Gitlab官方提供了 Helm 的方式在 Kubernetes 集群中来快速安装，但是在使用的过程中发现 Helm 提供的 Chart 包中有很多其他额外的配置，所以我们这里使用自定义的方式来安装，也就是自己来定义一些资源清单文件。

Gitlab主要涉及到3个应用：Redis、Postgresql、Gitlab 核心程序，实际上我们只要将这3个应用分别启动起来，然后加上对应的配置就可以很方便的安装 Gitlab 了，我们这里选择使用的镜像不是官方的，而是 Gitlab 容器化中使用非常多的一个第三方镜像：sameersbn/gitlab，基本上和官方保持同步更新，地址：[http://www.damagehead.com/docker-gitlab/](http://www.damagehead.com/docker-gitlab/" \t "_blank)

如果我们已经有可使用的 Redis 或 Postgresql 服务的话，那么直接配置在 Gitlab 环境变量中即可，如果没有的话就单独部署。

创建一个用于存储密码的secret文件:

创建username和password文件:

$ echo -n "admin" > ./username

$ echo -n "1f2d1e2e67df" > ./password

用kubectl生成secret对象:

$ kubectl create secret generic db-user-pass --from-file=./username --from-file=./password

secret "db-user-pass" created

创建PVC和storageclass做持久化:

[复制代码](javascript:void(0);)

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: gitlab-redis-pvc

namespace: kube-ops

annotations:

volume.beta.kubernetes.io/storage-class: "gitlab-storageclass"

spec:

accessModes:

- ReadWriteMany

resources:

requests:

storage: 1Gi

[复制代码](javascript:void(0);)

部署需要的 Redis 服务，对应的资源清单文件如下：(gitlab-redis.yaml)

[复制代码](javascript:void(0);)

apiVersion: apps/v1beta1

kind: Deployment

metadata:

name: redis

namespace: kube-ops

labels:

name: redis

spec:

template:

metadata:

name: redis

labels:

name: redis

spec:

containers:

- name: redis

image: sameersbn/redis

imagePullPolicy: IfNotPresent

ports:

- name: redis

containerPort: 6379

volumeMounts:

- mountPath: /var/lib/redis

name: data

livenessProbe:

exec:

command:

- redis-cli

- ping

initialDelaySeconds: 30

timeoutSeconds: 5

readinessProbe:

exec:

command:

- redis-cli

- ping

initialDelaySeconds: 5

timeoutSeconds: 1

volumes:

- name: data

persistentVolumeClaim:

claimName: gitlab-redis-pvc

---

apiVersion: v1

kind: Service

metadata:

name: redis

namespace: kube-ops

labels:

name: redis

spec:

ports:

- name: redis

port: 6379

targetPort: redis

selector:

name: redis

[复制代码](javascript:void(0);)

vim gitlab-postgresql-pvc.yaml:

[复制代码](javascript:void(0);)

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: gitlab-postgresql-pvc

namespace: kube-ops

annotations:

volume.beta.kubernetes.io/storage-class: "gitlab-storageclass"

spec:

accessModes:

- ReadWriteMany

resources:

requests:

storage: 1Gi

[复制代码](javascript:void(0);)

然后是数据库 Postgresql，对应的资源清单文件如下：(gitlab-postgresql.yaml)

[复制代码](javascript:void(0);)

apiVersion: apps/v1beta1

kind: Deployment

metadata:

name: postgresql

namespace: kube-ops

labels:

name: postgresql

spec:

template:

metadata:

name: postgresql

labels:

name: postgresql

spec:

containers:

- name: postgresql

image: sameersbn/postgresql

imagePullPolicy: IfNotPresent

env:

- name: DB\_USER

value: gitlab

- name: DB\_PASS

value: passw0rd

- name: DB\_NAME

value: gitlab\_production

- name: DB\_EXTENSION

value: pg\_trgm

ports:

- name: postgres

containerPort: 5432

volumeMounts:

- mountPath: /var/lib/postgresql

name: data

livenessProbe:

exec:

command:

- pg\_isready

- -h

- localhost

- -U

- postgres

initialDelaySeconds: 30

timeoutSeconds: 5

readinessProbe:

exec:

command:

- pg\_isready

- -h

- localhost

- -U

- postgres

initialDelaySeconds: 5

timeoutSeconds: 1

volumes:

- name: data

persistentVolumeClaim:

claimName: gitlab-postgresql-pvc

---

apiVersion: v1

kind: Service

metadata:

name: postgresql

namespace: kube-ops

labels:

name: postgresql

spec:

ports:

- name: postgres

port: 5432

targetPort: postgres

selector:

name: postgresql

[复制代码](javascript:void(0);)

vim gitlab-pvc.yaml

[复制代码](javascript:void(0);)

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: gitlab-pvc

namespace: kube-ops

annotations:

volume.beta.kubernetes.io/storage-class: "gitlab-storageclass"

spec:

accessModes:

- ReadWriteMany

resources:

requests:

storage: 1Gi

[复制代码](javascript:void(0);)

然后就是我们最核心的 Gitlab 的应用，对应的资源清单文件如下：(gitlab.yaml)

[复制代码](javascript:void(0);)

apiVersion: apps/v1beta1

kind: Deployment

metadata:

name: gitlab

namespace: kube-ops

labels:

name: gitlab

spec:

template:

metadata:

name: gitlab

labels:

name: gitlab

spec:

containers:

- name: gitlab

image: sameersbn/gitlab:12.1.6

imagePullPolicy: IfNotPresent

env:

- name: TZ

value: Asia/Shanghai

- name: GITLAB\_TIMEZONE

value: Beijing

- name: GITLAB\_SECRETS\_DB\_KEY\_BASE

value: long-and-random-alpha-numeric-string

- name: GITLAB\_SECRETS\_SECRET\_KEY\_BASE

value: long-and-random-alpha-numeric-string

- name: GITLAB\_SECRETS\_OTP\_KEY\_BASE

value: long-and-random-alpha-numeric-string

- name: GITLAB\_ROOT\_PASSWORD

valueFrom:

secretKeyRef:

name: git-user-pass

key: password

# value: admin321

- name: GITLAB\_ROOT\_EMAIL

value: 1369472116@qq.com

- name: GITLAB\_HOST

value: gitlab.fuyuteng.com

- name: GITLAB\_PORT

value: "80"

- name: GITLAB\_SSH\_PORT

value: "30022"

- name: GITLAB\_NOTIFY\_ON\_BROKEN\_BUILDS

value: "true"

- name: GITLAB\_NOTIFY\_PUSHER

value: "false"

- name: GITLAB\_BACKUP\_SCHEDULE

value: daily

- name: GITLAB\_BACKUP\_TIME

value: 01:00

- name: DB\_TYPE

value: postgres

- name: DB\_HOST

value: postgresql

- name: DB\_PORT

value: "5432"

- name: DB\_USER

value: gitlab

- name: DB\_PASS

value: passw0rd

- name: DB\_NAME

value: gitlab\_production

- name: REDIS\_HOST

value: redis

- name: REDIS\_PORT

value: "6379"

ports:

- name: http

containerPort: 80

- name: ssh

containerPort: 22

volumeMounts:

- mountPath: /home/git/data

name: data

livenessProbe:

httpGet:

path: /

port: 80

initialDelaySeconds: 180

timeoutSeconds: 5

readinessProbe:

httpGet:

path: /

port: 80

initialDelaySeconds: 5

timeoutSeconds: 1

volumes:

- name: data

persistentVolumeClaim:

claimName: gitlab-pvc

---

apiVersion: v1

kind: Service

metadata:

name: gitlab

namespace: kube-ops

labels:

name: gitlab

spec:

ports:

- name: http

port: 80

targetPort: http

- name: ssh

port: 22

targetPort: ssh

nodePort: 30022

type: NodePort

selector:

name: gitlab

---

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: gitlab

namespace: kube-ops

annotations:

kubernetes.io/ingress.class: traefik

spec:

rules:

- host: gitlab.fuyuteng.com

http:

paths:

- backend:

serviceName: gitlab

servicePort: http

[复制代码](javascript:void(0);)

我们这里应用数据都做数据持久化，还有敏感数据用了secret引入到环境变量,比如添加 PV/PVC 或者 StorageClass。

要注意的是其中 Redis 和 Postgresql 相关的环境变量配置，另外，我们这里添加了一个 Ingress 对象，来为我们的 Gitlab 配置一个域名git.qikqiak.com，这样应用部署完成后，我们就可以通过该域名来访问了，然后直接部署即可：

$ kubectl create -f gitlab-redis.yaml gitlab-postgresql.yaml gitlab.yaml

创建完成后，查看 Pod 的部署状态：

$ kubectl get pods -n kube-ops

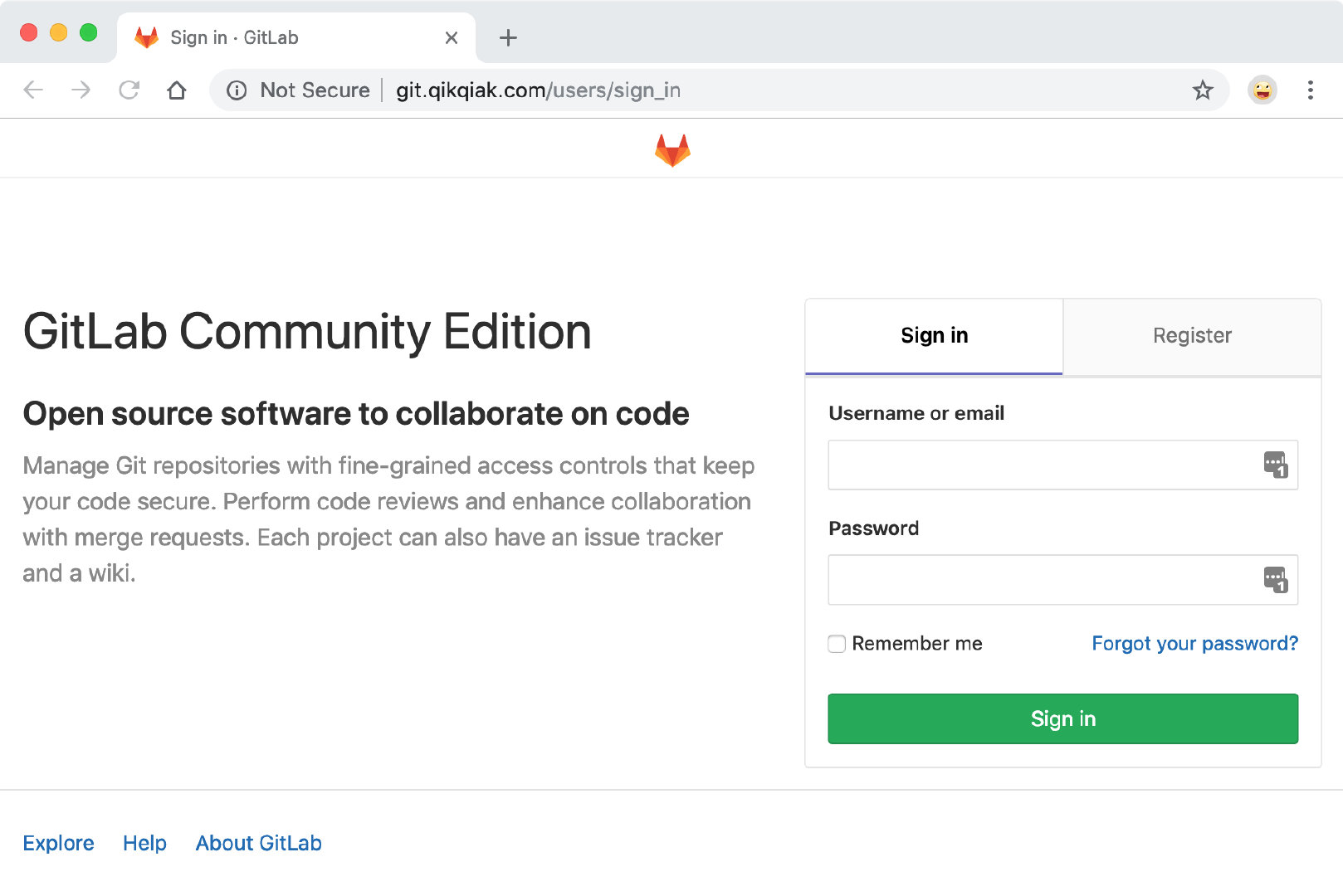
NAME READY STATUS RESTARTS AGE

gitlab-7d855554cb-twh7c 1/1 Running 0 10m

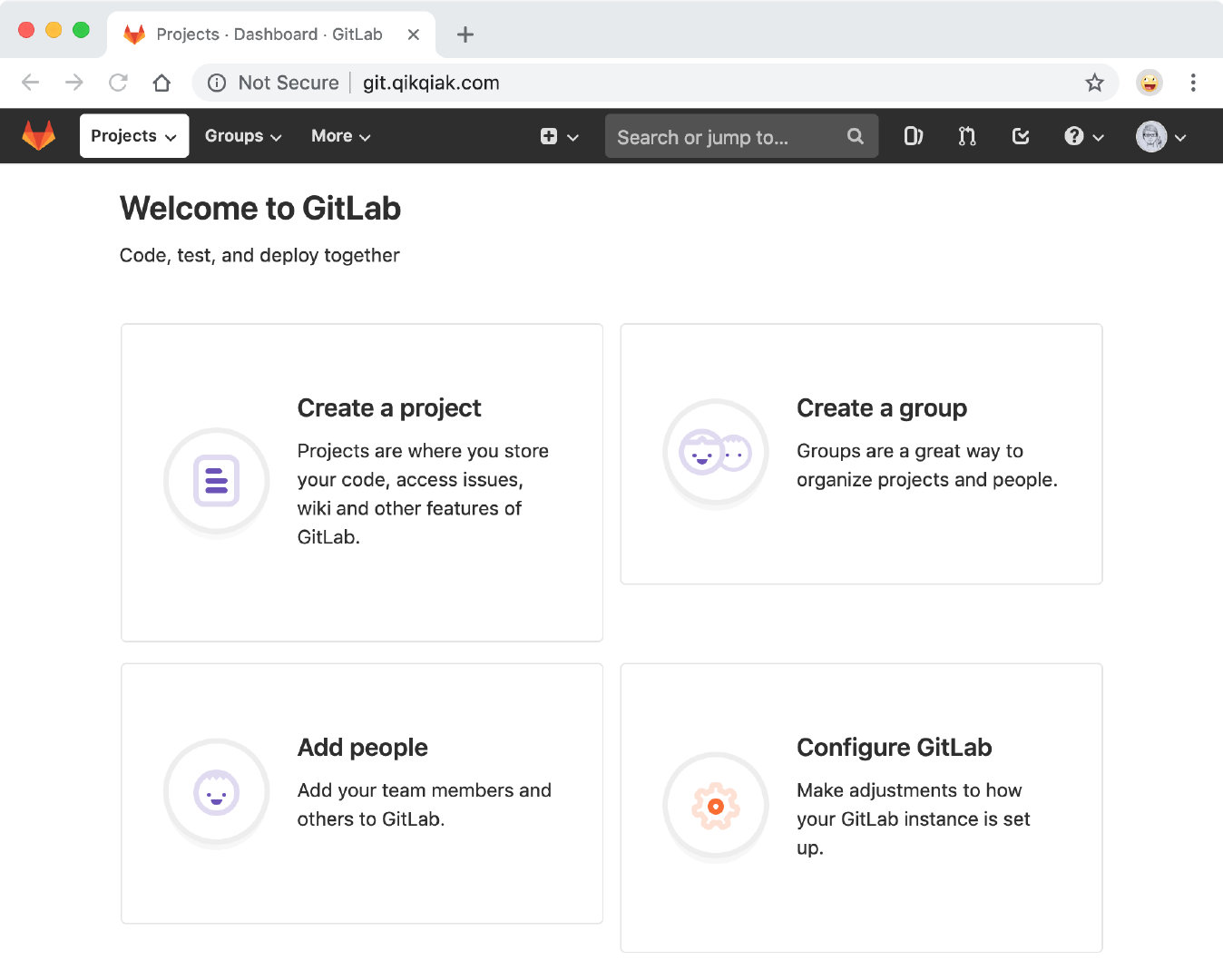
postgresql-8566bb959c-2tnvr 1/1 Running 0 17h

redis-8446f57bdf-4v62p 1/1 Running 0 17h

可以看到都已经部署成功了，然后我们可以通过 Ingress 中定义的域名git.qikqiak.com(需要做 DNS 解析或者在本地 /etc/hosts 中添加映射)来访问 Portal：

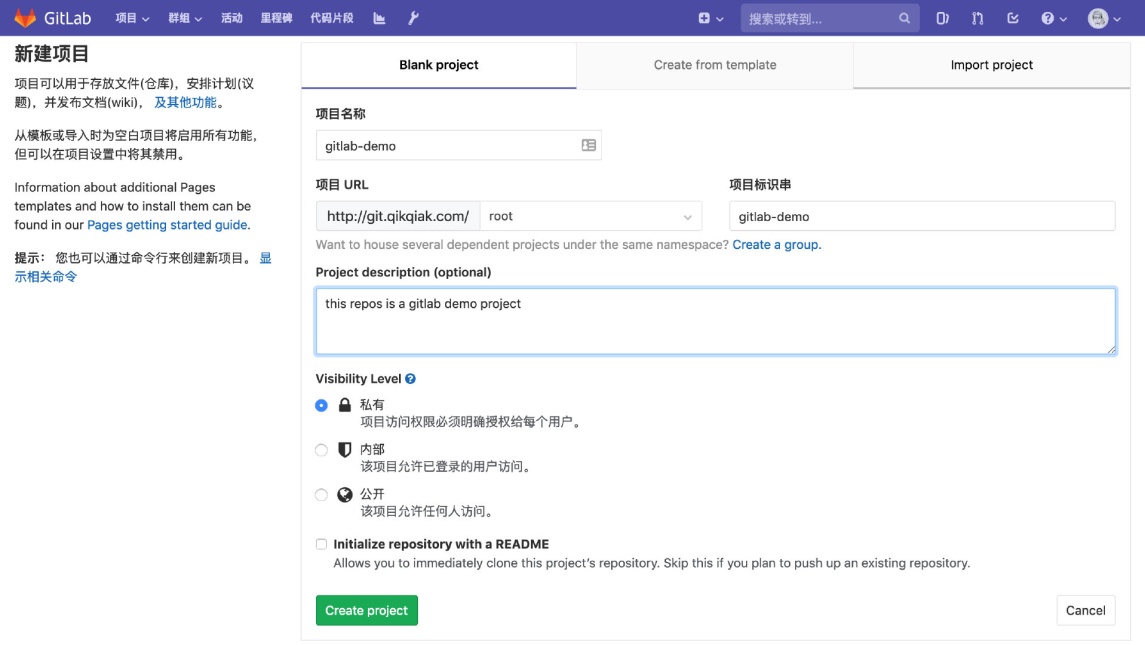
gitlab portal

使用用户名 root，和部署的时候指定的超级用户密码GITLAB\_ROOT\_PASSWORD=admin321即可登录进入到首页：

gitlab homepage

Gitlab 运行后，我们可以注册为新用户并创建一个项目，还可以做很多的其他系统设置，比如设置语言、设置应用风格样式等等。

点击Create a project创建一个新的项目，和之前 Github 使用上没有多大的差别：

create gitlab project

创建完成后，我们可以添加本地用户的一个SSH-KEY，这样我们就可以通过 SSH 来拉取或者推送代码了。SSH 公钥通常包含在~/.ssh/id\_rsa.pub 文件中，并以ssh-rsa开头。如果没有的话可以使用ssh-keygen命令来生成，id\_rsa.pub里面的内容就是我们需要的 SSH 公钥，然后添加到 Gitlab 中。

由于平时使用的 ssh 默认是 22 端口，现在如果用默认的 22 端口去连接，是没办法和 Gitlab 容器中的 22 端口进行映射的，因为我们只是通过 Service 的 22 端口进行了映射，要想通过节点去进行 ssh 链接就需要在节点上一个端口和容器内部的22端口进行绑定，所以这里我们可以通过 NodePort 去映射 Gitlab 容器内部的22端口，比如我们将环境变量设置为GITLAB\_SSH\_PORT=30022，将 Gitlab 的 Service 也设置为 NodePort 类型：

apiVersion: v1

kind: Service

metadata:

name: gitlab

namespace: kube-ops

labels:

name: gitlab

spec:

ports:

- name: http

port: 80

targetPort: http

- name: ssh

port: 22

targetPort: ssh

nodePort: 30022

type: NodePort

selector:

name: gitlab

注意上面 ssh 对应的 nodePort 端口设置为 30022，这样就不会随机生成了，重新更新下 Deployment 和 Service，更新完成后，现在我们在项目上面 Clone 的时候使用 ssh 就会带上端口号了：

gitlab ssh

现在就可以使用Clone with SSH的地址了，由于上面我们配置了 SSH 公钥，所以就可以直接访问上面的仓库了：

$ git clone ssh://git@git.qikqiak.com:30022/root/gitlab-demo.git

Cloning into 'gitlab-demo'...

warning: You appear to have cloned an empty repository.

Checking connectivity... done.

然后随便在该项目下面添加一些资源：

$ echo "# hello world" > README.md

$ git add .

$ git commit -m 'hello world'

[master (root-commit) 63de7cb] hello world

1 file changed, 1 insertion(+)

create mode 100644 README.md

$ git push origin master

Counting objects: 3, done.

Writing objects: 100% (3/3), 224 bytes | 0 bytes/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To ssh://git@git.qikqiak.com:30022/root/gitlab-demo.git

\* [new branch] master -> master

然后刷新浏览器，就可以看到刚刚创建的 Git 仓库中多了一个 README.md 的文件：

