Python + Jupyter for Performance Testing

Setting the stage.

- Working on performance improvement to ZFS (sync writes)
- To verify my changes, I needed to:
    1. Measure the performance of the system **without** my changes.
    2. Measure the performance of the system **with** my changes.
    3. Analyze the difference(s) in performance with and without my changes.
    4. Collect tangential information from the system, to support (or refute) my conclusions.

Visualizations required?

- While not strictly required, visualizations are often powerful.
- Examples:
  - Flamegraphs for on-CPU Analysis.
  - Heatmaps and/or Histograms for multi-modal latency data.
  - Simple 2D line graphs for high level application metrics.
- Thus, visualizations are *kind of* required…
  - Analysis is prohibitively difficult without them.

Performance testing overview.

- Generally, performance testing takes the following approach:
    1. Run some (usually known) workload.
    2. Collect application and/or system metrics in some "random" format.
        - The format depends on the metric being collect.
        - Different metrics output data in different formats.
    3. Consume metric data with a tool to generate visualizations.
    4. Analyze raw data and/or visualizations to form conclusions.
        - Analysis must be easy to share...
        - So it can be scrutinized by others.
    5. Learn, Refine, Repeat.

Always use the right tool for the job.

- Without proper tooling, any of the prior steps:
  - can become tedious.
  - can be done incorrectly (and lead to incorrect conclusions).
  - can be insufficiently documented.
- Without proper documentation:
  - mistakes can go unrecognized.
  - methods cannot be shared.
  - analysis cannot be scrutinized.
  - conclusions can be forgotten.
  - results cannot be reproduced.
- The "right tool" must enable solutions to these complications.

This must be a solved problem... right?

- Rather than re-invent the wheel, lets learn from my co-workers.
- What tools were used for past performance related work?
  - Excel
  - Google Spreadsheets
- How was data transferred into the spreadsheet?
  - CSV file
  - Copy/Paste
- Everything done in an ad-hoc basis, specific to each project.
  - Workload chosen by developer, using tools familiar to them.
  - Usually, all steps in the process undocumented (often forgotten).

OK, Google Spreadsheets it is... take one.

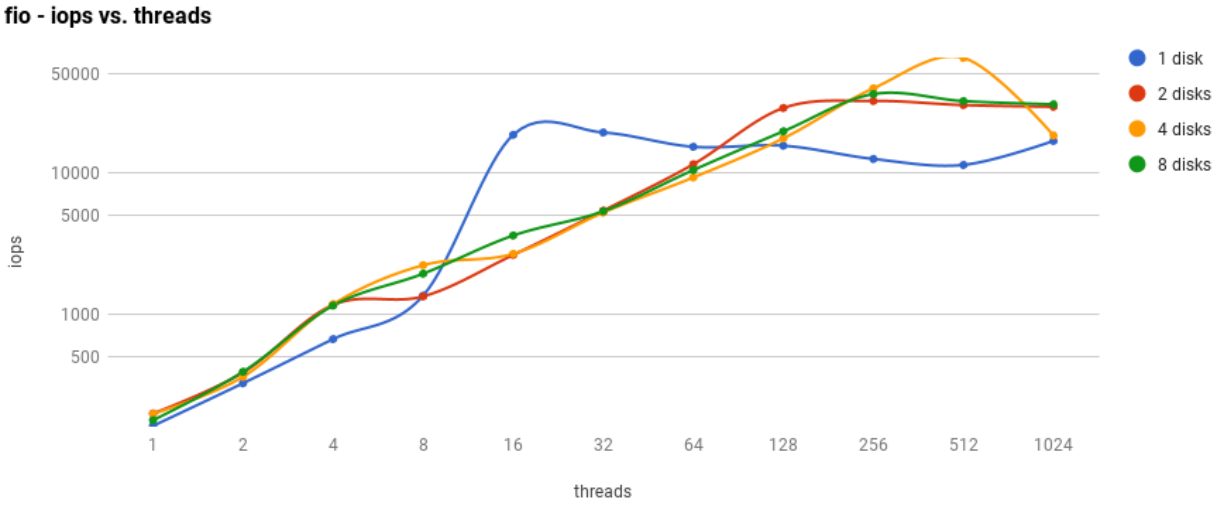
- "fio" was used; it output metrics about the IO it performs:

```
write: io=4171.3MB, bw=70549KB/s, iops=8818, runt= 60544msec
  clat (usec): min=680, max=2260.4K, avg=115400.25, stdev=214661.67
   lat (usec): min=681, max=2260.4K, avg=115401.19, stdev=214661.63
```

- 40 unique test configurations:
  - fio run with 10 different thread counts; 1 to 1024 threads.
  - zpool used with 4 different disk counts; 1 to 8 disks.
- 80 tests in total; 40 **with** my changes, 40 **without** my changes.
- For each test, I would manually copy/paste fio data into spreadsheet.
- Graphs generated from the data was nice...
- Inputting data into the spreadsheet was terrible.

End result.

iops	1	2	4	8	16	32	...
1 disk	164.79	328.89	672.61	1361.03	18414.90	19130.33	...
2 disks	201.11	390.75	1171.71	1342.84	2630.02	5389.89	...
4 disks	200.31	364.79	1184.56	2228.02	2677.17	5223.90	...
8 disks	180.57	395.95	1158.49	1940.64	3602.46	5340.03	...



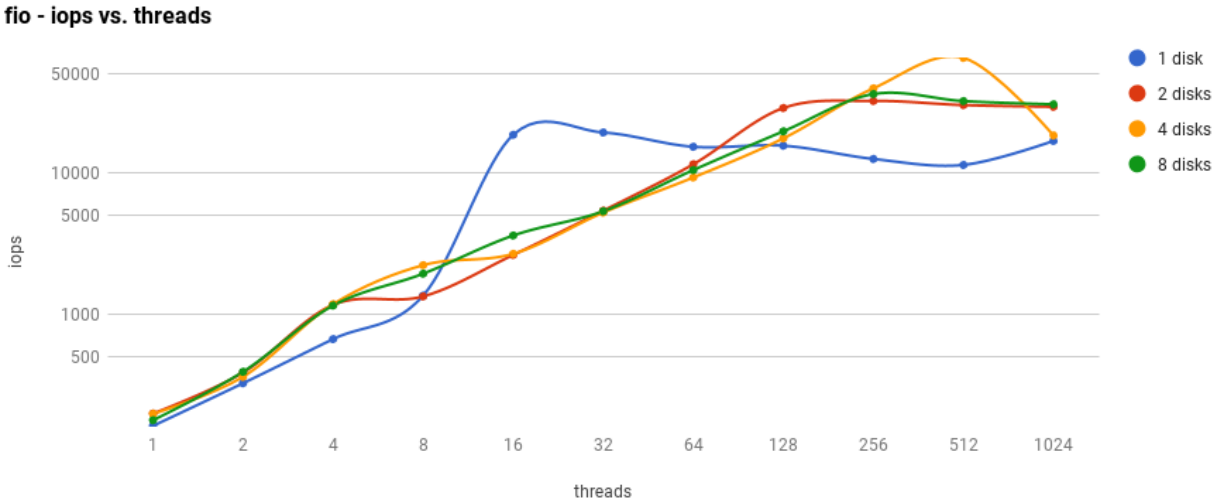
- Would use a meeting to discuss results, share analysis, etc.

Google Spreadsheets; take two.

- "Take one" was lame... So, I started looking for ways to improve it.
- Discovered that fio can output JSON data using "--output-format".
- Maybe that, combined with "jq" and some Bash would help?
- Wrote a Bash script to:
  - iterate over all fio JSON output files
  - use jq to parse IOPs data for each test iteration
  - output CSV (to stdout) for all 40 test configurations
- I would then manually copy/paste CSV data into spreadsheet.
- Using a CSV file rather than copy/paste isn't much different.

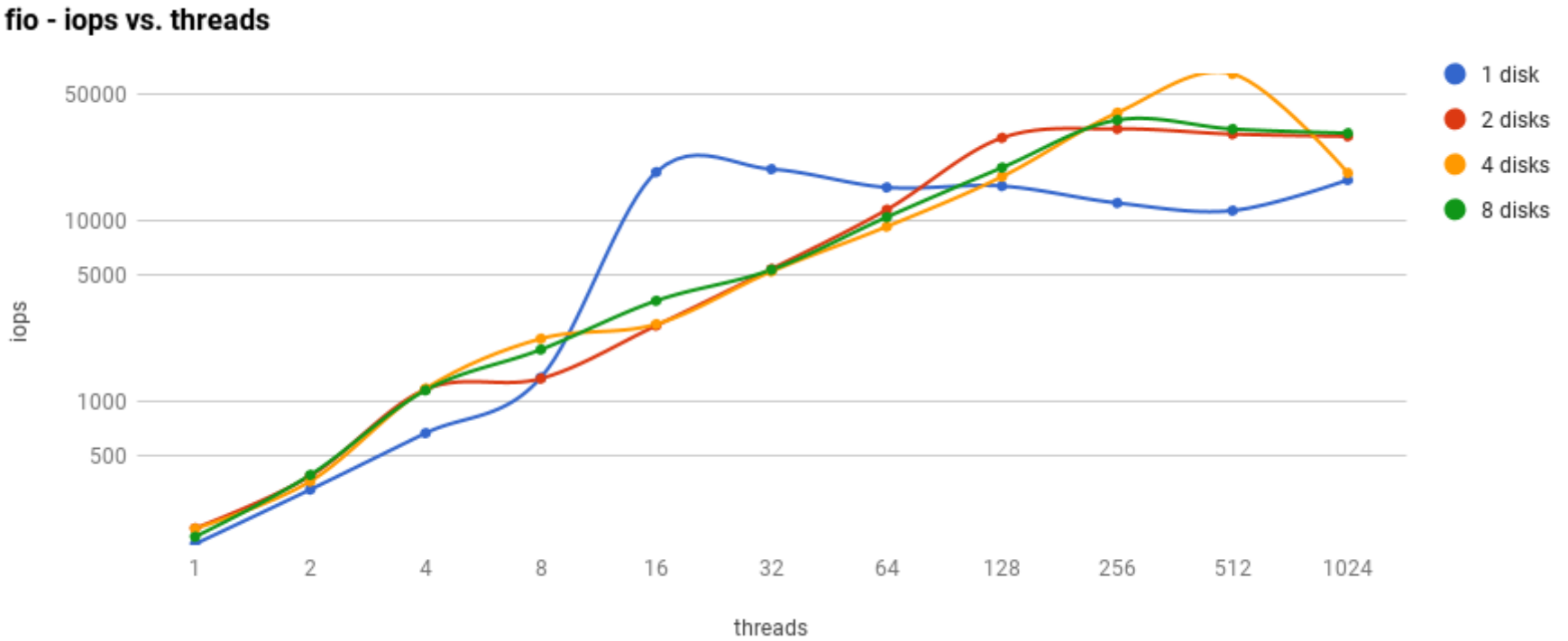
Same result; easier to generate.

iops	1	2	4	8	16	32	...
1 disk	164.79	328.89	672.61	1361.03	18414.90	19130.33	...
2 disks	201.11	390.75	1171.71	1342.84	2630.02	5389.89	...
4 disks	200.31	364.79	1184.56	2228.02	2677.17	5223.90	...
8 disks	180.57	395.95	1158.49	1940.64	3602.46	5340.03	...



- Still no supporting documentation to explain process or results.

Huh.. The blue line looks different.. Why?

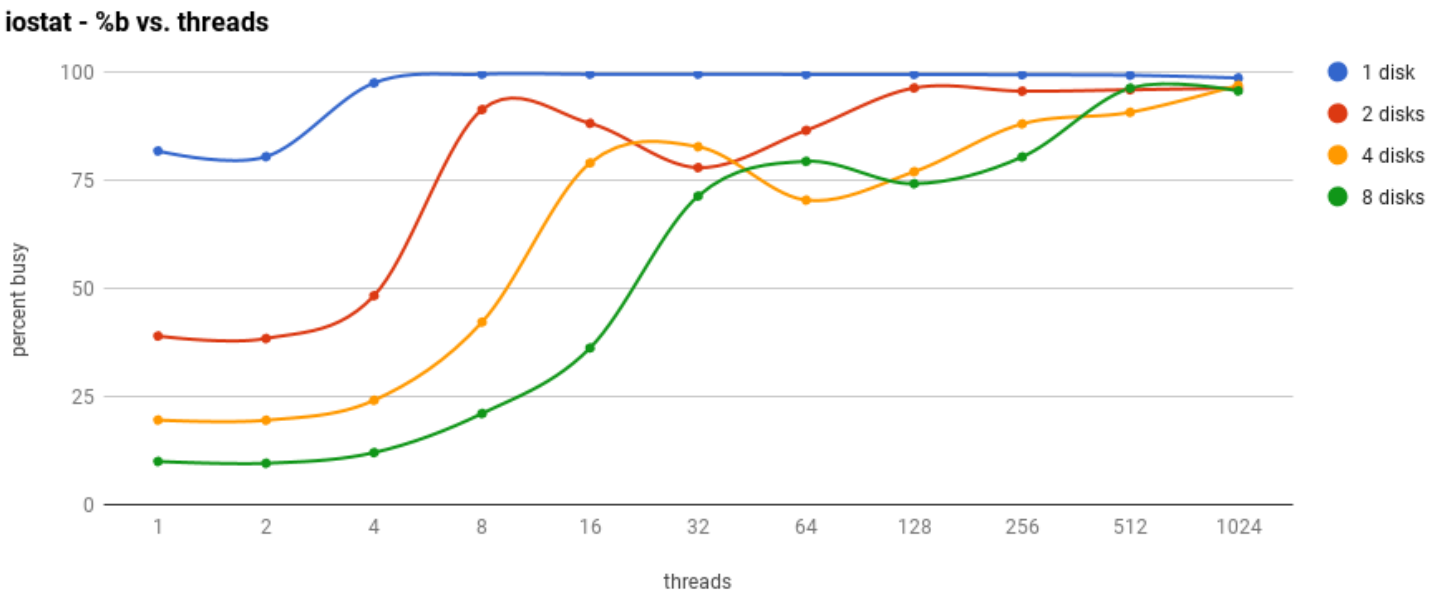


Application metrics is not sufficient.

- Started looking at data from "iostat"
- Used a script to:
  - log "iostat" output to a file for each test configuration…
  - then parse the output files for each configuration…
  - then output CSV to standard output.
- CSV data would be manually copied into the spreadsheet
- Now, there were copy/pasted tables (and graphs) for:
  - fio IOPs vs. fio threads
  - iostat "%b", "actv", "asvc_t", and "w/s" (each vs. fio threads)
- Starting to encroach on original problem; too much copy/paste.

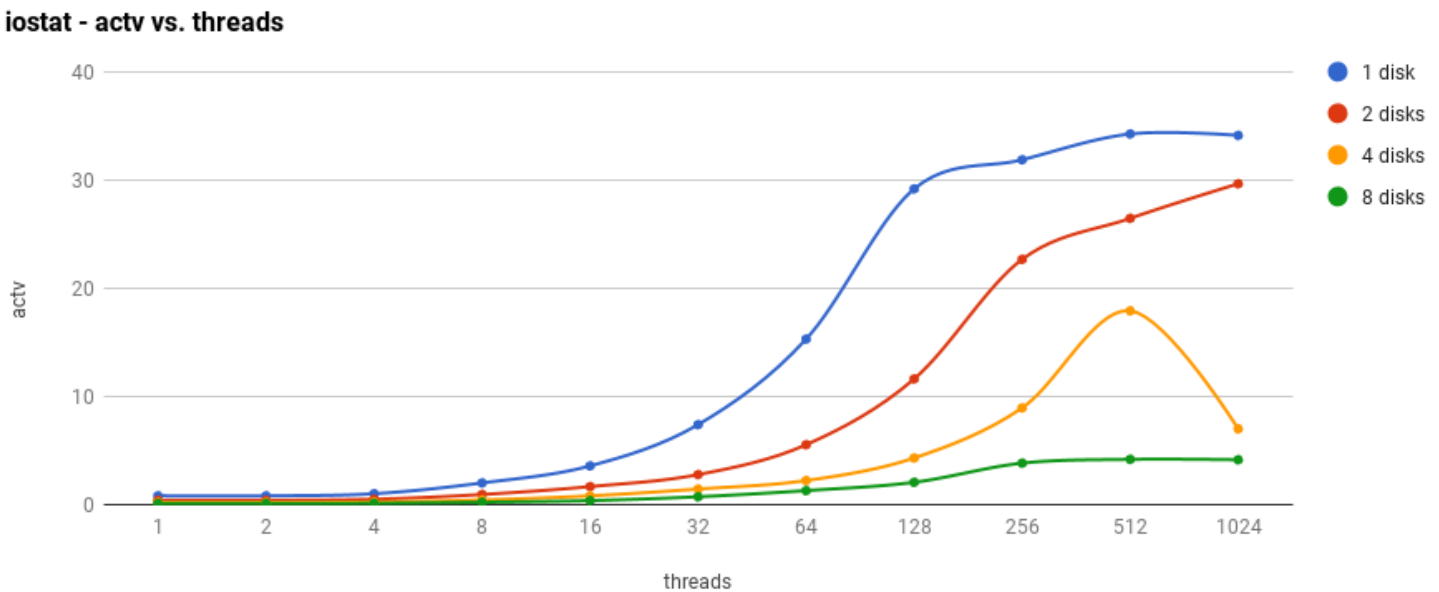
iostat's %b vs. fio threads

%b	1	2	4	8	16	32	...
1 disk	81.68	80.37	97.40	99.42	99.42	99.42	...
2 disks	38.93	38.36	48.25	91.25	88.07	77.83	...
4 disks	19.52	19.47	24.08	42.15	78.91	82.66	...
8 disks	9.96	9.54	12.00	21.02	36.17	71.26	...



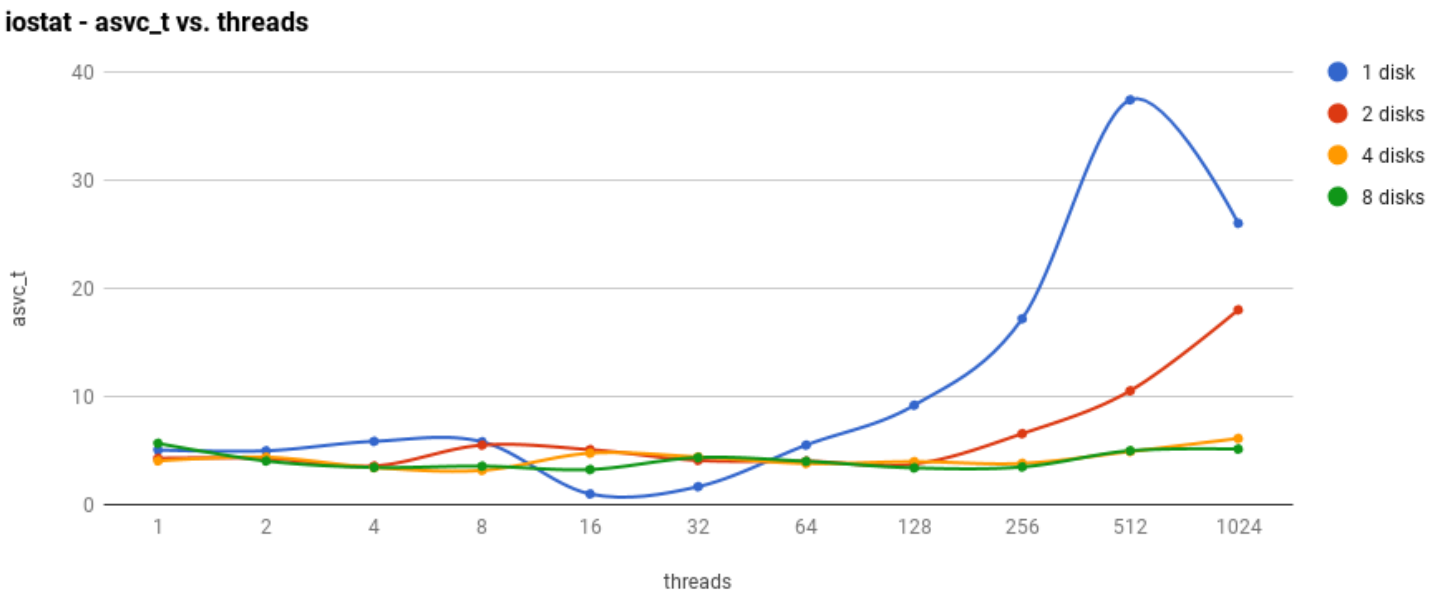
iostat's actv vs. fio threads

actv	1	2	4	8	16	32	...
1 disk	0.81	0.81	1.00	2.00	3.57	7.39	...
2 disks	0.40	0.39	0.49	0.93	1.66	2.77	...
4 disks	0.19	0.20	0.24	0.42	0.80	1.43	...
8 disks	0.10	0.10	0.12	0.21	0.36	0.72	...



iostat's asvc_t vs. fio threads

asvc_t	1	2	4	8	16	32	...
1 disk	5.04	4.97	5.84	5.80	0.98	1.65	...
2 disks	4.27	4.26	3.56	5.49	5.07	4.08	...
4 disks	4.03	4.38	3.42	3.15	4.75	4.40	...
8 disks	5.65	4.02	3.43	3.55	3.23	4.34	...



Spreadsheets: decent, but far from ideal.

- Pros:
  - No setup required.
  - Visualizations helped understand data.
- Cons:
  - Data input is manual and error prone.
  - Available visualizations can be limiting; e.g. flamegraphs?
  - Code is seperate from data/visualizations.
  - No way to review code/data for correctness.[1]
  - No way to annotate data/visualizations with explanations.

[1] http://www.nytimes.com/2013/04/19/opinion/krugman-the-excel-depression.html

Is there a better way?

- Spent some time googling around for different ideas/approaches.
- Discovered Jupyter and Jupyter Notebooks.
- With Jupyter, I am able to:
  - Generate complex visualizations using Python libraries
  - Perform data analysis using Python libraries
  - Embed text/explanations inline with visualizations
  - Embed python data analysis code directly in the notebook

What is Jupyter?

- Excerpt taken from "What is Jupyter?" article[1]

> ...
>
> But without attracting the hype, Jupyter Notebooks are revolutionizing
> the way engineers and data scientists work together. If all important
> work is collaborative, the most important tools we have are tools for
> collaboration, tools that make working together more productive.
>
> That's what Jupyter is, in a nutshell: it's a tool for collaborating.
> It’s built for writing and sharing code and text, within the context of
> a web page.
>
> ...
>
> Code is never just code. It's part of a thought process, an argument,
> even an experiment. This is particularly true for data analysis, but
> it's true for almost any application. Jupyter lets you build a "lab
> notebook" that shows your work: the code, the data, the results, along
> with your explanation and reasoning.
>
> ...

[1] https://www.oreilly.com/ideas/what-is-jupyter

Basic example 1: Visualizaing fio IOPs

- Link to notebook
- Link to nbviewer

Basic example 2: Visualizaing iostat

- Link to notebook
- Link to nbviewer

Real example: Results for OpenZFS #447

- Link to Max Rate Submit on HDDs
- Link to Max Rate Submit on SSDs
- Link to Fixed Rate Submit on HDDs
- Link to Fixed Rate Submit on HDDs

*https://www.prakashsurya.com/post/2017-09-08-performance-testing-results-for-openzfs-447/

My Jupyter Tips

- Use relavant Python libraries; e.g. Pandas, Matplotlib, etc.
- Format output data to allow easier ingestion.
  - e.g. use `pandas.read_csv` rather than custom parsing.
- Use `pandas.DataFrame`; makes data analysis and graphing easy:
  - `df.plot()` to plot data.
  - `df3 = df1 - df2` to determine the difference
- `seaborn` library can help make graphs subjectively "prettier".
  - As simple as `import seaborn` to change defaults

How can YOU use Jupyter?

- Official documentation: Jupyter Notebook Quickstart
- My notes: Using Python and Jupyter for Performance Testing and Analysis
- The "`jupyter`" DCenter image is Ubuntu 17.04 with Jupyter pre-installed.
    - No LDAP; log in using `delphix` user and run `jupyter`

```
$ jupyter notebook --ip=0.0.0.0

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://0.0.0.0:8888/?token=431434aa3c192dd33613c4bff990e4207a3af5e402f48012
```

How can WE use Jupyter?

- For notebooks that we don't want publically accessible:
  - Create an internal Jupyter service; e.g. `jupyter.delphix.com`
  - Create an internal nbviewer service; e.g. `nbviewer.delphix.com`
- Taking this a step further: Scaling Knowledge at Airbnb
  - Teaching debugging techniques
  - Sharing novel, intesting, and/or complicated RCA of bugs
  - "Marketing" what one is working on to peers/organization
  - Disseminating random, but useful, TIL stories
- Would require a cultural shift, to adopt Jupyter effectively.

End