Gitlab in Kubernetes - Norman Shipman

Setting up Redis

I started off with creating the Redis deployment yams which will handle the pod creation and replicaset.

```
---
```

~~~~
kind: Service
apiVersion: v1
metadata:
  labels:
    app: redis
  name: redis
  namespace: norman-test
spec:
  ports:
    - port: 6379
      targetPort: 6379
  selector:
    app: redis
```
~~~~

Once the deployment has been created, I created the redis service which will handle the routing from our gitlab application to the redis pod.

~~~~

---
kind: Service
apiVersion: v1
metadata:
  labels:
    app: redis
  name: redis
  namespace: norman-test
spec:
  ports:
    - port: 6379
      targetPort: 6379
  selector:
    app: redis
```
~~~~

## Provision Infrastructure

The following infrastructure are required for Gitlab to run:

1. Postgresql Database which handles gitlab references and relational data.

- [ ] Deployed the database using terraform.
- [ ] Create a new database user called `gitlab` with default permissions.
- [ ] Using the root user, created the extension pg_trgm to avoid permission issues during the pod build. `create EXTENSION pg_trgm;`

2. Elastic Block Storage volume for the gitlab data such as repositories and container registries.

- [ ] There are multiple ways of approaching this, but for this setup I opted for dynamic provisioning using storage classes. (Shout out to Luis Viant for working with this approach and making it work.)

- [ ] To provision the EBS volumes dynamically we will begin by creating a storage class in Kubernetes.  We are taking this approach so Kubernetes will spin up a new encrypted EBS volume. This is important because if you are going through the trouble of standing up a private gitlab server,  you likely want to add an additional layer of protection to your data.

  We will then create the Persistent Volume Claim (PVC) will ensure storage devices will not be removed from the cluster.

  gitlab-storageclass.yml

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gitlab-sc
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  encrypted: "true"
  kmsKeyId: arn:aws:kms:us-east-1:492865049799:alias/aws/ebs
reclaimPolicy: Retain
volumeBindingMode: Immediate
```

  gitlab-pvc.yml

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gitlab-volume
  annotations:
     volume.beta.kubernetes.io/storage-class: "gitlab-sc"
  namespace: norman-test
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```
```

3. (Optional) S3 Bucket for container registries.

- [ ] Deployed the bucket via terraform

## Create Gitlab.rb and Gitlab Runner Secrets

```
kubectl create secret generic -n norman-test gitlab-rb --from-file=./gitlab.rb
```

```
kubectl create secret generic -n norman-test config-toml --from-file=./config.toml
```

gitlab.rb

```
gitlab_rails['db_username'] = 'gitlab'
gitlab_rails['db_password'] = 'REDACTED'
postgresql['enable'] = true
gitlab_rails['db_host'] = "gitlabdb.dev.ue1.quovo.com"
gitlab_rails['db_port'] = "5432"
gitlab_rails['db_database'] = "gitlabdb"
gitlab_rails['db_adapter'] = 'postgresql'
gitlab_rails['db_encoding'] = 'utf8'
redis['enable'] = false
gitlab_rails['redis_host'] = 'redis'
gitlab_rails['redis_port'] = '6379'
```
```

~~~
gitlab_rails['gitlab_shell_ssh_port'] = '22'
external_url 'https://norman-gitlab.dev.ue1.quovo.com'
registry_external_url 'https://registry-norman-gitlab.dev.ue1.quovo.com/'
gitlab_rails['registry_port'] = "4567"
registry_nginx['ssl_certificate'] = "/etc/gitlab/ssl/norman-
gitlab.dev.ue1.quovo.com.crt"
registry_nginx['ssl_certificate_key'] = "/etc/gitlab/ssl/norman-
gitlab.dev.ue1.quovo.com.key"
letsencrypt['enable'] = false
nginx['redirect_http_to_https'] = true
prometheus_monitoring['enable'] = false
registry['token_realm'] = "https://norman-gitlab.dev.ue1.quovo.com"
registry['storagedriver_health_enabled'] = false
git_data_dirs({
  "default" => {
    "path" => "/data/gitlab"
   }
})
registry['storage'] = {
  's3' => {
    'accesskey' => 'REDACTED',
    'secretkey' => 'REDACTED',
    'bucket' => 'quovo-norman-gitlab-registry-bucket',
    'region' => 'us-east-1'
  }
}
```
~~~

config.toml

```
concurrent = 4
[[runners]]
  name = "Gitlab Runner"
  url = "https://norman-gitlab.dev.ue1.quovo.com/"
  token = "REDACTED"
  executor = "kubernetes"
  [runners.kubernetes]
    privileged = true
    namespace = "norman-test"
```

```
```

Create Gitlab

Now we will create the Gitlab stack. We will treat the gitlab-runner as it's own stack.

SSL Enabled

We want to upload our ssl certificate for an added layer of protection so we can place our already generated ssl certificate and key file to a Kubernetes secret.

```
kubectl create secret -n norman-test norman-gitlab-certificate --from-file=./
norman-gitlab.crt --from-file=./norman-gitlab.key
```

We need to be mindful of the following components when creating the gitlab pod.

1. The image being used. I Strongly suggest using a predefined stable version of gitlab.
2. The ports needed for gitlab:
   1. HTTP (80)
   2. HTTPS (443)
   3. SSH (22)
   4. Docker Registry (4567)
3. Volumes:
   1. The gitlab volume referred to as `gitlab-volume` will hold the repository data for our gitlab server. You can adjust the location of the gitlab repositories in the `gitlab.rb`. Be sure to change the location otherwise, you will lose all your repository data upon redeploying the pod; which will lead to 404 errors since the reference still exists in the database.
   2. The gitlab-certificate volume is created from our ssl secret and will be attached to the instance.
4. Volume Mount points:
   1. `gitlab-volume` was mountedted to an arbitrary spot.
   2. `gitlab-certificate` was mounted to `/etc/gitlab/ssl` since gitlab will look for certificates in that directory.

```

---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: gitlab
  namespace: norman-test
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: gitlab
    spec:
      containers:
      - image: gitlab/gitlab-ce:11.3.6-ce.0
        name: gitlab
        securityContext:
          privileged: true
        ports:
        - containerPort: 80
        - containerPort: 443
        - containerPort: 22
        - containerPort: 4567
        env:
        - name: GITLAB_OMNIBUS_CONFIG
          valueFrom:
            secretKeyRef:
              name: gitlab-rb
              key: gitlab.rb
        volumeMounts:
        - mountPath: /data/gitlab
          name: gitlab-volume
          readOnly: false
        - mountPath: /etc/gitlab/ssl
          name: norman-gitlab-certificate
          readOnly: false
      volumes:
      - name: gitlab-volume
        persistentVolumeClaim:
          claimName: gitlab-volume
      - name: norman-gitlab-certificate
        secret:
          defaultMode: 0755
          secretName: norman-gitlab-certificate
```

Gitlab Service

The same with our redis pod, we will need to create a service object for gitlab to route traffic. By using an AWS internal load balancer we can avoid exposing our gitlab pod to the internet. The `selector` field will be used to route the traffic to gitlab, so be sure you define the same app name in the `gitlab-deployment.yml`

```
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: gitlab
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-internal: 0.0.0.0/0
  name: gitlab-service
  namespace: norman-test
spec:
  type: LoadBalancer
  ports:
  - port: 80
    name: http
    protocol: TCP
    targetPort: 80
  - port: 443
    name: https
    protocol: TCP
    targetPort: 443
  - port: 22
    name: ssh
    protocol: TCP
    targetPort: 22
  - port: 4567
    name: registry
    targetPort: 4567
  selector:
    app: gitlab
```

DNS Propagation

With our current setup, I utilized Terraform to create the DNS name and attach it to the load balancer created in the `gitlab-svc.yml` The load balancer should be stable, but if the entire cluster goes down, or your service is redeployed, we will need to update the entry in terraform.

Gitlab Runner

The gitlab runner is mostly straight forward. We will need to create a Service account to allow the gitlab runner to make pods and run our build jobs.

We will also need to go into the pod and start register a new runner to our gitlab server (there are multiple ways of doing this.)

1. Exec into the pod:
    1. `kubectl exec -it gitlab-runner-REDACTED /bin/bash -n norman-test`
    2. `gitlab-runner register` and input the requested information.
    3. Head into your gitlab web interface to collect the runner token and enter the token in the config.toml. After updating delete and create the secret again
    4. Delete the pod or deployment to get the updated secrets.
2. From your local machine or a linux node.
    1. Run `gitlab-runner register` command and input requested information.
    2. Head into your gitlab web interface to collect the runner token and enter the token in the config.toml.
    3. Create the secret.
    4. Deploy the gitlab-runner pod.

gitlab-runner-deployment.yml

```

---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: gitlab-runner
```

~~~
  namespace: norman-test
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: gitlab-runner
    spec:
      serviceAccountName: gitlab-runner
      containers:
      - image: gitlab/gitlab-runner:ubuntu-v11.3.1
        name: gitlab-runner
        volumeMounts:
        - mountPath: /etc/gitlab-runner
          name: config
      volumes:
      - name: config
        secret:
          secretName: config-toml
```
~~~

gitlab-runner-rbac.yml

```

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: gitlab-runner
  namespace: norman-test
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: norman-test
  name: gitlab-runner
rules:
  - apiGroups: [""]
    resources: ["pods", "pods/exec", "pods/log", "secrets"]
```

~~~yaml
    verbs: ["*"]
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: RoleBinding
metadata:
  name: gitlab-runner
  namespace: norman-test
subjects:
- kind: ServiceAccount
  name: gitlab-runner
  namespace: norman-test
roleRef:
  kind: Role
  name: gitlab-runner
  apiGroup: ""
```
~~~

Backing up Gitlab

Gitlab comes with a script which handles the application and database backup. In order to run the backup, we will want to create a Kubernetes cron job.

ubuntu-kubectl docker image

```dockerfile

FROM ubuntu:18.04

ENV DEBIAN_FRONTEND=noninteractive

RUN apt-get update && apt-get install -y \
        apt-transport-https \
        curl \
        gnupg2

RUN curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add
-

RUN echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | tee -a /etc/
apt/sources.list.d/kubernetes.list
```

~~~
RUN apt-get update && apt-get install -y \
        kubectl



RUN apt-get purge -y \
        apt-transport-https \
        curl \
        gnupg2 \
    && rm -rf /var/lib/apt/lists/*

CMD ["tail -f /dev/null"]
```
~~~

gitlab-backup-cron.yml

~~~

---
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: gitlab-backup-job
  namespace: norman-test
spec:
  schedule: "26/14 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          serviceAccountName: gitlab-backup
          containers:
          - name: gitlab-backup-job
            image: REDACTED:ubuntu-kubectl:1.0.0
            args:
            - /bin/sh
            - -c
            - kubectl exec -it $(kubectl get pod -n norman-test -l "app=gitlab"
-o jsonpath='{.items[0].metadata.name}') -n norman-test -- gitlab-rake
gitlab:backup:create SKIP=registry
          restartPolicy: Never
```
~~~

Followed by a service account to access pods in the cluster.

gitlab-cron-rbac.yml

~~~yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: gitlab-backup
  namespace: norman-test
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: norman-test
  name: gitlab-backup
rules:
  - apiGroups: [""]
    resources: ["pods", "pods/exec"]
    verbs: ["*"]
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: RoleBinding
metadata:
  name: gitlab-backup
  namespace: norman-test
subjects:
- kind: ServiceAccount
  name: gitlab-backup
  namespace: norman-test
roleRef:
  kind: Role
  name: gitlab-backup
  apiGroup: ""
```
~~~

Additional Notes

At this point you should have a running mostly functional gitlab container. I will need to add additional configurations to get mailing working.

I will also need to add a mechanism that allows us to handle backing up the gitlab secrets file.