Reproducible Performance Evaluation Pipelines for Ceph

Mariette Souppe
UC Santa Cruz
msouppe@ucsc.edu

***Abstract*—Ceph, a free-software distributed system, is commonly studied and practiced among students with a background or interest in Computer Science. Setting up any distributed system is not a straightforward task and Ceph is no exception. This paper provides a reproducible automated setup of a Ceph distributed system with minimal configuring to first time users of Ceph or an experienced user on a macOS or Linux environment for end to end testing.**

I. Introduction

A. Background

Setting up a distributed system like Ceph is hard and complex because there are many different components that make up Ceph. The goal of this project is implementing a workflow for Ceph experimentation. Instead of going the traditional route of manually installing everything in ad-hoc ways, a SciOps (DevOps for science) methodology is followed to implement experimentation workflows for Ceph. This results in having reproducible studies, transparency (openscience), experiments that are amenable to collaboration and extension. In summary, we apply the SciOps methodology to Ceph experimentation that is typical of storage and data-management R&D settings (e.g think of companies such as Toshiba, WD, Samsung, etc.). In this report we describe what we did.

Anecdotally, when students take a course in distributed systems and need to setup a distributed environment, it can take several weeks *just* to get the system up and running. If a student were in a ten week class, that does not give students enough time to play around with the system and report interesting observations or results since time is very limited. Reducing the overhead of the setup time will allow for more time working on a project versus spending time getting the system up and running. The workflow that has been implemented will allow for a variety of testing, but this paper one benchmarking test will be performed for a proof of concept basis. Some of these tests which can be performed within Ceph include performance, scalability, correctness, availability, and overhead. The contributions of this project include:

- Applying SciOps methodology to implement a Ceph experimentation workflow;
- A template to deploy and test a Ceph cluster.

The remainder of this paper is organized as the following, Section II describes the approach and technologies used to create the workflow. Section III goes into detail about the different stages in the pipeline. Section IV reflects on the challenges while creating the experimentation workflow. Section V describes the experimental results and outcome of the experimentation workflow. Lastly, Section VI describes the future work and how this project can be futher developed.

B. Ceph

Ceph [1] is an open source software that provides excellent performance, reliability, highly scalable objects, block, and file-based storage in a distributed system. Other features that are integrated within are having no single point of failure, uses commodity hardware and dynamically increase or decrease nodes where the system will be able to fully recover with underlying algorithms.

As previously mentioned Ceph is scalable. A Ceph cluster consists of many roles such as monitors, agents, OSDs, clients, managers, and rgws. For this project, a small Ceph cluster will get created. When first starting out with Ceph, there is a minimum requirement of three nodes needed to get the system running. Each of these three nodes have their own role: a monitor node, and two object storage device (OSD) nodes. The monitor node contains a "cluster map" which is the set of maps comprising of the roles of a monitor, OSDs, placement group (PG), MDS, and CRUSH map [2]. These five different maps in the "cluster map" have knowledge of the whole cluster topology. Another feature that the monitor provides are logging and authentication services which are used when setting up the system. The OSD node is in charge of storing objects on a local file system and providing access to them over the network [2]. It is also important that there is one monitor node in a Ceph cluster because the monitor node acts as a leader for the cluster. Additionally, it is important to always have an odd amount of nodes in the system because we want the system to reach consistency so if there is an even amount of nodes there is a possibility of not obtaining a majority. When there is no majority, a system can get stuck and not reach consistency. Figure 1 shows a high level architecture of Ceph and communication between the different components. Although there are other extra components of Ceph that pictured in Figure 1, but for a proof of concept this paper focuses on a very small system running.

II. Approach

In order to get a Ceph cluster running for experimentation, there are different tools that are used to automate and create this Ceph system; Cloudlab, Ceph Benchmarking Tool, Ceph-Ansible, Docker, and Popper.

Cloudlab [3], is an online service that hosts bare metal machines for users to create and compute a cloud environment. This is how the Ceph system will be created. It is also important to note that since Cloudlab is a hosting service, requested machines are only available for a certain amount of time. Data that has not been saved before the expiration time will be erased from the machines. Since the allocation of the machines, also referred as nodes throughout the paper, is automated it doesn't matter too much that the machines expire because the machines can get reallocated fairly quickly.

Ceph Benchmarking Tool [4] also referred as CBT

Ceph-ansible [5] is an Ansible playbook for Ceph and helps automate the installation of Ceph.

Docker [6], a container platform, helps containerize packages and environments which can be shared among multiple applications. Docker will help create the Ceph environment into a container so all of Ceph's dependencies will be encapsulated together. As mentioned in the abstract, this workflow is only for MAC and Linux operating systems because of Docker. Docker does not work as easily and nicely on a Windows operating system which was observed from past experience. Once an account on Docker has been set and Docker has been installed onto your machine, then the obtaining images on Docker will be easy to pull from Docker hub which consists of many many images. For this workflow, a base Docker images will contain all of the dependencies for some of the other mentioned tools will be contained within a container.

Popper [7] is a CLI tool and convention to create reproducible scientific articles and experiments. The convention is based on the open source software (OSS) development model which creates self-contained experiments that doesn't require external dependencies than what is already contained in the experiment. The Popper convention uses a pipeline that consists of shell scripts which executes an entire experiment. When a pipeline has been initialized, the pipeline consists of several default stages: the setup, run, postrun, validate, and teardown stage. These default stages are not required for every experiment and stages can be renamed accordingly to what makes sense for your experiment. For this experiment the stage names that are used are setup, deploy, run-benchmarks, teardown, and validate. In the setup stage, a user would usually download all the necessary files to run the project. These files are, for example, data files, libraries, and other dependencies. The run stage executes the script that is used to run the main part of the experiment. The postrun stage is where data can be manipulated so it is setup nicely to the validation stage. The validate stage is where a user would display the results obtained in the postrun stage. This stage could be used to open a log file that shows the results of the experiment or run a script that graphs and displays the results, for our case Jupyter will be used to showcase the results.

III. Pipeline

A. Prerequisites

There are three main prerequisites to run this experiment; Popper (v1.1.2), Docker (v2.0.0.3), and a Cloudlab account. Any other additional dependecies that are needed to set up the cluster and benchmarking tool are contained in a Docker container, therefore no additional installs are required.

B. Worflow

Lst. 1 shows the complete pipeline for this experiment. This pipeline consists of the following main stages; setup, deploy, run-benchmark, teardown, and validate. One can notice in Fig. **??** that the setup stage is surrounded in a different color compared to the remainder of the stages. The reasoning behind this is to allow for different services for resources for this end-to-end testing environment. For this experiment, the resources are being allocated from Cloulab and the idea is to use other services such as Amazon Web Services or Chameleon for resources. The setup for Amazon Web Services and Chameleon would consist of different processes for requesting resources therefore the setup scripts would differ from the Cloudlab setup script.

In addition to the scripts to conduct the experiment, there are also other folders in the pipeline such as cbt, ceph-anisble, Docker, and GENI. These folders indicate the different tools are being used throughout the experiment. Each tool will be futher explained in their appropriate stages of how that tool is used.

{#Fig:pipeline_flow} - MIA

C. Setup.sh

The setup stage consists of requesting resources. As mentioned earlier, the resources for this experiment will come from Cloudlab. For the resources that are being used for this project, Ceph is using Cloudlab's cluster Clemson and type c6320. Lst. 2 shows the Clemson cluster hardware specifications for the type c6320. The Clemson cluster was choosen versus the other available clusters because of the amount of storage this specific hardware has.

For this experiment, using Cloudlab, there are five environment variables that need to be declared in order to begin the request of the nodes. The following environmental variables are; `CLOUDLAB_USER`, `CLOUDLAB_PASSWORD`, `CLOUDLAB_PROJECT`, `CLOUDLAB_PUBKEY_PATH`, `CLOUDLAB_CERT_PATH`.

Once the following environmental variables have been declared, these variables are passed into a Docker container to request resources autonmously through the GENI tool. GENI is an API that allows for .... Furthermore, the request API allows users choose which hardware they want to use. For our case Clemson c6320 is choosen. After the request of the resources completes, an `cl-clemson.xml` file is formed with all of the specifications for X amount of nodes, where X is the amount of nodes a user wants for their experiment. Next, a `machines` file is written to group the allocated resources from Cloudlab. Lastly, the monitor IP address gets copied from the `cl-clemson.xml` into the the `all.yml` file in the ceph-ansible folder to prepare for the deployment of the cluster. Below shows which files are used for this stage and the descriptions of a files' functionality. Lst. 3 shows a table of the files and descriptions of their functionality.

Listing 1 Contents of Ceph Popper repository.

```
Ceph-repo
| README.md
| .popper.yml
| pipelines
|   |-- ceph
|   |   |-- setup.sh
|   |   |-- deploy.sh
|   |   |-- run-benchmarks.sh
|   |   |-- teardown.sh
|   |   |-- cbt/
|   |   |     |-- ceph/
|   |   |         |-- ceph.client.admin.key
|   |   |         |-- ceph.conf
|   |   |         |-- rbdmap
|   |   |-- ceph-ansible/
|   |   |     |-- group_vars/
|   |   |         |-- all.yml
|   |   |         |-- osds.yml
|   |   |     |-- purge_cluster.yml
|   |   |     |-- site.yml
|   |   |-- Docker/
|   |   |     |-- cbt/
|   |   |         |-- Dockerfile
|   |   |         |-- install.sh
|   |   |     |-- ceph/
|   |   |         |-- Dockerfile
|   |   |         |-- install.sh
|   |   |-- geni/
|   |   |     |-- clemson.xml
|   |   |     |-- machines
|   |   |     |-- release.py
|   |   |     |-- request.py
|   |   |     |-- renew.py
|   |   |     |-- monitor_config.py
| paper
|   |-- build.sh
|   |-- figures/
|   |-- paper.md
|   |-- paper.pdf
|    -- references.bib
```

D. Deploy.sh

After allocating resources, a Ceph cluster can be deployed. The tool that is used to create the Ceph cluster is Ceph-ansible. Ceph-ansible requires a few configuration files to deploy the cluster which can be referred to Lst. 1. First, all of the allocated nodes gets cleared and erased to make sure no previous version of Ceph has been installed. This will guarantee a clean slate when Ceph is installed. Next, the cluster design gets deployed and the monitor and OSDs roles are assigned to server groups.

The way to verify that the Ceph cluster is up, is by checking the status of the Ceph cluster by one of three ways. First, looking at the end of the log from the deploy stage or secondly `ssh` into the monitor node and use the command `sudo ceph -s` to view the status of the cluster. Lst. 5 shows a sample output of a running cluster. Note that `health` has a warning. That warning can be ignored because in the configuration files when deploying the cluster, the mgr role was omitted. Another way to make sure that the cluster is configured properly is verfiying that the OSDs are up and running. This can be checked my looking at the `services > osd` from the output. In this case, it is shown that two OSDs are up. If the osds are not up and running, that indicates a cluster doesn't exsist.

Listing 2 Cloudlab Clemson c6320 hardware specifications.

```
CPU   Two Intel E5-2683 v3 14-core CPUs
      at 2.00 GHz (Haswell)
RAM   256GB ECC Memory
Disk  Two 1 TB 7.2K RPM 3G SATA HDDs
NIC   Dual-port Intel 10Gbe NIC (X520)
NIC   Qlogic QLE 7340 40 Gb/s Infiniband
      HCA (PCIe v3.0, 8 lanes)
```

Listing 3 Setup stage file breakdown

```
clemson.xml
   - Specifications for each node
machines
   - Allocated resources grouped
request.py
   - Request resources from Cloudlab
renew.py
   - Renew resources from Cloudlab
monitor_config.py
   - Copying monitor IP address for
     deploy stage
```

E. Run-benchmarks.sh

https://www.snia.org/sites/default/files/SDC/2016/presentations/performan

F. Teardown.sh

Once a user has completed their experiment there are two ways to terminates all of the allocated machines back to Cloudlab which will erase all of the data that was installed. The first method is letting the resources expire. The second way is using Cloudlab's release API to release the resources back into the resource pool. The teardown stage script will differ depending where the resources has been allocated, just like the setup stage's script.

IV. Challenges

With any project, there are always obstacles that are faced through the process. The overall main challenge is and was setting up the environment. The first main challenge going into this project was having very little knowledge of Ceph. With

Listing 4 Deploy stage file breakdown

```
all.yml
   - Cluster network configuration settings
osd.yml
   - Osd configuration
site.yml
   - Defined deployment design and assigns
     role to server groups
purge-cluster.yml
   - Purge and clean out nodes
Dockerfile
   - Container containing Ceph-ansible and
     install install.sh
install.sh
   - Clone Ceph-anisble and requirements
```

Listing 5 Cluster configuration output.

```
  cluster:
    id:     xxx-xxx-xxx-xxx-xxxxx
    health: HEALTH_WARN
            no active mgr

  services:
    mon: 1 daemons, quorum node0
    mgr: no daemons active
    osd: 2 osds: 2 up, 2 in

  data:
    pools:   0 pools, 0 pgs
    objects: 0  objects, 0 B
    usage:   0 B used, 0 B / 0 B avail
    pgs:
```

some help I was able ramp up my understanding of Ceph the current methodologies exit for deployed Ceph.

A key part of this workflow is for the pipeline to be reproducible. If reproducibility occurred after a pipeline has been completed, there is a risk that the pipeline will break and the programmer, such as myself, will later be stuck in a loop of fixing the pipeline and maintaining the which can result into complicated setup. Incorporating reproducibility from the beginning is important so this entire workflow will be portable from machine to another.

Another challenge that pushed back the timeline of getting this pipeline working was having to change cloud hosts two times along the way. The first cloud host subscription expired, so then another alternative was suggested and create a local Ceph system. However, the local distribution failed due to permission rights on a Linux - Ubuntu machine and having to change some hardware settings that I didn't want played with. In hindsight a virtual machine would have been a better solution for the local Ceph system, but another main goal of the project was not to use a virtual machine. Virtual machines are useful, but as mentioned in the Abstract this pipeline is intended to have minimal setup.

Another one of the challenge encountered was allocating nodes in Cloudlab. Cloudlab nodes consists of a variety of hardware, so when first starting out and allocating nodes, the nodes that were being allocated from the Cloudlab Utah cluster where the node had 256GB on the disk. There was a memory storage limit that was encountered since Ceph has a lot dependencies. Switching to the Cloudlab Clemson solved this issue.

V. Results

VI. Future Work

VII. Conlcusion

In conclusion, building a Ceph distributed system is not an easy, straight forward task because there are different components that need to be put together. Additionally, it can take multiple weeks just to get the system running so in a class setting where weeks are limited most of the time is spent troubleshooting and debugging the system versus spending time obtaining interesting results to analyze and then make further tests within Ceph. This workflow, once understood, provides a lightweight automated, reproducible Ceph deployment system with an end to end testing workflow.

1) RESERVE/ IGNORE: How to add in an image *Reproduced results using Popper.*

Testing correctness in Ceph will look for the time when nodes fail in the system and check for consistency of the data. Testing availability will also look when nodes fail but then check to see how long it takes for Ceph to recover, which also adds overhead. Lastly, testing the overhead of replication techniques such as whether data will need one copy, two copies of data or use an erasure coding algorithm.

References

[1] S.A. Weil, S.A. Brandt, E.L. Miller, D.D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," *Proceedings of the 7th symposium on operating systems design and implementation*, USENIX Association, 2006, pp. 307–320.

[2] "Ceph docs," *http://docs.ceph.com.*

[3] "Cloudlab."

[4] "Ceph benchmarking tool."

[5] "Ceph ansible."

[6] "Docker."

[7] I. Jimenez, M. Sevilla, N. Watkins, C. Maltzahn, J. Lofstead, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "The popper convention: Making reproducible systems evaluation practical," *Parallel and distributed processing symposium workshops (ipdpsw), 2017 ieee international*, IEEE, 2017, pp. 1561–1570.