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
  annotations:
    cert-manager.io/issuer: letsencrypt-prod
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  tls:
  - hosts:
    - app.bekaiym.biz
    secretName: bekaiym-biz-secret
  rules:
  - host: app.bekaiym.biz
    http:
      paths:
      - path: /?(.*)
        pathType: Prefix
        backend:
          service:
            name: frontend-service
            port:
              number: 80
      - path: /api/?(.*)
        pathType: Prefix
        backend:
          service:
            name: backend-service
            port:
              number: 3000
```

Ingress is essential to manage external access to services within a Kubernetes cluster!!!
It is very *moody* resource, so please be careful with it.
-annotations: it manages the Ingress-nginx controller and SSL by cert-manager.
-TLS part: it makes the use of HTTPS with a certificate in the secret bekaiym-biz-secret for my domain app.bekaiym.biz.
-routing rules: paths (/?(.*)) (any paths) are routed to frontend-service on port 80. Paths starting with /api -> (/api/?(.*)) are routed to backend-service on port 3000.

It is better to use HELM charts, here's we gonna deploy both ExternalDNS and the Ingress-NGINX controller through CLI:

```
$ helm repo add bitnami https://charts.bitnami.com/bitnami
$ helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
$ helm repo update
```

```
$ kubectl create namespace ingress-nginx
$ helm install ingress-nginx ingress-nginx/ingress-nginx --namespace ingress-nginx
$ helm install my-release oci://registry-1.docker.io/bitnamicharts/external-dns
```

HELM CHARTS are cool guys!!!
Helm charts include predefined configurations for service accounts, roles, and role bindings needed for ExternalDNS to operate within your Kubernetes cluster. It also manages credentials needed and stores it properly.

```
$ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
awesome-cats-backend-579787cd7f-6s7hh     1/1     Running   0        3h54m
awesome-cats-backend-579787cd7f-lwl8g     1/1     Running   0        3h54m
awesome-cats-frontend-86984cc789-5tr5g    1/1     Running   0        7h30m
awesome-cats-frontend-86984cc789-g9pc2    1/1     Running   0        7h30m
my-release-external-dns-6bbff8ffc4-s9swz  1/1     Running   0        7h40m
```

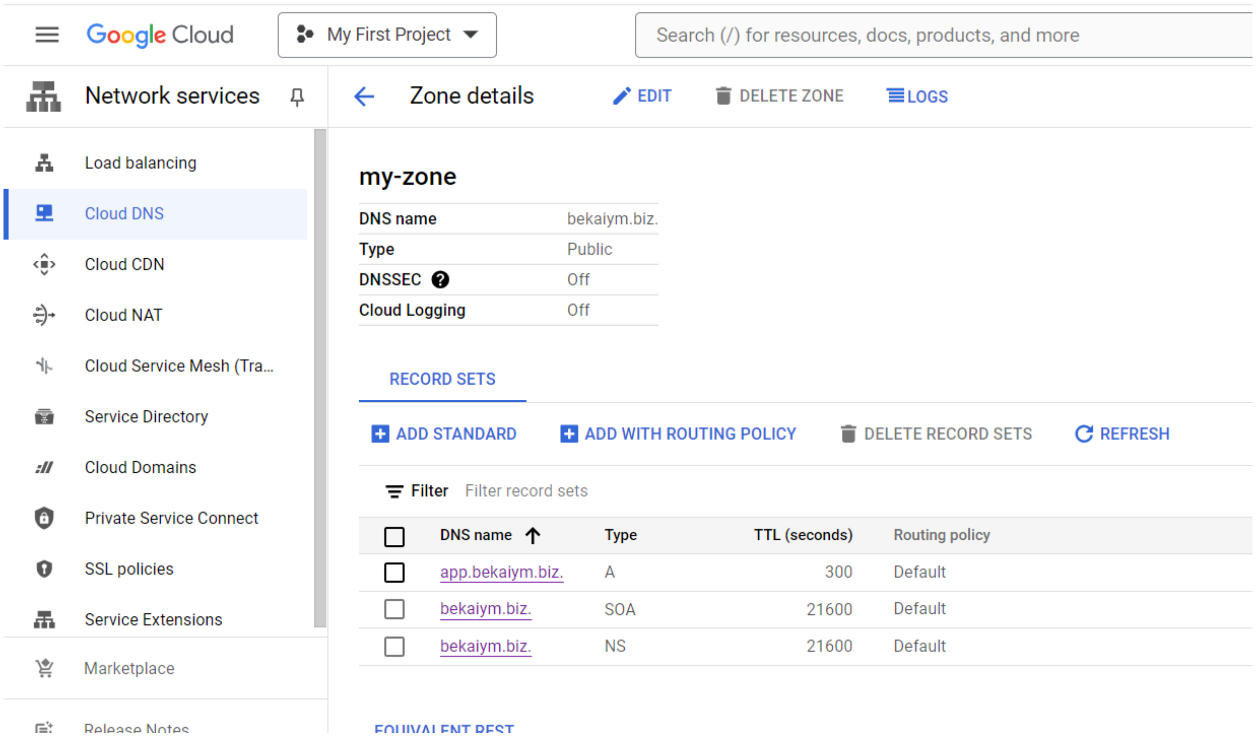
Done by Bekaiym Egemkulova

```
$ kubectl get svc
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP    PORT(S)        AGE
backend-service         ClusterIP    34.118.234.136  <none>      3000/TCP          4h
frontend-service        ClusterIP    34.118.237.69   <none>       80/TCP            7h37m
kubernetes             ClusterIP    34.118.224.1     <none>      443/TCP            2d
my-release-external-dns  ClusterIP    34.118.230.125  <none>       7979/TCP           7h47m
nginx-ingress-controller  LoadBalancer  34.118.238.234  34.138.253.222  80:32736/TCP,443:30633/TCP  13h
```

```
$ kubectl get ing
NAME          CLASS    HOSTS           ADDRESS         PORTS    AGE
nginx-ingress   <none>   app.bekaiym.biz   34.148.105.28   80, 443   7h38m
```

== --- ==

Some setup, In Network Services in GCP: we add a new zone under our domain name, in my case it is bekaiym.biz that I bought from the Namecheap website. We add the A record and point it to the nginx-ingress Port as shown below (in my case **34.148.105.28**):

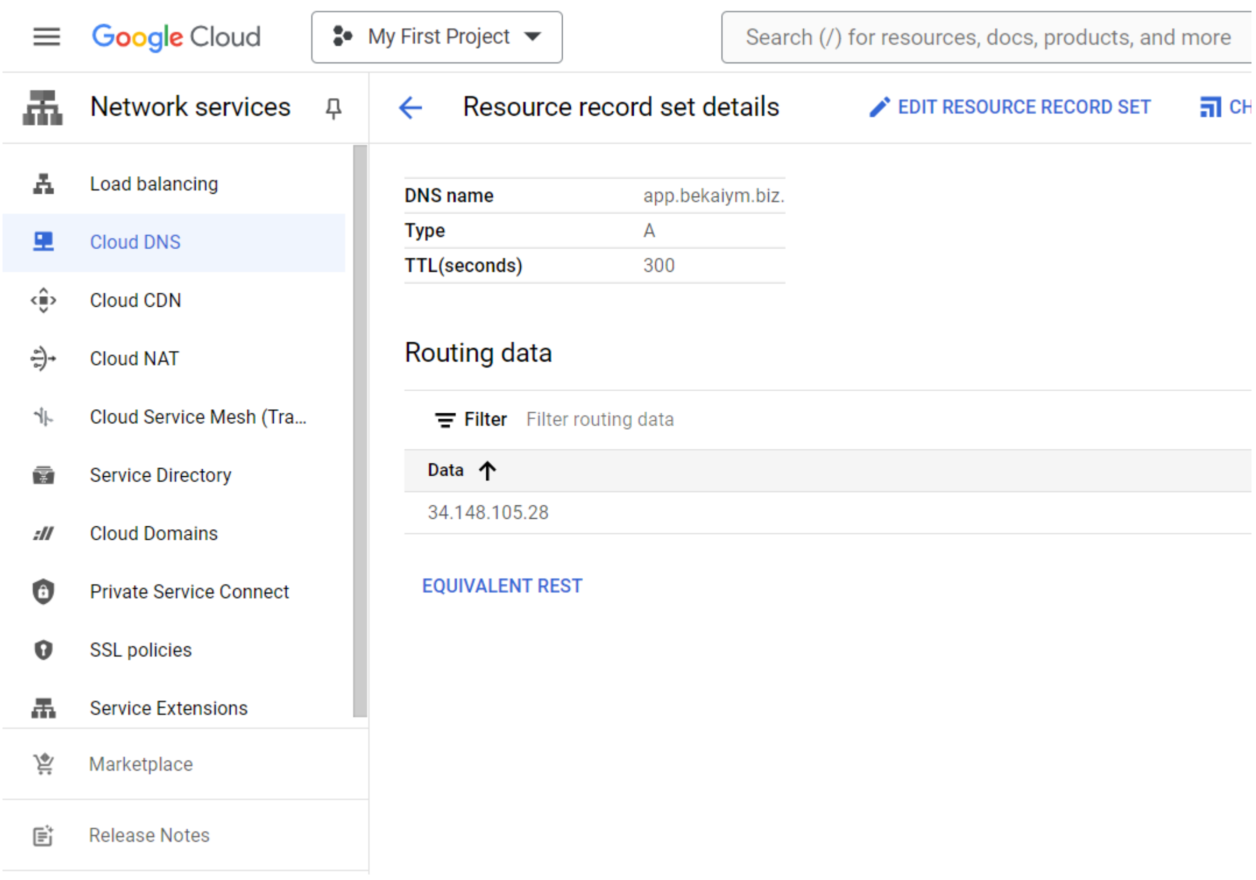


my-zone

DNS name	bekaiym.biz.
Type	Public
DNSSEC	Off
Cloud Logging	Off

DNS name	Type	TTL (seconds)	Routing policy
app.bekaiym.biz.	A	300	Default
bekaiym.biz.	SOA	21600	Default
bekaiym.biz.	NS	21600	Default

Done by Bekaiym Egemkulova



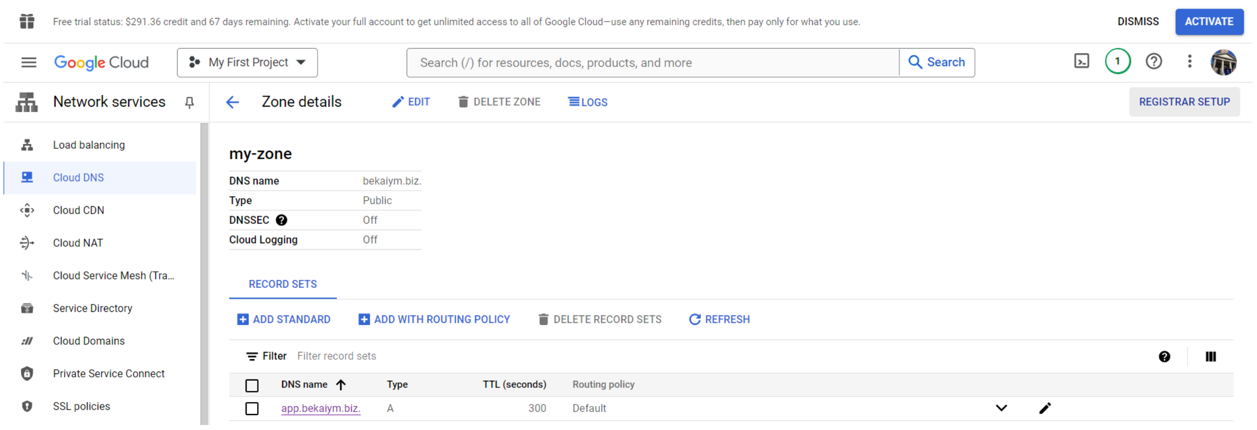
Resource record set details

DNS name	app.bekaiym.biz.
Type	A
TTL(seconds)	300

Routing data

Data
34.148.105.28

Also, in the NameCheap dashboard, we need to add Nameservers from the GCP:
Go to the Registrar Setup in CloudDNS-Zones.


Free trial status: $291.36 credit and 67 days remaining. Activate your full account to get unlimited access to all of Google Cloud—use any remaining credits, then pay only for what you use.

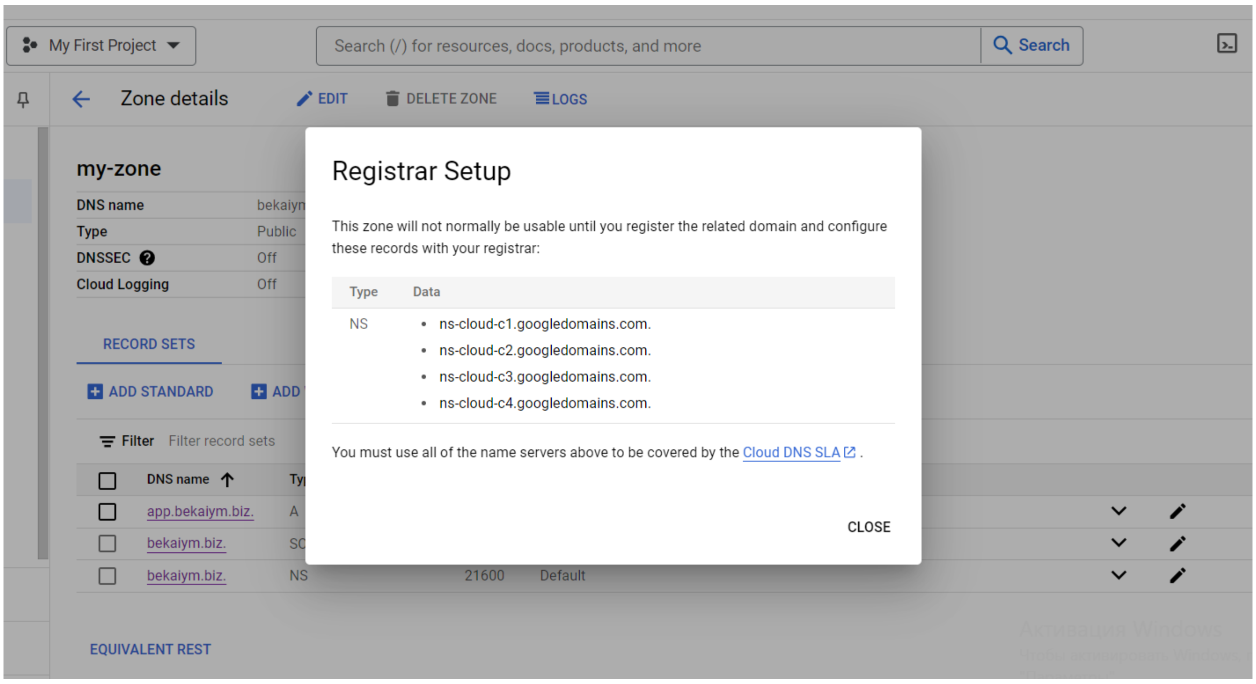
Zone details

my-zone

DNS name	bekaiym.biz.
Type	Public
DNSSEC	Off
Cloud Logging	Off

DNS name	Type	TTL (seconds)	Routing policy
app.bekaiym.biz.	A	300	Default

Done by Bekaiym Egemkulova



my-zone

DNS name	bekaiym
Type	Public
DNSSEC	Off
Cloud Logging	Off

Registrar Setup

This zone will not normally be usable until you register the related domain and configure these records with your registrar:

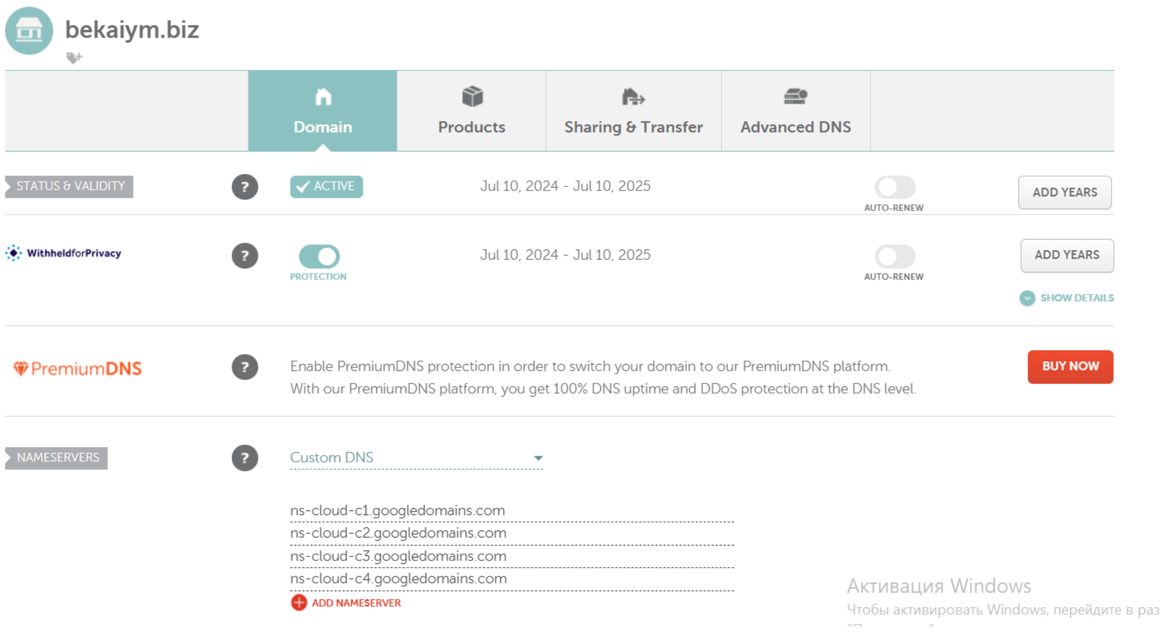
Type	Data
NS	• ns-cloud-c1.googledomains.com. • ns-cloud-c2.googledomains.com. • ns-cloud-c3.googledomains.com. • ns-cloud-c4.googledomains.com.

You must use all of the name servers above to be covered by the Cloud DNS SLA .

CLOSE

DNS name	Ty		
app.bekaiym.biz.	A		
bekaiym.biz.	SO		
bekaiym.biz.	NS	21600	Default

Add this information to the Dashboard of the Domain Provider (NameCheap in my case as the Custom DNS in the NameServers section):



STATUS & VALIDITY	ACTIVE	Jul 10, 2024 - Jul 10, 2025	AUTO-RENEW
WithheldforPrivacy	PROTECTION	Jul 10, 2024 - Jul 10, 2025	AUTO-RENEW
PremiumDNS	Enable PremiumDNS protection in order to switch your domain to our PremiumDNS platform. With our PremiumDNS platform, you get 100% DNS uptime and DDoS protection at the DNS level.		
NAMESERVERS	Custom DNS		

ns-cloud-c1.googledomains.com
ns-cloud-c2.googledomains.com
ns-cloud-c3.googledomains.com
ns-cloud-c4.googledomains.com

Note: you can delete frontend and backend deployments and services and ingress and apply them again, so everything will be set up and work properly after all these manipulations.

Check the health and status of the pods, svc, ing, deployments, etc:

```
$ kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
awesome-cats-backend-579787cd7f-6s7hh    1/1     Running   0          5h
```

```
awesome-cats-backend-579787cd7f-lwl8g    1/1    Running  0       5h
awesome-cats-frontend-86984cc789-5tr5g    1/1    Running  0       8h
awesome-cats-frontend-86984cc789-g9pc2    1/1    Running  0       8h
my-release-external-dns-6bbff8ffc4-s9swz  1/1    Running  0       8h

$ kubectl get svc
NAME               TYPE        CLUSTER-IP     EXTERNAL-IP    PORT(S)              AGE
backend-service        ClusterIP     34.118.234.136  <none>       3000/TCP             5h1m
frontend-service       ClusterIP     34.118.237.69   <none>       80/TCP               8h
kubernetes           ClusterIP     34.118.224.1    <none>       443/TCP              2d1h
my-release-external-dns   ClusterIP     34.118.230.125  <none>       7979/TCP             8h
nginx-ingress-controller  LoadBalancer  34.118.238.234  34.138.253.222
80:32736/TCP,443:30633/TCP   14h


$ kubectl get ing
NAME         CLASS    HOSTS          ADDRESS       PORTS    AGE
nginx-ingress   <none>   app.bekaiym.biz   34.148.105.28   80, 443   8h

$ kubectl get deployment
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
awesome-cats-backend     2/2    2          2        5h2m
awesome-cats-frontend    2/2    2          2        8h
my-release-external-dns   1/1    1          1        8h
```