Author: Kevin Meyer

Date: Saturday, 2020 12 Dec

# Kevin Meyer's Rockset take-home writeup

# The actual problem:

This exercise is intended to gauge your ability to learn how to use a new tool based on documentation and produce a working prototype. It should take around 2 hours, but you should not spend more than 4 hours on it. There is no time limit, but we do ask that you submit your answer within a week of receiving this assignment. If you cannot find time in the next week to complete it, please let us know and we would be happy to work around your schedule. You may use any resources available to you, but we do ask that the work you submit is done completely by you. We realize this is a large time commitment, so we have replaced our coding and technical phone screen with this in order to minimize the time burden on you.

If you have any questions or need clarification on any of the requirements, please do not hesitate to ask. If you get stuck, you can ask for hints. You can email Dai at dai@rockset.com.

You will need an AWS account if you do not already have one (a regular Amazon account will work). $XXXXCOMPANYXXXX will reimburse you for any expenses you incur while completing this exercise. We will let you know as soon as we finish reviewing your submission so that you can turn off the AWS resources to stop incurring further fees.

In AWS, use the Elastic Container Service for Kubernetes (EKS) to create a new Kubernetes cluster. Add 3 worker nodes to this cluster using the instance type t3.small. Choose one of the US regions for this cluster. You can use this to help get started: https://docs.aws.amazon.com/eks/latest/userguide/getting-

started.html

1) Using your favorite web server image, create a single Kubernetes deployment of 3 pods in the cluster such that each pod runs an instance of the web server, and there is one pod running on each of the 3 worker nodes.

You will need a Docker image of the web server, which you can either build yourself or use an existing one. You can use these for reference:

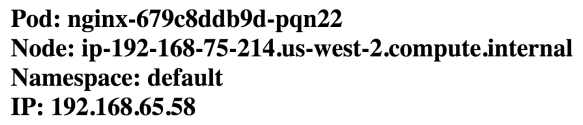
<https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>

<https://kubernetes.io/docs/reference/kubectl/overview/>

A possible choice for web server is nginx: <https://docs.docker.com/samples/library/nginx/>

2) Configure the web servers such that when an HTTP GET request is made to /, the server returns the pod name, node name, pod namespace, and pod IP.

Here is an example page:



3) Create a Kubernetes service of type 'LoadBalancer' that has the 3 web server pods as backends so that you can now issue requests to the web servers from the internet using a web browser. Refreshing the page in the browser should periodically return different information depending on which of the 3 web server pods the request was routed to.

You can use this for reference: <https://kubernetes.io/docs/concepts/services-networking/service/>

4) Bonus if you can automate this entire process. (not required)

## **Submission:**

Submit a write up of all the steps you took in detail (including any commands or scripts you had to run), and mention any areas where you got stuck or were particularly difficult and how you worked past it. Also submit all Kubernetes configuration files you used, any scripts or automation you wrote, and the URL of the external load balancer that was created so that we can test sending requests to the web servers you created.

—- My submission is below —-

# How to test:

#### On EC2:

for i in {1..10}; do curl a074e1c0d454243ccae0632a1675d8c8-1807468475.us-east-2.elb.amazonaws.com; done

#### Generically:

**Start on a Mac running OS X:**

% sw\_vers

ProductName: Mac OS X

ProductVersion: 10.15.7

BuildVersion: 19H15

% uname -a

Darwin <hostname> 19.6.0 Darwin Kernel Version 19.6.0: <TIME>; root:xnu-6153.141.2.2~1/RELEASE\_X86\_64 x86\_64

**Install Docker Desktop:**

meyerkev% docker --version

Docker version 20.10.0, build 7287ab3

**Install `brew`, `python3`, and `pip3` and add them to your PATH:**

meyerkev% brew --version

Homebrew 2.6.1

-Homebrew/homebrew-core (git revision 6865fd; last commit 2020-12-12)

-Homebrew/homebrew-cask (git revision 003c8e; last commit 2020-12-12)

meyerkev% pip3 --version

pip 20.3.1 from /usr/local/lib/python3.9/site-packages/pip (python 3.9)

meyerkev% python3 --version

Python 3.9.0

**Untar the provided tar.gz into a directory and**tar -xzvf meyerkev\_rockset\_submission.tar.gz  
cd rockset

* **Change the two account ID references from 386145735201 to yours**
* **Change SSH**
* **If desired, change the two regional references to run in a different region, the image name (Note: untested), and/or the cluster name**

**vars.sh**

*ACCOUNT\_ID=386145735201*

*REGION=us-east-2*

*CLUSTER\_NAME=meyerkev-rockset-interview*

*IMAGE=rockset*

**eks\_config/cluster\_config/rockset\_deployment.yaml**

*image: 386145735201.dkr.ecr.us-east-2.amazonaws.com/rockset:latest*

**eks\_config/rockset\_eks\_cluster.yaml**

*metadata:*

*name: meyerkev-rockset-interview*

*region: us-east-2*

*nodeGroups:*

*...*

*ssh:*

*publicKeyPath: ~/.ssh/ec2\_id\_rsa.pub # Attached in the ssh/ directories*

**Run:**

bin/from\_end\_to\_end.sh

In order, this will

* Install everything including a new virtualenv - bin/install\_deps.sh
* Create the docker image - bin/build\_image.sh
* Upload that image to EKS - bin/upload\_to\_eks.sh
* Turnup the cluster - bin/turnup\_cluster.sh
  + Note: The first 3 usually work, this last is a bit flaky, it may make sense to skip to retrying this at need.

When prompted, run aws configure with your account that has access

AWS Access Key ID [\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*IZ32]: <Your Key>

AWS Secret Access Key [\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Cc2q]: <Your Secret Key>

Default region name [us-east-2]: <YOUR REGION>

Default output format [json]: json

*[Ed Note: I am provisionally willing to give you mine on request on the condition that I be allowed to take some time to craft a super-tiny permissions set that can only launch clusters first. 5 rounds of playing IAM whack-a-mole in, I looked at the clock, gave up, and made the interview user IAM Admin]*

# The actual writeup:

## Locally build as much as we can in minikube without spending money or adding AWS complexities

AKA my first time doing anything serious with Kubernetes.

Because of that, let’s do this in slightly reverse order.

### Build a server.

*Configure the web servers such that when an HTTP GET request is made to /, the server returns the pod name, node name, pod namespace, and pod IP.*

Let's use flask because I don't know flask, but the examples look pretty simple and I do know Python and boto3

Create requirements.txt, add flask and preemptively add boto3 (Future Edit: unneeded)

Create rockset\_server.py with a basic hello\_world

Run it and connect… It works

Now apply a basic multi-line string output in plain-text.

Pod: <Kubernetes pod name, aka hostname>

Node: <EC2 hostname, currently incomplete>

Namespace: <Pod namespace>

IP: <Look up server ip address via boto3? Update: Possibly, but request.host.split also works in theory>

All of these come from env via spec based on <https://kubernetes.io/docs/tasks/inject-data-application/environment-variable-expose-pod-information/> , and since we’re not in Pods yet…

(From this point forward, since I am using this as a copy-paste doc, I will use Courier New for all terminal input and output, using Italics to denote output from input)

curl localhost:8080

*Pod: HomeMBP-4.attlocal.net # Hostname is the pod name in Kubernetes, so we use that as backup*

*Node: Not running in a pod*

*Namespace: Not running in a pod*

*IP: 127.0.0.1*

Alright, good enough.

Actually, let’s add a SERVER\_PORT environment variable to configure the open port and add a final "\n" to make curl cleaner and then good enough.

### Docker image (as DockerFile)

Start with Python 3.9.0, copy src to src and requirements.txt to requirements.txt, upgrade pip just in case, pip install -r requirements.txt, run flask app directly.

Cross fingers this works and move into Kubernetes.

### 1 Pod

Create a single pod that runs the web server

Because running Kubernetes locally with minikube doesn’t work with automatically picking up new versions, let's version every build and we'll have to delete the deployment every time.

Run once:

minikube start

eval $(minikube docker-env)

Repeat:

minikube kubectl -- delete deployment rockset

export DOCKER\_VERSION="rockset:$(date -u +%s)"

docker build -t ${DOCKER\_VERSION?} .

minikube kubectl -- create deployment rockset --image=${DOCKER\_VERSION?}

minikube kubectl -- get pods

*<some output>*

minikube kubectl -- exec --stdin --tty rockset-<something> -- /bin/bash

$ curl localhost:8080

*Pod: rockset-57b7b8d4b8-88g9s*

*Node: Not running in a pod # Ed Note: Correctly, since we haven't configured environment variables yet*

*Namespace: Not running in a pod*

*IP: localhost*

$

(Bonus: Learned how to disable smart quotes in GDocs today)

### 3 pods

*Using your favorite web server image, create a single Kubernetes deployment of 3 pods in the cluster such that each pod runs an instance of the web server, and there is one pod running on each of the 3 worker nodes. You will need a Docker image of the web server, which you can either build yourself or use an existing one.*

#### Moving this into a config file (and fixing the output via env vars finally)

I suspect there's a standard filepath for kubernetes configs, but I don't know what it is so we're using /kconfig/..

Update the sample from <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.19/#deployment-v1-apps> with the env variables from <https://kubernetes.io/docs/tasks/inject-data-application/environment-variable-expose-pod-information/> and use 3 replicas.

And we're hard-coding SERVER\_PORT=containerPort=8080 at the moment, since it's not immediately obvious how to tell SERVER\_PORT to use whatever the generated value of containerPort is. It's theoretically possible via ConfigMaps, but let's not complicate this too much. Bonus: Int/string confusion, chased the rabbit for a little bit.

And… minikube isn't finding my image...

HomeMBP-4:rockset meyerkev% minikube kubectl -- get pods

*pod/rockset-6f7d64954f-wljm7 0/1 ImagePullBackOff 0 3m36s*

*pod/rockset-74845887b4-78mgv 0/1 ErrImagePull 0 11s*

*pod/rockset-748cd8c8f5-dhvs2 0/1 ErrImageNeverPull 0 13m*

*pod/rockset-7895f4dc68-vlknj 0/1 ImagePullBackOff 0 12m*

HomeMBP-4:rockset meyerkev% minikube kubectl -- logs pod/rockset-7895f4dc68-vlknj

*Error from server (BadRequest): container "rockset" in pod "rockset-7895f4dc68-vlknj" is waiting to start: trying and failing to pull image*

Ah.

eval $(minikube docker-env) in this new terminal, build the image \*again\* to get it into minikube, use minikube kubectl instead of kubectl (Retroactively fixed in up-doc scripts because I actually use those) and it really helps if we don't delete all --all because that also kills the Docker daemon which needs a minikube start.

Also set imagePullPolicy: IfNotPresent, which will require a restart

minikube kubectl -- apply -f kconfig/

minikube kubectl -- get all -o wide

*HomeMBP-4:rockset meyerkev% minikube kubectl -- get all -o wide*

*\NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES*

*pod/rockset-74845887b4-4dzd9 1/1 Running 0 18s 172.17.0.2 minikube <none> <none>*

*pod/rockset-74845887b4-6pnc5 1/1 Running 0 18s 172.17.0.7 minikube <none> <none>*

*pod/rockset-74845887b4-hgkl9 1/1 Running 0 18s 172.17.0.5 minikube <none> <none>*

*NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE SELECTOR*

*service/kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 79s <none>*

*NAME READY UP-TO-DATE AVAILABLE AGE CONTAINERS IMAGES SELECTOR*

*deployment.apps/rockset 3/3 3 3 18s rockset rockset app=rockset-server*

*NAME DESIRED CURRENT READY AGE CONTAINERS IMAGES SELECTOR*

*replicaset.apps/rockset-74845887b4 3 3 3 18s rockset rockset app=rockset-server,pod-template-hash=74845887b4*

HomeMBP-4:development meyerkev% kubectl exec --stdin --tty rockset-74845887b4-4dzd9 -- /bin/bash

root@rockset-74845887b4-4dzd9:/usr/src/app# curl localhost:8080

*Pod: rockset-74845887b4-4dzd9*

*Node: minikube*

*Namespace: default*

*IP: localhost*

root@rockset-74845887b4-4dzd9:/usr/src/app#

Localhost feels wrong… and it's because I'm using the wrong environment variable IP\_ADDR instead of POD\_IP. Let's rebuild and try this again…

minikube kubectl -- delete deployment rockset # Needed because it won't pick up new images b/c minikube, local build, and imagePullPolicy: IfNotPresent  
minikube kubectl -- apply -f kconfig

minikube kubectl -- get all -o wide # is sane

*pod/rockset-74845887b4-4p5wg 1/1 Running 0 87s 172.17.0.7 minikube <none> <none>*

root@rockset-74845887b4-4p5wg:/usr/src/app# curl localhost:8080

*Pod: rockset-74845887b4-4p5wg*

*Node: minikube*

*Namespace: default*

*IP: 172.17.0.7*

Servers are done.

### Load balance those servers

*Create a Kubernetes service of type 'LoadBalancer' that has the 3 web server pods as backends so that you can now issue requests to the web servers from the internet using a web browser. Refreshing the page in the browser should periodically return different information depending on which of the 3 web server pods the request was routed to.*

*You can use this for reference:* [*https://kubernetes.io/docs/concepts/services-networking/service/*](https://kubernetes.io/docs/concepts/services-networking/service/)

#### Load Balancer config file

Create a new file rockset\_load\_balancer.yaml and apply -f. It works 2nd try b/c of a typo. And… no curl.

#### Curl doesn't work

HomeMBP-4:~ meyerkev% curl localhost:8080

curl: (56) Recv failure: Connection reset by peer

Let's -o wide. Oh, that would do it.

HomeMBP-4:rockset meyerkev% minikube kubectl -- get svc -o wide

*NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE SELECTOR*

*kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 40m <none>*

*rockset ClusterIP 10.110.90.88 <none> 80/TCP 2m35s app=rockset*

https://kubernetes.io/docs/tasks/access-application-cluster/create-external-load-balancer/

Add type: LoadBalancer to your config file

ExternalIP is now <pending>

And still pending.

https://stackoverflow.com/questions/44110876/kubernetes-service-external-ip-pending

https://kubernetes.io/docs/tasks/debug-application-cluster/debug-service/

Ok, minikube tunnel needs to be running in a separate terminal

https://minikube.sigs.k8s.io/docs/handbook/accessing/#loadbalancer-access

Let's consistently do that.

Lots of back and forth, the hello-node deploy from https://kubernetes.io/docs/tutorials/hello-minikube/ works on that same 8080 port, but our flask API doesn't.

nmap shows an open port on 8080

HomeMBP-4:~ meyerkev% nmap localhost

*Starting Nmap 7.91 ( https://nmap.org ) at 2020-12-06 01:12 EST*

*Nmap scan report for localhost (127.0.0.1)*

*Host is up (0.00019s latency).*

*Other addresses for localhost (not scanned): ::1 fe80::1*

*Not shown: 963 closed ports, 29 filtered ports*

*PORT STATE SERVICE*

*22/tcp open ssh*

*8080/tcp open http-proxy*

minikube kubectl -- get service hello-node -o yaml and minikube kubectl -- get service rockset -o yaml produce pretty similar results. Maybe it's a flask interaction with Docker?

#### It's a flask interaction with Docker

Turns out it's a flask issue. Flask needs to run as host 0.0.0.0 or things break. h/t https://github.com/pallets/flask/issues/2109 and https://forums.docker.com/t/docker-curl-56-recv-failure/54172/7

So rebuild the image, redeploy the deployment…

HomeMBP-4:~ meyerkev% curl localhost:8080

*Pod: rockset-65ccf9685c-zqw7p*

*Node: minikube*

*Namespace: default*

*IP: 172.17.0.3*

And change the LB port to 80, let's be kind.   
  
HomeMBP-4:~ meyerkev% curl localhost

*Pod: rockset-65ccf9685c-zqw7p*

*Node: minikube*

*Namespace: default*

*IP: 172.17.0.3*

### Quick Cleanup:

1. We don't need Boto3, so remove it from requirements.txt (which is now one line....)
2. Let's quickly automate the rebuild. bin/local\_rebuild.sh now rebuilds and locally deploys minikube.

We have a working deployment, now time for:

## Deploying to AWS

*In AWS, use the Elastic Container Service for Kubernetes (EKS) to create a new Kubernetes cluster. Add 3 worker nodes to this cluster using the instance type t3.small. Choose one of the US regions (us-east-2) for this cluster.*

*Using your favorite web server image, create a single Kubernetes deployment of 3 pods in the cluster such that each pod runs an instance of the web server, and there is one pod running on each of the 3 worker nodes. You will need a Docker image of the web server, which you can either build yourself or use an existing one.*

### Getting tools

* virtualenv
  + dev\_requirements.txt:
    - awscli
    - boto3
* aws
* eks\_ctl
  + Building this on a Mac, so using brew
* kubernetes-cli brew package
  + The configuration misbehaves a bit going through minikube

Luckily, I already had several variants of an install\_deps.sh script lying around, so let's steal one and edit it. And since there's a lot of output, go get a [Stackoverflow question](https://stackoverflow.com/questions/65165875/redirect-output-to-dev-null-only-if-verbose-is-not-set) and 2299 reputation.

### Build out an AWS account for the interview

* Fix the MFA issues with my personal account.
* In the console, create a new IAM user - rockset
  + `aws configure` with those IAM keys
* Give the rockset IAM user permissions using a custom policy to do anything it wants in ecr or eks in us-east-2
  + Broader than I'd like, but I can fix permissions inch by inch or I can fix code.

### Getting the image into EKS

EKS + ECR + Docker Push

https://docs.aws.amazon.com/AmazonECR/latest/userguide/docker-push-ecr-image.html

We need registry login first.

HomeMBP-4:rockset meyerkev% aws ecr get-login --region us-east-2 --no-include-email

An error occurred (AccessDeniedException) when calling the GetAuthorizationToken operation: User: arn:aws:iam::386145735201:user/rockset-interview is not authorized to perform: ecr:GetAuthorizationToken on resource: \*

OK, let's add that to our EKS policy.

aws ecr get-login --region us-east-2 --no-include-email

*docker login -u AWS -p <REDACTED> https://386145735201.dkr.ecr.us-east-2.amazonaws.com*

docker login -u AWS -p <REDACTED> https://386145735201.dkr.ecr.us-east-2.amazonaws.com

*Login Succeeded*

Create a repository rockset in the console following https://docs.aws.amazon.com/AmazonECR/latest/userguide/repository-create.html

docker images

*REPOSITORY TAG IMAGE ID CREATED SIZE*

*rockset latest 6e214d17b592 18 hours ago 897MB*

This was finicky enough I pre-wrote it as a script…. and now it runs. Very slowly because Dad has 25/3 internet.

*REGION=us-east-2*

*ACCOUNT\_ID=386145735201*

*IMAGE=rockset*

*ECR\_REGISTRY=${ACCOUNT\_ID?}.dkr.ecr.us-east-2.amazonaws.com/${IMAGE?}*

*# Runs `docker login` to give us keys into AWS*

*$(aws ecr get-login --region us-east-2 --no-include-email)*

*docker tag ${IMAGE?} ${ECR\_REGISTRY?}*

*docker push ${ECR\_REGISTRY?}*

And it worked. Sort of, tagged it as latest. Good enough for now, we can fix that later (or never because this has taken some time).

### Turning up a cluster of anything

<https://eksctl.io/usage/creating-and-managing-clusters/#using-config-files>

1. Copy that into rockset\_eks\_cluster.yaml and edit to taste
   1. Divide the local configs and the EKS configs
2. Create a new SSH public key to upload.
   1. Which is included with the submission.
3. Fix an IAM permission or two.
4. Or Five…

eksctl create cluster -v 5 -f eks\_config/rockset\_eks\_cluster.yaml

There's a very real tradeoff between my willingness to play permissions whack-a-mole on 5 minute cycles, and a narrowly crafted IAM. Since this is timed, albeit broadly, spending a solid hour if not more building the perfect IAM permission is a poor use of my time, but would be an \*excellent\* use of my time at a real company where that scales and there's very real monetary issues with giving IAM Admin to a random "Turn up Kubernetes clusters" user.

Final Result with EC2 Read-only, IAM Admin (Don't do this, but I wasted 5 cycles getting 5 different permissions and gave up), and every policy with EKS in the name and we still needed this:

{

"Statement": [

{

"Effect": "Allow",

"Action": [

"eks:\*"

],

"Resource": "arn:aws:eks:us-east-2:\*"

},

{

"Effect": "Allow",

"Action": [

"eks:DescribeCluster"

],

"Resource": "\*"

},

{

"Effect": "Allow",

"Action": [

"ecr:\*"

],

"Resource": "arn:aws:ecr:us-east-2:\*"

},

{

"Effect": "Allow",

"Action": [

"ecr:GetAuthorizationToken"

],

"Resource": "\*"

},

{

"Effect": "Allow",

"Action": [

"ssm:GetParameter"

],

"Resource": "arn:aws:ssm:us-east-2:386145735201:parameter/\*"

},

{

"Effect": "Allow",

"Action": [

"cloudformation:CreateStack"

],

"Resource": "\*"

}

]

}

And.. now we have a cluster.... 73 minutes later? I'm going to be honest here and admit that I didn't count this as part of my 4 hours. Also, since I did this on -v 5, I'm not going to copy this output.

*[Ed Note: Apparently, we needed verification. "You recently requested an AWS Service that required additional validation. Your request has now been validated for AWS US East (Ohio) region(s)."]*

eksctl create cluster -v 5 -f eks\_config/rockset\_eks\_cluster.yaml

### Getting the configs into AWS

#### Pointing kubectl at our new cluster now that the endpoint exists

HomeMBP-4:bin meyerkev% aws eks --region us-east-2 update-kubeconfig --name rockset-interview

*Added new context* arn:aws:eks:us-east-2:386145735201:cluster/rockset-interview to /Users/meyerkev/.kube/config

HomeMBP-4:bin meyerkev% kubectl get svc

*NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE*

*kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 36m*

#### Turning up a cluster

Let's try applying the local configs (I doubt this will work...)

HomeMBP-4:rockset meyerkev% kubectl apply -f local\_config/

*deployment.apps/rockset created*

*service/rockset created*

HomeMBP-4:rockset meyerkev% kubectl get all

*NAME READY STATUS RESTARTS AGE*

*pod/rockset-bd9b7779d-4z5vz 0/1 ImagePullBackOff 0 8s*

*pod/rockset-bd9b7779d-5q54t 0/1 ErrImagePull 0 8s*

*pod/rockset-bd9b7779d-dz8rg 0/1 ImagePullBackOff 0 8s*

*NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE*

*service/kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 40m*

*service/rockset LoadBalancer 10.100.10.4 a1bdfec5efbd4432f82b89e1dc16b640-1384802103.us-east-2.elb.amazonaws.com 80:31282/TCP 8s*

*NAME READY UP-TO-DATE AVAILABLE AGE*

*deployment.apps/rockset 0/3 3 0 10s*

*NAME DESIRED CURRENT READY AGE*

*replicaset.apps/rockset-bd9b7779d 3 3 0 10s*

Probably the image name is wrong… yes

HomeMBP-4:rockset meyerkev% kubectl logs pod/rockset-bd9b7779d-4z5vz

*Error from server (BadRequest): container "rockset" in pod "rockset-bd9b7779d-4z5vz" is waiting to start: image can't be pulled*

Let's check ecr for the image name… 386145735201.dkr.ecr.us-east-2.amazonaws.com/rockset:latest looks like a reasonable value.

reapply…

kubectl apply -f eks\_config/cluster\_config/

HomeMBP-4:rockset meyerkev% kubectl get all

*NAME READY STATUS RESTARTS AGE*

*pod/rockset-f5d7c5d99-2zpwq 1/1 Running 0 67s*

*pod/rockset-f5d7c5d99-l767n 1/1 Running 0 48s*

*pod/rockset-f5d7c5d99-p6f7j 1/1 Running 0 29s*

*NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE*

*service/kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 46m*

*service/rockset LoadBalancer 10.100.10.4* ***a1bdfec5efbd4432f82b89e1dc16b640-1384802103.us-east-2.elb.amazonaws.com*** *80:31282/TCP 6m30s*

*NAME READY UP-TO-DATE AVAILABLE AGE*

*deployment.apps/rockset 3/3 3 3 6m31s*

*NAME DESIRED CURRENT READY AGE*

*replicaset.apps/rockset-bd9b7779d 0 0 0 6m32s*

*replicaset.apps/rockset-f5d7c5d99 3 3 3 68s*

HomeMBP-4:rockset meyerkev% **curl a1bdfec5efbd4432f82b89e1dc16b640-1384802103.us-east-2.elb.amazonaws.com**

*Pod: rockset-f5d7c5d99-2zpwq*

*Node: ip-192-168-50-100.us-east-2.compute.internal*

*Namespace: default*

*IP: 192.168.46.79*

HomeMBP-4:rockset meyerkev% **curl a1bdfec5efbd4432f82b89e1dc16b640-1384802103.us-east-2.elb.amazonaws.com**

*Pod: rockset-f5d7c5d99-l767n*

*Node: ip-192-168-70-247.us-east-2.compute.internal*

*Namespace: default*

*IP: 192.168.72.15*

HomeMBP-4:rockset meyerkev% **curl a1bdfec5efbd4432f82b89e1dc16b640-1384802103.us-east-2.elb.amazonaws.com**

*Pod: rockset-f5d7c5d99-2zpwq*

*Node: ip-192-168-50-100.us-east-2.compute.internal*

*Namespace: default*

*IP: 192.168.46.79*

*HomeMBP-4:rockset meyerkev%*

**----------THIS IS HOW FAR I MADE IT---------------**

### Building a Route53 endpoint so I'm not constantly updating the docs to use the new endpoint

It would be very nice if as part of cluster turnup, we waited until we weren't in pending, then pointed a Route53 endpoint at the result.

As it stands, my last "up" cluster was the one linked above.

### Loosely automating this entire process

Im going to be honest and admit that I'd been building local scripts from the get-go, but from\_start\_to\_finish.sh is…. new. in the process, CloudFormation decided to not delete the underlying CloudFormation template, so our new cluster name is meyerkev-rockset-interview

Other than that, adding a clean script (cleaning up the env), a tar script (merge specifically the files we want into a tarball), and an untar->run script would've been useful.

Another thing that I found really elegant even if they used it wrong was that a previous employer would generate requirements.transitive.txt which would let them have floating dependencies, but hard-coded dependencies for the actual deploys. I have built out variants of that for other Python builds, but when your dependencies are literally "flask" and nothing else, I can let it slide from here.

## Final Cleanup

Error-checking and cleanup.

### What names made sense at the time that don't make sense now?

Arguably, a lot of places where "rockset" and "rockset-interview" should have been more deliberate. I ended up having to convert to meyerkev-rockset-interview just to get something running halfway through when CloudFormation decided to stop deleting stacks (Tracked that down to an issue deleting the load balancer).

Pretty much all the shell scripts could probably stand to be renamed, but they get the job done.

### Are there 8 names the same and can we get them from one spot?

Region, account\_id, cluster\_name, service\_name, Image name

The two places I need to do work are:

1. Syncing env vars to kube configs
2. Building a central config in my kube\_config.

In particular, #2 means there's a pretty reasonable number of places where the magic string is "rockset" and I know I don't understand what they all are.

## 