KloudBuster Documentation

Release 6.1.0

OpenStack Foundation

Jul 03, 2017

Contents

Contents:

CHAPTER 1

KloudBuster version 6

How good is your OpenStack **data plane** or **storage service** under real heavy load?

KloudBuster is a tool that can load the data plane or storage infrastructure of any OpenStack cloud at massive scale and measure how well the cloud behaves under load where it matters: from the VMs standpoint, where cloud applications run.

Accessible to anybody with basic knowledge of OpenStack, installs in minutes and runs off the box with sensible default workloads in a fully automated way. CLI/REST or Web User Interface.. you pick what works best for you.

Feature List

- Neutron configuration agnostic (any encapsulation, any overlay, any plugin)
- OpenStack Storage backend agnostic
- Real VM-level performance and scale numbers (not bare metal)
- Punishing scale (thousands of VMs and enough load to fill even the fastest NIC cards or load any storage cluster with ease)
- Data plane with HTTP traffic load:
  - Can load the data plane with one OpenStack cloud (single-cloud operations for L3 East-West scale) or 2 OpenStack clouds (dual-cloud operations with one cloud hosting the HTTP servers and the other loading HTTP traffic for L3 North-South scale testing)
  - Real HTTP servers (Nginx) running in real Linux images (Ubuntu 14.04)
  - Can specify any number of tenants, routers and networks
  - Can specify any number of HTTP servers per tenant (as many as your cloud can handle)
  - High performance and highly scalable HTTP traffic generators to simulate huge number of HTTP users and TCP connections (hundreds of thousands to millions of concurrent and active connections)

  - Overall throughput aggegation and loss-less millisecond-precision latency aggregation for every single HTTP request (typically millions per run)
  - Traffic shaping to specify on which links traffic should flow
  - Can support periodic reporting and aggregation of results
- Storage load:
  - VM-level Cinder volume (block storage) or Ephemeral disk file I/O performance measurement using FIO running inside VMs (not bare metal)
  - Supports random and sequential file access mode
  - Supports any mix of read/write
  - IOPs, bandwitdh and loss-less millisecond-precision latency aggregation for every IO operation (typically millions per run)
  - User configurable workload sequence
- Supports automated scale progressions (VM count series in any multiple increment) to reduce dramatically scale testing time
- Highly efficient and scalable metric aggregation
- Automatic cleanup upon termination
- Regular expression based cleanup script (*OpenStack Resources Cleanup*)
- KloudBuster server mode to drive scale test:
  - from any browser (KloudBuster Web UI)
  - or from any external programs (KloudBuster REST API)
- Aggregated results provide an easy to understand way to assess the scale of the cloud under test
- KloudBuster VM image pre-built and available from the OpenStack Community App Catalog (https://apps.openstack.org/)

Diagrams describing how the scale test resources are staged and how the traffic flows are available in *Architecture*.

Scale results are available in json form or in html form with javascript graphical charts generated straight off the tool.

Examples of results are available in *Gallery*.

Limitations and Non-Goals

- Requires Neutron networking (does not support Nova networking)
- Only supports HTTP and storage traffic in this version

Unlike some other scaling test frameworks, KloudBuster does **not** attempt to:

- Provide a scale test framework that works across different cloud technologies (OpenStack + AWS + Google Cloud + ...) - we are only focusing on OpenStack
- Provide a scale test framework that is flexible and programmable to do everything - we just focus on opinionated and well targeted performance and scale areas with sensible use cases and available in a fully integrated and easy to consume packaged format
- Replace bare metal and domain specific native performance and scale frameworks (line level traffic generators, ceph specific performance and scale tools...)

Contributions and Feedbacks

If you are interested in OpenStack Performance and Scale, contributions and feedbacks are welcome!

If you have any feedbacks or would like to make small or large contributions, simply send an email to openstack-dev@lists.openstack.org with a '[kloudbuster]' tag in the subject.

Licensing

KloudBuster VM images contain multiple open source license components:

- nginx: BSD License (http://nginx.org/LICENSE)
- wrk2: Apache License 2.0 (https://raw.githubusercontent.com/giltene/wrk2/master/LICENSE)
- Redis: BSD License (http://redis.io/topics/license)
- FIO: GPL v2 (https://raw.githubusercontent.com/axboe/fio/master/MORAL-LICENSE)

Although the VM image includes a binary copy of the FIO code, it does not include the source code used to build it. In accordance to the GPL V2 license related to the inclusion of binary copies of FIO, the source code used to build the FIO binary copy was not modified and can be found directly at https://github.com/axboe/fio or can be obtained by email request to the maintainer of KloudBuster.

Links

- Complete documentation: http://kloudbuster.readthedocs.org
- KloudBuster REST API documentation Preview
- Source: https://github.com/openstack/kloudbuster
- Supports/Bugs: http://launchpad.net/kloudbuster
- Mailing List: kloudbuster-core@lists.launchpad.net

CHAPTER 2

Architecture

Data Plane Scale Test

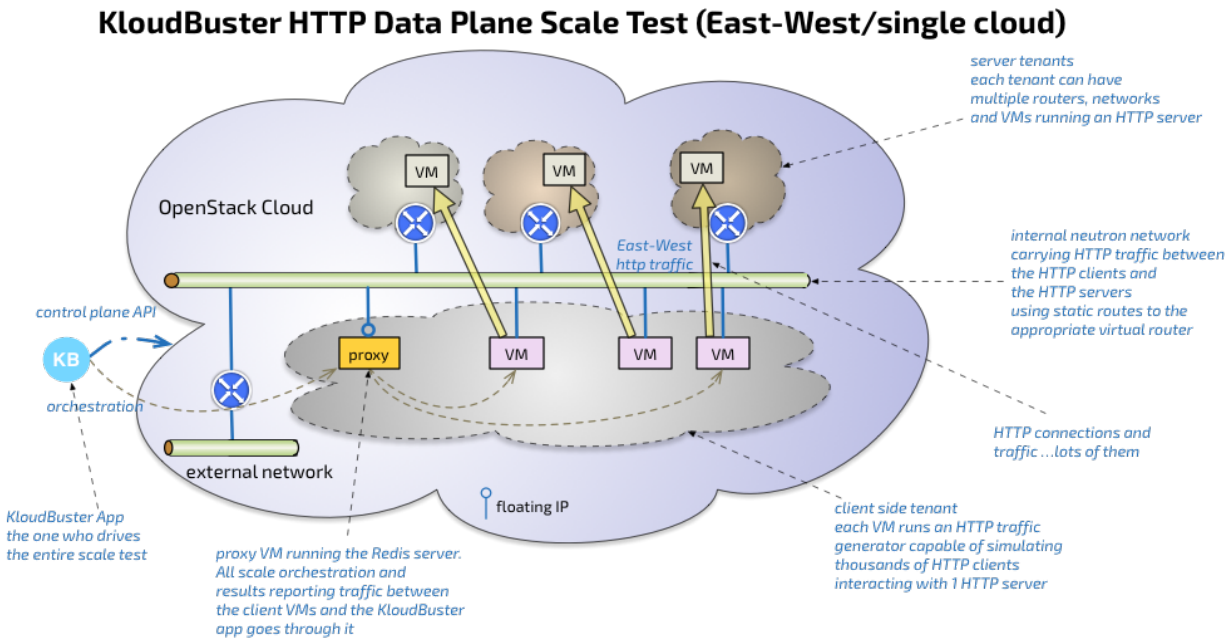
Although many types of traffic can run on an OpenStack data plane, KloudBuster focuses on measuring HTTP traffic scale on any OpenStack cloud because it is a well understood and very popular traffic type and there are many great open source tools that can scale to the task and can be leveraged to implement the HTTP scale test.

East-West Data Plane Scale Test

East-West traffic refers to the traffic that is exchanged between VM instances that run inside the same cloud. Such traffic can involve:

- Only packet switching if the 2 end points belong to the same Neutron network (that is the packets do not have to go through a router) - often called L2 East-West
- Packet routing if the 2 end points belong to different Neutron networks (packets have to go through router) - often called L3 East-West

The KloudBuster data plane scale test exercises L3 East-West traffic by running a set of HTTP servers in their respective tenant private network and a set of HTTP traffic generators in a client tenant network with HTTP traffic flowing from the client tenant network to the various server networks through the corresponding server router as illustrated in the following diagram:



The KloudBuster App typically runs outside the cloud under test on any server that has a python interpreter (MacBook, Linux workstation...) with the requirement to have access to the OpenStack API of the cloud under test.

The KloudBuster app basically reads the requested scale config (which contains parameters such as test duration, how many HTTP servers are to be used, how many tenants, networks and routers, how many clients, rate of HTTP requests...), stages the resources using the OpenStack API, orchestrates the start of the test, collects then aggregates all the results, then cleans up the resources.

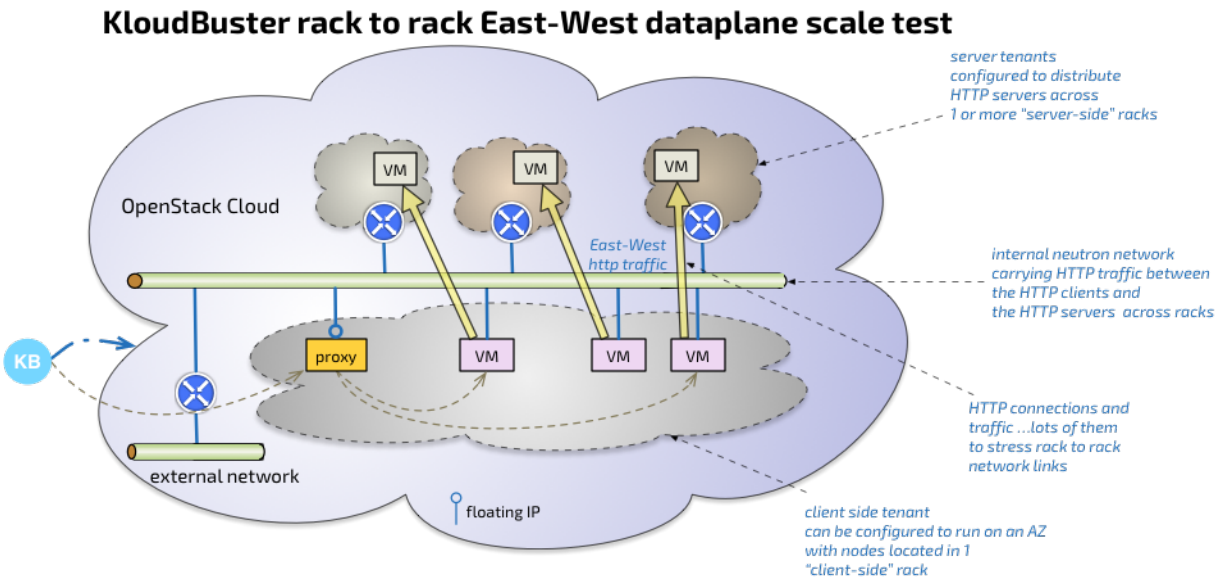
Each HTTP traffic generator instance runs in a VM and is capable of simulating a large number of HTTP clients that will send HTTP requests to the same HTTP server instance at a configurable rate (there is a 1:1 mapping between each client VM and server VM).

Orchestration and results collection is done through the Redis server which runs in the first staged VM. Many scale tools use SSH to drive the test agents and SSH has shown to be hard to scale beyond a few hundred sessions while a solution based on Redis can scale very easily to thousands of end points. Another important benefit of using a Redis server is that the scale test only requires 1 floating IP (for the Redis server) since all communication to the client VMs are performed on the internal network using private addresses. Using SSH directly to the client VMs would require a lot of floating IPs or would require a proxy which makes the solution even more brittle.

Rack to Rack Data Plane Scale

By default KloudBuster will rely on the Nova scheduler to place the various client and server VMs. As a result these VMs will be load balanced across all servers and causing the data path of the HTTP traffic to be quite random. This can be good to measure the scale on a random traffic pattern but sometimes it is more interesting to shape the HTTP traffic can be shaped to follow certain paths.

One good example is to assess the scale of the data plane across racks since most deployments involve the use of a top of rack switch to service a potentially large number of servers in each rack. For that purpose, KloudBuster provides a way to specify the placement of client and server VMs using specific availability zones as illustrated in the following diagram:

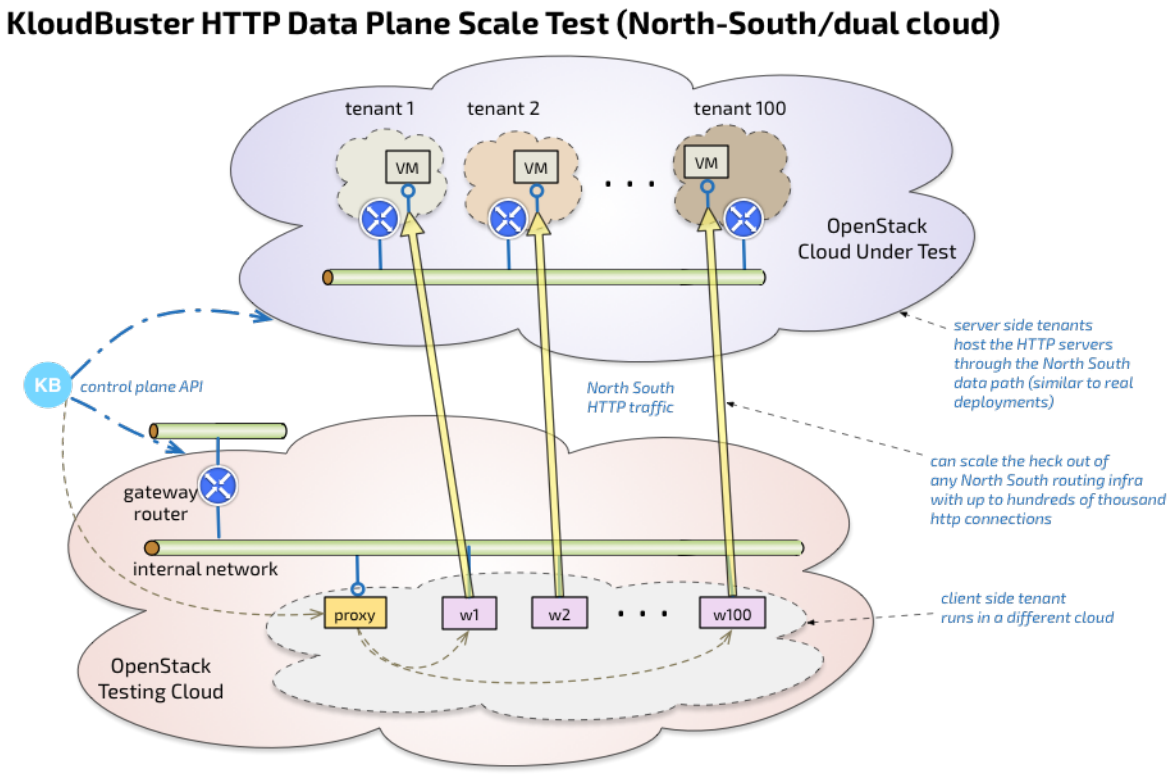


The client VMs are placed in 1 rack while the server VMs are placed in a set of different racks. Such arrangement will cause the traffic to flow exclusively between these racks, allowing a good measurement of the data plane capabilities of each rack.

North South Data Plane Scale Test

The North South traffic refers to traffic flowing between external sources and VMs running in the cloud. Such traffic follows a very different path than East-West traffic as it is generally always routed and requires the used of IP address translation (SNAT and DNAT). One exception to this is the use of a provider network which may avoid routing and NAT completely.

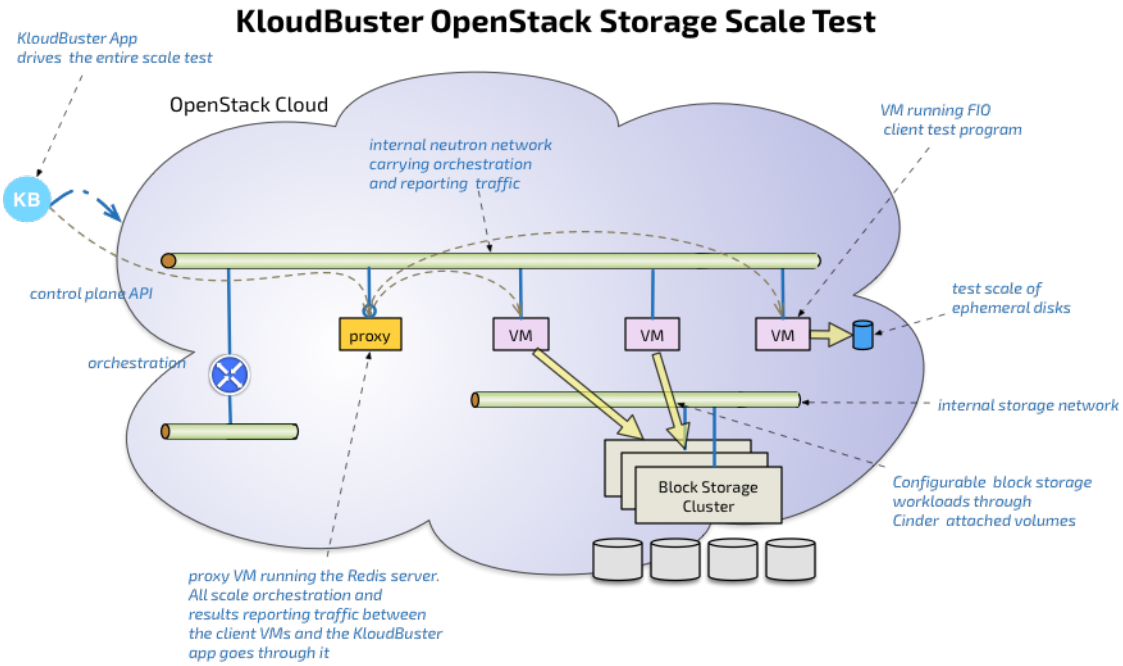
KloudBuster provides a option to test the North-South data plane traffic by separating the client VMs and server VMs into 2 different OpenStack clouds.



In this mode, KloudBuster will stage and orchestrate the test on 2 distinct clouds.

Storage Scale Test

The storage scale test is a relatively simpler version of the data plane scale test as it only involves 1 tenant, 1 network and 1 router. Each test VM runs one instance of the FIO test client (FIO which is a widely adopted open source storage test client).



VM staging, storage plumbing (using Cinder or Nova for ephemeral disks) is done by the KloudBuster app using OpenStack APIs. Because the Cinder API abstracts the storage back-end, it is agnostic of the technology used (Ceph, EMC...). After the test, all resources (volumes, VMs, network, router) are cleaned up in the proper order.

Progression Runs

VM staging is a very lengthy phase of any scale test especially when dealing with scales of hundreds or thousands of VMs.

Progression runs are a very convenient feature as it allows to produce result for series in a much shorter time by reusing the same set of staged VMs and iterating the scale test to produce measurements at different scale level.

For example, to get storage performance measurement for 100 to 1000 VMs in increments of 100, would require staging and unstaging 100+200+300+....+1,000 = 5,500 VM Instances without progression runs while it would only require staging 1,000 instances with VM reuse.

Latency and Distributed Latency at Scale

Latency is a critical metric and reporting correctly latency at scale in a distributed environment is a difficult problem.

An example of HTTP scale test could simulate 1 million HTTP users sending a combined 100,000 HTTP requests per second for 1,000 seconds across say 1,000 HTTP servers and 1,000 HTTP traffic generators. A good characterization of how well a cloud supporting these 1,000 HTTP servers behaves is not only to measure the actual combined HTTP requests per seconds achieved (e.g. 70,000 HTTP request/per second) but also the latency of these HTTP requests with a precision of 1 millisecond. For this kind of scale, the only proper way to measure latency is to have the complete latency distribution percentile for all 70,000 * 1,000 = 70 million HTTP operations. The problem is that these 70 M operations are distributed across 1,000 client VMs and as such each traffic generator has only the latency distribution of those requests issued locally (or about 70,000 HTTP operations per VM).

Similarly, assessing the storage scale of 500 VMs doing 400 IOPs each results in tracking the latency of a combined 200K IO operations per second. A mere 10-minute run results in tracking the latency for over 100M IO operations,

distributed across 500 VMs.

Many scaling tools take the shortcut of only reporting an average per client VM (or even min or max - each client only has to report a small number of metrics per run). The aggregation of all these averages makes the reported result (average of averages, min/max of averages...) very weak because it completely loses sight of outliers which is precisely the type of detail you need to assess accurately the scale of a large distributed system.

To solve that problem, KloudBuster uses the HdrHistogram open source library to do loss-less compression/decompression of full latency histograms on the fly in a highly scalable way.

CHAPTER 3

Gallery

This page has links to examples of scale test reports in HTML format generated by KloudBuster. These reports were generated within minutes after starting the scale test from a bare bone OpenStack cloud (not running anything). Click on the thumbnail images to view the result HTML file in your browser (you will need access to the Internet to view these files as they reference multiple Java script libraries in CDN).

Sample HTTP Scale Report

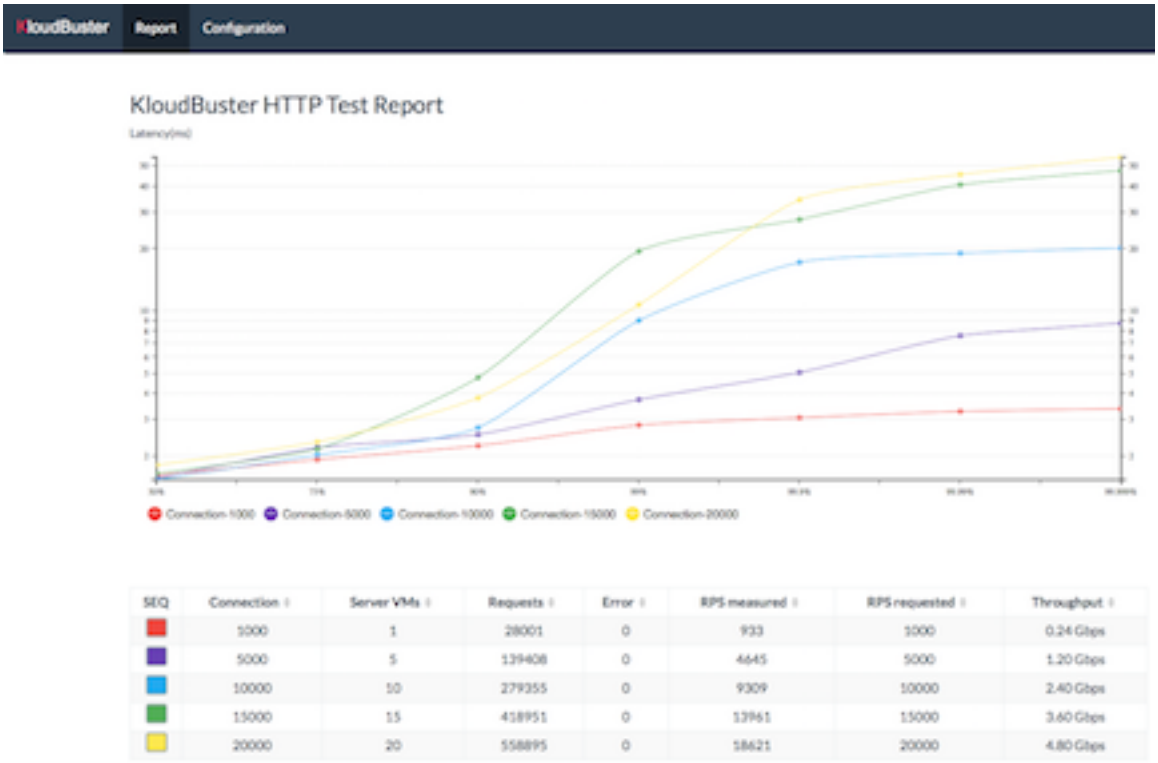
The following report shows an HTTP scale run with results for 1 to 20 HTTP servers (running in as many server VMs) in increment of 5, where each HTTP server is receiving HTTP requests from 1 HTTP traffic generator that runs in a separate VM and emulates 1,000 users sending 1 request per second each (for a total of 1000 requests per second per HTTP server). The topology used for the test is 1 tenant, 1 router, 4 networks and 5 HTTP servers per network. Each iteration is programmed to run for 30 seconds. This scale settings can be viewed in the Configuration tab.

The table shows the results for each iteration step, with the requested and measured RPS (HTTP requests per second) and the corresponding aggregated download throughput (the sum of all downloads for all clients).

Each line in the chart represents the latency distribution for one load level (or iteration in the progression). Lines can be individually shown/hidden by clicking on the corresponding legend item.

For example, the largest scale involves 20,000 simultaneous users sending an aggregate of 18,621 HTTP requests per second and the latency chart tells us that 99.9% of these 18,621 requests are replied within 34ms, which is actually excellent.

Note that this test is configured to reuse HTTP connections meaning that we do not have the overhead of creating a new TCP connection for every HTTP request. This also means that this cloud had 20,000 TCP active connections at all times during the scale test.



SEQ	Connection	Server VMs	Requests	Error	RPS measured	RPS requested	Throughput
	1000	1	28001	0	933	1000	0.24 Gbps
	5000	5	139408	0	4645	5000	1.20 Gbps
	10000	10	279355	0	9309	10000	2.40 Gbps
	15000	15	418951	0	13961	15000	3.60 Gbps
	20000	20	558895	0	18621	20000	4.80 Gbps

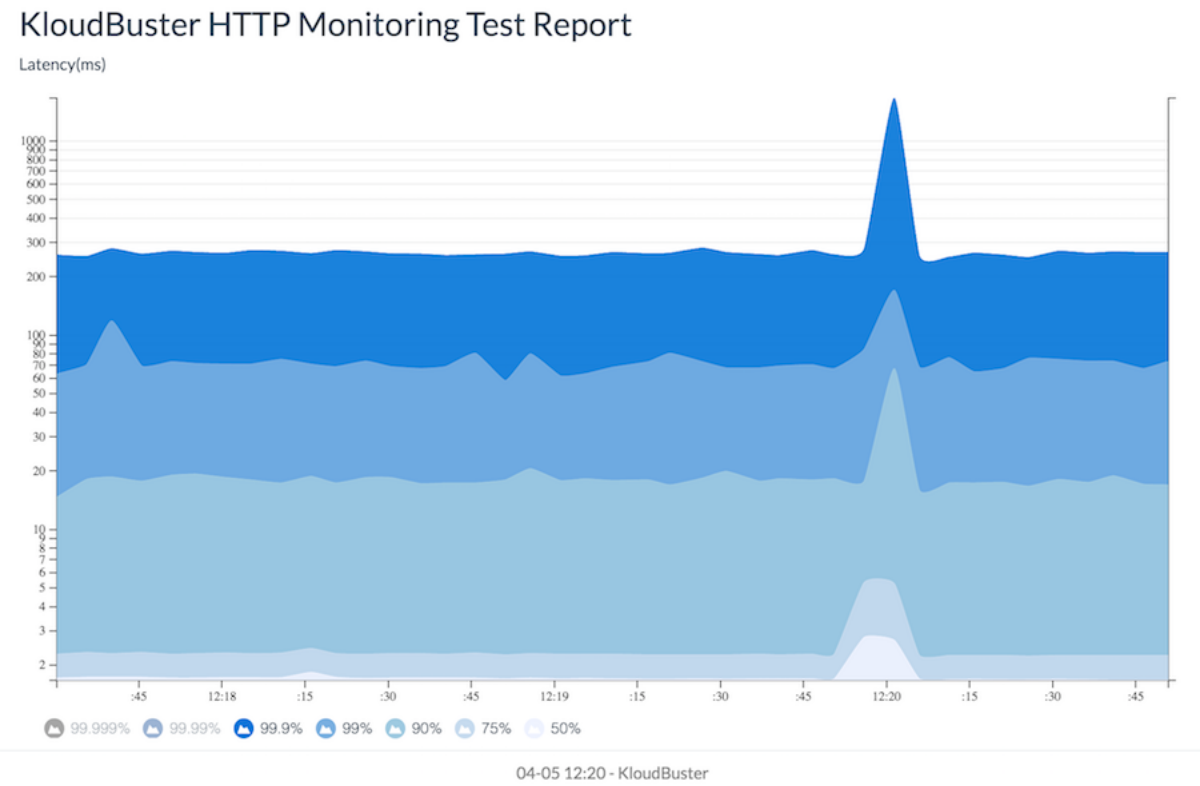
Sample HTTP Monitoring Report

The report below shows an HTTP monitoring run with 15 HTTP servers where each HTTP server is receiving HTTP requests from 1 HTTP traffic generator that runs in a separate VM and emulates 1,000 users sending 1 request per second each (for a total of 1000 requests per second per HTTP server). The topology used for the test is 1 tenant, 1 router, 3 networks and 5 HTTP servers per network. The total duration is set to 300 seconds. This scale settings can beviewed in the Configuration tab.

This stacked chart updates in real time by scrolling to the left and shows how the latency of HTTP requests evolves over time for each percentile group (50%, 75%, 90%, 99%99.9% 99.99%, 99.999%). Lines can be individually shown/hidden by clicking on the corresponding legend item.

The compute node where the HTTP servers run is protected against individual link failures by using a link aggregation with 2 physical links connected to 2 different switches.

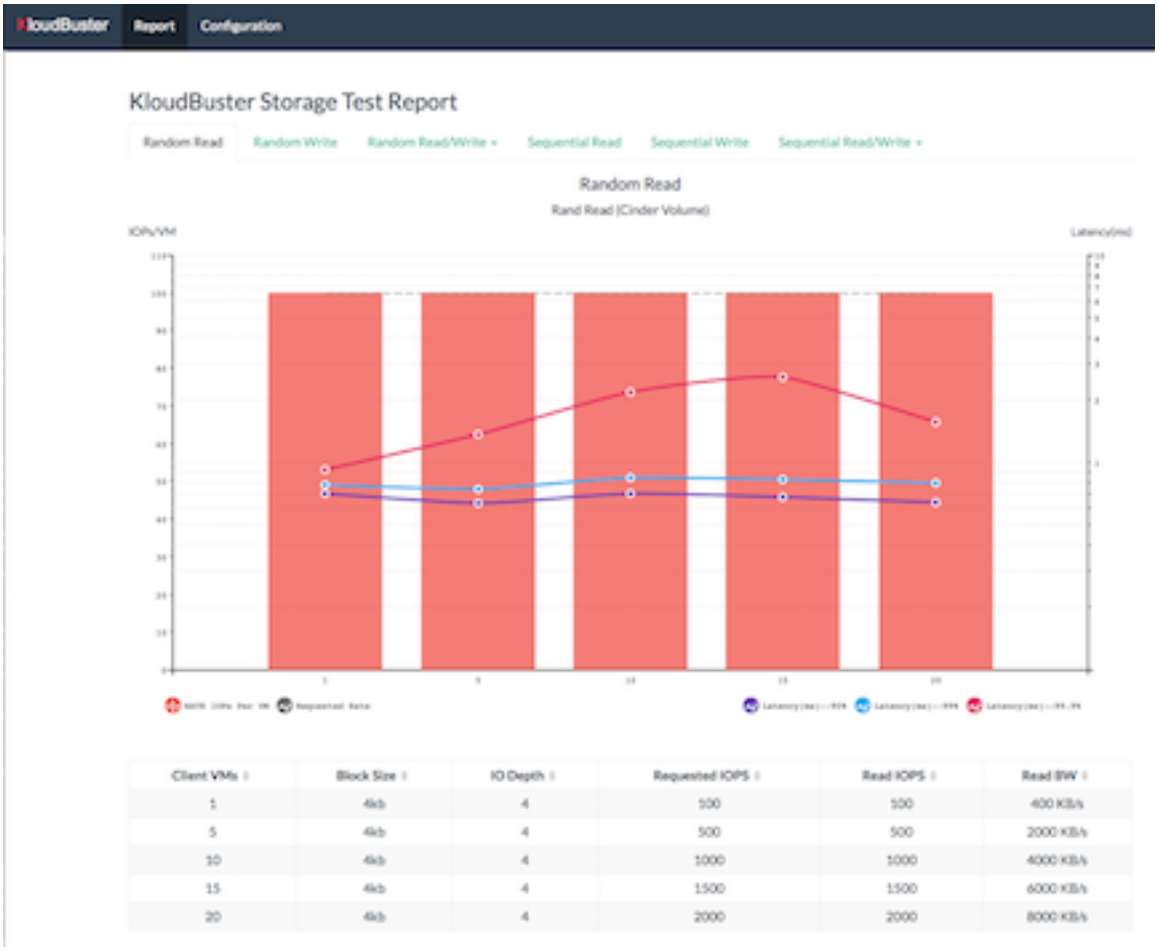
At 12:19:53, one of the 2 physical links is purposely shut down. As can be seen, the latency spikes as high as 1664 msec and returns to normal after about 10 seconds.



Sample Storage Scale Report

This is a report for a storage scale test using the default workload suite with a progression run from 1 VM to 20 VMs in increment of 5, and 30 second run per iteration. This results in 6 tabs of results (1 per workload). The mixed read/write tabs further split in 2 sub tabs (1 for read and 1 for write results).

The random read tab shows that each VM could achieve its requested 100 IOPs across the progression. The lines represent the latency value at given percentile and can be individually shown/hidden by clicking the corresponding legend item. As an example, 20 VMs represents a combined 2,000 IOPs measured for a total of 60,000 random read operations. The latency line tells us that 99.9% of these 60,000 read operations are completed within 1.576 msec.



The sequential write results are more challenging as they show that the VMs cannot achieve their requested write bandwidth (60MB/s) and can only get 49MB/s each when there are 20 of such VMs. The latency lines also reflect that stress by peaking at 500 ms for 99.99% of all write operations (although latency is not nearly as critical for sequential access than for random access).

CHAPTER 4

Usage and Quick Start Guides

KloudBuster User Interfaces

KloudBuster provides 3 interfaces, the easiest being the **Web User Interface**. It offers the most user friendly interface and needs the least learning to get started. **CLI** is the traditional way to run applications. It has the most comprehensive feature sets when compared to the other methods. **REST API** gives another way to access and control KloudBuster from another application.

The Web UI is fully implemented on top of the REST API.

OpenStack Cloud Pre-Requisites

OpenStack cloud pre-requisites to run KloudBuster:

- Neutron networking
- Admin access to the cloud under test (non-admin might work with some tweaks and limitations)
- 3 available floating IPs if running the HTTP data plane scale test
- 2 available floating IPs if running the Storage scale test

KloudBuster Installation Options

There are 4 different ways to install KloudBuster:

- use a pre-built Docker container (recommended if you already use Docker)
- use a pre-built VM image (if you prefer to run the KloudBuster application in a VM and do not need CLI)
- install from PyPI (if you prefer to use pip install)
- install directly from GitHub (git clone, for code development or if you want to browse the code)

Users of KloudBuster who prefer to use the CLI or who prefer to run KloudBuster locally on their workstation or laptop can use the PyPI based installation (pip install) or the Docker container.

Docker container, Web Service and PyPI based installation will satisfy most use cases and are the recommended ways for running KloudBuster under production environments or through an automated or scheduled job.

Quick Start Guides

KloudBuster Docker Container Quick Start Guide

The KloudBuster Docker container provides a quick way to use KloudBuster if you are already familiar with Docker.

Prerequisites

This quick start guide assumes you have already installed Docker. All command line examples below are based on Linux (which could be either native or in a VM) and require Internet access to Docker Hub.

1. Pull latest Docker container image

KloudBuster is available as a container in Docker Hub at berrypatch/kloudbuster

```
$ docker pull berrypatch/kloudbuster
```

2. Get the openrc file from your OpenStack Horizon dashboard

Using the Horizon dashboard, download the openrc file (Project|Compute|API Access then click on "Download OpenStack RC File"). It is best to use the admin user to run KloudBuster as much as possible (otherwise there are restrictions on what you can do). Instructions below assume a copy of that file is saved under the local directory with the name "admin-openrc.sh"

3. Upload the KloudBuster VM image to the cloud under test

If your OpenStack cloud has full access to the Internet, you can skip this step as KloudBuster will instruct Glance to download the KloudBuster VM image directly from the OpenStack (skip to next step).

Otherwise, *download the latest kloudbuster image* from the OpenStack App Catalog.

In addition to the method described to upload the image using the Horizon dashboard or the glance CLI, you can also use the glance CLI that is already available in the KloudBuster container. Start a bash shell in the container and map the local directory to '/opt/kb' in the container so that you have access to the image and the RC file:

```
docker run -v $PWD:/opt/kb --rm -it berrypatch/kloudbuster bash
```

Then from inside the container bash prompt, source the openrc file, invoke the glance CLI to upload the VM image (should take a few minutes) then exit and terminate the container:

```
source /opt/kb/admin-openrc.sh
glance image-create --name "kloudbuster_v6" --visibility public --disk-format qcow2 --
↪container-format bare --file /opt/kb/kloudbuster_v6.qcow2
```

Now you should be back to the host and should see the kloudbuster image in the current directory.

4. Running the KloudBuster CLI

If you do not really need a Web UI or REST interface, you can simply run KloudBuster scale test straight from CLI in the container.

```
docker run -v $PWD:/opt/kb --rm -t berrypatch/kloudbuster kloudbuster -h
```

We assume in the below example that you have an openrc file available called "admin-openrc.sh" in the local directory and that the corresponding OpenStack password is "admin".

Run the default HTTP data plane scale test

The default HTTP scale test is described *here*.

```
docker run --rm -t -v $PWD:/opt/kb berrypatch/kloudbuster kloudbuster --tested-rc /
↪opt/kb/admin-openrc.sh --tested-passwd admin
```

Run the default storage scale test

The default storage scale test is described *here*.

```
docker run --rm -t -v $PWD:/opt/kb berrypatch/kloudbuster kloudbuster --tested-rc /
↪opt/kb/admin-openrc.sh --tested-passwd admin --storage
```

Run KloudBuster with a custom configuration

To get a copy of the default KloudBuster configuration and store it to a file called "kb.cfg":

```
docker run --rm berrypatch/kloudbuster kloudbuster --show-config >/opt/kb/kb.cfg
less kb.cfg
```

You can then edit kb.cfg and modify it appropriately. To run KloudBuster with the custom configuration, simply pass it to container after mapping the host local directory to "/opt/kb" (for example):

```
docker run --rm -t -v $PWD:/opt/kb berrypatch/kloudbuster kloudbuster --tested-rc /
↪opt/kb/admin-openrc.sh --tested-passwd admin --config /opt/kb/kb.cfg
```

5. Running KloudBuster as a WebUI/REST Server

By default KloudbBuster will listen on port 8080 in the container. This port must be mapped to a host level port using the -p argument. For example, to use the same port number at the host level:

```
docker run -p 8080:8080 --rm berrypatch/kloudbuster kb_start_server&
```

The first port number is the host listen port (any port of your choice) while the second one after the column is the container listen port (always 8080 for KloudBuster). For example, to use port 9090 on the host and map it to the KloudBuster port in the container, you would use -p 9090:8080

To stop the KloudBuster container, you can use the "docker kill <id>" command.

Assuming the host port used is 8080, the Web UI URL to use from any browser is:

```
http://<host_ip>:8080
```

The KloudBuster REST base URL is the above URL with "/api" appended:

```
http://<host_ip>:8080/api
```

How to use the Web UI

How to use the REST interface

KloudBuster Pip Install Quick Start Guide

KloudBuster is available in the Python Package Index (PyPI) KloudBuster PyPI and can be installed on any system that has python 2.7.

1. Install pip and the python virtualenv (if not installed already)

You will need to have python 2.7, pip, and some dependencies installed before installing KloudBuster depending on the operating system at the installation site. These pre-requisites can be skipped if the corresponding dependencies are already installed.

Ubuntu/Debian based:

```
$ sudo apt-get install python-dev python-pip python-virtualenv libyaml-dev
```

RHEL/Fedora/CentOS based:

```
$ sudo yum install gcc python-devel python-pip python-virtualenv libyaml-devel
```

MacOSX:

```
$ # Download the XCode command line tools from Apple App Store
$ xcode-select --install
$ sudo easy_install pip
$ sudo pip install virtualenv
$
$ # If you need to run KloudBuster Web UI from PyPI installation,
$ # coreutils needs to be installed using Homebrew.
$ # Refer here for the steps to install Homebrew on Mac:
$ # http://brew.sh/
$ brew install coreutils
```

2. Install KloudBuster in a virtual environment

Create a virtual environment for Python, and install KloudBuster:

```
$ virtualenv vkb
$ source vkb/bin/activate
$ pip install kloudbuster
```

Alternatively, if you have virtualenvwrapper installed:

```
$ mkvirtualenv kloudbuster
$ pip install kloudbuster
```

Note: "A Virtual Environment is a tool to keep the dependencies required by different projects in separate places, by creating virtual Python environments for them." It is optional but recommended. We could use:

```
$ sudo pip install kloudbuster
```

instead if isolation among multiple Python projects is not needed.

To verify kloudbuster is installed properly, just type:

```
kloudbuster --help
```

3. Upload the KloudBuster VM image

Follow the *steps* to upload the KloudBuster VM image to the OpenStack cloud under test.

4. Download the openrc file

Using the Horizon dashboard, download the openrc file (Project|Compute|API Access then click on "Download OpenStack RC File"). It is best to use the admin user to run KloudBuster as much as possible (otherwise there are restrictions on what you can do).

5. Running the KloudBuster CLI

Run the default HTTP data plane scale test

The default HTTP scale test is described *here*.

```
kloudbuster --tested-rc admin-openrc.sh --tested-passwd admin
```

Run the default storage scale test

The default storage scale test is described *here*.

```
kloudbuster --tested-rc admin-openrc.sh --tested-passwd admin --storage
```

Run KloudBuster with a custom configuration

To get a copy of the default KloudBuster configuration and store it to a file called "kb.cfg":

```
kloudbuster --show-config >kb.cfg
less kb.cfg
```

You can then edit kb.cfg and modify it appropriately. To run KloudBuster with the custom configuration:

```
kloudbuster --tested-rc admin-openrc.sh --tested-passwd admin --config kb.cfg
```

6. Running KloudBuster as a WebUI/REST Server

```
kb_start_server&
```

You should see a message similar to the one below, which indicates the server is up running:

```
Starting server in PID 27873
serving on 0.0.0.0:8080, view at http://127.0.0.1:8080
```

By default KloudbBuster will listen on port 8080. The KloudBuster Web UI URL to use from any browser is:

```
http://<host_ip>:8080
```

The KloudBuster REST base URL is the above URL with "/api" appended:

```
http://<host_ip>:8080/api
```

How to use the Web UI

How to use the REST interface

KloudBuster VM Application Quick Start Guide

The pre-built KloudBuster qcow2 VM image contains the KloudBuster Web server and is ready to service HTTP and REST requests once up and running. To get the KloudBuster Web server running in any OpenStack cloud:

1. Upload the KloudBuster VM image

Follow the *steps* to upload the KloudBuster VM image to the OpenStack cloud that will host your KloudBuster web server

---

Note: The KloudBuster web server can run in the cloud under test or in another OpenStack cloud.

---

2. Create a Neutron tenant router and network

If necessary, and as for any VM-based web server application bring up, create and configure the Neutron router and network where the KloudBuster web server VM instance will be attached. You can also reuse an existing tenant network and router.

3. Create a Security Group

Create or reuse a security group which allows ingress TCP traffic on port 8080.

4. Launch the KloudBuster VM Application

Launch an instance using the KloudBuster image with the proper security group, and connect to the appropriate network. Leave the Key Pair as blank, as we don't need the SSH access to this VM.

5. Associate a floating IP

Associate a floating IP to the newly created VM instance so that it can be accessible from an external browser

6. Connect to the web UI with a browser

The Web UI URL to use from any browser is:

```
http://<floating_ip>:8080
```

The base URL to use for REST access is:

```
http://<floating_ip>:8080/api
```

7. Download the openrc file

Using the Horizon dashboard, download the openrc file (Project|Compute|API Access then click on "Download OpenStack RC File"). It is best to use the admin user to run KloudBuster as much as possible (otherwise there are restrictions on what you can do).

8. Login to KloudBuster

Follow *instructions* on how to use the web UI.

KloudBuster Git Quick Start Guide

This is the default installation method for code development.

It is recommended to run KloudBuster inside a virtual environment. However, it can be skipped if installed in a dedicated VM.

1. Install Dependencies and Clone Git Repository

Quick installation on Ubuntu/Debian

```
$ sudo apt-get install python-dev python-virtualenv git git-review qemu-utils
$ # create a virtual environment
$ virtualenv ./vkb
$ source ./vkb/bin/activate
$ git clone https://github.com/openstack/kloudbuster.git
$ cd kloudbuster
$ pip install -r requirements-dev.txt
```

Quick installation on RHEL/Fedora/CentOS

```
$ sudo yum install python-devel python-virtualenv git qemu-img
$ # create a virtual environment
$ virtualenv ./vkb
$ source ./vkb/bin/activate
```

```
$ git clone https://github.com/openstack/kloudbuster.git
$ cd kloudbuster
$ pip install -r requirements-dev.txt
```

Quick installation on MacOSX

KloudBuster can run natively on MacOSX.

First, download XCode from App Store, then execute below commands:

```
$ # Download the XCode command line tools
$ xcode-select --install
$ # Install pip
$ sudo easy_install pip
$ # Install python virtualenv
$ sudo pip install virtualenv
$ # create a virtual environment
$ virtualenv ./vkb
$ source ./vkb/bin/activate
$ git clone https://github.com/openstack/kloudbuster.git
$ cd kloudbuster
$ pip install -r requirements-dev.txt
```

If you need to run the KloudBuster Web UI you need to install coreutils (you can skip this step if you do not run the KloudBuster Web server):

```
$ # If you need to run KloudBuster Web UI,
$ # coreutils needs to be installed using Homebrew.
$ # Refer here for the steps to install Homebrew on Mac:
$ # http://brew.sh/
$ brew install coreutils
```

Verify installation

To verify kloudbuster is installed, from the root of the kloudbuster repository type:

```
$ python kloudbuster/kloudbuster.py --help
```

2. Upload the KloudBuster VM image

Follow the *steps* to upload the KloudBuster VM image to the OpenStack cloud under test.

3. Download the openrc file

Using the Horizon dashboard, download the openrc file (Project|Compute|API Access then click on "Download OpenStack RC File"). It is best to use the admin user to run KloudBuster as much as possible (otherwise there are restrictions on what you can do). The examples below assume the openrc file is saved at the root of the kloudbuster git repository with the name "admin-openrc.sh" and the password is "admin".

4. Running the KloudBuster CLI

Run the default HTTP data plane scale test

The default HTTP scale test is described *here*.

```
python kloudbuster/kloudbuster.py --tested-rc admin-openrc.sh --tested-passwd admin
```

Run the default storage scale test

The default storage scale test is described *here*.

```
python kloudbuster/kloudbuster.py --tested-rc admin-openrc.sh --tested-passwd admin --
↪storage
```

Run KloudBuster with a custom configuration

The default KloudBuster configuration file is in cfg.scale.yaml. You can make a copy of it in "kb.cfg":

```
cp kloudbuster/cfg.scale.yaml kb.cfg
```

You can then edit kb.cfg and modify it appropriately. To run KloudBuster with the custom configuration:

```
python kloudbuster/kloudbuster.py --tested-rc admin-openrc.sh --tested-passwd admin --
↪config kb.cfg
```

5. Running KloudBuster as a WebUI/REST Server

```
python kloudbuster/start_server.py&
```

You should see a message similar to the one below, which indicates the server is up running:

```
Starting server in PID 27873
serving on 0.0.0.0:8080, view at http://127.0.0.1:8080
```

By default KloudbBuster will listen on port 8080.

How to use the Web UI

How to use the REST interface

To terminate the server, simply use the kill command on the server pid.

Using the KloudBuster Web UI

Using any browser, point to the provided URL. You will get a Login page where you will need to enter:

- The type of scale test (HTTP data plane or storage)
- The location of the openrc file for the cloud under test and the corresponding OpenStack password

You could modify the scale test configuration options or simply start the scale test with the default scale configuration. Click on Stage button to instruct KloudBuster to stage all the OpenStack resources. This can take time depending on how many VMs are requested and how fast is the cloud under test.

Once staging is done, click on the Run button to run the scale test.

Interacting with the KloudBuster REST Interface

REST Documentation

Once the server is started, you can use different HTTP methods (GET/PUT/POST/DELETE) to interact with the KloudBuster REST interface using the provided URL at port 8080.

- KloudBuster REST API Documentation Preview
- REST API Documentation (Swagger yaml)

Examples of REST requests

KloudBuster VM Image Upload

Before you can use KloudBuster you must upload the KloudBuster VM image to your OpenStack cloud under test. KloudBuster needs one "universal" test VM image (referred to as "KloudBuster image") that contains the necessary test software. The KloudBuster image is then instantiated by the KloudBuster application in potentially large number of VMs using the appropriate role (HTTP server, HTTP traffic generator...).

Pre-built VM images are available for download from the OpenStack App Catalog.

Note: The same KloudBuster VM image can be instantiated for running the test functions (HTTP servers, HTTP traffic generators, file access tools) and for running KloudBuster as a web service.

Note: If your OpenStack Glance is able to access the Internet and you only use the CLI to launch KloudBuster, you can skip this section (KloudBuster CLI will request Glance to download the image from the OpenStack App Catalog when it is not present in Glance).

Download the KloudBuster VM image to the local directory

You must download a local copy if your OpenStack cloud does not have dirct access to the Internet. Download the latest image directly from OpenStack App Catalog using your favorite browser (search for "kloudbuster") or using wget. KloudBuster VM images are qcow2 images named "kloudbuster_v<version>.qcow2" (e.g. "kloudbuster_v6.qcow2"). Look for an image named with the "kloudbuster_v" prefix and download the latest version from the list.

Example for downloading the v6 image using wget:

```
wget http://storage.apps.openstack.org/images/kloudbuster_v6.qcow2
```

Upload the KloudBuster VM image using the Horizon Dashboard

From the dashboard, create a new image and select either "Image File" if you want to uplaod from the local copy of the image or "Image Location" if you want to upload directly from the OpenStack App Catalog (you will need the complete URL of the image).

The name of the image in Glance *must* match exactly the image name in the App Catalog (without the .qcow2 extension, e.g. "kloudbuster_v6").

Upload the KloudBuster VM image using the Glance CLI

This assumes that you have installed the OpenStack Glance API and have sourced the appropriate openrc file.

To upload the image from a local copy of that image using the Glance CLI:

```
glance image-create --file kloudbuster_v6.qcow2 --disk-format qcow2 --container-
↪format bare --visibility public --name kloudbuster_v6
```

CHAPTER 5

Configuration Options

Default HTTP Scale Test

The default HTTP scale test will run on a single cloud and perform the following steps:

- Create 2 tenants, 2 users, and 2 routers;
- Create 1 shared network for both servers and clients tenants
- Create 1 VM running the Redis server (for orchestration)
- Create 1 VM running as an HTTP server
- Create 1 VM running the HTTP traffic generator (defaults to 1000 connections, 1000 requests per second, and 30 seconds duration)
- Measure/aggregate throughput and latency
- Bring down and cleanup

Default Storage Scale Test

The default storage scale test will use the following settings:

- Create 1 tenant
- Create 1 router
- Create 1 private network
- Create 1 VM and attach a 10 GB Cinder volume to it
- **Perform the default storage workload sequence:**
  - **random access 4KB block size, IO depth 4, 100 IOPs for 30 seconds**
    - 100% read

        * 100% write
        * 70% read, 30% write
    - **sequential access 64KB block size, IO depth 64, 60 MB/sec for 30 seconds**
        * 100% read
        * 100% write
        * 70% read, 30% write
- Measure/aggregate IOPs, throughput and latency
- Bring down and cleanup

The run should take a few minutes (depending on how fast the cloud can create the resources) and you should see the actions taken by KloudBuster displayed on the console. Once this minimal scale test passes, you can tackle more elaborate scale testing by increasing the scale numbers or providing various traffic shaping options. See below sections for more details about configuring KloudBuster.

KloudBuster Configuration File

To create a custom scale test configuration, make a copy of the default configuration (this can be obtained by redirecting the output of *–show-config* to a new file, as described in the quick start guide) and modify that file to satisfy our own needs. The configuration file follows the yaml syntax and contains options that are documented using yaml comments.

Note: The default configuration is always loaded by KloudBuster and any default option can be overridden by providing a custom configuration file that only contains modified options. So you can delete all the lines in the configuration file that you do not intend to change

---

Once modified, you can pass the configuration file to KloudBuster using the *–config* option.

General Configuration Options

Each item in the configuration file is well documented. Below is just a quick-start on some important config items that need to be paid more attention to.

- **vm_creation_concurrency**

This controls the level of concurrency when creating VMs. There is no recommended values, as it really varies and up to the cloud performance. On a well-deployed cloud, you may able to push the values to more than 50. The default value of 5 concurrent VM creations should be OK for most deployments.

Note: For deployment prior to Kilo release, you may hit this bug if the concurrency level is too high. Try to lower down the value if you are hitting this issue.

- **server:number_tenants, server:routers_per_tenant, server:networks_per_router, server:vms_per_network**

These are the four key values which controls the scale of the cloud you are going to create. Depends on how you want the VM to be created, sets these values differently. For example, if we want to create 180 Server VMs, we could do either of the following settings:

(1) 30 tenants, 1 router per tenant, 2 networks per router, and 3 VMs per network (so-called 30*1*2*3);

(2) 20 tenants, 3 routers per tenant, 3 networks per router, and 1 VMs per network (so-called 20*3*3*1);

- **server:secgroups_per_network**

Reference Neutron router implementation is using IPTABLES to perform security controls, which should be OK for small scale networks. This setting for now is to investigate the upper limit capacity that Neutron can handle. Keep the default to 1 if you don't have the concerns on this part yet.

- **client:progression**

KloudBuster will give multiple runs (progression) on the cloud under this mode.

If enabled, KloudBuster will start with certain amount of VMs, and put more VMs into the testing for every iteration. The increment of the VM count is specified by *vm_multiple*. The iteration will continue until it reaches the scale defined in the upper sections, or the stop limit.

The stop limit is used for KloudBuster to determine when to stop the progression, and do the cleanup if needed earlier.

In the case of HTTP testing:

> It is defines as: [number_of_err_packets, percentile_of_packet_not_timeout(%)]. For example: [50, 99.99] means, KloudBuster will continue the progression run only if **ALL** below conditions are satisfied:
>
> 1. The error count of packets are less or equal than 50;
> 2. 99.99% of the packets are within the timeout range;

In the case of Storage testing:

> It is a single integer indicating the degrading percentile. In the mode of random read and random write, this value indicates the percentile of degrading on IOPS, while in the mode of sequential read and sequential write, this value indicates the percentile of degrading on throughput.
>
> Assume the IOPS or throughput per VM is a fixed value, usually we are expecting higher values when the VM count grows. At certain point where the capacity of storage is reached, the overall performance will start to degrade.
>
> e.g. In the random read and random write mode, for example the IOPS is limited to 100 IOPS/VM. In the iteration of 10 VMs, the requested IOPS for the system is 100 * 10 = 1000. However, the measured IOPS is degraded to only 800 IOPS. So the degraded percentile is calculated as 800/1000=20% for this set of data.

HTTP Test Specific Options

- **client:http_tool_configs**

This section controls how the HTTP traffic will be generated. Below are the two values which determine the traffic:

```
# Connections to be kept concurrently per VM
connections: 1000
# Rate limit in RPS per client (0 for unlimited)
rate_limit: 1000
```

Each testing VM will have its targeting HTTP server for sending the requests. Simply to say, connections determines the how many concurrent users that the tool is emulating, and rate_limit determines how fast the HTTP request will be sent. If the connections are more than the capacity of the cloud can handle, socket errors or timeouts will occur; if the requests are sending too fast, you will likely to have lots of requests responded very slow (will be reflected in the latency distribution spectrum generated by KloudBuster).

Different cloud has different capacity to handle data plane traffics. The best practice is to have an estimate first, and get started. In a typical 10GE VLAN deployment, the line rate is about 9Gbps, or 1.125 GB/s. For pure HTTP traffic,

the effective rate minus the overhead is approximately 80% of the line rate, which is about 920 MB/s. Each HTTP request will consume 32KB traffic for loading the HTML page (HTML payload size is configurable), so the cloud capacity is about 30,000 req/sec. If you are staging a cloud with 20 testing pairs, the rate_limit for each VM settings will be about (30000 / 20 = 1500).

The capacity for handling connections varies among factors including kernel tuning, server software, server configs, etc. and hard to have an estimate. It is simple to start with the same count as the rate_limit to have (1 request/connection) for each VM, and we can adjust it later to find out the maximum value. If you see socket errors or timeouts, means the scale you are testing is more than the cloud capacity.

Some other values which are self-explained, and you can change them as needed.

Storage Test Specific Options

- **client:storage_stage_configs**

This section defines the storage specific configs in the staging phase:

```
# The number of VMs for running storage tests
vm_count: 1
# KloudBuster supports to run storage tests on Cinder Volumes or Ephemeral
# Disks. Available options to be configured: ['volume', 'ephemeral'].
target: 'volume'
# Volumes size in GB for each VM
disk_size: 10
# The size of the test file for running IO tests in GB. Must be less or
# equal than disk_size.
io_file_size: 1
```

- **client:storage_tool_configs**

This section controls how the Storage tests will be performed. All the fields are self-explained, and you can create your own test case with customized parameters.

CHAPTER 6

KloudBuster Standard Scale Profile

Standard scale profile definition

Multiple factors can impact data plane scale numbers measured by KloudBuster: VM count, number of connections per VM, number of requests per seconds per VM, timeout, etc... To help obtaining quick and easy results without having to tweak too many parameters, KloudBuster defines an off the shelf *default scale profile*.

In the default scale profile for running HTTP load:

- The number of connections per VM is set to 1000;
- The number of requests per seconds per VM is set to 1000;
- The HTTP request timeout is set to 5 seconds;
- The stop limit for progression runs will be error packets greater than 50;
- The size of the HTML page in the server VMs will be 32768 Bytes;

As a reference, KloudBuster can run approximately 21 VMs (with 21,000 connections and 21,000 HTTP requests/sec) and achieve approximately 5 Gbps of HTTP throughput on a typical multi-node Kilo OpenStack deployment (LinuxBridge + VLAN, 10GE NIC card).

In the default scale profile for running Storage load:

- A standard set of 6 test cases (random read/write/mixed, sequential read/write/mixed);
- The IOPS limit per VM is set to 100 for random read/write/mixed test cases, and Rate limit per VM is set to 60MB/s for sequential read/write/mixed test cases;
- Block size is set to 4K for random read/write/mixed test cases, and 64K for sequential read/write/mixed test cases;
- IO Depth is set to 4 for random read/write/mixed test cases, and 64 for sequential read/write/mixed test cases;
- The stop limit for progression runs is degrading more than 20% of the target;

Note that it is hard to give a reference on storage testing since the performance varies a lot on different hardware or solutions.

How to run the standard scale profile

In order to perform a run using the default scale profile, set the max VM counts for the test, enable progression run and leave everything else with their default values. KloudBuster will start the iteration until reaching the stop limit or the max scale. Eventually, once the KloudBuster run is finished, the cloud performance can be told by looking at how many VMs KloudBuster can run to and by looking at the latency charts.

Steps:

1. Enable progression runs:

   Running from CLI: Edit the config file, and set **client:progression:enabled** to True

   Running from Web UI: Navigate to "Interactive Mode" from the top menu bar, unfold the left panel for detail settings, under "Progression Test" section, and check the "Progression Test" checkbox.

2. Set up the max scale:

   The max scale basically means the max VM counts that KloudBuster will try to reach. In the case of HTTP testing, for a typical 10GE NIC card with VLAN encapsulation, 25 will be a good value; in the case of Storage testing, depends on the solution the deployment is using, pick a number from 10 to 25 would usually be fine. Remember you can always adjust it to a more reasonable value based on your deployment details.

   Running from CLI: Edit the config file, and set **server:vms_per_network** to a proper value.

   Running from Web UI: Navigate to "Interactive Mode" from the top menu bar, unfold the left panel for detail settings, under "Staging Settings" section, and set "VMs/Network" to a proper value.

Interpret the results

From the CLI, check the log and find the warning that KloudBuster gave, similar to this:

```
WARNING KloudBuster is stopping the iteration because the result reaches the stop␣
↪limit.
```

One line before is the json output of last successful run, which has the number in the "total_server_vms" field.

From the Web UI, in the "Interactive Mode" tab, you will see how many sets of data are you getting. The second last set of data shows the last successful run, which has the number in the "Server VMs" column.

CHAPTER 7

Advanced Features

Control the VM Placement

By default, VMs are placed by NOVA using its own scheduling logic. However, traffic can be shaped precisely to fill the appropriate network links by using specific configuration settings. KloudBuster can change that behavior, and force NOVA to place VMs on desired hypervisors as we defined by supplying the topology file.

The format of the topology file is relatively simple, and group into two sections. See file "cfg.topo.yaml" for an example.

The "servers_rack" section contains the hypervisors that the server side VMs will be spawned on, and the "clients_rack" section contains the hypervisors that the client side VMs will be spawned on. The hypervisor names can be obtained from Horizon dashboard, or via "*nova hypervisor-list*". Note that the name in the config files must exactly match the name shown in Horizon dashboard or NOVA API output.

A typical use case is to place all server VMs on one rack, and all client VMs on the other rack to test Rack-to-Rack performance. Similarly, all server VMs on one host, and all client VMs on the other host to test the Host-to-Host performance.

To use this feature, just pass *-t <path_to_topo_file>* to the kloudbuster command line.

Note: Admin access is required to use this feature.

Running KloudBuster without admin access

When there is no admin access to the cloud under test, KloudBuster does support to run and reused the existing tenant and user for running tests. You have to ask the cloud admin one time to create the resources in advance, and KloudBuster will create the resources using the pre-created tenant/user.

When running under the tenant/user reusing mode:

- Only one tenant will be used for hosting both server cloud and client cloud resources;

- Only two users will be used for creating resources, and each cloud has its own user;

And also there are some limitations that you should aware:

- The VM placement feature will not be supported;
- The flavor configs will be ignored, and the KloudBuster will automatically pick the closest flavor settings from the existing list;
- KloudBuster will not automatically adjust the tenant quota, and give warnings when quota exceeded;

See file "cfg.tenants.yaml" for an example. Modify the settings to match your cloud.

To use this feature, just pass *-l <path_to_tenants_file>* to the kloudbuster command line.

Displaying the Results

Results can be saved in a file in json format or in HTML format. The json format is more appropriate for usage by any post-processing tool or script while the HTML file is more adapted for human usage.

The KloudBuster Web UI will display the results using charts and tables when the test is finished running. The KloudBuster CLI provides an option to generate the HTML file from the results (*–html* option). It can also generate the HTML file from the JSON results (*–charts-from-json* option).

Examples of running KloudBuster

Assuming the OpenStack RC file is stored at ~/admin_openrc.sh, and the password is "admin". Running the program is relatively easy, some examples are given to help get started quickly.

Note: Before going to large scale test, it is strongly recommended to start with a small scale. The default config is a good point to start with. It will make sure KloudBuster is talking to the clouds well.

Example 1: Single-cloud Mode

Kloudbuster will create both server VMs and client VMs in the same cloud if only one RC file is provided:

```
$ kloudbuster --tested-rc ~/admin_openrc.sh --tested-passwd admin
```

Example 2: Dual-cloud Mode, Save results

Assume the cloud for server VMs is ~/admin_openrc1.sh, and the cloud for client VMs is ~/admin_openrc2.sh. The password for both clouds is "admin". Also save the results to a JSON file once the run is finished:

```
$ kloudbuster --tested-rc ~/admin_openrc1.sh --tested-passwd admin --testing-rc ~/
↪admin_openrc2.sh --testing-passwd admin --json result.json
```

Example 3: Single-cloud Mode, Customized VM placements

```
$ kloudbuster --tested-rc ~/admin_openrc.sh --tested-passwd admin -t cfg.topo.yaml
```

Example 4: Single-cloud Mode, Running storage test, Save results to JSON

```
$ kloudbuster --tested-rc ~/aio-openrc.sh --tested-passwd lab --storage --json aio.
↪json
```

CHAPTER 8

OpenStack Resources Cleanup

KloudBuster may exit with resources lingering in the cloud under test when there are uncaught exceptions or when the configuration file explicitly disables any resource cleanup upon exit (this option can be useful for debugging for example).

KloudBuster provides a time saving *force_cleanup* python script to cleanup resources created by a previous KlousBuster run. This script can also be used to cleanup OpenStack resources with a name matching a given regular expression.

Resources in a given selection set are deleted along with their dependencies in the correct order. For example to delete a router you need to delete first all the interfaces before you can delete the router. To delete a volume you need to first detach the volume (if attached) before it can be deleted. Furthermore, some resource deletions require dependent resources to be actually deleted first (which can tale more or less time) before they can succeed. A volume detach command for example can take time and if you do not want long enough the volume deletion will fail.

The script takes care of all these dependencies and timing considerations.

The current version of the script can delete the following resources with a name that matches a given regular expression:

- **Storage**
  - volumes (detach and delete)
- **Compute**
  - instances
  - flavors
  - key pairs
- **Network**
  - security groups
  - floating IPs
  - routers (including all associated interfaces)
  - networks

- **Keystone:**
  - users
  - tenants

In some cases, because of timing reasons, you may have to run the force_cleanup script a few times to get rid of all selected resources.

How to Select Resources to Delete

Resource list (–file <pathname>)

KloudBuster generates a cleanup log file when it exits without deleting all resources. This file is a text file with 1 row per resource, where each row has the following format>:

```
<resource type>|<resource name>|<uuid>
```

Example of cleanup log file:

```
flavors|kb.client|58dededb-4c04-444f-8779-d0487ff08035
flavors|kb.proxy|2613f998-9a77-4f49-bd1b-7cd83c1038cf
keypairs|KBc-T0-U-K|
users|KBc-T0-U|c50be73385a84acdb5bdfa565d2b613c
routers|KBc-T0-U-R0|15678e46-a437-42d1-96a7-a57e9b96edcd
floating_ips|10.23.228.204|7a93a1ba-2356-4dbe-a387-006e90be4462
instances|KB-PROXY|67cb2da6-b05e-40b5-bc8f-df061035b945
instances|KBc-T0-U-R0-N0-I0|ab895ccb-1632-42ed-8fa2-d0dd08b15641
instances|KBc-T0-U-R0-N0-I1|ea416057-3a16-4f1c-875d-f4e0bb5b55c8
instances|KBc-T0-U-R0-N0-I3|a95a9ff9-5c9c-470b-b6a3-a729e0cd7857
instances|KBc-T0-U-R0-N0-I2|aff78e4f-cf59-49c9-81b3-53c9c2417d78
instances|KBc-T0-U-R0-N0-I15|bda2b43e-fc5f-448a-a7ca-f152f2c62bb3
instances|KBc-T0-U-R0-N0-I16|02e93acc-89b8-4bcf-94af-ec2369aee6b5
volumes|KBc-T0-U-R0-N0-V0|ebd5f46e-7cfd-4bd4-a140-196a8cd3df38
volumes|KBc-T0-U-R0-N0-V1|df84e730-1a1c-4b01-bd44-315d7128959a
volumes|KBc-T0-U-R0-N0-V17|5c1b1765-248d-4f08-8b3c-4f43aa9bcc0d
volumes|KBc-T0-U-R0-N0-V18|1c681d85-de43-4407-be97-98cc6a8f5a73
volumes|KBc-T0-U-R0-N0-V19|291edb4a-9a79-40ed-b193-7e3e9b08e1f6
sec_groups|KBc-T0-U-R0-N0-SG0|1a97ee38-bc10-4ca1-b7cc-8da09991595b
tenants|KBc-T0|59d98fa36536490a8746c517b3ed7383
networks|KBc-T0-U-R0-N0|7047d34c-ad77-4453-8d69-5dd41b102159
```

If such file is provided to the cleanup script using the *–file* option, only the resources described in the file will be deleted.

Discovery with Resource name filter (–filter <regex>)

If no cleanup log file is provided, resources are discovered from OpenStack and selected using a regular expression on the resource name (–filter <regex>). You can specify any valid python regular expression to select the resource by name.

If you do not specify a cleanup log file nor a filter, the script will discover all resources with a name starting with "KB" which is the prefix for all KloudBuster resources.

Some examples (refer to the python regex documentation for a detailed description of regular expressions):

Regular expression	(default) any OpenStack resource with a name starting with "KB
ext$	any OpenStack resource with a name starting ending with "ext"
.*net	any resource with a name containing "net" in any position
glancelneutron	any resource with a name starting with "glance" or "neutron"

Warning: You can of course also specify '.*' to list all resources but you probably do not want to delete all of them!

Credentials (RC) file (–rc <pathname>)

Specify the openrc file (downloaded from the Horizon API Access page) to provide the credentials to access OpenStack. Alternatively you can also source that file from the shell before invoking the force_cleanup.py script.

Dry Run (–dryrun)

The script also provides a dry run mode, meaning that you can just check what the script would do without actually deleting anything.

Installation and Dependencies

The script is available in the OpenStack KloudBuster repository under kloudbuster/force_cleanup.py. If you need to run the script outside of the usual KloudBuster installation, the script requires the usual OpenStack python client libraries and credentials.py (from the kloudbuster module). Otherwise, pick one of the kloudbuster installation method to install the script (the KloudBuster docker container looks to be the simplest).

Known Issues and Limitations

Volumes attached to instances that are no longer present cannot be deleted thorugh Nova or Cinder APIs. Such volumes will show up as attached to "None" and in the "in-use" or "available" state from the Horizon dashboard. In this case, the script will print a warning with the volume ID:

```
WARNING: Volume 6080fdce-f894-4c41-9bc0-70120e8560a8 attached to an instance that no␣
↪longer exists  (will require manual cleanup of the database)
```

Cleanup of such volumes will require first setting the attach_status of the corresponding volume to "detached" in the Cinder database directly. You have to SSH to the controller host, and login to the MySQL shell:

```
[root@gg34-2 ~]# mysql cinder
MariaDB [cinder]> UPDATE volumes SET attach_status='detached' WHERE id='18ed7f10-be49-
↪4569-9e04-2fc4a654efee';
```

Then re-run the script (or manually delete the volume from Horizon).

Examples

KloudBuster resources cleanup:

```
$ python force_cleanup.py -r admin-openrc.sh
Please enter your OpenStack Password:
Discovering Storage resources...
Discovering Compute resources...
Discovering Network resources...
Discovering Keystone resources...


SELECTED RESOURCES:
+------------+--------------------+--------------------------------------+
| Type       | Name               | UUID                                 |
|------------+--------------------+--------------------------------------|
| volumes    | KBc-T0-U-R0-N0-V34 | 8a7746b1-5c31-4db8-b80e-58baeb21b2e9 |
| volumes    | KBc-T0-U-R0-N0-V36 | b1f007e6-e46f-4b25-beca-8418f8680377 |
| volumes    | KBc-T0-U-R0-N0-V4  | 5168c8fb-2124-4c00-9365-0767551a1861 |
| volumes    | KBc-T0-U-R0-N0-V3  | d02dd62b-cd12-4e75-8356-cf41f3d3bc86 |
| volumes    | KBc-T0-U-R0-N0-V7  | 32f50b20-3d8c-46f8-8e0e-1e642fe52a67 |
| volumes    | KBc-T0-U-R0-N0-V5  | 4ee5710f-8cb6-454d-8661-ac5daa0dec35 |
| volumes    | KBc-T0-U-R0-N0-V31 | 5eae2777-6680-4d63-907f-9b9280bdab36 |
| volumes    | KBc-T0-U-R0-N0-V17 | cd44d985-468c-4d15-a26a-3205966f56bf |
| volumes    | KBc-T0-U-R0-N0-V29 | 20cfd301-6f24-4727-a2e6-ec4c7979f24a |
| volumes    | KBc-T0-U-R0-N0-V9  | ab7a09cd-4176-4119-89bb-44f22e42ac57 |
| volumes    | KBc-T0-U-R0-N0-V1  | 467c6203-b30a-460d-9654-79e3798814ad |
| volumes    | KBc-T0-U-R0-N0-V13 | 9b8c1697-a691-4ca8-b8aa-0ba5126f4330 |
| volumes    | KBc-T0-U-R0-N0-V20 | 2fae40bd-b7f8-4ad0-8b49-28199cc20219 |
| volumes    | KBc-T0-U-R0-N0-V33 | 29949338-9fb0-4a6f-8df5-65a97cfc5b5c |
| volumes    | KBc-T0-U-R0-N0-V10 | 562a7f29-e0d4-479d-a916-deb7b062d826 |
| volumes    | KBc-T0-U-R0-N0-V35 | 9643b353-ac1b-4088-940d-babdfed8239a |
| volumes    | KBc-T0-U-R0-N0-V25 | 1d605aed-ad92-469a-a3ae-d8763793b764 |
| volumes    | KBc-T0-U-R0-N0-V22 | 895ba475-debb-4b06-9372-dabebfd26b1c |
| volumes    | KBc-T0-U-R0-N0-V6  | f0c3659a-b9ef-4b15-a015-35fc845a8509 |
| volumes    | KBc-T0-U-R0-N0-V37 | df749f20-f2a9-4d8e-b1c5-667c3c64bf15 |
| volumes    | KBc-T0-U-R0-N0-V32 | 5cca56d7-9543-470e-a964-1f6a314ee3a7 |
| volumes    | KBc-T0-U-R0-N0-V0  | eb4e82d7-131e-417a-9bbb-0aedbd3c2263 |
| volumes    | KBc-T0-U-R0-N0-V38 | 65737d70-c41d-4a3d-853e-ae4c9ecae44d |
| volumes    | KBc-T0-U-R0-N0-V23 | 04c5bcdb-49b5-4006-9479-1f15b530cfcc |
| volumes    | KBc-T0-U-R0-N0-V11 | 181c2dc4-56fd-4f42-ab5d-5e9f9b8a3be5 |
| volumes    | KBc-T0-U-R0-N0-V18 | 6f78f429-6603-4dba-9fa0-cbc601c170a1 |
| volumes    | KBc-T0-U-R0-N0-V39 | b9878b28-9a34-43b0-a5ea-46f7598b23f7 |
| volumes    | KBc-T0-U-R0-N0-V19 | 1a2ef52a-a990-4cb8-974e-2e7bfde07e64 |
| volumes    | KBc-T0-U-R0-N0-V12 | 78761313-89d0-47df-b8a6-6d6baac5a48d |
| volumes    | KBc-T0-U-R0-N0-V8  | 712c06bb-75a1-4d3b-8e7e-1d1845e2636e |
| volumes    | KBc-T0-U-R0-N0-V30 | baaffd6c-ed0c-41c8-9f81-a59e8cef8318 |
| volumes    | KBc-T0-U-R0-N0-V21 | 4ef6e3fd-e102-45f2-b69f-cc28049667b4 |
| volumes    | KBc-T0-U-R0-N0-V28 | 728edd5d-df01-4eae-8811-1e8e0c1357d6 |
| volumes    | KBc-T0-U-R0-N0-V14 | 33fe1128-a4da-4d68-b3fe-e160856c2b46 |
| volumes    | KBc-T0-U-R0-N0-V15 | 7fac9831-2ade-487f-9c79-126b5981df5a |
| volumes    | KBc-T0-U-R0-N0-V26 | 801f95d4-1100-4bbd-9ec1-5fbe925b70d5 |
| volumes    | KBc-T0-U-R0-N0-V27 | 61802296-9201-4d7a-aeda-62f2ad8b2de2 |
| volumes    | KBc-T0-U-R0-N0-V24 | 9fab9127-a496-41ad-b8ab-7bdc83d0df7e |
| volumes    | KBc-T0-U-R0-N0-V2  | ed95d6c3-497e-4e5f-99b1-8f9c5bd82a54 |
| volumes    | KBc-T0-U-R0-N0-V16 | 7083ac1d-1383-4a6f-b95c-cc11c5fe4eda |
| sec_groups | KBc-T0-U-R0-N0-SG0 | b324ce05-384a-40e5-95f9-4e7e9dccb9d8 |
| routers    | KBc-T0-U-R0        | 143a6fc6-5558-41c9-90cf-a08c4d26d37e |
| networks   | KBc-T0-U-R0-N0     | d300fe6d-260b-4a99-99bc-a6a187c0fbc3 |
| tenants    | KBc-T0             | 5d344c4be893420d9d94c7434143b09d     |
| users      | KBc-T0-U           | d26097b180c64e34b80bfa4e73418267     |
+------------+--------------------+--------------------------------------+
```

```
Warning: You didn't specify a resource list file as the input. The script will delete␣
↪all resources shown above.
Are you sure? (y/n) y
*** STORAGE cleanup
    + VOLUME KBc-T0-U-R0-N0-V34 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V36 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V4 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V3 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V7 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V5 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V31 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V17 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V29 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V9 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V1 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V13 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V20 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V33 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V10 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V35 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V25 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V22 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V6 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V37 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V32 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V0 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V38 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V23 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V11 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V18 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V39 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V19 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V12 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V8 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V30 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V21 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V28 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V14 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V15 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V26 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V27 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V24 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V2 is successfully deleted
    + VOLUME KBc-T0-U-R0-N0-V16 is successfully deleted
*** COMPUTE cleanup
*** NETWORK cleanup
    + SECURITY GROUP KBc-T0-U-R0-N0-SG0 is successfully deleted
    + Router Gateway KBc-T0-U-R0 is successfully deleted
    + Router Interface 10.1.0.3 is successfully deleted
    + ROUTER KBc-T0-U-R0 is successfully deleted
    + NETWORK KBc-T0-U-R0-N0 is successfully deleted
*** KEYSTONE cleanup
    + USER KBc-T0-U is successfully deleted
    + TENANT KBc-T0 is successfully deleted
```

Delete all resources with a name starting with “HA”:

```
$ python force_cleanup.py -r admin-openrc.sh --filter 'HA'
Discovering Storage resources...
Discovering Compute resources...
Discovering Network resources...
Discovering Keystone resources...

SELECTED RESOURCES:
+----------+-------------------------------------------------------+----------------------
↪-----------------+
| Type     | Name                                                  | UUID                 ␣
↪                 |
|----------+-------------------------------------------------------+----------------------
↪-----------------|
| networks | HA network tenant b4d72c4ec4254c789ee11700e3f6d7a4 | ed2912db-4a56-4673-
↪828c-c825e9f8d7ac |
| networks | HA network tenant 890190a4482448d197606d663702efc2 | 32ee3483-8aee-4a97-
↪a2d2-62ac7e521c67 |
| networks | HA network tenant 0550a6a1045a40a1aa9cf3b92731ef00 | 586cc6e2-eec8-4927-
↪8100-993027b6c925 |
| networks | HA network tenant 3c0a953100964440ac1bc8c1611ce96e | fa3ff23e-7a62-458d-
↪911f-299f938685a0 |
| networks | HA network tenant 74a1ec7f4155403cbb482ea6be857295 | 09cee2bc-a2b7-4680-
↪a6f0-542881f0fcd2 |
| networks | HA network tenant 45f2158c9fd2496ab68c51ef69d0cb80 | df6e0506-9ede-4df9-
↪adc1-11f3046a94c6 |
| networks | HA network tenant 19dec7d3b39c48ef85b9d5e2500361f5 | 227c1e27-b117-43d6-
↪9f0e-e1bd11993c05 |
| networks | HA network tenant 5d344c4be893420d9d94c7434143b09d | c3c2eebb-95b0-4a0c-
↪b700-5591b4992ce1 |
+----------+-------------------------------------------------------+----------------------
↪-----------------+

Warning: You didn't specify a resource list file as the input. The script will delete␣
↪all resources shown above.
Are you sure? (y/n) y
*** STORAGE cleanup
*** COMPUTE cleanup
*** NETWORK cleanup
    + NETWORK HA network tenant b4d72c4ec4254c789ee11700e3f6d7a4 is successfully␣
↪deleted
    + NETWORK HA network tenant 890190a4482448d197606d663702efc2 is successfully␣
↪deleted
    + NETWORK HA network tenant 0550a6a1045a40a1aa9cf3b92731ef00 is successfully␣
↪deleted
    + NETWORK HA network tenant 3c0a953100964440ac1bc8c1611ce96e is successfully␣
↪deleted
    + NETWORK HA network tenant 74a1ec7f4155403cbb482ea6be857295 is successfully␣
↪deleted
    + NETWORK HA network tenant 45f2158c9fd2496ab68c51ef69d0cb80 is successfully␣
↪deleted
    + NETWORK HA network tenant 19dec7d3b39c48ef85b9d5e2500361f5 is successfully␣
↪deleted
    + NETWORK HA network tenant 5d344c4be893420d9d94c7434143b09d is successfully␣
↪deleted
*** KEYSTONE cleanup
```

Dry run mode, regular expression, environment variable credentials, find all resources with a name ending with "ext":

```
$ python force_cleanup.py --dryrun --filter '.*ext$'
Discovering Storage resources...
Discovering Compute resources...
Discovering Network resources...
Discovering Keystone resources...

!!! DRY RUN - RESOURCES WILL BE CHECKED BUT WILL NOT BE DELETED !!!

SELECTED RESOURCES:
+----------+-------------+--------------------------------------+
| Type     | Name        | UUID                                 |
|----------+-------------+--------------------------------------|
| networks | storm-b-ext | a9e91d24-bb21-4321-a0d5-3408d15b25b4 |
+----------+-------------+--------------------------------------+


*** STORAGE cleanup
*** COMPUTE cleanup
*** NETWORK cleanup
    + NETWORK storm-b-ext should be deleted (but is not deleted: dry run)
*** KEYSTONE cleanup
```

CHAPTER 9

Frequently Asked Questions

KloudBuster in a nutshell?

A self contained, fully automated and open source OpenStack VM-level tenant network and storage scale measurement tool.

Why is a tool like KloudBuster useful?

Before KloudBuster it was practically very difficult and time consuming to load an OpenStack cloud at a scale that reflects large deployments with traffic on the data plane or the storage plane and to measure its impact where it counts: at the VM application level. To the point that practically very few people would take the pain of conducting such experiment except perhaps for very small scale setups (such as single rack, 2 compute nodes). Just to give an idea, to replicate manually what a 15-minute KloudBuster run can do on a medium size cloud (2 racks, less than 100 nodes and 40GE links), would require at the very least several days of hard-to-repeat work assuming the person doing that test knows how to do all the different small tasks needed to obtain similar results:

- create a VM image with the appropriate test packages (not trivial to find which packages to use)
- create all the tenants/users/routers/resources/VMs
- place the VMs in a way that makes sense for scale testing (such as rack based placement)
- provision the test VMs with the right configuration
- orchestrate the test VMs to start at the same time (cannot be manual due to the scale, we're talking about hundreds of client VMs to coordinate)
- repatriate all the results from all the client VMs when the test is finished (which itself can represent a very large amount of data)
- consolidate all the results in a way that makes sense system wise and is easy to interpret
- present results in a nice digestible format

And this is just for the simplest of the KloudBuster runs. Dual cloud scale testing (where 1 testing cloud loads the cloud under test to scale up the North South traffic) requires juggling with 2 OpenStack clouds at the same time, KloudBuster handles that mode by design and as easily as the single-cloud scale test. Latest features add support for periodic reporting (e.g. latency stats every 5 seconds), server mode with RESTFUL API control by external orchestrators (this is required for any form of automated HA testing) or scale progressions (e.g. collect latency numbers for 10,000 clients to 200,000 clients by increment of 10,000 clients, at that level of scale recreating every resource set from scratch at every iteration is going to take too much time). All of these advanced features are clearly impossible to do manually or semi-manually.

What do you mean by comprehensive end to end scale testing?

You could start with a completely idle OpenStack cloud with zero resources and zero data plane traffic (as if you just deployed OpenStack on it). Within minutes you have a cloud that is fully loaded with tenants, users, routers, networks, VMs and with all network pipes filled to capacity (if the network architecture and configuration is tuned properly) with a massive amount of live HTTP traffic or storage traffic. After the test, you revert back to the original idle state and you have a nice HTML report with the data plane characterization of your cloud or charts representing the scalability of your storage back end viewed from VM applications.

How scalable is KloudBuster itself?

All the runs so far have shown bottlenecks residing in the cloud under test. KloudBuster is designed to scale to an extremely large number of VM end points thanks to the use of an efficient distributed key value store and the associated publish/subscribe service (Redis) for the scale orchestration. Redis has shown to scale to thousands of subscribers without any problem while more traditional scaling tools that use SSH to control the end points will have trouble keeping up past a few hundred sessions.

General Usage Questions

Is there a way to prevent KloudBuster from deleting all the resources?

In cfg.scale.yaml, there is a “cleanup_resources” property which is True by default. Set it to False and KloudBuster won’t clean up the resources after the run.

Is there a way to cleanup all lingering resources created by KloudBuster?

All resources created by KlousBuster have a “KB_” prefix in their name. The “force_cleanup” script will clean up all resources that have a name starting with “KB_”.

How are KloudBuster VM images managed?

KloudBuster VM images are built using OpenStack diskimage-builder (or DIB) and have a version (single number). The default name of an image is “kloudbuster_v<version>” (e.g. “kludbuster_v6”). Normally each KloudBuster application is associated to a recommended KloudBuster VM image version.

This is indicated in the output of –version:

```
$ python kloudbuster.py --version
6.0.3, VM image: kloudbuster_v6
```

In this example, the KloudBuster application is version 6.0.3 and the matching VM image is v6. By default KloudBuster will use the Glance image that is named "kloudbuster_v6" (this can be overridden in the configuration file).

Note that if the user specifies a different VM image version in the configuration file, a warning will be issued to indicate that there might be some incompatibilities (but the run will proceed):

```
2015-08-26 10:47:10.915 38100 INFO kb_runner [-] Waiting for agents on VMs to come up.
↪..
2015-08-26 10:47:15.918 38100 INFO kb_runner [-] 0 Succeed, 0 Failed, 1 Pending...␣
↪Retry #0
2015-08-26 10:47:20.920 38100 INFO kb_runner [-] 1 Succeed, 0 Failed, 0 Pending...␣
↪Retry #1
2015-08-26 10:47:20.961 38100 WARNING kb_runner [-] The VM image you are running (2.
↪0) is not the expected version (6) this may cause some incompatibilities
```

It is recommended to always use the appropriate VM image version to avoid any potential incompatibility.

HTTP Data Plane Testing

How many TCP connections exactly are created, how many requests are generated and how long do the connections stay?

KloudBuster will create the exact number of HTTP connections configured and will keep them active and open until the end of the scale test. There is a 1:1 mapping between an HTTP connection/client and 1 TCP connection (the same TCP connection is reused for all requests sent by the same client). For example, with 100 HTTP servers, 1000 HTTP connections and 500 HTTP requests/sec per HTTP server, the total number of simultaneous HTTP connections will be 100,000 at any time during the scale test and the number of HTTP requests generated will be 50,000 rps.

Why pick wrk2 to generate HTTP traffic?

This tool was chosen among many other open source tools because it was tested to be the most scalable (highest number of connections/rps per CPU) and provided very accurate HTTP throughput and latency results (which cannot be said of most other tools - see FAQ on how latency is calculated).

Storage Scale Testing

What kind of VM storage are supported?

KloudBuster cam measure the performance of ephemeral disks and Cinder attached volumes at scale.

How to measure the fastest IOPs or Throughput from a VM ?

This feature is only available from the CLI by using a properly defined configuration file. To measure the fastest IOPs, omit the "rate_iops" and "rate" parameters from the workload definition in the configuration file.

The file kloudbuster/cfg.1GB.yaml provides and example of configuration file to measure the highest IOPs and throughput for random/sequential, read/write for 1 VM on 1 1GB file residing on an attached volume.

How to interpret the generated results in json?

General parameters:

- test_mode: is always "storage"
- storage_target: indicates if the storage used is a Cinder block storage ("volume") or an ephemeral disk,
- time: time the test was executed
- version: KloudBuster version
- tool: the FIO version used to generate the results
- block_size: the unit in the value indicates the unit (e.g "4k" = 4 kilobytes)
- iodepth: number of in-flight operations,
- total_client_vms: total number of VMs running an FIO client
- timeout_vms: number of VM/fio clients that did not return a result within the allocated time (this parameter is absent if there was no VM timing out, should not be present for most runs)

These parameters represent aggregated values for all VMs (to get a per VM count, divide the value by the number of client Vms (total_client_vms):

- read_runtime_ms, write_runtime_ms: aggregated time the fio tests ran in msec as measured by fio
- rate_iops: aggregated requested number of IOPs, 0 or missing = unlimited (i.e. test as high as possible)
- read_iops, wrote_iops: aggregated read or write IO operations per second as measured by fio (if rate_iops is not zero, will be <= rate_iops)
- rate: aggregated requested kilobytes per second, 0 or missing = unlimited (i.e. test as high as possible)
- read_bw, write_bw: aggregated read or write bandwidth in KB/sec (if rate is not zero, will be <= rate)
- read_KB, write_KB: aggregated number of kilobytes read or written as measured by fio

Latency values are reported using a list of pre-defined percentiles:

- read_hist: a list of pairs where each pair has a percentile value and a latency value in micro-seconds e.g. [99.9, 1032] indicates that 99.9% of all I/O operations will take 1032 usec or less to complete

Common Pitfalls and Limitations

AuthorizationFailure and SSL Exception when running KloudBuster

```
2016-05-12 17:20:30 CRITICAL AuthorizationFailure: Authorization Failed: SSL␣
↪exception connecting to https://172.29.86.5:5000/v2.0/tokens: [SSL: CERTIFICATE_
↪VERIFY_FAILED] certificate verify failed (_ssl.c:765)
```

This exception most likely indicates that the OpenStack API uses SSL and that the CA certificate file is missing in the openrc file used. Check that the openrc file used:

- has OS_AUTH_URL using https
- either has OS_CACERT missing or pointing to an invalid or missing certificate file path

To fix this you will need to have the OS_CACERT variable in your openrc file point to a valid certificate file (you will need to get this certificate file from the cloud admin).

Creating the image with diskimage-builder fails with an "import yaml" error

This error means that the python PyYAML package is not installed or that your /etc/sudoers file is configured in a way that causes a sudo script in diskimage- builder to fail. To check if PyYAML is installed: pip list | grep PyYAML If PyYAML is installed, comment out this line in /etc/sudoers (use "sudo visudo" to modify that file):

```
#Defaults   secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
```

CHAPTER 10

Development

Build the KloudBuster VM Image

This section describes how to rebuild:

- the KloudButer VM Image from the git repository source code
- the KloudBuster Docker image

Build on Linux

Your Linux server must have python, git and qemu utilities installed.

Ubuntu/Debian based:

```
$ sudo apt-get install python-dev git qemu-utils
$ # Source the virtual environment if you have one
$ pip install PyYAML
```

Redhat/Fedora/CentOS based:

```
$ sudo yum install python-devel git qemu-img
$ # Source the virtual environment if you have one
$ pip install PyYAML
```

Build the image with below commands:

```
$ # Clone the kloudbuster repository if you have not done so
$ git clone https://github.com/openstack/kloudbuster.git
$ # Go to the dib directory
$ cd kloudbuster/kb_dib
$ # Run the build image script, which will install DIB and start the build
$ ./build-image.sh
```

After a few minutes, the qcow2 image will be built and available in the same directory. You can then upload it to OpenStack using the glance CLI.

If you get an error message saying that import yaml fails (seems to happen only on Ubuntu):

```
dib-run-parts Thu Jul 2 09:27:50 PDT 2015 Running /tmp/image.ewtpa5DW/hooks/extra-
↪data.d/99-squash-package-install

"/tmp/image.ewtpa5DW/hooks/extra-data.d/../bin/package-installs-squash",
line 26, in <module>
    import yaml
ImportError: No module named yaml
```

You need to comment out the secure_path option in your /etc/sudoers file (use "sudo visudo" to edit that file):

```
#Defaults   secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
```

Build on MacOSX

You need to install first:

- Virtualbox
- Vagrant

And build the image with below commands:

```
$ # Clone the kloudbuster repository if you have not done so
$ git clone https://github.com/openstack/kloudbuster.git
$ # Go to the dib directory
$ cd kloudbuster/kb_dib
$ # Run vagrant and start building the image
$ vagrant up
```

After a few minutes (depending on virtualbox overhead), the qcow2 image will be built and available in the same directory. You can then upload it to OpenStack using the glance CLI, destroy the vagrant VM ("vagrant destroy") and dispose of the kloudbuster directory (if no longer needed).

Build the KloudBuster Docker Container Image

The KloudBuster Docker images are published in the DockerHub berrypatch repository: https://hub.docker.com/r/berrypatch/kloudbuster/

The Dockerfile at the root of the git repository can be used to build a new container based on Ubuntu 14.04.

To build for tag 6.0.3 (replace as needed with the real tag), go to the root of the repository then execute the docker build command:

```
sudo docker build --tag=berrypatch/kloudbuster:6.0.3 .
```

To publish you need to be a member of the berrypatch kloudbuster team. After the login (requires your DockerHub username and password), push the appropriate version to berrypatch:

```
sudo docker login
sudo docker push berrypatch/kloudbuster:6.0.3
```

CHAPTER 11

Contributing

Contribute to KloudBuster

Below are a simplified version of the workflow to work on KloudBuster. For complete instructions, you have to follow the Developer's Guide in OpenStack official documents. Refer to *below section* for links.

Start working

Before starting, a GitHub/OpenStack respository based installation must be done. Refer *here* for detailed documentation.

1. From the root of your workspace, check out a new branch to work on:

```
$ git checkout -b <TOPIC-BRANCH>
```

2. Happy working on your code for features or bugfixes;

Before Commit

There are some criteria that are enforced to commit to KloudBuster. Below commands will perform the check and make sure your code complys with it.

3. PEP 8:

```
$ tox -epep8
```

Note: The first run usually takes longer, as tox will create a new virtual environment and download all dependencies. Once that is the done, further run will be very fast.

4. Run the test suite:

```
$ tox -epython27
```

5. If you made a documentation change (i.e. changes to .rst files), make sure the documentation is built as you expected:

```
$ cd <kloudbuster-ws-root>/doc
$ make html
```

Once finished, the documentation in HTML format will be ready at <kloudbuster-ws-root>/doc/build/html.

Submit Review

6. Commit the code:

```
$ git commit -a
```

Note: For a feature commit, please supply a clear commit message indicating what the feature is; for a bugfix commit, please also containing a launchpad link to the bug you are working on.

7. Submit the review:

```
$ git review <TOPIC-BRANCH>
```

The members in the KloudBuster team will get notified once the Jenkin verification is passed. So watch your email from the review site, as it will contain the updates for your submission.

8. If the code is approved with a +2 review, Gerrit will automatically merge your code.

File Bugs

Bugs should be filed on Launchpad, not GitHub:

https://bugs.launchpad.net/kloudbuster

Build the KloudBuster Docker Image

Two files are used to build the Docker image: *Dockerfile* and *.dockerignore*. The former provides all the build instructions while the latter provides the list of files/directories that should not be copied to the Docker image.

In order to make the Docker image clean, remove all auto generated files from the root of your workspace first. Specify the image name and the tag, and feed them to docker build. Examples to build the image with name "berrypatch/kloudbuster", tag "6.0.4":

```
$ cd <kloudbuster-ws-root>
$ sudo docker build --tag=berrypatch/kloudbuster:6.0.4 .
```

The images should be available for use:

```
$ sudo docker images
REPOSITORY               TAG                 IMAGE ID            CREATED
↪VIRTUAL SIZE
berrypatch/kloudbuster   6.0.4               0f17ae788c69        8 minutes ago
↪443.1 MB
```

To push the new image to the KloudBuster Docker Hub repository (berrypatch), you need to login to Docker Hub (sudo docker login) and you need to have write access to the berrypatch/kloudbuster repository before you can push the new container:

```
sudo docker login
sudo docker push berrypatch/kloudbuster:6.0.4
```

It is also good practice to build and override the latest tag:

```
sudo docker build --tag=berrypatch/kloudbuster:latest .
sudo docker push berrypatch/kloudbuster:latest
```

Developer's Guide of OpenStack

Feedbacks and contributions to KloudBuster are welcome.

KloudBuster follows the same workflow as any other OpenStack project.

If you would like to contribute to the development of OpenStack, you must follow the steps in this page:

> http://docs.openstack.org/infra/manual/developers.html

If you already have a good understanding of how the system works and your OpenStack accounts are set up, you can skip to the development workflow section of this documentation to learn how changes to OpenStack should be submitted for review via the Gerrit tool:

> http://docs.openstack.org/infra/manual/developers.html#development-workflow

Pull requests submitted through GitHub will be ignored.

CHAPTER 12

Indices and tables

- genindex
- modindex
- search