Measuring Cloud Network Performance with PerfKit Benchmarker

A Google Cloud Technical White Paper
January 2020

Google Cloud

For more information visit **google.com/cloud**

Table of Contents

1. Abstract

Network performance is of vital importance to any business or user operating or running part of their infrastructure in a cloud environment. As such, it is also important to test the performance of a cloud provider's network before deploying a potentially large number of virtual machines and other components of virtual infrastructure. Researchers at SMU's AT&T Center for Virtualization (see smu.edu/provost/virtualization) have been working in conjunction with a team at Google to run network performance benchmarks across various cloud providers using automation built around PerfKit Benchmarker (see github.com/GoogleCloudPlatform/PerfKitBenchmarker) to track changes in network performance over time. This paper explains how cloud adopters can perform their own benchmarking.

2. Introduction

When choosing a cloud provider, users are often faced with the task of figuring out which one best suits their needs. Beyond looking at the advertised metrics, many users will want to test these claims for themselves or see if a provider can handle the demands of their specific use case. This brings about the challenge of benchmarking the performance of different cloud providers, configuring environments, running tests, achieving consistent results, and sifting through the gathered data. Setting up these environments and navigating the APIs and portals of multiple different cloud providers can escalate this challenge and takes time and skill. Despite the sometimes difficult nature of this, benchmarking is a necessary endeavor.

This document demonstrates how to run a variety of network benchmarks on the largest public cloud providers using PerfKit Benchmarker (PKB). We begin with an overview of the PKB architecture and how to get started running tests, then describe specific test configurations to cover a variety of deployment scenarios. These configurations can be used to immediately compare the performance of different use cases, or run on a schedule to track network performance over time.

2.1 PerfKit Benchmarker

PerfKit Benchmarker is an open source tool created at Google that allows users to easily run benchmarks on various cloud providers without having to manually set up the infrastructure required for those benchmarks. PerfKit Benchmarker follows the 5 step process detailed in **Figure 1** to automate each benchmark run. The Configuration phase processes command line flags, configuration files, and benchmark defaults to establish the final specification used for the run. The Provisioning phase creates the networks, subnets, firewalls and firewall rules, virtual machines, drives, and other cloud resources required to run the test. Benchmark binaries and dependencies like datasets are also loaded in this phase. The Execution phase is responsible for running the benchmarks themselves, and Teardown releases any resources created during the Provision phase. The Publishing phase packages the test results into a format suitable for further analysis such as loading into a reporting system. The metadata returned from the Publishing phase can include verbose details about the actual infrastructure used during the test and timing information for each phase of the run along with the metrics returned from the benchmark itself, providing the level of detail needed to understand the benchmark results in context.

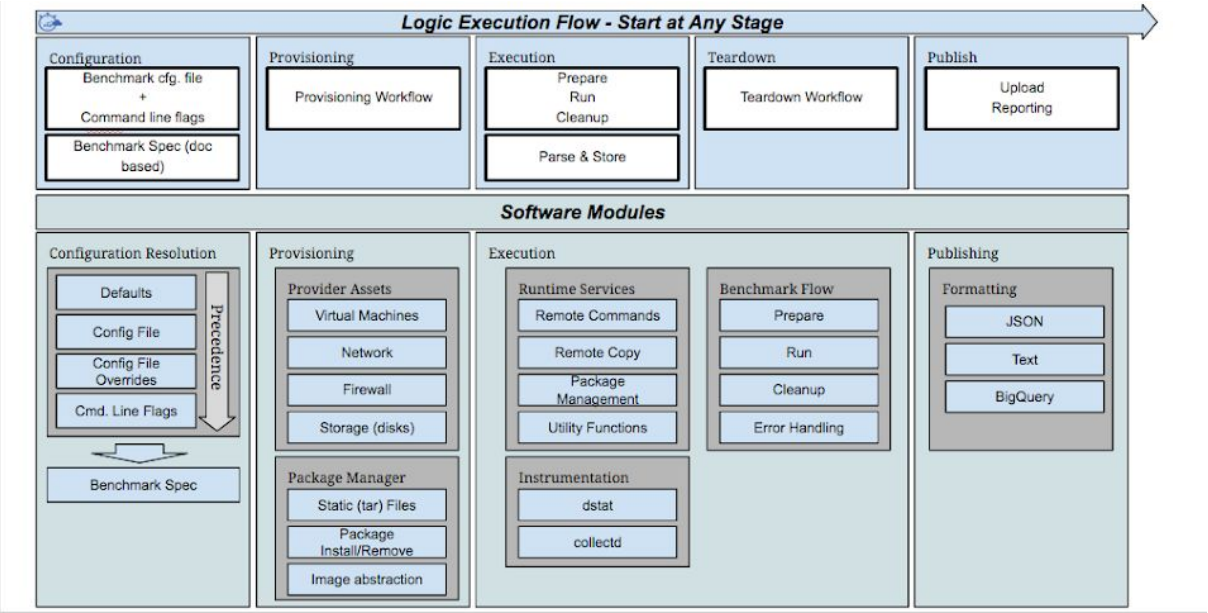


Fig. 1: PerfKitBenchmarker Architecture Diagram

Perfkit Benchmarker, along with an additional automation framework built around it, allows you to schedule and automate a large number of tests on a daily basis.

2.1.1 PerfKit Benchmarker Basic Example

Once PKB has been downloaded from github.com/GoogleCloudPlatform/PerfKitBenchmarker and its dependencies have been installed following the directions on that page, running a single benchmark with PerfKit Benchmarker is simple. You give it the benchmark you want to run and where you want to run it. For example, here is a ping benchmark between two VMs that will be located in zone us-east1-b on Google Cloud Platform:

```
./pkb.py --benchmarks=ping --zones=us-east1-b --cloud=GCP
```

If the zone or cloud is not given, a default value will be used. You can also specify the machine type with the **--machine_type** flag. If this is not set, a default single CPU VM will be used.

2.1.2 PerfKit Benchmarker Example with Config file

For more complicated benchmark setups, users define configurations using files in the .yaml format, as shown in the following example.

At the top of the config file is the benchmark that is being run. Next, give it the name of a flag matrix to use, in this case we'll call it fmatrix. Then define a filter to apply to the flag matrix and define the flag matrix itself. PKB works by taking the lists defined for each flag in the matrix (in our case this is zones, extra_zones, and machine_type) and finding every combination of those flags. It will then run the benchmark once for each combination of flags defined under fmatrix, as long as it evaluates to true with the flag matrix filters. The flags defined under ***flags*** at the bottom will be used for all benchmarks runs.

```
netperf:
  flag_matrix: fmatrix
  flag_matrix_filters:
      fmatrix: "zones != extra_zones"
  flag_matrix_defs:
      fmatrix:
        zones: [us-west1-a, us-west1-b,us-west1-c]
        extra_zones: [us-west1-a, us-west1-b,us-west1-c]
  flags:
    cloud: GCP
    netperf_histogram_buckets: 1000
    netperf_benchmarks: TCP_RR,TCP_STREAM,UDP_RR,UDP_STREAM
    netperf_test_length: 30
    netperf_num_streams: 1,4,32
    machine_type: n1-standard-16
    netperf_tcp_stream_send_size_in_bytes: 131072
```

This config file can be run with the command:

```
./pkb.py --benchmarks=netperf --benchmark_config_file=interzone_us_west1.yaml
```

Using this config file will run netperf TCP_RR, TCP_STREAM, UDP_RR and UDP_STREAM between pairs of n1-standard-16 instances in the us-west1-a, us-west1-b and us-west1-c zones. Because of the flag matrix filter, it will exclude tests from the same zone. Each test will be of 30 seconds duration and will be repeated for 1, 4, and 32 parallel streams. So from one config file and command line, we will get 72 benchmarks run (6 zone combinations * 4 netperf benchmarks * 3 different stream counts).

In the following sections of this paper, we will see several more examples of how to run specific tests with PKB. Generally, they all use this same format; the structure and parameters of the benchmark are defined in a config file and a relatively simple command is used to start the benchmark with the specified config file.

3. Benchmark Configurations

All of the benchmarks that are presented here are simple and easy to reproduce should anyone want to run their own tests. In this section we will discuss the configurations for various test runs.

There are several general types of network benchmarks you may want to run, including: same zone (intra-zone), cross zone (inter-zone), and cross region (inter-region) tests. Intra-zone tests are between VMs within the same zone, which usually means that they are situated in the same datacenter. Inter-zone tests run between VMs in different zones within the same cloud region and Inter-region tests run between VMs in separate cloud regions. These kinds of groupings are necessary as network performance can vary dramatically across these three scales.

Additionally, benchmarks can be run to test network performance across VPN connections, on different levels of network performance tiers, using different server operating systems, and on Kubernetes clusters.

3.1 Basic Benchmark Specific Configurations

In this subsection, we cover the basic flags and configurations that are most commonly used for network tests. These benchmarks are fairly standard and used to gather metrics on latency, throughput, and packets per second.

3.1.1 Latency (ping and netperf TCP_RR)

Ping is a commonly used utility for measuring latency between two machines and uses ICMP. The flag you should know for running a ping benchmark is **--ping_also_run_using_external_ip=True**. Just as the name implies, this will tell PKB to get latency results using both internal and external IP addresses.

```
./pkb.py --benchmarks=ping --ping_also_run_using_external_ip=True
--zone=us-central1-a --zone=us-west1-b --cloud=GCP
```

Ping, with its default once-a-second measurement is quite sufficient for inter-region latency. If you wish to measure intra-region latency (either intra-zone or inter-zone) a netperf TCP_RR test will show results that are more representative of application-level performance.

```
./pkb.py --benchmarks=netperf --netperf_histogram_buckets=1000 \
--netperf_benchmarks=TCP_RR --netperf_test_length=60 \
--zone=us-west1-b --cloud=GCP
```

3.1.2 Throughput (iperf and netperf)

Both iperf and netperf can be used to gather throughput data about both TCP and UDP connections with various numbers of parallel streams, so that you can test single stream throughput performance as well as aggregate.

The relevant flags for iperf are shown in the following. The first sets the length of time the throughput tests are run (default: 60s) and the second flag sets how many threads iperf will use to send traffic (default: 1).

iperf_runtime_in_seconds=60
Iperf_sending_thread_count=<num_threads>

```
./pkb.py --benchmarks=iperf --iperf_runtime_in_seconds=120 \
--iperf_sending_thread_count=32 --zone=us-central1-a --cloud=GCP
```

To perform UDP tests or a request/response test in PKB, one should use netperf. We can also set the number of streams, the test length in seconds, which netperf benchmarks are being run, and how many buckets are in the optional histogram.

```
./pkb.py --benchmarks=netperf \
--netperf_histogram_buckets=1000 \
--netperf_benchmarks=TCP_STREAM,UDP_STREAM \
--netperf_test_length=30 \
--netperf_num_streams=4 \
--zone=us-central1-a --cloud=GCP
```

For any of the example benchmark configurations in sections 3.2 and after, you can use iperf instead of ping, ping instead of netperf, etc. depending on what type of metrics you would like to gather.

3.1.3 Packets per Second

Packets per second tests are performed using a script that runs multiple instances of netperf UDP request/response (UDP_RR) using small message sizes to achieve the maximum possible packets per second the VM can achieve in the configured situation. In PerfKit Benchmarker, it is called netperf_aggregate and uses 3 virtual machines to test packets per second performance, as can be seen in the configuration file:

```
netperf_aggregate:
  vm_groups:
    vm_1:
      vm_spec:
        GCP:
          machine_type: n1-standard-4
          zone: us-east4-b
    vm_2:
      vm_spec:
        GCP:
          machine_type: n1-standard-4
          zone: us-east4-c
    vm_3:
      vm_spec:
        GCP:
          machine_type: n1-standard-4
          zone: us-east4-c
```

This config file can be run with the following command:

```
./pkb.py --benchmarks=netperf_aggregate \
--benchmark_config_file=/path/to/config.yaml
```

At the time of this writing the packets per second benchmark is still a pending contribution, but should be available to use soon.

3.2 On-Premises to Cloud Benchmarks

On-premise to cloud benchmarks are highly specific to the user's location, so unlike most cloud to cloud benchmarks, you can't simply look up results on a table online. PerfKit Benchmarker makes it simple to setup your own benchmarking for your on-premise situation. There are two ways to perform On-Prem to Cloud Benchmarks within the paradigm of PerfKit Benchmarker. The first is to use a Static, On-Prem System (either VM or bare-metal). This requires you to set up said on-prem system and can ssh to it. Then in a config file, you can specify that machine be the static VM you have set up, and the other will be a VM that will be created on the cloud provider of your choice. A config file to run a netperf test between a sample static VM and a n1-standard-2 machine in GCP zone us-central1-a would look the following:

```
netperf:
  vm_groups:
    vm_1:
      static_vms:
        ip_address: 192.168.0.1
        ssh_private_key: <ssh_key>
        user_name: <username>
        zone: local
    vm_2:
      vm_spec:
        GCP:
          machine_type: n1-standard-2
          zone: us-central1-a
```

A potentially quicker option is to use Docker. If you have Docker installed, you can run tests between a Docker container running locally and a VM in the cloud. For this, the config file to use would look something like this:

```
netperf:
  vm_groups:
    vm_1:
      cloud: Docker
    vm_2:
      vm_spec:
        GCP:
```

```
          machine_type: n1-standard-2
          zone: us-central1-a
```

In the config file, specify two vm_groups: vm_1 and vm_2. In vm_1, tell it to use Docker as the cloud. In vm_2, use vm_spec to set the machine_type and zone manually, as shown in the example. Doing this will create a new Docker image if you have not used the Docker provider previously and a new Docker container on your local machine (wherever you execute PKB from) that will function as a VM for the benchmark.

The command to run the benchmark from either of the preceding config files would be

```
./pkb.py --benchmarks=netperf --benchmark_config_file=/path/to/config.yaml
```

3.3 Cross-cloud Benchmarks

If you use multiple cloud providers, it may be of interest to run cross cloud benchmarks. With PKB, this can be achieved simply with a config file similar to the one we used for the on prem to cloud with Docker benchmark.

```
netperf:
  vm_groups:
    vm_1:
      cloud: AWS
      vm_spec:
        AWS:
          machine_type: m4.4xlarge
          zone: us-east-1a
    vm_2:
      cloud: GCP
      vm_spec:
        GCP:
          machine_type: n1-standard-16
          zone: us-central1-a
```

This will create one VM on AWS and another on GCP with the specified machine types in the specified zones and run netperf between them. The command to run the benchmark would be:

```
./pkb.py --benchmarks=netperf --benchmark_config_file=/path/to/config.yaml
```

3.4 VPN Benchmarks

Running benchmarks across an IPSec VPN is possible using the PKB VPN service.
Base requirements for IPSec VPNs across the Internet:

- Public IP address on both ends of the tunnel.
- Unique subnet ranges behind each VPN GW. CIDRs can't overlap unless using multiple tunnels.
- Pre-shared key

By default, GCP and some other providers in PKB run benchmarks from within a single VPC and subnet range. To meet the requirement for mutually exclusive subnet ranges, you can distinguish using the **cidr** vm_group property in your benchmark config file as follows:

```
iperf:
  description: Run iperf on custom cidr
  vm_groups:
    vm_1:
      cloud: GCP
      cidr: 10.0.1.0/24
      vm_spec:
        GCP:
            zone: us-west1-b
            machine_type: n1-standard-4
    vm_2:
      cloud: GCP
      cidr: 192.168.1.0/24
      vm_spec:
        GCP:
            zone: us-central1-c
            machine_type: n1-standard-4
```

Then to establish the VPN for a benchmark config you can add --**use_vpn** to the flags passed to PKB and include the desired parameters to the **vpn_service** section of the configuration:

```
ping:
 description: Run ping over vpn
 flags:
   use_vpn: True
```

```
    vpn_service_gateway_count: 1
  vpn_service:
    tunnel_count: 2
    ike_version: 2
    routing_type: static
  vm_groups:
    vm_1:
      cloud: GCP
      cidr: 10.0.1.0/24
      vm_spec:
        GCP:
            zone: us-west1-b
            machine_type: n1-standard-4
    vm_2:
      cloud: GCP
      cidr: 192.168.1.0/24
      vm_spec:
        GCP:
            zone: us-central1-c
            machine_type: n1-standard-4
```

At the time of this writing VPN support is still a pending contribution to PerfKit Benchmarker, but should be available to use soon.

3.5 Kubernetes Benchmarks

There are two ways to execute Kubernetes tests on a cloud provider. The first is to create a Kubernetes cluster in the cloud provider and provide its config to PKB via the **`--kubeconfig=</path/to/.kube/config>`** flag. Using this method, PKB handles the setup and teardown of the Kubernetes pods, in the cluster you have setup manually. This will work for most benchmarks that you want to run on a cluster.

The second method involves using a config file that looks like the following with the benchmark **container_netperf**. Using this benchmark will set up a Kubernetes cluster for you and deploy pods that use a specialized netperf container image. In the config file, we have to specify the specs of both our containers that will be deployed and the cluster itself.

```
container_netperf:
  container_specs:
    netperf:
      image: netperf
      cpus: 2
      memory: 4GiB
```

```
  container_registry: {}
  container_cluster:
    vm_count: 2
    vm_spec:
      GCP:
        machine_type: n1-standard-4
        zone: us-east1-b
```

The command to run this benchmark will be:

```
./pkb.py --benchmarks=container_netperf \
--benchmark_config_file=</path/to/config.yaml>
```

3.6 Intra-zone Benchmarks

To run an intra-zone benchmark (two VMs in the same zone), you can simply specify the zone you want both VMs to be in and any other flags you want to specify. The following example runs an intra-zone netperf TCP_RR benchmark in GCP zone us-central1-a with n1-standard-4 machines. If you want to run another network benchmark, refer to section 3.1 for details on the flags available to use.

```
./pkb.py --benchmarks=netperf --cloud=GCP --zones=us-central1-a \
--machine_type=n1-standard-4 --netperf_benchmarks=TCP_RR
```

3.7 Inter-zone Benchmarks

Inter-zone tests, like most other tests can be executed in one of two ways. It can be done entirely from the command line using the --zone flag, as follows:

```
./pkb.py --benchmarks=iperf --cloud=GCP --zone=us-east4-b \
--zone=us-east4-c --machine_type=n1-standard-4
```

The same Inter-zone benchmark can also be set up using a config file:

```
iperf:
  vm_groups:
```

```
    vm_1:
      cloud: GCP
      vm_spec:
        GCP:
          machine_type: n1-standard-4
          zone: us-east4-b
    vm_2:
      cloud: GCP
      vm_spec:
        GCP:
          machine_type: n1-standard-4
          zone: us-east4-c
```

This config file can be run using the command:

```
./pkb.py --benchmarks=iperf --benchmark_config_file=/path/to/config.yaml
```

3.8 Inter-Region Benchmarks

Inter-Region benchmarks (between VMs located in separate geographic regions), can likewise be run using command line flags or with a config file.

```
./pkb.py --benchmarks=iperf --cloud=GCP --zone=us-central1-b \
--zone=us-east4-c --machine_type=n1-standard-4
```

The same Inter-Region benchmark can also be set up using the following config file:

```
iperf:
  vm_groups:
    vm_1:
      cloud: GCP
      vm_spec:
        GCP:
          machine_type: n1-standard-4
          zone: us-central1-b
    vm_2:
      cloud: GCP
      vm_spec:
        GCP:
          machine_type: n1-standard-4
          zone: us-east4-c
```

And this config file can be run with the following command:

```
./pkb.py --benchmarks=iperf --benchmark_config_file=/path/to/config.yaml
```

3.9 Multi Tier

Many cloud providers have multiple tiers of network performance. GCP has premium and standard tiers, which basically determines where traffic transitions between the Internet and Google's network, with the premium tier spending more time on Google's internal network. The network tier can be set with the **--gce_network_tier** flag. However, you will only see a difference between the tiers when testing between GCP and another network (cross cloud or on prem to cloud).

```
iperf:
  flags:
    Gce_network_tier: premium
  vm_groups:
    vm_1:
      cloud: AWS
      vm_spec:
        AWS:
          machine_type: m4.4xlarge
          zone: us-east-1a
    vm_2:
      cloud: GCP
      vm_spec:
        GCP:
          machine_type: n1-standard-16
          zone: us-central1-a
```

And this config file can be run with the following command:

```
./pkb.py --benchmarks=iperf --benchmark_config_file=/path/to/config.yaml
```

4. Inter-Region Latency Example and Results

As an illustrative example, we present the actual results of our Google Cloud all-region to all-region round trip latency tests, as shown in **Fig. 2.** This chart shows the average round trip

latency between regions from benchmarks run over the course of a month. The benchmarks were all executed on n1-standard-2 machine types with internal IP addresses. The statistics are all collected using PerfKit Benchmarker to run ping benchmarks between VMs in each pair of regions.

To reproduce this chart, you can run the following pkb command with the following config file. If you want to run a smaller subset of regions, just remove the regions you don't want included from the **zones** and **extra_zones** lists.

```
ping:
  flag_matrix: inter_region
  flag_matrix_filters:
    inter_region: "zones < extra_zones"
  flag_matrix_defs:
    inter_region:
      zones:
[asia-east1-a,asia-east2-a,asia-northeast1-a,asia-northeast2-a,asia-south1-a,asi
a-southeast1-a,australia-southeast1-a,europe-north1-a,europe-west1-c,europe-west
2-a,europe-west3-a,europe-west4-a,europe-west6-a,northamerica-northeast1-a,south
america-east1-a,us-central1-a,us-east1-b,us-east4-a,us-west1-a,us-west2-a]
      extra_zones:
[asia-east1-a,asia-east2-a,asia-northeast1-a,asia-northeast2-a,asia-south1-a,asi
a-southeast1-a,australia-southeast1-a,europe-north1-a,europe-west1-c,europe-west
2-a,europe-west3-a,europe-west4-a,europe-west6-a,northamerica-northeast1-a,south
america-east1-a,us-central1-a,us-east1-b,us-east4-a,us-west1-a,us-west2-a]

  flags:
    cloud: GCP
    machine_type: n1-standard-2
    ping_also_run_using_external_ip: True
```

You can also add the **--run_processes=<# of processes>** to tell it to run multiple benchmarks in parallel, but this will still likely take awhile (>12 hours). If you run too many benchmarks in parallel, you may run into quota issues, such as regional CPU quotas and per project subnet quotas, which limits you to around 8 processes. If you exceed a quota while running PKB, it will tell you the exception that was thrown and the benchmark will fail. Additionally, you can use the **`--gce_network_name=<network name>`** flag to have each benchmark use a GCP VPC that you have already created, so that each benchmark doesn't make their own, which adds up to a significant amount of time. This will also ensure that you don't run into subnet quota issues.

```
./pkb.py --benchmarks=ping --benchmark_config_file=/path/to/config.yaml
```

milliseconds	*receiving_region*																			
sending_region	asia-east1	asia-east2	asia-northeast1	asia-northeast2	asia-south1	asia-southeast1	australia-southeast1	europe-north1	europe-west1	europe-west2	europe-west3	europe-west4	europe-west6	northamerica-northeast1	southamerica-east1	us-central1	us-east1	us-east4	us-west1	us-west2
asia-east1		13	36	37	107	46	139	282	250	244	258	253	262	183	289	153	185	175	118	128
asia-east2	13		48	48	95	37	127	294	263	257	267	266	274	195	302	164	197	190	130	142
asia-northeast1	36	49		9	127	67	114	254	220	215	226	222	234	154	260	122	156	146	89	98
asia-northeast2	37	48	9		138	79	122	260	228	224	235	231	241	163	269	131	165	156	98	106
asia-south1	108	95	127	138		60	151	374	341	337	346	348	359	276	384	245	276	267	212	220
asia-southeast1	46	37	67	77	60		92	318	285	278	290	286	296	215	323	187	219	209	152	161
australia-southea	139	127	114	122	152	92		303	271	264	276	273	283	203	302	171	196	196	160	137
europe-north1	283	296	253	261	374	318	303		33	40	32	31	39	115	241	132	124	113	165	167
europe-west1	251	263	219	228	342	285	271	33		7	7	7	14	82	210	99	92	81	131	134
europe-west2	245	258	215	224	337	279	265	40	7		13	10	20	77	203	94	87	76	127	129
europe-west3	259	268	226	234	349	290	276	32	7	13		7	8	88	214	105	98	87	137	140
europe-west4	254	267	223	231	348	286	273	30	7	10	7		14	84	211	102	94	83	136	137
europe-west6	263	275	234	242	358	297	283	39	14	20	8	14		95	221	112	105	93	147	147
northamerica-nor	183	196	154	163	277	216	203	115	82	77	88	84	95		142	31	25	14	67	66
southamerica-ea	290	303	260	269	383	323	301	241	210	204	215	212	221	142		144	117	130	172	166
us-central1	154	164	122	131	245	188	172	131	99	94	105	103	111	31	143		35	25	34	35
us-east1	186	198	157	165	276	220	197	124	92	87	98	94	104	26	118	35		12	67	60
us-east4	176	190	146	156	267	209	196	113	81	76	86	84	93	14	129	25	12		58	60
us-west1	119	131	89	98	212	153	161	164	131	127	137	136	147	67	172	34	67	58		25
us-west2	128	142	98	106	219	161	137	166	133	129	140	137	146	66	166	36	60	60	26	

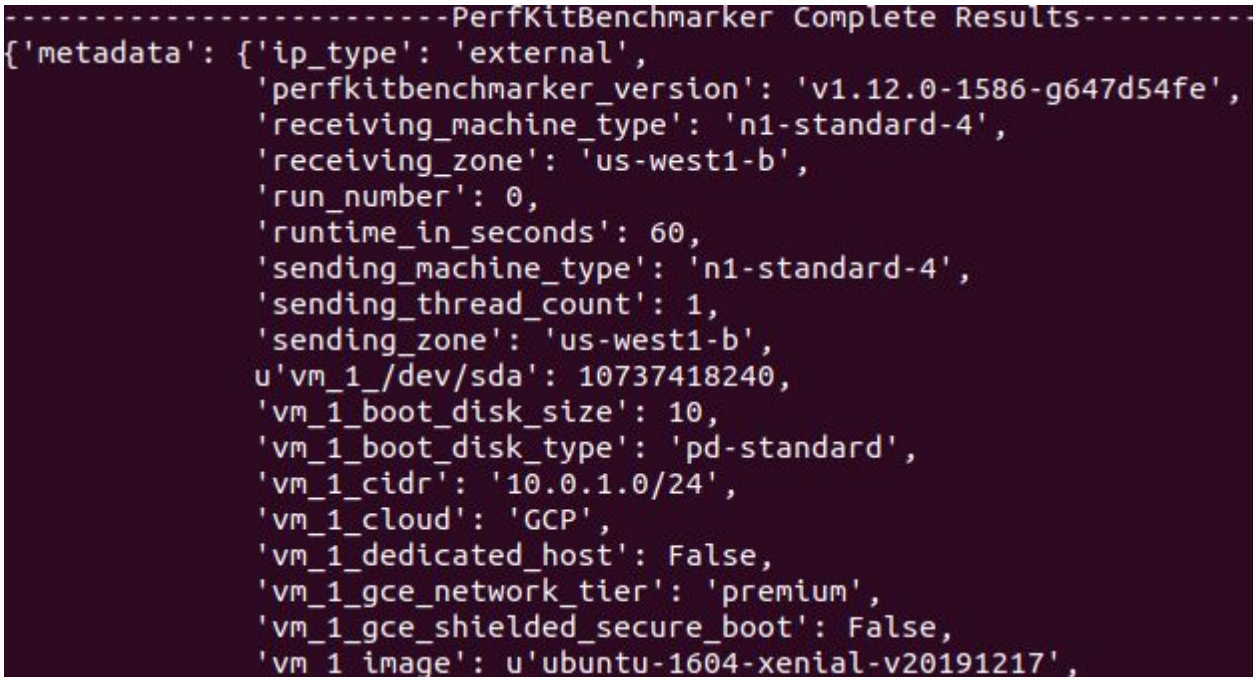
Fig. 2: Inter-Region Latency results for Google Cloud. All numbers are in milliseconds.

In the matrix shown in **Fig. 2,** The labels on the y-axis (left side) represent the sending region and the labels on the x-axis (across the top) represent the receiving region. So if we look at the intersection of asia-east2 on the y-axis and asia-east1 on the x-axis, this represents the average of results from ping benchmarks executed from a VM in asia-east2 to a VM in asia-east1.

5. Viewing and Analyzing Results

The report generated from a PKB run includes the results of the benchmark test along with a significant quantity of metadata about the test environment. The raw report is a JSON formatted

dictionary of key:value pairs. The default location for this file is <tmp_dir>/perfkitbenchmarker/runs/<run_uri>/perfkitbenchmarker_results.json



PKB includes a number of publishing targets as well, which can be specified on the command line when the test is launched to store the results in a backend like BigQuery or ElasticSearch automatically. It is then possible to query these runs from a dashboard provider to visualize the data.

5.1 Visualizing Results with BigQuery and Data Studio

To use the BigQuery PKB publisher, first create a BigQuery table in your GCP project (the schema will be created when you first push a sample), and then include the table name and project name in the PKB run flags:

```
./pkb.py --benchmarks=iperf --benchmark_config_file=/path/to/config.yaml
--bigquery_table=<bq.table> --bq_project=<bq.project>
```

The schema for each sample published by a run is described in the table below. Each run can (and usually does) produce multiple samples. In a network test like ping for example, the latency from zone_1 to zone_2 and the latency from zone_2 to zone_1 are recorded in separate samples. Likewise, there are separate samples created when using public and private networks, as well as samples that describe system metadata like lscpu and procmap. All of the samples for a single run share the same run_uri and can be joined on this field for grouping in queries.

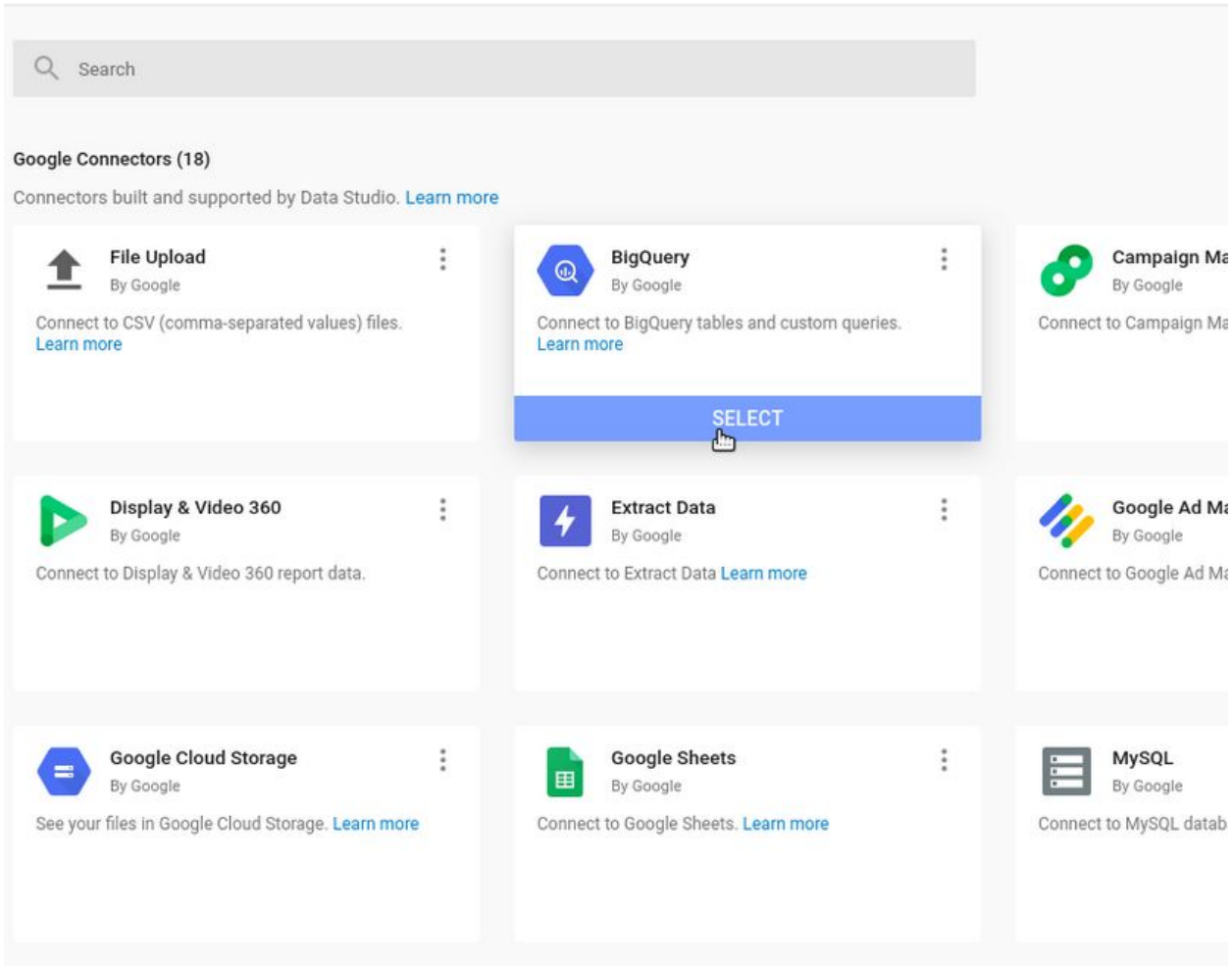
FIELD NAME	TYPE	MODE	DESCRIPTION

unit	STRING	NULLABLE	Unit type of the test/metric. (sec, ms, Mbit/sec, etc)
labels	STRING	NULLABLE	Catch all field that stores any information about the benchmark that does in any other field. This will contain a variety of information depending on the specific benchmark and test setup
timestamp	FLOAT	NULLABLE	Timestamp of benchmark in epoch time
product_name	STRING	NULLABLE	Name of the testing tool (this will always be 'PerfkitBenchmarker')
test	STRING	NULLABLE	Name of the specific benchmark that is being run (iperf, netperf, ping, etc)
official	BOOLEAN	NULLABLE	This will always be false
metric	STRING	NULLABLE	The specific metric that the value and unit type is for. (Avg latency, TCP Throughput, etc). A test can have multiple metrics.
value	FLOAT	NULLABLE	The value of the specific test and metric
owner	STRING	NULLABLE	The user who executed PerfKitBenchmarker
run_uri	STRING	NULLABLE	A unique value assigned to each test run
sample_uri	STRING	NULLABLE	A unique value assigned to each metric in each test run

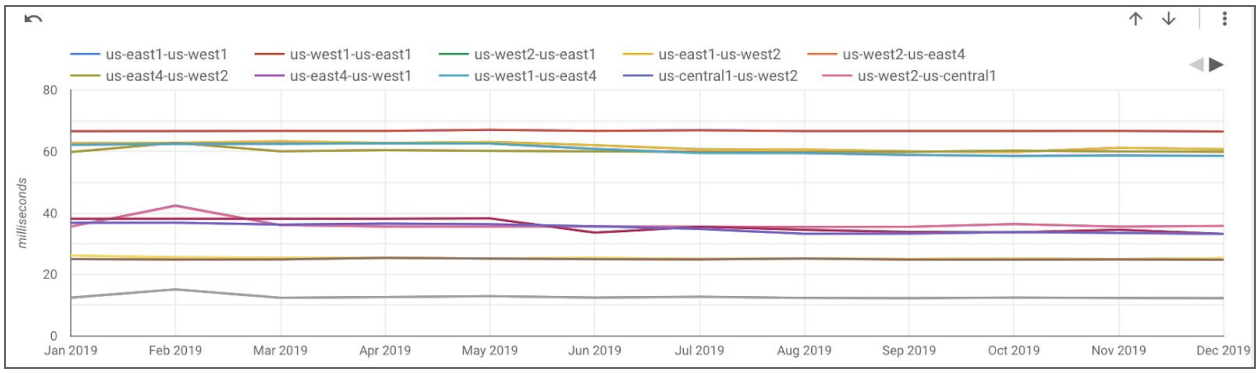
Once the table is populated you can query run results directly for reporting. If you are capturing several test types or tests with different parameters in the same table it may be useful to create views for each test used in your reports. The following BigQuery Standard SQL query shows how you can capture specific key:value pairs nested in the labels field and how to work with the time format for use in reporting.

```
SELECT
    value,
    unit,
    metric,
    test,
    TIMESTAMP_MICROS(CAST(timestamp * 1000000 AS int64)) AS thedate,
    REGEXP_EXTRACT(labels, r"\|vm_1_cloud:(.*?)\|") AS vm_1_cloud,
    REGEXP_EXTRACT(labels, r"\|vm_2_cloud:(.*?)\|") AS vm_2_cloud,
    REGEXP_EXTRACT(labels, r"\|sending_zone:(.*?)\|") AS sending_zone,
    REGEXP_EXTRACT(labels, r"\|receiving_zone:(.*?)\|") AS receiving_zone,
    REGEXP_EXTRACT(labels, r"\|sending_zone:(.*?-.*?)-.*?\|") AS sending_region,
    REGEXP_EXTRACT(labels, r"\|receiving_zone:(.*?-.*?)-.*?\|") AS receiving_region,
    REGEXP_EXTRACT(labels, r"\|vm_1_machine_type:(.*?)\|") AS machine_type,
    REGEXP_EXTRACT(labels, r"\|ip_type:(.*?)\|") AS ip_type
FROM <PROJECT>.<dataset>.<table>
```

To create a visualization using Data Studio, start by adding a connection to the BigQuery table you specified above. If using separate views, you can make each view its own data source.



Once Data Studio can see the PKB results table, you can design your charts and visualizations accordingly using the full range of reporting tools available. The example report below shows inter-region ping latency results:



Example PerfKit Benchmarker report in Google Data Studio

6. Summary

Perfkit Benchmarker simplifies cloud network performance testing, allowing you to collect your measurements of interest in an easy and repeatable manner. In this whitepaper we have covered benchmark testing network latency and throughput using familiar tools like iperf, netperf, and ping. The scenarios we described allow you to verify network performance claims within a single cloud, across cloud providers, or from your site to the cloud. For more information about PKB including the other available benchmarks (~100), supported cloud providers (~12), or to reach out to the community, please visit:
https://github.com/GoogleCloudPlatform/PerfKitBenchmarker

Authors & Acknowledgements

Authors: Derek Phanekham (SMU), Matthew Zaber (SMU), Suku Nair (SMU)

Reviewers/Contributors: Steve Deitz (Google), Rick Jones (Google), Manasa Chalasani (Google), Mike Truty (Google)

Publication Date: January 13, 2020