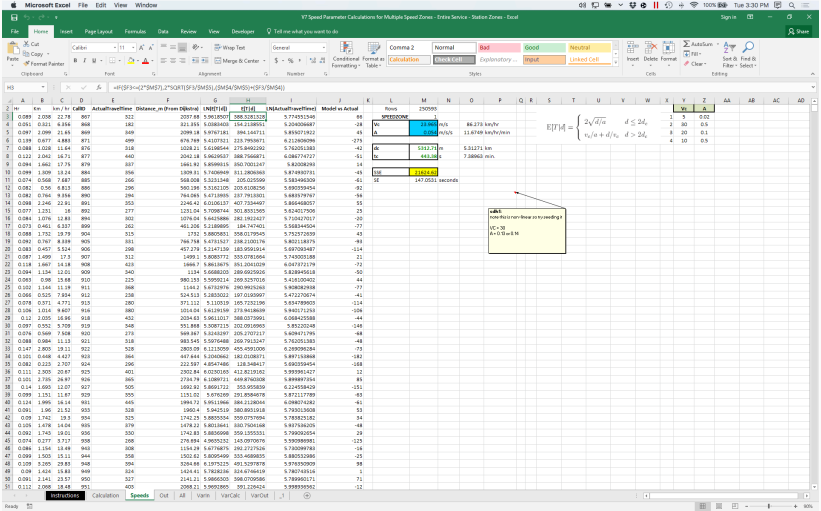
A talker on Docker:

How containers can make your work more reproducible, accessible, and ready for production.

Finbarr Timbers, Analyst, Darkhorse Analytics

Three stories.

One: Moving a nonlinear regression from Excel to Python.



V7 Speed Parameter Calculations for Multiple Speed Zones - Entire Service - Station Zones - Excel

H3 =IF($F3<=(2*$M$7),2*SQRT($F3/$M$5),($M$4/$M$5)+($F3/$M$4))

Row	Hr	Km	km / hr	CallID	ActualTravelTime	Distance_m (From Dijkstra)	LN(E[T|d])	E[T|d]	LN(ActualTravelTime)	Model vs Actual
3	0.089	2.038	22.78	867	322	2037.68	5.9618507	388.3281328	5.774551546	66
4	0.051	0.321	6.356	868	182	321.355	5.0383403	154.2138551	5.204006687	-28
5	0.097	2.099	21.65	869	349	2099.18	5.9767181	394.144711	5.855071922	45
6	0.139	0.677	4.883	871	499	676.769	5.4107321	223.7953671	6.212606096	-275
7	0.088	1.028	11.64	876	318	1028.21	5.6198544	275.8492292	5.762051383	-42
8	0.122	2.042	16.71	877	440	2042.18	5.9629537	388.7566871	6.086774727	-51
9	0.094	1.662	17.75	879	337	1661.92	5.8599315	350.7001247	5.82008293	14
10	0.099	1.309	13.24	884	356	1309.31	5.7406949	311.2806363	5.874930731	-45
11	0.074	0.568	7.687	885	266	568.008	5.3231348	205.025599	5.583496309	-61
12	0.082	0.56	6.813	886	296	560.196	5.3162105	203.6108256	5.690359454	-92
13	0.082	0.764	9.356	890	294	764.065	5.4713935	237.7913301	5.683579767	-56
14	0.098	2.246	22.91	891	353	2246.42	6.0106137	407.7334497	5.866468057	55
15	0.077	1.231	16	892	277	1231.04	5.7098744	301.8331565	5.624017506	25
16	0.084	1.076	12.83	894	302	1076.04	5.6425886	282.1922427	5.710427017	-20
17	0.073	0.461	6.337	899	262	461.206	5.2189895	184.747401	5.568344504	-77
18	0.088	1.732	19.79	904	315	1732	5.8805831	358.0179545	5.752572639	43
19	0.092	0.767	8.339	905	331	766.758	5.4731527	238.2100176	5.802118375	-93
20	0.083	0.457	5.524	906	298	457.279	5.2147139	183.9591914	5.697093487	-114
21	0.087	1.499	17.3	907	312	1499.1	5.8083772	333.0781664	5.743003188	21
22	0.118	1.667	14.18	908	423	1666.7	5.8613675	351.2041029	6.047372179	-72
23	0.094	1.134	12.01	909	340	1134	5.6688203	289.6925926	5.828945618	-50
24	0.063	0.98	15.68	910	225	980.153	5.5959214	269.3257016	5.416100402	44
25	0.102	1.144	11.19	911	368	1144.2	5.6732976	290.9925263	5.908082938	-77
26	0.066	0.525	7.934	912	238	524.513	5.2833022	197.0193997	5.472270674	-41
27	0.078	0.371	4.771	913	280	371.112	5.110319	165.7232196	5.634789603	-114
28	0.106	1.014	9.607	916	380	1014.04	5.6129159	273.9418639	5.940171253	-106
29	0.12	2.035	16.96	918	432	2034.63	5.9611017	388.0373991	6.068425588	-44
30	0.097	0.552	5.709	919	348	551.868	5.3087215	202.0916963	5.85220248	-146
31	0.076	0.569	7.508	920	273	569.367	5.3243297	205.2707217	5.609471795	-68
32	0.088	0.984	11.13	921	318	983.545	5.5976488	269.7913247	5.762051383	-48
33	0.147	2.803	19.11	922	528	2803.09	6.1213059	455.4591006	6.269096284	-73
34	0.101	0.448	4.427	923	364	447.644	5.2040662	182.0108371	5.897153868	-182
35	0.082	0.223	2.707	924	296	222.597	4.8547486	128.348417	5.690359454	-168
36	0.111	2.303	20.67	925	401	2302.84	6.0230163	412.8219162	5.993961427	12
37	0.101	2.735	26.97	926	365	2734.79	6.1089721	449.8760308	5.899897354	85
38	0.14	1.693	12.07	927	505	1692.92	5.8691722	353.955839	6.224558429	-151
39	0.099	1.151	11.67	929	355	1151.02	5.676269	291.8584678	5.872117789	-63
40	0.124	1.995	16.14	931	445	1994.72	5.9511966	384.2128044	6.098074282	-61
41	0.091	1.96	21.52	933	328	1960.4	5.942519	380.8931918	5.793013608	53
42	0.09	1.742	19.3	934	325	1742.25	5.8835334	359.0757694	5.783825182	34
43	0.105	1.478	14.04	935	379	1478.22	5.8013641	330.7504168	5.937536205	-48
44	0.092	1.743	19.01	936	330	1742.83	5.8836998	359.1355331	5.799092654	29
45	0.074	0.277	3.717	938	268	276.694	4.9635232	143.0970676	5.590986981	-125
46	0.086	1.154	13.49	943	308	1154.29	5.6776875	292.2727526	5.730099783	-16
47	0.099	1.503	15.11	944	358	1502.62	5.8095499	333.4689835	5.880532986	-25
48	0.109	3.265	29.83	948	394	3264.66	6.1975225	491.5297878	5.976350909	98
49	0.09	1.424	15.83	949	324	1424.41	5.7828236	324.6746419	5.780743516	1
50	0.091	2.141	23.57	950	327	2141.21	5.9866303	398.0709586	5.789960171	71
51	0.112	2.068	18.48	951	403	2068.21	5.9692865	391.226424	5.998936562	-12

Rows	250593			
SPEEDZONE	1			
Vc	23.965	m/s	86.273	km/hr
A	0.054	m/s/s	11.6749	km/hr/min
dc	5312.71	m	5.31271	km
tc	443.38	s	7.38963	min.
SSE	21624.62			
SE	147.0531	seconds		

$$E[T|d] = \begin{cases} 2\sqrt{d/a} & d \leq 2d_c \\ v_c/a + d/v_c & d > 2d_c \end{cases}$$

	Vc	A
1	5	0.02
2	30	0.5
3	20	0.1
4	10	0.5

sdh1:
note this is non-linear so try seeding it

VC = 30
A = 0.13 or 0.14

Instructions | Calculation | Speeds | Out | All | VarIn | VarCalc | VarOut | _1

One: Moving a nonlinear regression from Excel to Python.

The solution:

NumPy
Base N-dimensional array package

SciPy library
Fundamental library for scientific computing

But...

“Hey Finbarr, can you help? The code doesn’t seem to run.”

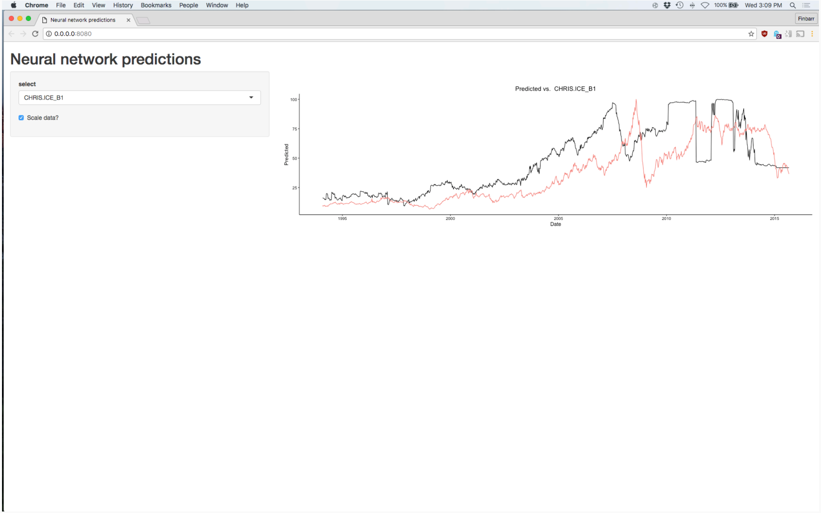
The solution?

Fiddle with the computer for 20 minutes.

Two: Sharing exploratory models

TensorFlow

Two: Sharing exploratory models



Three: Running statistical model on client's system

Windows Server® 2012

(If you're a consultant, this happens a lot).

Three: Running statistical model on client's system

All we knew was:

1. We had access to a database.

Three: Running statistical model on client's system

All we knew was:

1. We had access to a database.
2. We had to create an application that would talk to that database.

The solution?

Three: Running statistical model on client's system

1. Attend a series of meeting with the client's IT team discussing their systems and our needs.

Three: Running statistical model on client's system

1. Attend a series of meeting with the client's IT team discussing their systems and our needs.
2. Write a comprehensive test suite that ensured every possible point of failure was covered.

Three: Running statistical model on client's system

1. Attend a series of meeting with the client's IT team discussing their systems and our needs.
2. Write a comprehensive test suite that ensured every possible point of failure was covered.
3. Pray.

Is there a common thread?

Problems

1. Unmet dependencies.

Problems

1. Unmet dependencies.
2. Undefined production environments.

Problems

1. Unmet dependencies.
2. Undefined production environments.
3. Lengthy setup/install processes.

If only there was something that could help us...

An Ideal solution would be:

1. Portable. It works on every computer in the same way.

An Ideal solution would be:

1. Portable. It works on every computer in the same way.
2. Easy to set up.

An Ideal solution would be:

1. Portable. It works on every computer in the same way.
2. Easy to set up.
3. Easy to deploy.

An Ideal solution would be:

1. Portable. It works on every computer in the same way.
2. Easy to set up.
3. Easy to deploy.
4. Fast— as close to running the code natively as possible.

docker

What is Docker?

- Allows for the creation of “containers”

What is Docker?

- Allows for the creation of “containers”
- Containers are lightweight VMs that wrap up code with everything needed to run it

What is Docker?

- Allows for the creation of “containers”
- Containers are lightweight VMs that wrap up code with everything needed to run it
- “Write once run everywhere”

What is Docker?

- Allows for the creation of “containers”
- Containers are lightweight VMs that wrap up code with everything needed to run it
- “Write once run everywhere”
- Easy to write and use

Let’s revisit our three stories...

One: Moving a nonlinear regression from Excel to Python.

- After we have the Python script (`nonlinear-regression.py`), add a Dockerfile:

```
FROM python:3.5.2-slim
RUN pip install numpy pandas pymssql
CMD python nonlinear-regression.py
```

One: Moving a nonlinear regression from Excel to Python.

- After we have the Python script (`nonlinear-regression.py`), add a Dockerfile:

```
FROM python:3.5.2-slim
RUN pip install numpy pandas pymssql
CMD python nonlinear-regression.py
```

- Time to build from scratch: 1:58.47

One: Moving a nonlinear regression from Excel to Python.

- After we have the Python script (`nonlinear-regression.py`), add a Dockerfile:

```
FROM python:3.5.2-slim
RUN pip install numpy pandas pymssql
CMD python nonlinear-regression.py
```

- Time to build from scratch: 1:58.47
- Time to update Python code and rebuild: 0.629s

One: Moving a nonlinear regression from Excel to Python.

- After we have the Python script (`nonlinear-regression.py`), add a Dockerfile:

```
FROM python:3.5.2-slim
RUN pip install numpy pandas pymssql
CMD python nonlinear-regression.py
```

- Time to build from scratch: 1:58.47
- Time to update Python code and rebuild: 0.629s
- Size: 648 MB (461.3 MB of that are the packages)

Two: Sharing exploratory models

- Dockerfile:

```
FROM tensorflow/tensorflow

RUN pip install numpy sklearn pandas
ADD world_oil_forecast_data.csv /home
ADD model.py /home
WORKDIR /home
CMD python model.py
```

Two: Sharing exploratory models

- Dockerfile:

```
FROM tensorflow/tensorflow

RUN pip install numpy sklearn pandas
ADD world_oil_forecast_data.csv /home
ADD model.py /home
WORKDIR /home
CMD python model.py
```

- Time to build from scratch: 2.55.19

Two: Sharing exploratory models

- Dockerfile:

```
FROM tensorflow/tensorflow

RUN pip install numpy sklearn pandas
ADD world_oil_forecast_data.csv /home
ADD model.py /home
WORKDIR /home
CMD python model.py
```

- Time to build from scratch: 2.55.19
- Size: 863.2 MB (mostly packages, but some upstream bloat).

Three: Running statistical model on client's system

- Dockerfile:

```
FROM python:3.5.2-slim

# Install build-essential, git and other dependencies
RUN pip install numpy pandas sklearn \
    scipy pymssql hypothesis
ADD weighting_algorithm.py /home
ADD test_wa.py /home
WORKDIR /home
CMD python test_wa.py
```

Three: Running statistical model on client's system

- Dockerfile:

```
FROM python:3.5.2-slim

# Install build-essential, git and other dependencies
RUN pip install numpy pandas sklearn \
    scipy pymssql hypothesis
ADD weighting_algorithm.py /home
ADD test_wa.py /home
WORKDIR /home
CMD python test_wa.py
```

- Time to build from scratch: 2:00.50

Three: Running statistical model on client's system

- Dockerfile:

```
FROM python:3.5.2-slim

# Install build-essential, git and other dependencies
RUN pip install numpy pandas sklearn \
    scipy pymssql hypothesis
ADD weighting_algorithm.py /home
ADD test_wa.py /home
WORKDIR /home
CMD python test_wa.py
```

- Time to build from scratch: 2:00.50
- Size: 681.3 MB (packages are 483.5 MB of that).

Docker basics

Dockerfiles:

```
FROM python:3.5.2-slim

RUN pip install numpy pandas sklearn scipy \
    pymssql hypothesis
ADD weighting_algorithm.py /home
ADD test_wa.py /home
WORKDIR /home
CMD python test_wa.py
```

Dockerfiles

1. Base Image:

```
FROM python:3.5.2-slim
```

2. Directives:

```
RUN pip install numpy pandas sklearn scipy pymssql \
    hypothesis
ADD weighting_algorithm.py /home
ADD test_wa.py /home
WORKDIR /home
```

3. The command:

```
CMD python test_wa.py
```

CLI Basics

- Once you have a Dockerfile, build a container with `docker build -t weighting-algorithm .`

CLI Basics

- Once you have a Dockerfile, build a container with `docker build -t weighting-algorithm .`
- This builds a container called `weighting-algorithm` from the file named `Dockerfile` sitting in your current folder (works similar to `Make`)

CLI Basics

- Once you have a Dockerfile, build a container with `docker build -t weighting-algorithm .`
- This builds a container called `weighting-algorithm` from the file named `Dockerfile` sitting in your current folder (works similar to `Make`)
- Once built, run anywhere on your path with `docker run weighting-algorithm`

Example

- We have a Shiny app (R code that displays images in HTML)

Example

- We have a Shiny app (R code that displays images in HTML)
- Code is in two files: `server.R` and `ui.R`, with three data files (`data.csv`, `preds_actuals.csv`, `output.csv`).

Example

- We have a Shiny app (R code that displays images in HTML)
- Code is in two files: `server.R` and `ui.R`, with three data files (`data.csv`, `preds_actuals.csv`, `output.csv`).
- We run the app with the command `R -e 'shiny::runApp(".", host="0.0.0.0", port=8080)'`

Example

- We have a Shiny app (R code that displays images in HTML)
- Code is in two files: `server.R` and `ui.R`, with three data files (`data.csv`, `preds_actuals.csv`, `output.csv`).
- We run the app with the command `R -e 'shiny::runApp(".", host="0.0.0.0", port=8080)'`
- How can we turn this into a Docker container?

Example

- Dockerfile:

```
FROM rocker/shiny

RUN R -e "install.packages(c('ggplot2'))"
ADD preds_actuals.csv /home
ADD data.csv /home
ADD output.csv /home
ADD server.R /home
ADD ui.R /home
WORKDIR /home
EXPOSE 8080
CMD R -e \
    'shiny::runApp(".", host="0.0.0.0", port=8080)'
```

Example

- `docker build -t tf-shinyapp`

Example

- `docker build -t tf-shinyapp`
- `docker run -p 8080:8080 tf-shinyapp`

One more thing...

We can instantly deploy this to Google's Cloud (assuming we have a cluster running on Google Container Engine):

```
gcloud docker push \
    gcr.io/applied-ridge-137723/tf-shinyapp
kubectl run tf-shinyapp \
    --image=gcr.io/applied-ridge-137723/tf-shinyapp \
    --port=8080
kubectl expose deployment \
    tf-shinyapp --type=”LoadBalancer”
kubectl get service tf-shinyapp
```

The Cloud

- Super easy to push to the cloud because containers are self-contained

The Cloud

- Super easy to push to the cloud because containers are self-contained
- Most cloud providers (GCE, AWS) don't charge any extra for containers

The Cloud

- Super easy to push to the cloud because containers are self-contained
- Most cloud providers (GCE, AWS) don't charge any extra for containers
- Only cost is from reduced performance

Performance

Felter, Wes, et al. "An updated performance comparison of virtual machines and linux containers." Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On. IEEE, 2015.

Performance

TABLE I. RESULTS FOR PXZ, LINPACK, STREAM, AND RANDOMACCESS. EACH DATA POINT IS THE ARITHMETIC MEAN OF TEN RUNS. DEPARTURE FROM NATIVE EXECUTION IS SHOW WITHIN PARENTHESES "()". THE STANDARD DEVIATION IS SHOWN WITHIN SQUARE BRACKETS "[]".

Workload		Native	Docker	KVM-untuned	KVM-tuned
PXZ (MB/s)		76.2 [±0.93]	73.5 (-4%) [±0.64]	59.2 (-22%) [±1.88]	62.2 (-18%) [±1.33]
Linpack (GFLOPS)		290.8 [±1.13]	290.9 (-0%) [±0.98]	241.3 (-17%) [±1.18]	284.2 (-2%) [±1.45]
RandomAccess (GUPS)		0.0126 [±0.00029]	0.0124 (-2%) [±0.00044]	0.0125 (-1%) [±0.00032]	Tuned run not warranted
Stream (GB/s)	Add	45.8 [±0.21]	45.6 (-0%) [±0.55]	45.0 (-2%) [±0.19]	
	Copy	41.3 [±0.06]	41.2 (-0%) [±0.08]	40.1 (-3%) [±0.21]	
	Scale	41.2 [±0.08]	41.2 (-0%) [±0.06]	40.0 (-3%) [±0.15]	
	Triad	45.6 [±0.12]	45.6 (-0%) [±0.49]	45.0 (-1%) [±0.20]	

Performance

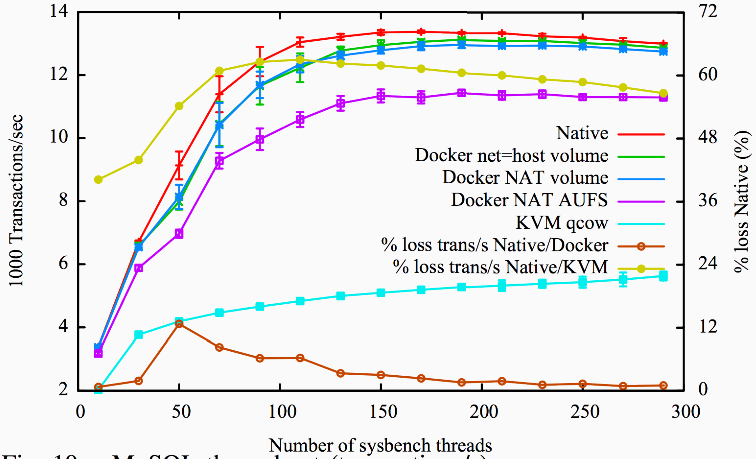


Fig. 10. MySQL throughput (transactions/s) vs. concurrency.

Performance

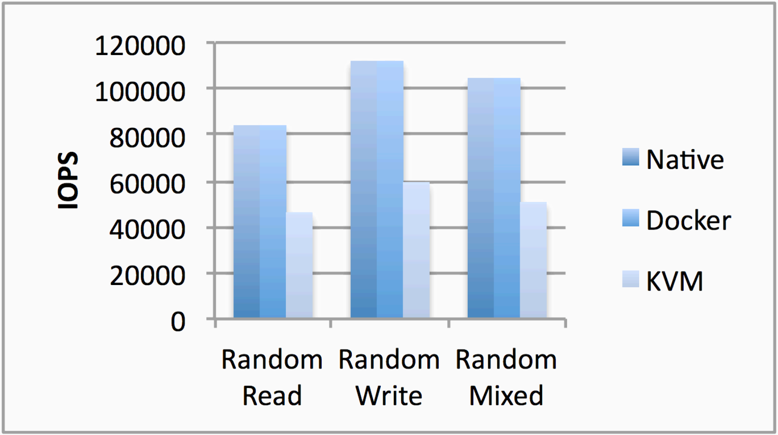


Fig. 6. Random I/O throughput (IOPS).