Guide to run repo & quickies

2022年2月24日 18:41

Guide to run repo

1.
prerequisites
	Prepare AWS account, key, secret creation, service role
	Complete AWS CLI configuration
	github account, PAT
	git local config configuration
	EC2 key
	Install eksctl, istioctl, k9s, jq, helm
	
	$ git clone https://github.com/scp756-221/term-project-sfu_data_best_data.git
	$ make -f k8s.mak dynamodb-clean
	
	(Optional) tools/shell.sh (Still need tools container because of gatling)

2 account preparation	$ mv cluster/tpl-vars-blank.txt cluster/tpl-vars.txt configure tpl-var.txt configure cluster/ghcr.io-token.txt
3 instantiates all makefiles/scripts	$ make -f k8s-tpl.mak templates
4 ghcr login	$ make -f k8s.mak registry-login
5 Startup eks cluster	$ make -f eks.mak start

6. Change ns	$ kubectl config use-context aws756
	$ kubectl create ns project-services
	$ kubectl config set-context aws756 --namespace=project-services
	$ kubectl config get-contexts

6. Install istio, Prometheus, create and set namespace, enable istio injected, deploy gw,s1,s2, playlist, db, monitoring, Kiali svc	$ make -f k8s.mak provision
7. Run loader to push data into db	$ make -f k8s.mak loader

8.
Run your thing
	Run client:
	$ cd mcli
	$ make build-mcli
	$ make PORT=80 SERVER=… SERVICE=user/music/playlist run-mcli
	
	Do load test:
	$ tools/shell.sh
	/home/k8s# tools/gatling <user amount> ReadUSerSim/ReadMusicSim

9. Stop everything	make -f eks.mak stop
10. Delete dynamodb table	make -f k8s.mak dynamodb-clean
11. Delete logs	rm logs/*.log

Some quickies

kubectl config get-contexts

kubectl config current-context

kubectl get namespace

kubectl -n istio-system get service istio-ingressgateway | cut -c -140

make -f k8s.mak grafana-url
User: admin
Password: prom-operator

make -f k8s.mak prometheus-url
flask_http_request_total{service="cmpt756s2"}

make -f k8s.mak kiali-url

Wanna re-push image, need delete logs/*.repo.log

Digging

2022年2月20日 22:41

```
CREG=ghcr.io
DK=docker
REGID=qiushuo222
LOG_DIR=logs
KC=kubectl
APP_NS=c756ns
S2_VER=v1
```

Login github

```
registry-login:
	@/bin/sh -c 'cat cluster/${CREG}-token.txt | $(DK) login $(CREG) -u $(REGID) --password-stdin'
```

Launch s1 service:

```
s1: $(LOG_DIR)/s1.repo.log cluster/s1.yaml cluster/s1-sm.yaml cluster/s1-vs.yaml
	$(KC) -n $(APP_NS) apply -f cluster/s1.yaml | tee $(LOG_DIR)/s1.log
	$(KC) -n $(APP_NS) apply -f cluster/s1-sm.yaml | tee -a $(LOG_DIR)/s1.log
	$(KC) -n $(APP_NS) apply -f cluster/s1-vs.yaml | tee -a $(LOG_DIR)/s1.log

$(LOG_DIR)/s1.repo.log: s1/Dockerfile s1/app.py s1/requirements.txt
	make -f k8s.mak --no-print-directory registry-login
	$(DK) build $(ARCH) -t $(CREG)/$(REGID)/cmpt756s1:$(APP_VER_TAG) s1 | tee $(LOG_DIR)/s1.img.log
	$(DK) push $(CREG)/$(REGID)/cmpt756s1:$(APP_VER_TAG) | tee $(LOG_DIR)/s1.repo.log
```

Launch s2 service:

```
s2: rollout-s2 cluster/s2-svc.yaml cluster/s2-sm.yaml cluster/s2-vs.yaml
	$(KC) -n $(APP_NS) apply -f cluster/s2-svc.yaml | tee $(LOG_DIR)/s2.log
	$(KC) -n $(APP_NS) apply -f cluster/s2-sm.yaml | tee -a $(LOG_DIR)/s2.log
	$(KC) -n $(APP_NS) apply -f cluster/s2-vs.yaml | tee -a $(LOG_DIR)/s2.log

rollout-s2: $(LOG_DIR)/s2-$(S2_VER).repo.log cluster/s2-dpl-$(S2_VER).yaml
	$(KC) -n $(APP_NS) apply -f cluster/s2-dpl-$(S2_VER).yaml | tee $(LOG_DIR)/rollout-s2.log
	$(KC) rollout -n $(APP_NS) restart deployment/cmpt756s2-$(S2_VER) | tee -a $(LOG_DIR)/rollout-s2.log

$(LOG_DIR)/s2-$(S2_VER).repo.log: s2/$(S2_VER)/Dockerfile s2/$(S2_VER)/app.py s2/$(S2_VER)/requirements.txt
	make -f k8s.mak --no-print-directory registry-login
	$(DK) build $(ARCH) -t $(CREG)/$(REGID)/cmpt756s2:$(S2_VER) s2/$(S2_VER) | tee $(LOG_DIR)/s2-$(S2
_VER).img.log
	$(DK) push $(CREG)/$(REGID)/cmpt756s2:$(S2_VER) | tee $(LOG_DIR)/s2-$(S2_VER).repo.log
```

Launch db service:

```
db: $(LOG_DIR)/db.repo.log cluster/awscred.yaml cluster/dynamodb-service-entry.yaml cluster/db.yaml cluster/db-
sm.yaml cluster/db-vs.yaml
	$(KC) -n $(APP_NS) apply -f cluster/awscred.yaml | tee $(LOG_DIR)/db.log
	$(KC) -n $(APP_NS) apply -f cluster/dynamodb-service-entry.yaml | tee -a $(LOG_DIR)/db.log
	$(KC) -n $(APP_NS) apply -f cluster/db.yaml | tee -a $(LOG_DIR)/db.log
	$(KC) -n $(APP_NS) apply -f cluster/db-sm.yaml | tee -a $(LOG_DIR)/db.log
	$(KC) -n $(APP_NS) apply -f cluster/db-vs.yaml | tee -a $(LOG_DIR)/db.log

$(LOG_DIR)/db.repo.log: db/Dockerfile db/app.py db/requirements.txt
	make -f k8s.mak --no-print-directory registry-login
	$(DK) build $(ARCH) -t $(CREG)/$(REGID)/cmpt756db:$(APP_VER_TAG) db | tee $(LOG_DIR)/db.img.log
	$(DK) push $(CREG)/$(REGID)/cmpt756db:$(APP_VER_TAG) | tee $(LOG_DIR)/db.repo.log
```

Launch Dynamodb service

```
loader: dynamodb-init $(LOG_DIR)/loader.repo.log cluster/loader.yaml
	$(KC) -n $(APP_NS) delete --ignore-not-found=true jobs/cmpt756loader
	tools/build-configmap.sh gatling/resources/users.csv cluster/users-header.yaml | kubectl -n $(APP_NS) apply -f -
	tools/build-configmap.sh gatling/resources/music.csv cluster/music-header.yaml | kubectl -n $(APP_NS) apply -f -
	$(KC) -n $(APP_NS) apply -f cluster/loader.yaml | tee $(LOG_DIR)/loader.log

dynamodb-init: $(LOG_DIR)/dynamodb-init.log

$(LOG_DIR)/dynamodb-init.log: cluster/cloudformationdynamodb.json
	@# "|| true" suffix because command fails when stack already exists
	@# (even with --on-failure DO_NOTHING, a nonzero error code is returned)
	$(AWS) cloudformation create-stack --stack-name db-qiushuo222 --template-body file://$< || true | tee
$(LOG_DIR)/dynamodb-init.log
	# Must give DynamoDB time to create the tables before running the loader
	sleep 20

$(LOG_DIR)/loader.repo.log: loader/app.py loader/requirements.txt loader/Dockerfile registry-login
	$(DK) build $(ARCH) -t $(CREG)/$(REGID)/cmpt756loader:$(LOADER_VER) loader | tee $(LOG_DIR)/loader.img.log
	$(DK) push $(CREG)/$(REGID)/cmpt756loader:$(LOADER_VER) | tee $(LOG_DIR)/loader.repo.log
```

Launch gateway

```
gw: cluster/service-gateway.yaml
	$(KC) -n $(APP_NS) apply -f $< > $(LOG_DIR)/gw.log
```

Launch Kiali, Monitoring

```
monitoring: monvs
	$(KC) -n $(ISTIO_NS) get vs

monvs: cluster/monitoring-virtualservice.yaml
	$(KC) -n $(ISTIO_NS) apply -f $< > $(LOG_DIR)/monvs.log
```

Install Prometheus and Grafana

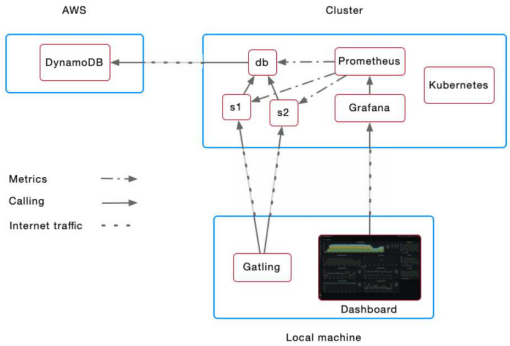
```
prom:
	make -f obs.mak init-helm --no-print-directory
	make -f obs.mak install-prom --no-print-directory

obs.mak init-helm:
	$(HELM) repo add prometheus-community https://prometheus-community.github.io/helm-charts
	$(HELM) repo update

obs.mak install-prom
	echo $(HELM) install $(RELEASE) --namespace $(ISTIO_NS) prometheus-community/kube-prometheus-stack >
$(LOG_DIR)/obs-install-prometheus.log
	$(HELM) install $(RELEASE) -f helm-kube-stack-values.yaml --namespace $(ISTIO_NS) prometheus-community/kube-
prometheus-stack | tee -a $(LOG_DIR)/obs-install-prometheus.log
	$(KC) apply -n $(ISTIO_NS) -f monitoring-lb-services.yaml | tee -a $(LOG_DIR)/obs-install-prometheus.log
	$(KC) apply -n $(ISTIO_NS) -f cluster/grafana-flask-configmap.yaml | tee -a $(LOG_DIR)/obs-install-prometheus.log
```

k8s.mak provision execution order

1. Install istio
2. Install and deploy prometheus and granfana
3. Create ns, enable ns istio injection
4. Launch gateway
5. Launch s1, s2, db
6. Launch monitoring



s1这段部署是为了写s1服务

1	build & push Dockerfile	quay.io/bitnami/python:3.8.6-prod-debian-10-r81 python app.py 30001
2	app.py	一个flask应用，部署于本地30000端口，定义http://cmpt756s1:30000/api/v1/user/health和/readiness返回200， 定义了user相关方法，在方法里向 db url发送请求，db/app.py来处理这些请求并更新Dynamodb
3	s1.yaml	定义了一个service名为cmpt756s1，开放端口30000 定义了一个Service account，叫svc-s1，label SVC_S2 定义了一个deployment，image用的ghcr.io/qiushuo222/cmpt756s1:v1，绑定service account svc-s1，健康探针api/v1/user/health和api/v1/user/readiness
4	s1-sm.yaml	定义了一个ServiceMonitor, /metrics
5	s1-vs.yaml	定义了一个virtual service，api版本networking.istio.io/v1alpha3，名字叫cmpt756s1，定义了url为cmpt756s1:30000并加了个/api/v1/user在url后缀 因为外网可见所以要用gateway字段，announce给c756-gateway

s2这段部署是为了写s2服务

1	build & push Dockerfile	quay.io/bitnami/python:3.8.6-prod-debian-10-r81 python app.py 30001
2	app.py	一个flask应用，部署于本地30001端口，定义http://cmpt756s2:30001/api/v1/music/health和/readiness返回200， 定义了music相关方法，在方法里向 db url发送请求，db/app.py来处理这些请求并更新Dynamodb
3	s2-dpl-v1.yaml	定义了一个Deployment，image用的ghcr.io/qiushuo222/cmpt756s2:v1，绑定service account svc-s2但是定义放在了s2-svc.yaml里，一个环境变量EXER: v1，健康探针/api/v1/music/health和/api/v1/music/readiness
4	rollout restart deployment/cmpt756s2-v1	cmpt756s2-v1是s2-dpl-v1.yaml里定义的deployment的名字
5	s2-svc.yaml	定义了一个Service，叫cmpt756s2 定义了一个Service account，叫svc-s2，label SVC_S2
6	s2-sm.yaml	定义了一个ServiceMonitor, /metrics
7	s2-vs.yaml	定义了一个virtual service，api版本networking.istio.io/v1alpha3，名字叫cmpt756s2，定义了url为cmpt756s2:30001并加了个/api/v1/music在url后缀, 因为外网可见所以要用gateway字段，announce给c756-gateway 定义了一个DestinationRule， # DestinationRule lists the subset destination Kubernetes Services # that will receive traffic from the Istio VirtualService. # VirtualService routes URLs to the destinations, splitting the traffic # across destinations according to weights.

DB 这一段部署是写了db的前端服务，为了处理指向db的http请求并转换为可以更新dynamodb的代码

1	Dockerfile	expose 30002端口，这样Dynamodb可以像这个端口发起请求 python app.py 30002 docker build and push to ghcr.io
2	db/app.py	一个flask应用，部署于本地30002端口，读取了aws region，aws key，SVC_LOADER_TOKEN, 这些都是db.yaml和awscred里定义的DYNAMODB_URL，并没有在任何地方有定义 其他函数都是在处理http请求并更新Dynamodb
3	awscred	secret服务，定义了aws region 和account key等
4	dynamodb-service-entry.yaml	不明白dynamodb-service-entry是干啥的，可能是dns
5	db.yaml	定义了一个service, 叫cmpt756db, 端口30002 定义了两个service accounts svc-db和svc-loader, 定义了secret svc-loader-token来绑定 svc-loader 定义了一个deployment cmpt756db，pull image ghcr.io/qiushuo222/cmpt756db:v1，引用awscred，SVC_LOADER_TOKEN, livenessProbe监听/api/v1/datastore/health和/api/v1/datastore/readiness
6	db-sm.yaml	ServiceMonitor, /metrics
7	db-vs.yaml	定义了一个virtual service, api版本networking.istio.io/v1beta1, 名字叫cmpt756db, 定义了url为cmpt756db:30002并加了个/api/v1/datastore在url后缀

Dynamodb/loader 这一段部署是为了创建dynamodb tables，并读取两个csv文件导入到dynamodb中

1	cluster/cloudformationdynamodb.json	存放了Dynamodb table的定义，aws cloudformation create-stack创建了aws table stack
2	loader app	Build container, 然后push到ghcr.io里 app.py定义了create user和create song， loader_token = os.getenv('SVC_LOADER_TOKEN'), 读取两个/data/music/music.csv和/data/users/users.csv，post读取的东西给http://cmpt756db:30002/api/v1/datastore/load Dockerfile没什么特殊的，就是运行python app.py
3	build-configmap.sh users-header.yaml music-header.yam	这两条命令是把users.csv和music.csv内容 append到configmap里，然后kubectl apply这俩config map
4	loader.yaml	定义了一个job，service account svc-loader, name: svc-loader-token, pull image ghcr.io/qiushuo222/cmpt756loader:v1, mount了两个文件夹/data/music和/data/users，然后又map了configmap

Service Gateway 部署了外网的gateway

1	cluster/service-gateway.yaml	定义了一个Gateway，使用istio的ingressgateway的80端口

Kiali, Monitoring

1	cluster/monitoring-virtualservice.yaml	定义了一个virtual service，叫monitoring，gateways c756ns/c756-gateway，端口20001，跟kiali有关
2	kubectl -n $(ISTIO_NS) get vs	don't know what is get vs

Prometheus & Grafana 部署两个监控软件

1	obs.mak init-helm	# add the latest active repo for Prometheus # Only needs to be done once for any user but is idempotent
2	obs.mak install-prom	第一步，install prometheus (附带Grafana) 第二步，似乎是建了个pod放prometheus和Grafana 第三步，monitoring-lb-services.yaml 定义了两个Service，一个叫prom-ingress，绑定9090端口，一个叫grafana-ingress，绑定3000端口 第四步，grafana-flask-configmap.yaml定义了一个Configmap，好多grafana_dashboard的参数

02/17/2022

2022年2月17日 21:05

Agenda:

- Scrum methodology for a distributed team
  - Merge a team in Coursys
  - Create github repo
  - Find a tool to build backlog and track ideas and works
  - Use github project Kanban to sync status of progress
  - Create four branches for each member
  - Meeting setup (Weekly meeting, MoM in Onenote or Google doc)
- Architecture
  - K8S cluster, AWS eks
  - Backend DB: DynamoDB
  - Gatling load test, Grafana and Prometheus monitor metrics
  - Three public micro-services (Music, User, Plus One)
  - Running markdown and test cases
- Interim Milestone
  - Report
- Report
  - How we apply scrum methodology
  - How we use Git
  - Process of developing cloud environment
  - Process of developing micro services
  - How we do load test and result
  - Exploring the failure modes of a system
  - Exploring various approaches to remediate such failures
- Video (20-30 mins) and Repo
  - A tour of the team's Github repo
  - A walk thru of a couple of issues/branches/PR to illustrate the collaboration between team members.
  - A deployment (e.g., a make -f k8s.mak provision or equivalent) of your system into an empty cluster.
  - A run of the load simulation on your system with no disturbance. As this run is long, you do not need to have it complete. About 5 minutes of run-time will suffice during which a narrator can point out any observations. If you have summary findings (e.g., Grafana summary reports) to present, collect those from a separate run.
  - A second run of the load simulation on your system during which there is some disturbance to the system: a change in load, a network fault (delays), a machine failure, introduction of a circuit breaker, etc. Include your follow-up actions to either re

Next:

- Check with George on following questions:
  - Ask if we can reuse assignment code
  - Exploring the failure modes of a system
  - Exploring various approaches to remediate such failures
  - Running markdown and test cases
- Decompose useable tasks to assignee

02/20/2022

2022年2月17日 21:27

Agenda:

- Check with George on following questions:
  - Ask if we can reuse assignment code
    Yes
  - Exploring the failure modes of a system
    Will discuss in later courses
  - Exploring various approaches to remediate such failures
    Will discuss in later courses
  - Running markdown and test cases
    No need to prepare AWS account for grading, no running markdown
  - No other project management tool, just use github issues and projects

- Decompose useable tasks to assignee
- How to Collaboration on eks cluster and dynamo db
- Plan:
  1. Lightweighted code: cluster,db,eks.mak, k8s.mak,loader, logs,mcli, profiles, s1,s2,tools

  2. Modify k8s.mak
     - Insert new service
     - Build new container
     - Modify namespace or not

  3. Code new image
  4. Code yaml script
  5. Create eks cluster
  6. Create gw, db, s1,s2, new service, loader, service mesh, (other gatling)
  7. Load Test

Next

Digging into existed service, learn how to write yaml and build up new app

02/24/2022

2022年2月24日 19:10

Agenda:

1. s1 s2 db loader services clear out

Next:

1. Dig into monitoring, gw, Service Mesh, Grafana, Prometheus, Gatling
2. Modify ns name
3. clear redundant files
4. test
5. Git push, write down guide how to run repo
6. Write new service
7. Question to George: s2 has two versions, can we counted s2-v2 as new service?
8. modify tools/shell.sh, copy /profiles
9. Kiali??

02/27/2022

2022年2月27日 16:21

Agenda:

1. Dig into monitoring, gw, Service Mesh, Grafana, Prometheus, Gatling
2. Modify ns name
3. clear redundant files
4. test
5. Git push, write down guide how to run repo
6. modify tools/shell.sh, copy /profiles

Next:

1. Write new service (including db, s3, client, container file, k8s yamls)
2. Fix mcli client(read all, support all microservices)
3. Kiali?? commented out from k8s.mak provision, # Nov 2021: Kiali is causing problems so do not deploy, test not working
4. Gatling running problem java.lang.ClassNotFoundException: proj756.ReadMusicSim

```
#!/usr/bin/env bash
docker container run --rm \
  -v ${PWD}/gatling/results:/opt/gatling/results \
  -v ${PWD}/gatling:/opt/gatling/user-files \
  -v ${PWD}/gatling/target:/opt/gatling/target \
  -e CLUSTER_IP=`tools/getip.sh kubectl istio-system svc/istio-ingressgateway` \
  -e USERS=1 \
  -e SIM_NAME=ReadMusicSim \
  --label gatling \
  ghcr.io/scp-2021-jan-cmpt-756/gatling:3.4.2 \
  -s proj756.ReadMusicSim
~
~
```

5. Gating test

03/03/2022

2022年3月3日 20:01

Agenda:

1. Write new service (including db, s3, client, container file, k8s yamls)
   Playlist app done, yaml done
2. Fix mcli client(read all, support all microservices)
   Support user and music, playlist tbd
3. Kiali?? commented out from k8s.mak provision, # Nov 2021: Kiali is causing problems so do not deploy, test not working
   Still need Kiali
4. Gatling running problem java.lang.ClassNotFoundException: proj756.ReadMusicSim
   Gatling can run now

5. Keys and token leakage

6. Gating test

Next step:

1. Get rid of tools container
2. Fix logic problem of playlist
3. Extend mcli functionality
4. Load test guide follow-up
5. Report

03/06/2022

2022年3月5日 12:11

Agenda:

1. Get rid of tools container
Chaucer Done, only gatling left
2. Complete playlist dev
Done
3. Extend mcli functionality
Done
4. Load test guide follow-up
Not done

5. Interim Report
Not done

Next step:

1. Follow up on all open issues in git repo
2. Do we need gatling for playlist? Scala script?
3. Load test guide follow-up
4. Interim Report