Reproducible and Automated Storage systems experimentation with Popper

Jayjeet Chakraborty
Mentored by Carlos Maltzahn, Ivo Jimenez, Jeff LeFevre

UC Santa Cruz

CENTER FOR RESEARCH IN
OPEN SOURCE SOFTWARE

UNIVERSITY OF CALIFORNIA
SANTA CRUZ

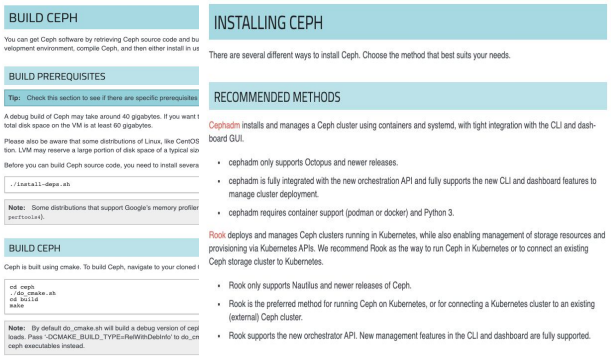
Problems in doing Systems experiments

CloudLab

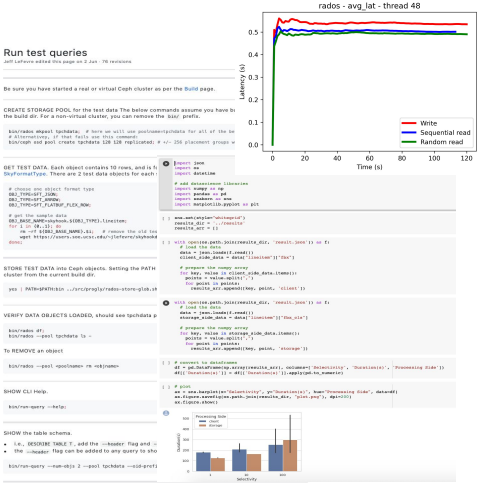
Google Cloud Platform

aws

Boot VMs or bare metal Nodes



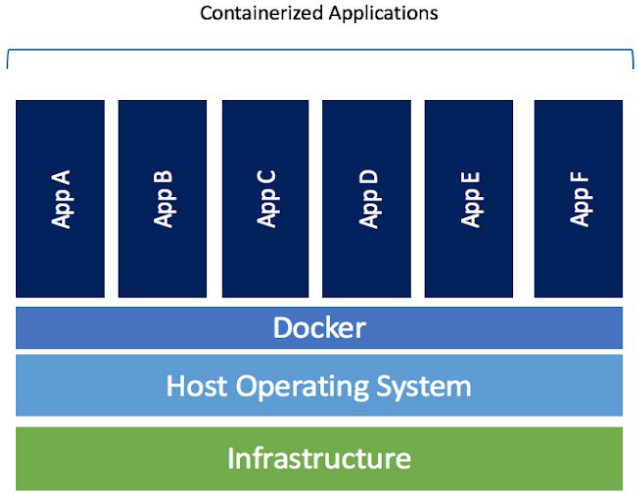
Build, Deploy and Run experiments

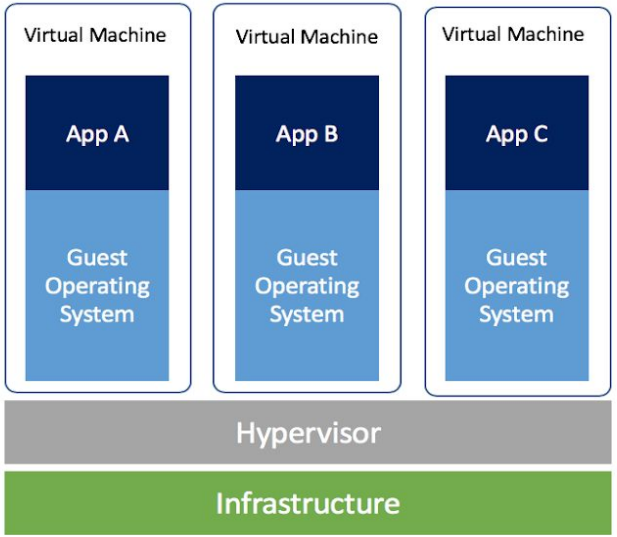


Prepare plots and notebooks

Should be platform independent and automated ! Otherwise time consuming and error-prone !

Overview of Containers





- Less resource usage than VMs
- Platform independent and portable software
- Consistent operation across environments
- Greater efficiency

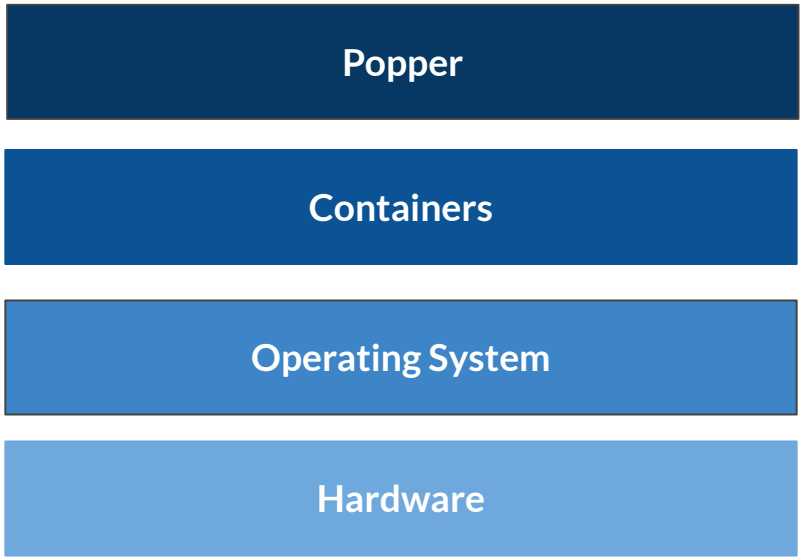
Containerizing Commands

```
$ docker run -e BLOCKDEVICE=sdb
             -e IODEPTH=32
             -v $PWD:/workspace
             --rm
             --entrypoint /bin/bash
             -w /workspace
             bitnami/kubectl:1.17.4
             ./run_benchmarks.sh
```

Solves platform dependency.

But still lacks automation !

What is Popper ?



Popper

docker

podman

circleci

Popper

```yaml
steps:
- id: install lulesh
  uses: popperized/spack@master
  args: [spack, install, -j8, lulesh+mpi]

- id: delete existing jobs
  uses: popperized/bin/sh@master
  args: [rm, -fr, sweep/jobs]

- id: install sweepj2
  uses: popperized/python-actions@master
  args: [pip, install, sweepj2]

- id: generate sweep
  uses: jefftriplett/python-actions@master
  args: [
    "sweepj2",
    "--template", "./sweep/script.j2",
    "--space", "./sweep/space.yml",
    "--output", "./sweep/jobs/",
    "--make-executable"
  ]

- id: run sweep
  uses: popperized/spack@master
  args: [run-parts, ./sweep/jobs]
```

Slide borrowed from Ivo Jimenez

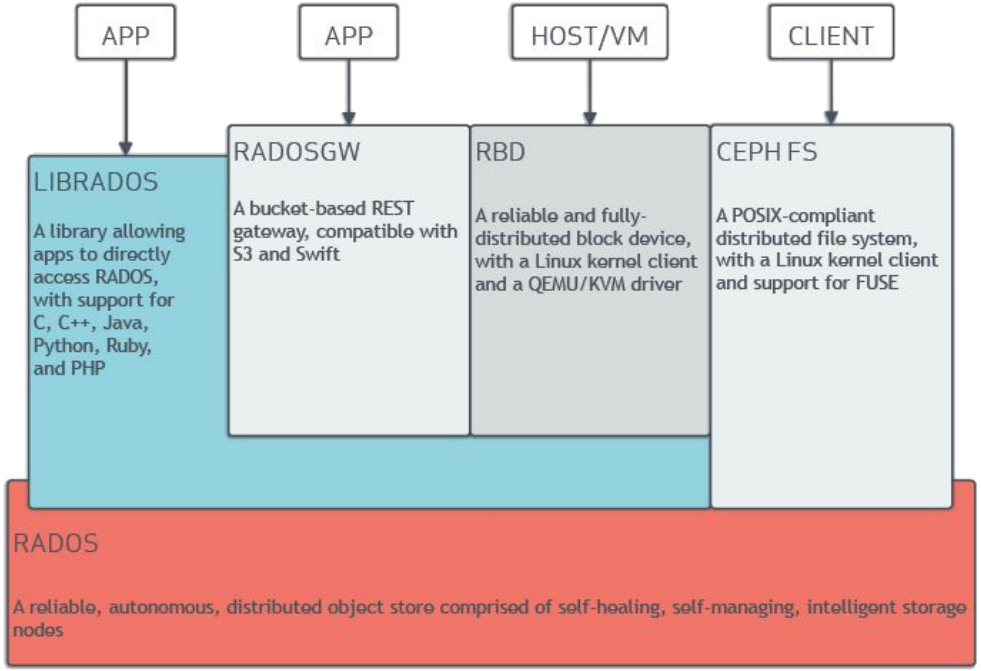
Reproducible and Scalable Ceph and SkyhookDM experimentation with Popper

Ceph

1. Provides 3 types of storage interface: File, Object, Block

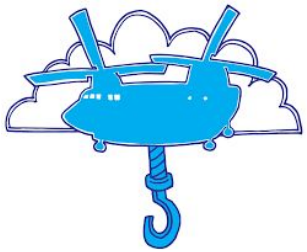
2. No central point of failure. Uses CRUSH maps that contains object - OSD mapping. A CRUSH map in each client. Client talks directly to OSD.

3. Highly extensible through plugins.



SkyhookDM

- Extends the Ceph object store using custom C++ Object classes for data management in the Storage layer.
- Allows push down of operations like SELECT, PROJECT, AGGREGATE to the storage layer.
- Supports querying both row oriented and column oriented data.

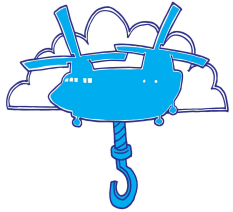


High-Level Workflow for SkyhookDM Experiments

Spawn nodes and deploy k8s. Get the config.

Baseline k8s cluster. Measure Disk I/O, Network latency, etc.

Deploy Ceph and benchmark the object store.

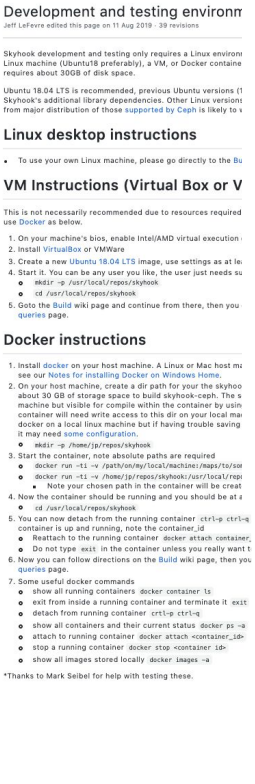


Benchmark throughput by running queries on hundreds of GBs of tabular data.

Study the Jupyter notebooks, Plots and Grafana snaps.

Should be platform independent and automated !

“*Popperizing*” the SkyhookDM Experimentation Workflow


Development and testing environm

Jeff LeFevre edited this page on 11 Aug 2019 · 39 revisions

Skyhook development and testing only requires a Linux environ
Linux machine (Ubuntu18 preferably), a VM, or Docker containe
requires about 30GB of disk space.

Ubuntu 18.04 LTS is recommended, previous Ubuntu versions (1
Skyhook's additional library dependencies. Other Linux versions
from major distribution of those supported by Ceph is likely to

Linux desktop instructions

- To use your own Linux machine, please go directly to the Bu

VM Instructions (Virtual Box or V

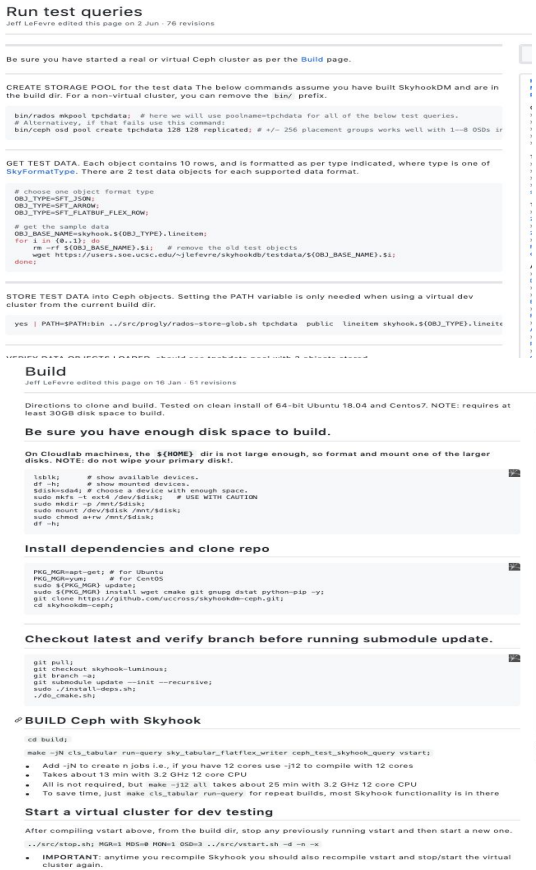
This is not necessarily recommended due to resources required
use Docker as below.

1. On your machine's bios, enable Intel/AMD virtual execution
2. Install VirtualBox or VMWare
3. Create a new Ubuntu 18.04 LTS image, use settings as at le
4. Start it. You can be any user you like, the user just needs su
   - mkdir ~p /usr/local/repos/skyhook
   - cd /usr/local/repos/skyhook
5. Goto the Build wiki page and continue from there, then you
   queries page.

Docker instructions

1. Install docker on your host machine. A Linux or Mac host ma
   see our Notes for installing Docker on Windows Home.
2. On your host machine, create a dir path for your the skyhoo
   about 30 GB of storage space to build skyhook-ceph. The s
   machine but visible for compile within the container by usin
   container will need write access to this dir on your local ma
   docker on a local linux machine but if having trouble saving
   it may need some configuration.
   - mkdir ~p /home/jp/repos/skyhook
3. Start the container, note absolute paths are required
   - docker run ~ti ~v /path/on/my/local/machine:/maps/to/so
   - docker run ~ti ~v /home/jp/repos/skyhook:/usr/local/repo
     - Note your chosen path in the container will be creat
4. Now the container should be running and you should be at a
   - cd /usr/local/repos/skyhook
5. You can now detach from the running container ctrl~p ctrl~q
   container is up and running, note the container_id
   - Reattach to the running container docker attach container_
   - Do not type exit in the container unless you really want t
6. Now you can follow directions on the Build wiki page, then you
   queries page.
7. Some useful docker commands
   - show all running containers docker container ls
   - exit from inside a running container and terminate it exit
   - detach from running container ctrl~p ctrl~q
   - show all containers and their current status docker ps ~a
   - attach to running container docker attach <container_id>
   - stop a running container docker stop <container id>
   - show all images stored locally docker images ~a

*Thanks to Mark Seibel for help with testing these.


Run test queries

Jeff LeFevre edited this page on 2 Jun · 76 revisions

Be sure you have started a real or virtual Ceph cluster as per the Build page.

CREATE STORAGE POOL for the test data The below commands assume you have built SkyhookDM and are in the build dir. For a non-virtual cluster, you can remove the bin/ prefix.

```
bin/rados mkpool tpchdata;  # here we will use poolname=tpchdata for all of the below test queries.
# Alternatively, if that fails use this command:
bin/ceph osd pool create tpchdata 128 128 replicated; # +/- 256 placement groups works well with 1~8 OSDs ir
```

GET TEST DATA. Each object contains 10 rows, and is formatted as per type indicated, where type is one of SkyFormatType. There are 2 test data objects for each supported data format.

```
# choose one object format type
OBJ_TYPE=SFT_JSON;
OBJ_TYPE=SFT_ARROW;
OBJ_TYPE=SFT_FLATBUF_FLEX_ROW;

# get the sample data
OBJ_BASE_NAME=skyhook.${OBJ_TYPE}.lineitem;
for i in {0..1}; do
    rm ~rf ${OBJ_BASE_NAME}.$i;   # remove the old test objects
    wget https://users.soe.ucsc.edu/~jlefevre/skyhookdb/testdata/${OBJ_BASE_NAME}.$i;
done;
```

STORE TEST DATA into Ceph objects. Setting the PATH variable is only needed when using a virtual dev cluster from the current build dir.

```
yes | PATH=$PATH:bin ../src/progly/rados~store~glob.sh tpchdata  public  lineitem skyhook.${OBJ_TYPE}.lineit
```

Build

Jeff LeFevre edited this page on 16 Jan · 61 revisions

Directions to clone and build. Tested on clean install of 64~bit Ubuntu 18.04 and Centos7. NOTE: requires at least 30GB disk space to build.

Be sure you have enough disk space to build.

On Cloudlab machines, the ${HOME} dir is not large enough, so format and mount one of the larger disks. NOTE: do not wipe your primary disk!.

```
lsblk;      # show available devices.
df ~h;      # show mounted devices.
$disk=sda4; # choose a device with enough space.
sudo mkfs ~t ext4 /dev/$disk;   # USE WITH CAUTION
sudo mkdir ~p /mnt/$disk;
sudo mount /dev/$disk /mnt/$disk;
sudo chmod a+rw /mnt/$disk;
df ~h;
```

Install dependencies and clone repo

```
PKG_MGR=apt~get; # for Ubuntu
PKG_MGR=yum;     # for CentOS
sudo ${PKG_MGR} update;
sudo ${PKG_MGR} install wget cmake git gnupg dstat python~pip ~y;
git clone https://github.com/uccross/skyhookdm~ceph.git;
cd skyhookdm~ceph;
```

Checkout latest and verify branch before running submodule update.

```
git pull;
git checkout skyhook~luminous;
git branch ~a;
git submodule update ~~init ~~recursive;
sudo ./install~deps.sh;
./do_cmake.sh;
```

BUILD Ceph with Skyhook

```
cd build;
make ~jN cls_tabular run~query sky_tabular_flatflex_writer ceph_test_skyhook_query vstart;
```

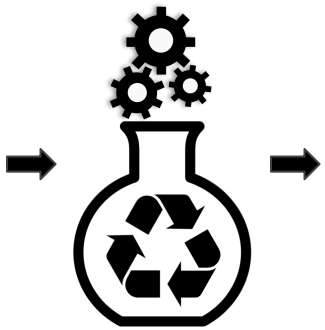
- Add ~jN to create n jobs i.e., if you have 12 cores use ~j12 to compile with 12 cores
- Takes about 13 min with 3.2 GHz 12 core CPU
- All is not required, but `make ~j12 all` takes about 25 min with 3.2 GHz 12 core CPU
- To save time, just `make cls_tabular run~query` for repeat builds, most Skyhook functionality is in there

Start a virtual cluster for dev testing

After compiling vstart above, from the build dir, stop any previously running vstart and then start a new one.

```
../src/stop.sh; MGR=1 MDS=0 MON=1 OSD=3 ../src/vstart.sh ~d ~n ~x
```

- **IMPORTANT**: anytime you recompile Skyhook you should also recompile vstart and stop/start the virtual cluster again.

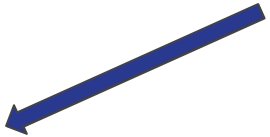


Building and Deploying Ceph on Bare metal

Building manually vs Building with Popper

```
$ git pull;
$ git checkout luminous;
$ git branch -a;
$ git submodule update --init
--recursive;
$ sudo ./install-deps.sh;
$ ./do_cmake.sh;
```

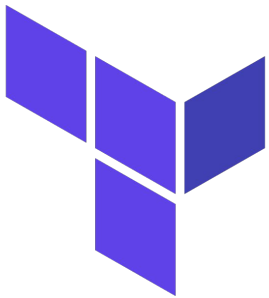
```
$ popper run dev-init

$ popper run build
```



```
- id: dev-init
  uses: docker://alpine/git:v2.24.3
  runs: [sh, -ec]
  args:
  - |
    git clone \
      --recursive \
      --depth 1 \
      --shallow-submodules \
      --branch nautilus \
      https://github.com/ceph/ceph
    ln -s ../../../src ceph/src/cls/tabular
    echo "add_subdirectory(tabular)" >> ceph/src/cls/CMakeLists.txt

- id: build
  uses: docker://uccross/skyhookdm-builder:nautilus
  runs: [bash]
  args: [scripts/build.sh]
  env:
    # travis config: 4 threads, release build
    CMAKE_FLAGS: "-DBOOST_J=4 -DCMAKE_BUILD_TYPE=Release -DWITH_MANPAGE=OFF
    BUILD_THREADS: "4"
```

Pack all these steps in a Popper workflow and let Popper handle the repetitive and error-prone work

Deploying Ceph manually



ceph-deploy -- Deploy Ceph with minimal infrastructure

`ceph-deploy` is a way to deploy Ceph relying on just SSH access to the servers, `sudo`, and some Python. It runs fully on your workstation, requiring no servers, databases, or anything like that.

- *Learn writing and managing multiple playbooks, vars files, commands in READMEs, host files, etc. Overwhelming for new Ceph developers.*
- *Require knowledge about and using a series of different DevOps tools. Makes the entry barrier high.*

Deploying Ceph using Popper

Popper workflows abstract away tools like Ansible, Terraform, etc. Put all the scripts , playbooks , configuration in a repository and orchestrate with Popper.

```
popper run -f wf.yml
```

A single command to deploy Ceph while using your favourite DevOps tools !

```
.
├── README.md
├── ansible
│   ├── deploy-libcls_tabular.yml
│   ├── deploy-mons-and-osds.yml
│   ├── files
│   ├── group_vars
│   │   └── all.yml
│   └── purge_cluster.yml
├── geni
│   ├── cloudlab_cmd.py
│   └── config.py
└── wf.yml
```

Building and Deploying on Kubernetes via Rook

Even Easier !

Why Kubernetes ? Why Rook ?

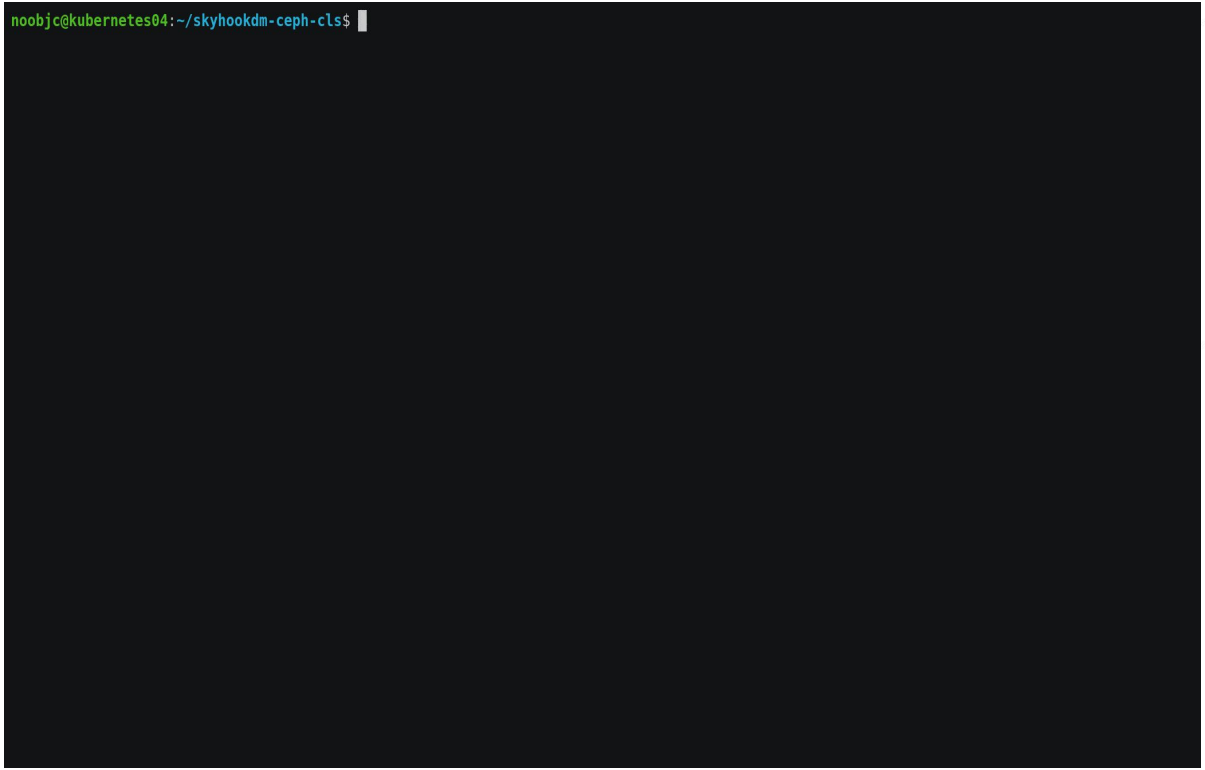
Turns storage software into self-managing, self-scaling, and self-healing storage services.

Uses the facilities provided by Kubernetes. Provisioning, scaling, upgrading, disaster recovery.

Make experimentation container native.

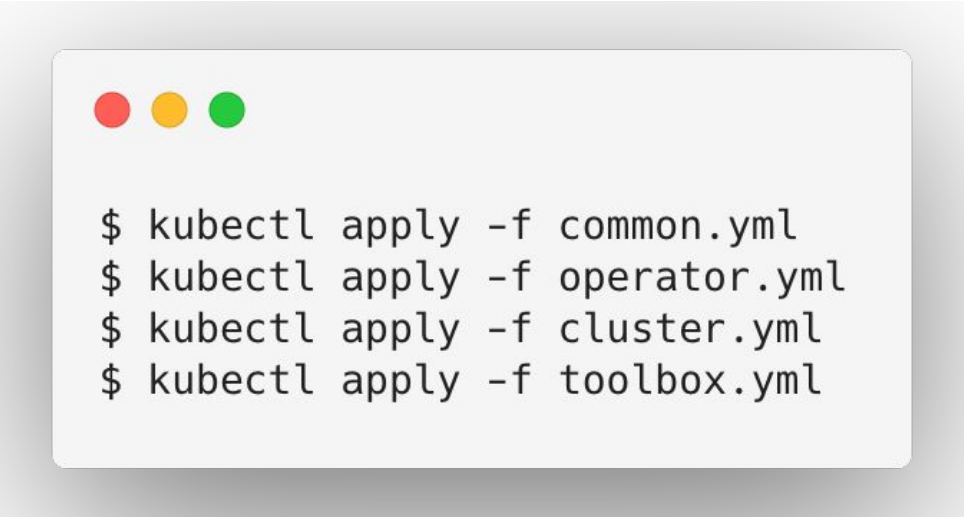
Allow using tools from the Kubernetes ecosystem. Get an operator for everything.

Building Ceph and SkyhookDM inside Docker




```
- id: build
  uses: docker://uccross/skyhookdm-builder:nautilus
  runs: [bash]
  args: [scripts/build.sh]
  env:
    # travis config: 4 threads, release build
    CMAKE_FLAGS: "-DBOOST_J=4 -DCMAKE_BUILD_TYPE=Release
    BUILD_THREADS: "4"

- id: build-rook-img
  uses: docker://docker:19.03.10
  args:
  - build
  -   --build-arg=CEPH_RELEASE=v14.2.9
  -   --tag=uccross/skyhookdm-ceph:v14.2.9
  -   --file=docker/Dockerfile.release
  -   .
```

Deploying Ceph on Kubernetes with Rook



Select the correct config files, install **`kubectl`**, run the **`kubectl apply`**'s in proper order.

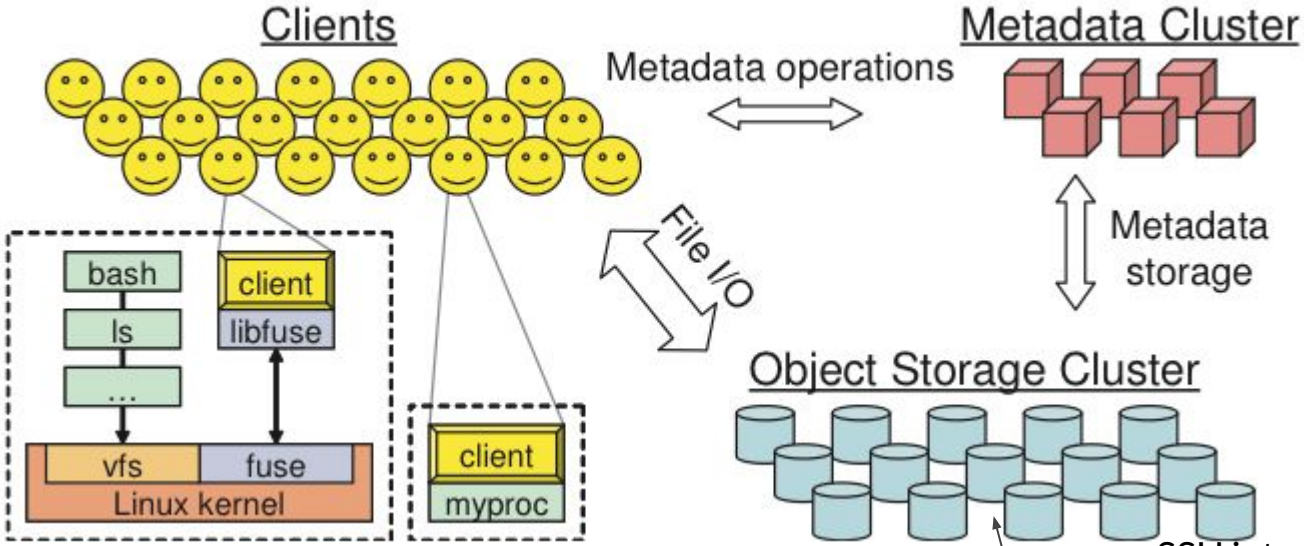
More time spent with Rook that is not really needed !

Making Rook Reproducible and Automated

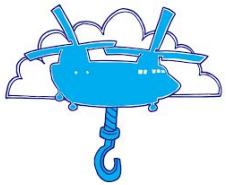
Clone the repository and **`$ popper run -f rook.yml setup-ceph`**

```
- id: setup-ceph
  uses: docker://bitnami/kubectl:1.17.4
  runs: [bash, -euc]
  args:
  - |
    kubectl apply -f ./rook/common.yaml
    kubectl apply -f ./rook/operator.yaml
    kubectl apply -f ./rook/cluster_ceph.yaml
    kubectl apply -f ./rook/toolbox.yaml
```

Easy upgrades with Rook than Bare metal



SSH into each node and upload the shared libraries at `/usr/lib64/rados-classes/`



`cls_tabular.cc` → `libcls_tabular.so`

Using Object class SDK

Upgrading Ceph to SkyhookDM manually

Use an existing Ceph cluster

Assumes Ceph luminous version. Just build and deploy our extensions library as below, there is no need to stop or restart the cluster.

1. Clone and build SkyhookDM as per our Build instructions, requires ~10 GB of disk space.
2. Copy the skyhookdm library file into the libcls directory on each of the OSDs, requires sudo
   - Centos7: `LIB_CLS_DIR=/usr/lib64/rados-classes/`
   - Ubuntu18: `LIB_CLS_DIR=/usr/lib/x86_64-linux-gnu/rados-classes/`

```
# from the BUILD dir:
for ((i = 0 ; i < $nosds ; i++)); do
    echo "copying shared lib to osd$i;"
    scp ./lib/libcls_tabular.so.1.0.0  osd$i:/tmp/;
    ssh  osd$i "sudo cp /tmp/libcls_tabular.so.1.0.0 ${LIB_CLS_DIR};";
    ssh  osd$i "cd ${LIB_CLS_DIR}; if test -f libcls_tabular.so.1; then sudo  unlink libcls_tabular.so.1; fi"
    ssh  osd$i "cd ${LIB_CLS_DIR}; if test -f libcls_tabular.so; then sudo unlink libcls_tabular.so; fi";
    ssh  osd$i "cd ${LIB_CLS_DIR}; sudo ln -s libcls_tabular.so.1.0.0 libcls_tabular.so.1;";
    ssh  osd$i "cd ${LIB_CLS_DIR}; sudo ln -s libcls_tabular.so.1 libcls_tabular.so;";
done;
```

```
osd_class_load_list = *
osd_class_default_list = *
```

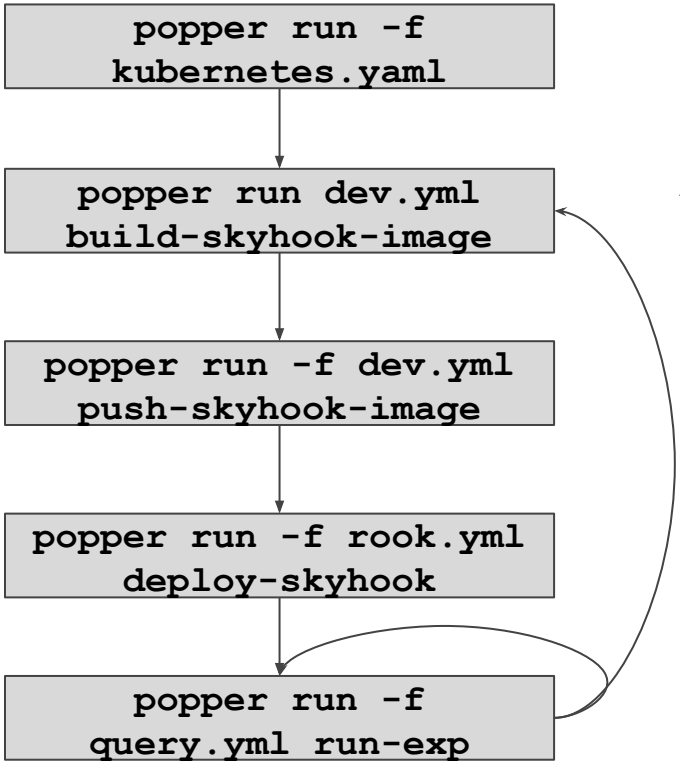
Upgrade the configuration file and SSH into every OSD and copy the libraries at the **`LIB_CLS_DIR`** of Ceph. Restart OSDs.

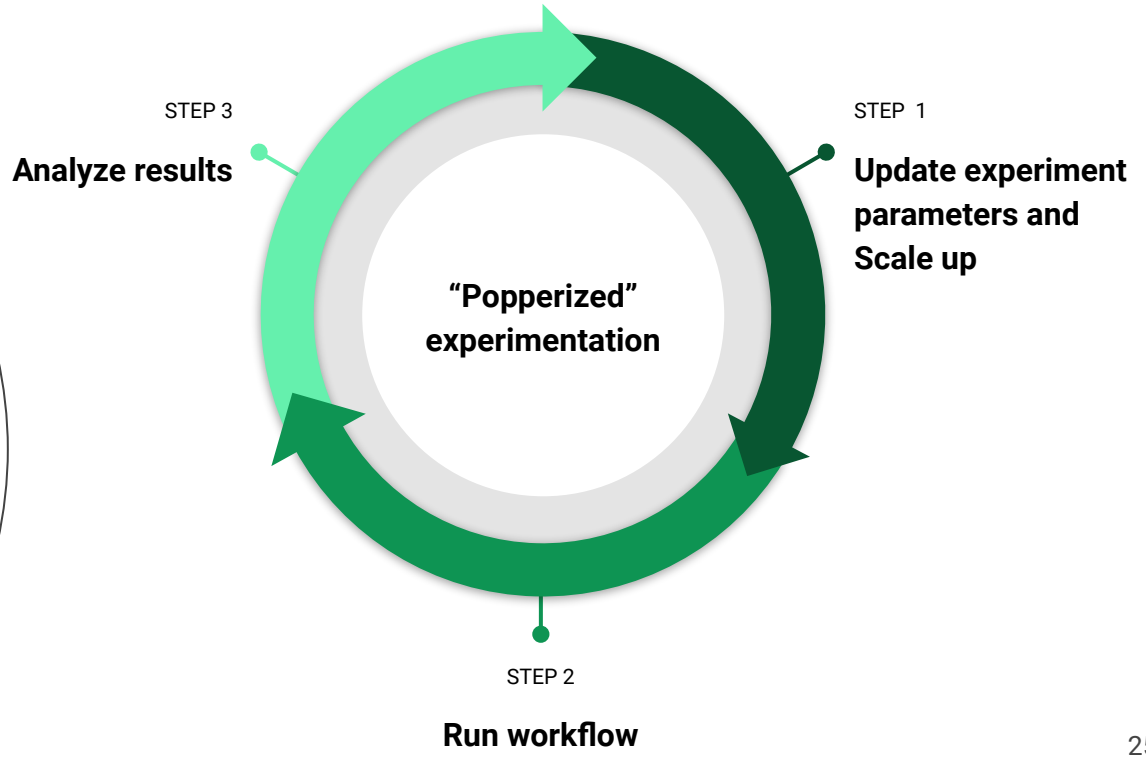
Heavy Manual Work !

Upgrade from Ceph to SkyhookDM on Rook using Popper

```
// deploy vanilla Ceph
$ popper run deploy-ceph

// run a popper step to inject libcls_tabular.so and upgrade to SkyhookDM
$ popper run deploy-skyhook-ceph
```

A Scalable experimentation Loop using Popper





Making Notebooks portable with Docker

```
docker run
   -p 8888:8888
   -v ~/notebooks:/home/jayjeet
   jupyter/tensorflow-notebook
   "nbconvert --execute
   --to=notebook
   ./run_query/notebook/plot.ipynb"
```

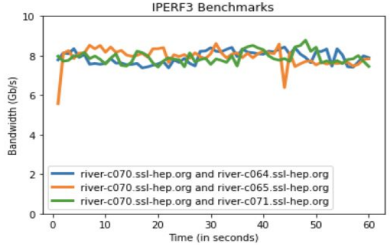
```
popper run plot-results
```

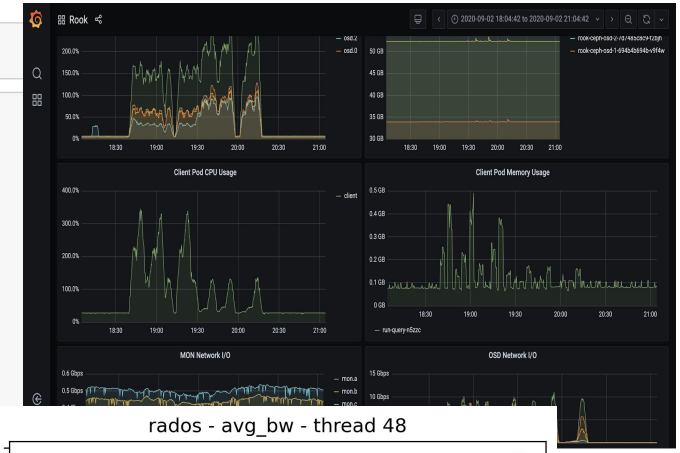
Let Popper take care of the error-prone components !

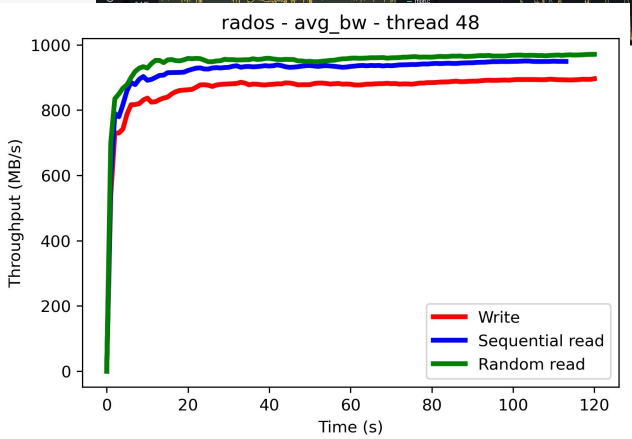
Notebooks , Grafana dashboards and Plots

In [1]:
```
import os
import json
import matplotlib.pyplot as plt
```

In [2]:
```
results_dir = '../results'
files = os.listdir(results_dir)

# iterate over each result file and plot the results
for file in files:
    if not file.endswith('.json'): continue
    with open(os.path.join(results_dir, file)) as f:
        results = json.load(f)

    seconds = []
    bandwidth = []

    for run in results["intervals"]:
        seconds.append(float(run["sum"]["end"]))
        bandwidth.append(float(run["sum"]["bits_per_second"])/(1000*1000*1000))

    plt.plot(seconds, bandwidth, markersize=10, linewidth=3.0, label=f"{os.environ['SERVER']} and {file[:-5]}")


# pin the range between 0 and 10
plt.ylim(0.0, 10.0)
plt.xlabel('Time (in seconds)')
plt.ylabel('Bandwidth (Gb/s)')
plt.title('IPERF3 Benchmarks')
plt.legend()
plt.savefig(os.path.join(results_dir, './iperf-benchmarks.png'), dpi=300, bbox_inches='tight')
plt.show()
```







Thank you !

Questions ?

 @heyjc25

The code is available here.