Goal 2: Research what type of Kubernetes components/services would best align with my code (and figure out how to run batch jobs and how to run mysql with persistent storage in a container)

First, a little bit of background on my program, stock-rsi.py. My python program is a client/server program that helps me make stock buying/selling decisions using the concept of RSI (relative strength index—the strength of a stock over a period of 30 days. https://www.investopedia.com/terms/r/rsi.asp). My program runs once a day, takes a file as an argument (a list of stock symbols), pulls 30 days of history for each stock in the list, computes an RSI value for each stock, stores that value in a mysql db using today's date as a primary key, and then prints to stdout the list of stocks sorted by strongest RSI value first. Here is a link to my code: https://github.com/olivelawn/python-analyze-stock-market/blob/master/stock-rsi.py

A sample run of the program looks like this:

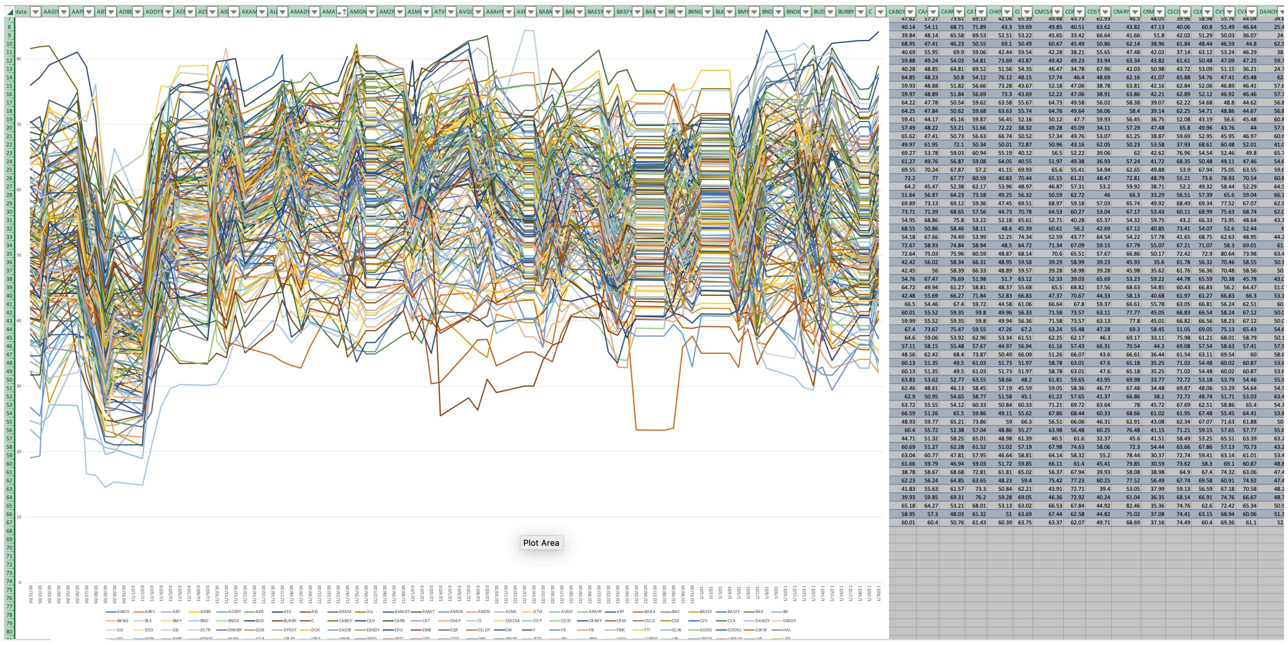
```
[parker@parker python-analyze-stock-market]$ ./stock-rsi.py symbols
[*********************100%***********************]  180 of 180 completed
EBKDY 83.98
CVS 79.29
ALL 72.48
MDLZ 71.55
UPS 70.05
ADDYY 70.03
ICLR 68.39
WMB 67.87
AKAM 67.64
TGT 67.62
NOC 67.62
NSRGY 67.59
…
…
```

And you can see from the below that each stock's RSI values are inserted into a mysql table named "stocks."

```
mysql> select * from stocks where date = '2021-05-18';
| date       | AAGIY | AAPL  | ABT   | ADBE  | ADDYY | AER   | AES   | AIG   | AKAM  | ALL   | AMADY | AMAT  | AMGN  | AMZN |
ASML  | ATVI  | AVGO  | AXAHY | AXP   | BABA  | BAC   | BAESY | BASFY | BAX   | BK    | BKNG  | BLK   | BMY   | BND   | BNDX  |
BUD   | BURBY | C     | CABGY | CAH   | CARR  | CAT   | CHKP  | CI    | CMCSA | COP   | COST  | CRARY | CRM   | CSCO | CSX   |
CVS  | CVX   | DANOY | DBSDY | DD    | DEO  | DIS   | DLTR  | DNHBY | DOX   | DPSGY | DUK   | EADSY | EBKDY | EDU   | EMB   |
EQR   | ESLOY | EW    | F     | FB    | FIS   | FMC   | FTI   | GLW  | GOOG  | GOOGL | GWW   | HAL   | HD    | HON   | HSBC |
HTHIY | HUM   | ICLR  | IDEXY | IDXX  | ILMN  | INFO | INTC  | IOO   | ITOT  | IWY   | JBAXY | JETS  | JNJ   | JPM   | LGLV  |
LGRDY | LIN   | LNSTY | LVMUY | LVS  | LYG  | MDLZ  | MET   | MO    | MPC  | MRK   | MRO   | MS    | MSCI  | MSFT  | NEE   |
NGLOY | NKE   | NOC   | NOW   | NRDBY | NSRGY | NTNX | NVDA | NVS   | ORCL  | OTIS | OXY   | PEP   | PFE   | PG    | PM   |
PNGAY | PRU   | PXD   | PYPL  | QCOM  | QQQ   | RCI   | RDSMY | RHHBY | RTX   | SAP   | SCHW | SNN   | SPLK | SQ    | SSDOY |
STNE | SU   | T     | TCOM  | TFC   | TMO   | TMSNY | TRYIY | TSM   | TSN   | TXN   | UBER  | UMICY | UNH   | UNM   | UNP   |
UPS   | V     | VDE  | VLO   | VMW  | VNQ  | VNQI  | VTI   | VZ    | WFC   | WMB   | ZTS   | ABNB | MCHI  | VB    | VO    |
VTRS  | VWO   | ARKK  | CNRG  | IWR   | JKH  | MSSMX | SCHD  | IMCG  | THNPY | TDUP  | ABBV  | ACN   | CHTR  | DOW   | EXC   |
GD    | GS    | IEMG  | KHC   | LLY   | LMT   | MCD   | NFLX  | SCHF  | SO    | SPG   | TFI   | TGT  | VEA  | VOO   | VTEB  |
VXF   | WMT   |

| 2021-05-18 | 60.28 | 45.75 | 41.71 | 45.72 | 71.68 | 48.79 | 34.88 | 63.31 | 69.49 | 75.09 |  NULL | 44.29 | 53.19 | 51.5 |
51.53 | 49.14 | 44.95 | 51.11 | 61.56 | 35.97 | 68.22 |  NULL | 50.86 | 36.46 | 66.83 | 40.89 | 60.48 | 58.64 | 42.98 | 31.84 |
68.46 |  NULL | 68.44 |  NULL | 37.27 | 52.39 | 65.94 | 44.94 | 61.14 | 48.83 | 65.25 | 65.27 |  NULL | 43.63 | NULL | 48.44 |
76.6 | 56.12 |  NULL | 49.45 | 69.04 | NULL | 28.42 | 39.21 | 51.89 | 53.58 |  62.9 | 60.33 | 48.46 | 81.31 | 29.66 | 55.24 |
51.52 |  NULL | 49.18 | 58.43 | 56.22 | 48.62 | 54.43 | 60.07 | 47.1 | 55.39 | 54.84 | 67.61 | 69.05 | 46.73 | 50.13 | NULL |
NULL | 51.78 | 59.35 |  NULL | 48.43 | 42.72 | NULL | 38.79 | 56.83 | 52.08 | 46.94 |  NULL | 56.33 | 63.46 | 67.91 | 57.18 |
66.21 | 62.18 |  NULL | 55.76 | NULL | NULL | 69.08 | 58.65 | 54.77 | NULL | 61.02 | 61.42 | 65.66 | 41.31 | 46.61 | 33.77 |
NULL | 56.56 | 69.69 | 33.07 |  NULL | 63.35 | 61.4 | 47.3 | 58.07 | 58.42 | NULL | 54.58 | 57.79 | 64.18 | 56.63 | 64.6 |
35.8 | 70.12 | 54.58 | 42.55 | 46.14 | 45.74 | NULL |    53 | 55.78 | 67.44 | 51.29 | NULL | 59.04 | 36.36 | 35.68 |  NULL |
NULL | 67.8 | 31.05 | 57.43 | 63.37 | 37.49 |    49 |  46.1 | 44.71 | 67.53 | 47.06 | 42.64 |  NULL | 61.53 | 63.85 | 50.51 |
73.92 | 53.03 | 67.5 | 64.23 | NULL | 52.6 | 55.58 | 51.96 | 41.29 | 70.42 | 74.82 | 52.05 |   26 | 45.08 | 50.42 | 51.24 |
75.33 | 51.28 | 35.57 | 36.13 | 52.04 | NULL | 35.51 | 61.64 | 42.95 |  54.5 | 48.82 | 63.52 | 50.16 | 56.74 | 69.01 | 55.33 |
55.73 | 66.21 | 50.22 | 68.82 | 58.69 | 62.55 | 52.12 | 39.48 | 57.88 | 49.66 | 54.86 | 48.04 | 56.1 | 57.7 | 53.74 | 52.01 |
46.23 | 62.41 |
```

At this point I've been storing these values over the last 140 or so trading days, so graphing the results over time using some ODBC hooks in excel looks like this:



I'm currently executing this on a VM which is automatically powered up once a day, and then powered down after program execution. This program typically takes 2-3 minutes to execute.

While it is technically possible, it is generally a bad idea to package two distinct applications (in my case "stock-rsi.py" and mysql) into a single container (https://cloud.google.com/architecture/best-practices-for-building-containers). While not entirely applicable to my program, scaling an application beyond a single container would present each application its own DB and hence multiple distinct sources of truth. Even though I probably could have coupled them, I decided to run stock-rsi.py and mysql in separate containers to follow best practices.

Admittedly, because stock-rsi.py is a batch program (and not a long running webservice), it isn't a good fit for Kubernetes. Kubernetes is best at monitoring applications that expose itself over the network. Because my code only makes an outbound request to the internet for stock data and then a subsequent call to the mysql DB, it is challenging for Kubernetes to understand the "health" of my application. As a container, stock-rsi.py will execute and finish, but then leave a zombie container running in perpetuity until manually cleaned up. On the other hand, my mysql DB container does expose a network service on port 3306. If Kubernetes sees that port down, it will automatically restart the mysql container.

When researching how other folks move "batch" types of workloads to Kubernetes, it looks like most people are firing off jobs using a CI/CD pipeline tool like Jenkins, and then using other programs in the pipeline to process the results for validity. Building a CI/CD pipeline is beyond the scope of my goal.

Despite not being a good fit for Kubernetes, my program is nonetheless an interesting academic use case for Kubernetes.

Most everything in Kubernetes is ephemeral. As a container orchestrator, Kubernetes load balances traffic against multiple identical containers, and insures that these containers are running properly. Behind the scenes, Kubernetes could be destroying faulty containers and subsequently creating new ones and typically does so without anyone noticing. Kubernetes strength is serving up stateless applications, so running mysql as a container in Kubernetes is something of a challenge. The good news is that there is a publicly available mysql container image on docker hub, so I didn't need to package it up on my own.

The Kubernetes master (specifically the scheduler) is responsible to making "just in time" decisions about where to place workloads. It generally makes these decisions by looking at which node in the cluster has available resources. In a cluster with 2 worker nodes, it is impossible to know in advance where the scheduler will place a mysql db container. Because of this we need a central storage location that either of the worker nodes can mount/access that persists the "stocks" table over container invocations and moreover stores RSI data over multiple days/weeks/years. This is where the NFS server (c1-nfs1) comes into play. By using the Kubernetes objects such as a PersistentVolume(PV), and then grabbing those storage resources in the form of a PersistentVolumeClaim(PVC), a mysql pod deployment can mount the central storage over the network and on the fly and subsequently read and write to the DB stock tables.

Here is my Kubernetes yaml description for the PersistentVolume(PV) required of the mysql container: https://github.com/olivelawn/cs199-indep-k8s/blob/main/k8s-configs/mysql/pv.yaml
My Kubernetes yaml description for the PersistentVolumeClaim(PVC) required of the mysql container: https://github.com/olivelawn/cs199-indep-k8s/blob/main/k8s-configs/mysql/pvc.yaml
My Kubernetes yaml description for the mysql deployment itself: https://github.com/olivelawn/cs199-indep-k8s/blob/main/k8s-configs/mysql/deploy.yaml
Finally, my Kubernetes yaml file that exposes the mysql DB service to other containers in the cluster: https://github.com/olivelawn/cs199-indep-k8s/blob/main/k8s-configs/mysql/svc.yaml

Here is what a running mysql service looks like in kubernetes after applying the above yaml configuration.

```
parker@c1-master1:~/mysql$ kubectl apply -f pv.yaml; kubectl apply -f pvc.yaml; kubectl apply -f deploy.yaml; kubectl apply -f
svc.yaml
persistentvolume/mysql created
persistentvolumeclaim/mysqlpvc created
deployment.apps/mysql created
service/mysqlsvc created
parker@c1-master1:~/mysql$ kubectl get pods -o wide
NAME                    READY   STATUS    RESTARTS   AGE   IP          NODE       NOMINATED NODE   READINESS GATES
mysql-68c777748-ftnm4   1/1     Running   0          44s   10.44.0.1   c1-node1   <none>           <none>
```

In the above, Kubernetes places the mysql db pod on host c1-node1. When the pod is deployed, it makes an NFS storage request to c1-nfs1 and mounts the files. We can see this from c1-node1:

```
parker@c1-node1:~$ sudo mount | grep -i nfs
c1-nfs1:/export/volumes/mysql on /var/lib/kubelet/pods/cc2cbbd5-b903-471d-9231-5482c34d59d4/volumes/kubernetes.io~nfs/mysql
type nfs4
(rw,relatime,vers=4.1,rsize=131072,wsize=131072,namlen=255,hard,proto=tcp,timeo=600,retrans=2,sec=sys,clientaddr=10.0.2.5,local
_lock=none,addr=10.0.2.7)
```

Finally, let's prove that mysql data persists over multiple container deployments. We exposed mysql on port in one of the previous steps. This is what it looks like:

```
parker@c1-master1:~/mysql$ kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
mysqlsvc     ClusterIP   10.105.59.47   <none>        3306/TCP   14s
```

Now let's connect to the database, delete a row in the stocks table, kill the mysql pod entirely, recreate it, and check for the existence of the row we deleted:

```
parker@c1-master1:~/mysql$ mysql --host=10.105.59.47 --user=root --password=mysqlpassword
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
mysql> select COUNT(date) from stocks where date = '2021-05-17';
+-------------+
| COUNT(date) |
+-------------+
|           1 |
+-------------+
1 row in set (0.04 sec)

mysql> delete from stocks where date = '2021-05-17';
Query OK, 1 row affected (0.10 sec)

mysql> select COUNT(date) from stocks where date = '2021-05-17';
+-------------+
| COUNT(date) |
+-------------+
|           0 |
+-------------+
1 row in set (0.02 sec)

mysql> ^DBye
```

Let's kill the mysql container and service and recreate.

```
parker@c1-master1:~/mysql$ kubectl delete svc mysqlsvc
service "mysqlsvc" deleted
parker@c1-master1:~/mysql$ kubectl delete deployments.apps mysql
deployment.apps "mysql" deleted
```

and recreate:

```
parker@c1-master1:~/mysql$ kubectl apply -f deploy.yaml
deployment.apps/mysql created
parker@c1-master1:~/mysql$ kubectl apply -f svc.yaml
service/mysqlsvc created
```

Get service info:

```
parker@c1-master1:~/mysql$ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
mysqlsvc     ClusterIP   10.109.20.156   <none>        3306/TCP   10s
```

Let's connect to the DB and check for the row previously deleted

```
parker@c1-master1:~/mysql$ mysql --host=10.109.20.156 --user=root --password=mysqlpassword
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
mysql> select COUNT(date) from stocks where date = '2021-05-17';
+-------------+
| COUNT(date) |
+-------------+
|           0 |
+-------------+
1 row in set (0.02 sec)

mysql> exit
Bye
```

This demonstration shows that persisting data in a Kubernetes cluster is possible for a mysql DB with centralized NFS storage using PersistentVolumes and PersistentVolumeClaims.