# Table of Contents

Table of Contents .................................................. 3
Revision History .................................................... 4
Integration information ............................................ 4
Introduction ......................................................... 5
Intended Audience .................................................. 5
Scope ................................................................. 5
Terminologies ......................................................... 5
Solution Introduction .............................................. 6
Solution Scope ....................................................... 7
Solution Limitations ............................................... 8
Solution Deployment .............................................. 9
CSO Interface Usage ............................................. 10
Test Architectures .................................................. 11
Functional Characterization ................................. 14
Performance Characterization .............................. 16
WRITE to IBM COS .............................................. 16
READ from IBM COS ............................................. 17
Performance Scaling ............................................ 19
Open Issues ......................................................... 21
Tools ................................................................. 21

Appendix A: Configuration Guide for
NGINX open source .............................................. 22
Assumptions ......................................................... 22
IBM COS ........................................................... 22
NGINX .............................................................. 22

Prerequisites ......................................................... 23
IBM COS ........................................................... 23
NGINX .............................................................. 23

Appendix B: Configuration Guide for
NGINX Plus ......................................................... 37
Assumptions ......................................................... 37
IBM COS ........................................................... 37
NGINX Plus ......................................................... 37

Prerequisites ......................................................... 38
IBM COS ........................................................... 38
NGINX Plus ......................................................... 38

Appendix-C: Highly Available Pair
Configuration Guide ............................................. 50
Assumptions ......................................................... 50
IBM COS ........................................................... 50
NGINX Plus ......................................................... 50

Prerequisites ......................................................... 51
IBM COS ........................................................... 51
NGINX Plus ......................................................... 51

References ......................................................... 60

Notices .............................................................. 61

Trademarks ......................................................... 63
Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>8/31/2017</td>
<td>Initial draft</td>
</tr>
<tr>
<td>0.2</td>
<td>9/15/2017</td>
<td>Draft read for internal review</td>
</tr>
<tr>
<td>0.3</td>
<td>10/02/2017</td>
<td>First draft ready for review</td>
</tr>
<tr>
<td>1.0</td>
<td>10/16/2017</td>
<td>Updates post review</td>
</tr>
</tbody>
</table>

Integration information

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration Engineer</td>
<td>Deanson Lin</td>
</tr>
<tr>
<td>Publication Date</td>
<td>10/16/2017</td>
</tr>
<tr>
<td>NGINX Review</td>
<td>10/3/2017</td>
</tr>
<tr>
<td>AIT Review</td>
<td>10/2/2017</td>
</tr>
<tr>
<td>Target ClevOS Version</td>
<td>ClevOS 3.10.2.96</td>
</tr>
<tr>
<td>Target Partner Product Version</td>
<td>Nginx/1.13.4 (nginx-plus-r13)</td>
</tr>
<tr>
<td>Target Partner Product OSS Version</td>
<td>Nginx/1.10.3 (Ubuntu)</td>
</tr>
</tbody>
</table>
Introduction
This document provides information on the inter-play between NGINX and IBM Cloud Object Storage for both On-Premise installations and public cloud in terms of functionality and performance. The actual procedure of integrating these two entities is documented in the configuration guide which is in Appendix A (NGINX Open Source), and Appendix B (NGINX Plus).

Intended Audience
This document is intended for anyone interested in understanding the functional and performance aspects of the integration of NGINX with IBM COS solutions.

Scope
This document assumes that the readers are aware the functional aspects of NGINX and NGINX Plus and wants to understand the performance aspects of the solution. Hence, the document focuses more on the performance aspects of the solution than the functional aspects of the integration. The intention of this document is to cover how you can use NGINX to reliably load balance incoming requests to IBM Cloud Object Storage.

This document is applicable to both the open source NGINX as well as the commercial NGINX Plus, as the performance characteristics and configuration are similar.

We will cover both a single open source NGINX instance for laboratory environments, as well as an HA Pair of NGINX Plus servers for production environments.

Terminologies
The following terms are helpful in understanding the NGINX/IBM COS solution:
• **NGINX OSS**: NGINX Open Source Software
• **VRRP**: Virtual Router Redundancy Protocol
• **Cloud storage**: TCP/IP based on-premise or cloud based storage pools
• **RTO**: Recovery Time Objective
• **RPO**: Recovery Point Objective
• **IDA**: Information Dispersal Algorithm
• **Manager**: performs FCAPS for IBM COS (Fault management, Configuration management, Accounting management, Performance management, Security management)
• **Accesser**: presents a stateless S3/CSO interface to client applications.
• **Slicestor**: IBM COS appliances that store slices of erasure coded objects
Solution Introduction

NGINX is an efficient asynchronous event-driven web server known for predictable performance under high loads. NGINX Plus, as presented in this solution guide, delivers an advanced load balancing solution for IBM Cloud Object Storage. NGINX OSS offers a basic subset of load balancing features as well but is limited in its advanced load balancing solutions compared to NGINX Plus.

Using core NGINX reverse proxy functionality available in both NGINX OSS and NGINX Plus, NGINX can serve as a high-performance load balancer supporting IBM COS workflows. Using NGINX OSS and NGINX Plus for load balancing is our primary focus in this solution guide.

The load balancer typically sits in between the client application(s) and the IBM COS Accessers.

NGINX offers high performance/high throughput load balancing for all object storage workflows. NGINX is also responsible for health checking IBM COS Accessers, and taking the accessers out of the load balanced resource pool when it detects the accessers not responding to requests properly via those health checks. Due to this functionality, the Client Application(s) can then treat NGINX’s Virtual IP as an always on object storage endpoint. This is an important function, as most client applications cannot be configured to use multiple endpoints simultaneously.

Using NGINX as a load balancer in front of IBM Cloud Object Storage allows client application(s) to scale near linearly simply by adding more accessers to the access pool.
Solution Scope

In our solution certification, we focused on one use case which we see NGINX / IBM Cloud Object Storage solution is the best fit for.

- Accesser Load Balancing

For lab environments, we recommend a single NGINX OSS instance. For production environments, we recommend a Highly Available Pair of NGINX Plus in Active/Standby configuration. As such, our solution guide will use NGINX Plus for the highly available load balancer. Open source NGINX can also be set up in an HA Pair using keepalived/pacemaker/corosync, similar to NGINX Plus, but we will not cover that in this guide.

NGINX OSS and NGINX Plus can also be configured as a high-performance content cache. Using NGINX as a content cache, can help offload high-IOPS requests to IBM COS, which can help scaling in certain use cases. Due to potential conflicts with IBM COS’s strong consistency guarantee, this would only be applicable only to very specific/custom workflows so we will not be covering using NGINX content caching in this guide.

We also wanted to point out that NGINX can also perform URL request/response rewrites natively or when used with the NGINX Lua module. This can be very useful for converting non-standard requests into standard S3 requests. We do have one current customer that uses NGINX in exactly this way, so their legacy software applications can use IBM COS without any application code changes. This isn’t a common use case, and we will not be covering REST request/response manipulation in this solution guide.
Solution Limitations

Currently, NGINX and NGINX Plus do not offer Global Server Load Balancing (GSLB). It is certainly possible to layer GSLB functionality on top of NGINX, but out of the box, NGINX doesn’t offer this feature.

NGINX Plus does offer specialized modules to help capture information about client location for other GSLB solutions to utilize.

We also wanted to point out the open source version of NGINX does not offer active health checks, and only supports passive health checks. We have seen good results with NGINX’s built in passive health checks. If active health checks are desired, NGINX Plus would be our recommendation.

We also wanted to point out that the Server Side Encryption with Customer Provided Encryption Keys (SSE-C), mandates that SSE-C headers are not being initiated over HTTP. As such, any SSE-C operation over HTTP will fail with a 400-level error (Bad Request). **So, if the customer workflow mandates SSE-C, then SSL termination at NGINX shouldn’t be performed.** If load balancing HTTPS with SSE-C is desired, NGINX can be configured in SSL passthrough mode directly to our accesser’s HTTPS endpoint. In SSL passthrough mode, you cannot add or modify HTTP headers, so you will lose client IP and other information contained in the X-forwarded-* headers. We will share the performance achieved by both NGINX terminating SSL as well as NGINX reverse proxying SSL directly to our accessers.
Solution Deployment

Typical deployment scenario for NGINX/IBM COS solutions are enterprise computing customers who are looking for an always on load balanced endpoint for their on-premise IBM Cloud Object Storage.

By using a load balancer, the customer no longer needs to worry about software upgrades affecting availability to IBM COS Accessers. Our IBM COS Manager already orchestrates software upgrades across an access pool, allowing only one Accesser to be upgraded at any single point in time. However, without a load balancer, the client application may still issue requests to an Accesser undergoing a software upgrade. The load balancer will detect, via passive health checks (NGINX OSS/NGINX Plus) and active health checks (NGINX Plus), that the Accesser is undergoing an upgrade, and will not send more requests to that Accesser until the Accesser is properly responding to requests. By using a load balancer in front of our Accessers, the customer can treat the NGINX’s virtual IP as an always-on endpoint.
CSO Interface Usage

NGINX itself doesn’t issue any REST API requests directly against our object storage. However, NGINX will pass through all request it receives itself from the client to our accessers.

Additionally, when running in SSL termination mode, we terminate SSL at the NGINX load balancer, and we add 2 additional headers to each request:
   - Host $host;
   - X-Forwarded-For $remote_addr;

In a packet capture, you can see the added headers:

```
Host: 172.20.26.104
X-Forwarded-For: 172.20.26.21
```

The Host header is just the server_name used in the NGINX configuration files. The way we have it configured is just the IP address of the NGINX load balancer.

The X-forwarded-For header, allows our internal IBM COS access logs to indicate which host the client request came from. Without this header, all incoming requests appear to come from the load balancer, and we have less client visibility. You’ll see the following in our access logs, with forwarded_for being the client’s IP address that made the request:

```
"server_name": "172.20.26.104",
"remote_address": "172.20.26.104",
"forwarded_for": "172.20.26.21",
```

Using this additional header allows for easier troubleshooting, as our access log will show the actual client IP address that issued the request. Note, when running in SSL passthrough mode, we can not add any additional headers, so you will not have this level of visibility when running in SSL passthrough mode.
Test Architectures

The following diagram depicts the topology of our testbed used for the performance tests.

**NGINX load balancer (Reverse Proxy)**

NGINX Hardware

Instance #1: The bare metal server hardware being used for the performance characterization was a repurposed IBM COS A4105. It has the following specifications:

- 48 cores, from 2x Intel® Xeon® CPU E5-2690 v3 2.6GHz
- 128 GiB RAM
- 120GB OS drive
- Intel XL710 40GbE QSFP+
Instance #2: This 2\textsuperscript{nd} instance was used for HA functional testing, as well as a VM to baremetal comparison. The server hardware being used for this are Dell FC630’s inside a PowerEdge FX2 chassis, provisioned with the following specifications:
- 48 cores, from 2x Intel\textregistered Xeon\textregistered CPU E5-2697A v4 2.6GHz
- 16-160 GiB RAM (For optimal performance, memory required depends on expected object sizes)
- 100GB OS drive.
- Intel XL710 40GbE QSFP+ in PCI-Passthrough mode

**NGINX Software**
- NGINX Plus \texttt{nginx/1.13.4 (nginx-plus-r13)}
- NGINX Open Source \texttt{nginx/1.10.3 (Ubuntu)}

**Load Generator Environment**

The load generator hardware being used to drive the performance tests are Dell FC830’s in a PowerEdge FX2 chassis, each provisioned with the following specifications:
- 64 cores, from 2x Intel\textregistered Xeon\textregistered CPU E5-4699 v4 2.2GHz
- 32 GiB RAM
- 16GB OS drive
- 500GB test staging drive
- 40Gbps networking via VMXNET3

The load generator hypervisor:
- VMWare ESXi 6.5 build 4887370

The load generator operating system:
- Ubuntu 16.04.2 LTS
- 4.4.0-62-generic #83-Ubuntu SMP

The load generator software:
- IBM COS OG v 1.4.1-SNAPSHOT
- ULIMIT=30000
- MEMORY=24000

**IBM COS – On-Premise**
- ClevOS 3.10.2.96
- 8x Accesser 4105 (40 Gbps client side links)
- 12x Slicestor 2448 – Single Site
- Vault Configuration
  - 12/7/9 IDA
  - 4.19 MB segment size
- SecureSlice: Disabled
- Versioning: Disabled
- Delete Restricted: No
- SSE-C: Disabled
- Name Index: Disabled
- Recovery Listing: Enabled
Functional Characterization

As stated earlier, this document focuses more on the performance aspects of the solution than the functional aspects of the integration. However, in addition to normal load balancing, we’ve also tested HA pair functionality with failover. NGINX Plus has built in support for active/passive Highly Available configuration. This is the mode we see most customers using in production environments.

This is a logical diagram of the testbed used for this functional testing:

NGINX Plus uses keepalived based solution for HA where there’s a floating VIP used by the Active Instance. In event of a failure of the Active Instance, keepalived will immediately transition the floating VIP to the Standby Instance which will become Active.

During the failover, we observed 2-6 seconds of failed requests. The failed requests all have HTTP status code 599/Network connect timeout error.

We also wanted to note that when the originally designated Active Instance becomes
available again, the floating VIP will transition automatically back to the originally designated *Active Instance*. 
Performance Characterization

We’ll cover the NGINX/IBM COS solution performance from the following perspectives:
- HTTP load balancing
- HTTPS load balancing
- SSL Termination vs SSL Passthrough

The performance numbers presented here, are the maximum throughput/OPs that can be achieved by using a single NGINX instance. Performance for a highly available pair of NGINX servers in active/standby configuration is identical, provided that both active/standby servers are using equivalent hardware.

WRITE to IBM COS

We ran a 100% write workflow, simulating overall throughput that can be sustained into our storage pool comprised of 12x IBM SS-2448 Slicestors. We used AWSv2 authentication for all tests. All write tests were one hour in duration.

<table>
<thead>
<tr>
<th>Object Size</th>
<th>OG Threads</th>
<th>HTTP/HTTPS</th>
<th>SSL</th>
<th>OPS</th>
<th>Overall throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 KiB</td>
<td>4000</td>
<td>HTTPS</td>
<td>Termination</td>
<td>17639.58</td>
<td>1.85 GB/s, 14.8 Gbps</td>
</tr>
<tr>
<td>1 MiB</td>
<td>4000</td>
<td>HTTPS</td>
<td>Termination</td>
<td>4351.09</td>
<td>4.56 GB/s, 36.48 Gbps</td>
</tr>
<tr>
<td>10 MiB</td>
<td>4000</td>
<td>HTTPS</td>
<td>Termination</td>
<td>430.35</td>
<td>4.52 GB/s, 36.16 Gbps</td>
</tr>
<tr>
<td>100 MiB</td>
<td>800</td>
<td>HTTPS</td>
<td>Termination</td>
<td>43.56</td>
<td>4.57 GB/s, 36.56 Gbps</td>
</tr>
<tr>
<td>1 GiB</td>
<td>120</td>
<td>HTTPS</td>
<td>Termination</td>
<td>4.17</td>
<td>4.37 GB/s, 34.96 Gbps</td>
</tr>
<tr>
<td>100 KiB</td>
<td>4000</td>
<td>HTTPS</td>
<td>Passthrough</td>
<td>18020.59</td>
<td>1.88 GB/s, 15.04 Gbps</td>
</tr>
<tr>
<td>1 MiB</td>
<td>4000</td>
<td>HTTPS</td>
<td>Passthrough</td>
<td>4397.23</td>
<td>4.62 GB/s, 36.96 Gbps</td>
</tr>
<tr>
<td>10 MiB</td>
<td>4000</td>
<td>HTTPS</td>
<td>Passthrough</td>
<td>432.03</td>
<td>4.53 GB/s, 36.24 Gbps</td>
</tr>
<tr>
<td>100 MiB</td>
<td>800</td>
<td>HTTPS</td>
<td>Passthrough</td>
<td>43.62</td>
<td>4.57 GB/s, 36.56 Gbps</td>
</tr>
<tr>
<td>1 GiB</td>
<td>240</td>
<td>HTTPS</td>
<td>Passthrough</td>
<td>4.43</td>
<td>4.64 GB/s, 37.12 Gbps</td>
</tr>
<tr>
<td>100 KiB</td>
<td>4000</td>
<td>HTTP</td>
<td>N/A</td>
<td>17862.11</td>
<td>1.87 GB/s, 14.98 Gbps</td>
</tr>
<tr>
<td>1 MiB</td>
<td>4000</td>
<td>HTTP</td>
<td>N/A</td>
<td>4412.83</td>
<td>4.63 GB/s, 37.04 Gbps</td>
</tr>
</tbody>
</table>
READ from IBM COS

We ran a 100% read workflow, simulating overall throughput that can be sustained out of our storage pool comprised of 12x IBM SS-2248 Slicestors. We used AWSv2 authentication for all tests. All read tests were 30 minutes in duration.

<table>
<thead>
<tr>
<th>Object Size</th>
<th>OG Threads</th>
<th>HTTP/HTTPS</th>
<th>SSL</th>
<th>OPS</th>
<th>Overall throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 MiB</td>
<td>4000</td>
<td>HTTP</td>
<td>N/A</td>
<td>445.38</td>
<td>4.67 GB/s 37.36 Gbps</td>
</tr>
<tr>
<td>100 MiB</td>
<td>800</td>
<td>HTTP</td>
<td>N/A</td>
<td>44.72</td>
<td>4.69 GB/s 37.52 Gbps</td>
</tr>
<tr>
<td>1 GiB</td>
<td>120</td>
<td>HTTP</td>
<td>N/A</td>
<td>4.47</td>
<td>4.68 GB/s 37.44 Gbps</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object Size</th>
<th>OG Threads</th>
<th>HTTP/HTTPS</th>
<th>SSL</th>
<th>OPS</th>
<th>Overall throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 KiB</td>
<td>4000</td>
<td>HTTPS</td>
<td>Termination</td>
<td>13125.25</td>
<td>1.37 GB/s 10.96 Gbps</td>
</tr>
<tr>
<td>1 MiB</td>
<td>4000</td>
<td>HTTPS</td>
<td>Termination</td>
<td>3733.17</td>
<td>3.92 GB/s 31.36 Gbps</td>
</tr>
<tr>
<td>10 MiB</td>
<td>4000</td>
<td>HTTPS</td>
<td>Termination</td>
<td>409.67</td>
<td>4.30 GB/s 34.4 Gbps</td>
</tr>
<tr>
<td>100 MiB</td>
<td>800</td>
<td>HTTPS</td>
<td>Termination</td>
<td>41.28</td>
<td>4.32 GB/s 34.56 Gbps</td>
</tr>
<tr>
<td>1 GiB</td>
<td>120</td>
<td>HTTPS</td>
<td>Termination</td>
<td>4.39</td>
<td>4.60 GB/s 36.80 Gbps</td>
</tr>
<tr>
<td>100 KiB</td>
<td>4000</td>
<td>HTTPS</td>
<td>Passthrough</td>
<td>10841.77</td>
<td>1.13 GB/s 9.08 Gbps</td>
</tr>
<tr>
<td>1 MiB</td>
<td>4000</td>
<td>HTTPS</td>
<td>Passthrough</td>
<td>4045.30</td>
<td>4.22 GB/s 33.76 Gbps</td>
</tr>
<tr>
<td>10 MiB</td>
<td>4000</td>
<td>HTTPS</td>
<td>Passthrough</td>
<td>400.37</td>
<td>4.18 GB/s 33.44 Gbps</td>
</tr>
<tr>
<td>100 MiB</td>
<td>800</td>
<td>HTTPS</td>
<td>Passthrough</td>
<td>39.60</td>
<td>4.15 GB/s 33.2 Gbps</td>
</tr>
<tr>
<td>1 GiB</td>
<td>400</td>
<td>HTTPS</td>
<td>Passthrough</td>
<td>3.54</td>
<td>3.71 GB/s 29.68 Gbps</td>
</tr>
<tr>
<td>100 KiB</td>
<td>4000</td>
<td>HTTP</td>
<td>N/A</td>
<td>14345.60</td>
<td>1.50 GB/s 12 Gbps</td>
</tr>
<tr>
<td>1 MiB</td>
<td>4000</td>
<td>HTTP</td>
<td>N/A</td>
<td>3884.48</td>
<td>4.08 GB/s 32.64 Gbps</td>
</tr>
<tr>
<td>10 MiB</td>
<td>4000</td>
<td>HTTP</td>
<td>N/A</td>
<td>434.40</td>
<td>4.55 GB/s 36.4 Gbps</td>
</tr>
<tr>
<td>100 MiB</td>
<td>800</td>
<td>HTTP</td>
<td>N/A</td>
<td>41.23</td>
<td>4.32 GB/s 34.56 Gbps</td>
</tr>
<tr>
<td>1 GiB</td>
<td>120</td>
<td>HTTP</td>
<td>N/A</td>
<td>3.50</td>
<td>3.674 GB/s 29.39 Gbps</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. Copyright 2017
Here’s a chart of the max throughput into IBM COS that can be sustained with a 100% write workflow for the 3 different reverse proxy configurations:

![100% Write Throughput in GB/s chart]

From the above chart, you can see that for most objects greater than 100 KiB in size, we’re writing objects into IBM COS near line rate/wire speed. At the 100 KiB object size, we were CPU limited at the Slicestor layer, with most of the CPU consumed in iowait, due to the Linux Kernel waiting on the drives themselves. With an optimized NGINX Plus configuration, we were not limited at the load balancer layer. You can see NGINX Plus is a high performance load balancer for all 3 different reverse proxy configurations at all object sizes for a 100% write workflow. With sufficient resources given to NGINX Plus, the load balancer was removed as the bottleneck in all these tests.

Moving on to the 100% read workflow, here’s a chart of the max throughput from IBM COS that can be sustained with a 100% read workflow for the 3 different reverse proxy configurations:
From the above chart, you can see that for most objects greater than 100 KiB in size, we're reading from IBM COS near line rate/wire speed. At the 100 KiB object size, we were CPU limited at the Slicestor layer.

As you've seen using IBM COS together with NGINX Plus gives the customer a very high performance, highly available, object storage solution for all workflows.

---

### Performance Scaling

The performance numbers presented above are the maximum that can be achieved by a single NGINX instance. If higher throughput is required, it is possible to set up multiple NGINX instances to achieve throughput greater than can be achieved by a single NGINX instance.

Each instance would be configured with its own Virtual IP address.

NGINX in such an active/active configuration requires another method to spread traffic across two or more active nodes. Some common methods are:

- Round robin DNS to map a single DNS name to multiple IP addresses
- Using another L3/L4/TCP load balancer device to distribute L3 traffic between the active NGINX servers
NGINX can also perform TCP load balancing, and can also be used to distribute traffic between the two (or more) active NGINX instances. Note that active NGINX instances in a load balanced cluster do not share configuration or state.

Caution should be used when running in active/active mode to increase the capacity of the load balanced cluster. In the event of a failure of one of the two nodes, the capacity of the cluster would be reduced by half. It is also possible to add a 3rd passive node to the active/active cluster to cover for a single node failure in the active/active configuration. It is also possible to have as many active nodes as desired to handle whatever throughput you require.
Open Issues

• None

Tools

The load generator tool we used for the performance testing was IBM’s Object Generator v1.4.1. It is open source, and you can obtain this at: https://github.ibm.com/cs-tools/og/tree/1.4.0

There’s a script we used to set irq affinity on the NIC. It is available from: https://gist.github.com/syuu1228/4352382

We also used openssl 1.0.2g to benchmark how fast our CPU could process SSL transactions. OpenSSL can be obtained from: https://www.openssl.org
Appendix A: Configuration Guide for NGINX open source

The significant segments of deploying NGINX load balancer together with IBM Cloud Object Storage are explained through the following steps taken.

1. Prepare IBM COS
2. Install base OS
3. Install NGINX
4. Configure NGINX
   a. Optionally configure SSL Termination
   b. Optionally configure SSL Passthrough
5. Tune the OS/Network Interface Card for best performance
   a. Optionally update kernel tunables
   b. Optionally increase NIC buffers
   c. Optionally disable IRQ balancing, and Affinitize interrupts to cores.
6. Testing the NGINX/IBM COS solution.

Assumptions

This guide assumes that existing IBM COS systems are operating, and with certain capabilities already configured.

IBM COS

- Storage pool
- COS Accesser pool

NGINX

- Ubuntu 16.04.3 Server with root level access
- DNS resolver configured. If using FQDN instead of IP address for the accessers, there should be both Primary AND Secondary DNS servers configured. Alternatively, /etc/hosts can also be updated to eliminate DNS lookups. If the environment is relatively stable/static, using a list of IP addresses minimizes points of failure in the solution.
- Public Internet access for software updates and NGINX package installation
• One or more dedicated NIC(s). For highest performance, NGINX should be installed on a bare metal server with direct access to the NIC(s). If that isn’t an option, we’ve also seen great performance with NGINX on an ESXi hypervisor, with the NIC(s) in PCI-Passthrough mode.

---

Prerequisites

IBM COS

• Tested Software Version
  o 3.10.2.96

NGINX

• Tested Software Version
  o nginx/1.10.3
  o OpenSSL 1.0.2g

1. Prepare COS

  All the Accessers being used should belong to an Access Pool. You’ll need a list of the FQDNs and/or IP addresses of the accessers in the access pool. We will use this list of accesser IP’s when we configure NGINX.

  ![List of Accesser IPs]

2. Install Ubuntu 16.04.3 LTS Server OS

  Please install the base operating system required for NGINX operation.

  You may obtain the ISO package from: https://www.ubuntu.com/download/server
You will also want to configure a “Management IP” address for this server. You should also configure a Virtual IP address to be used as the load balanced endpoint. The Management IP and Virtual IP can be the same IP address, depending on the resources you have available. Optionally, if you are in two-arm configuration, you may also want to configure a Subnet IP.

Once the OS is installed, ssh into the server and execute the following two commands:

```
sudo apt-get update
sudo apt-get upgrade
```

You may be prompted for the root password during these commands. Please enter the root password for these commands to complete.

<table>
<thead>
<tr>
<th>Management IP (optional)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual IP</td>
<td></td>
</tr>
<tr>
<td>Subnet IP (optional)</td>
<td></td>
</tr>
</tbody>
</table>

### 3. Install NGINX

Now that the base OS is installed, we can install the stable/production ready version of NGINX. Shell/ssh into the server and execute the following command:

```
sudo apt-get install nginx
```

When prompted “Do you want to continue?”, answer Y. The apt-get package manager will then fetch and install the required dependencies and libraries required to run the open source NGINX. NGINX will then be automatically started with a default configuration.

From your shell, execute the following command:

```
nginx -V
```
This will show the version of NGINX installed. This will also show which version of OpenSSL being used as well as the compilation options used to build this version of NGINX:

```
clessaf@bit-podi-nginxtmp:$ nginx -v
nginx version: nginx/1.10.3 (Ubuntu)
built with OpenSSL 1.0.2g 1 Mar 2016
```

Now, it is good to check that the default NGINX server can be properly accessed from the network. You can either point your web browser to the Virtual IP address you configured from Step 2 earlier, or you can issue a curl –vvv http://VIP. You should see a welcome message if you check using a browser, and you should see the version of NGINX as well as the welcome message if you check with curl:
4. **Configure NGINX load balancing/reverse proxy functionality.**
Now that NGINX is working with the default static welcome page, we will configure NGINX to accept requests on the Virtual IP and reverse proxy (load balance) those requests to IBM COS Accessers.

Obtain a root shell on the NGINX server, by executing `sudo bash`:

```
cleversafe@it-pod1-nginxtmp:~$ sudo bash
[sudo] password for cleversafe:
```

Change directory to `/etc/nginx`:

```
root@it-pod1-nginxtmp:~# cd /etc/nginx
root@it-pod1-nginxtmp:/etc/nginx# ls
conf.d  koi-utf  nginx.conf  sites-available  uwsgi_params
fastcgi.conf  koi-win  proxy_params  sites-enabled  win-utf
fastcgi_params  mime.types  scgi_params  snippets
root@it-pod1-nginxtmp:/etc/nginx#
```
There’s a file in this directory called nginx.conf. This is a key high level configuration file. For the most part, the default settings are good. From our testing, we have found better values to use for general purpose object storage workflows. Ensure that the following values within the http block are uncommented and set as follows:

<table>
<thead>
<tr>
<th>Directive</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sendfile</td>
<td>on</td>
</tr>
<tr>
<td>tcp_nopush</td>
<td>on</td>
</tr>
<tr>
<td>tcp_nodelay</td>
<td>on</td>
</tr>
<tr>
<td>keepalive_timeout</td>
<td>300</td>
</tr>
<tr>
<td>ssl_prefer_server_ciphers</td>
<td>on</td>
</tr>
</tbody>
</table>

Ensure the following values with the http block are commented out and set as follows:

<table>
<thead>
<tr>
<th>Directive</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#gzip</td>
<td>off</td>
</tr>
<tr>
<td>#gzip_disable</td>
<td>&quot;msie6&quot;</td>
</tr>
</tbody>
</table>

Ensure that the following values within the events block are uncommented and set as follows:

<table>
<thead>
<tr>
<th>Directive</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>worker_connections</td>
<td>10240</td>
</tr>
<tr>
<td>accept_mutex</td>
<td>off</td>
</tr>
<tr>
<td>multi_accept</td>
<td>off</td>
</tr>
</tbody>
</table>

Ensure that the following values within the events block are commented out and set as follows:

<table>
<thead>
<tr>
<th>Directive</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#use</td>
<td>epoll</td>
</tr>
</tbody>
</table>

Finally, in the main section, we want to manually set the following:

<table>
<thead>
<tr>
<th>Directive</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>worker_rlimit_nofile</td>
<td>10240</td>
</tr>
</tbody>
</table>

Now that the master nginx.conf is optimized, we shall now configure the load balancer/reverse proxy specific portion. The configuration file this resides in in /etc/nginx/sites-available/default.

From a root shell, please execute "mv /etc/nginx/sites-available/default /etc/nginx/sites-available/default.orig" to backup the original nginx server configuration file. Please pick from one of the following configurations:

a. Load balancer with SSL Termination on NGINX (recommended configuration)
   This is the by far the more common mode, and is the recommended configuration we recommend for general purpose object storage workflows. By terminating SSL on NGINX, you only need one certificate, no matter how many back end accessers are being used. When run in Highly Available mode, the same SSL certificate is used by both the Active and Standby
instances. It is important to note by terminating SSL on NGINX, unencrypted traffic does pass between the NGINX load balancer and the IBM COS Accessers. If your client’s policies prohibit unencrypted traffic, you may need to terminate SSL on the IBM COS Accessers. Also, if you are using the SSE-C feature, you will also need to terminate SSL on the IBM COS Accessers. Another advantage to terminating SSL on NGINX is, that it may be useful for debugging/troubleshooting purposes. By being able to terminate SSL on nginx, and passing HTTP directly to the accessers, we're able to packet capture in the clear, and inspect the HTTP headers.

We will be using self-signed certificates in this example. For production environments, you will definitely want to purchase certificates from a reputable CA.

As with any self signed certificates, the steps are:

a. Generate a private key
b. Generate a certificate signing request
c. Create a self-signed certificate using the private key and the CSR

First generate a private key:

```
cleversafe@ait-pod1-lb-nginx:~$ openssl genrsa -out 172.20.26.95private.key 2048
Generating RSA private key, 2048 bit long modulus
...............................................................+++
....................................................+++
e is 65537 (0x10001)
cleversafe@ait-pod1-lb-nginx:~$ ls -a
172.20.26.95private.key
```

Now that the private key is generated, we can just create a certificate signing request using that private key:

```
cleversafe@ait-pod1-lb-nginx:~$ openssl req -new -key ./172.20.26.95private.key -out 172.20.26.95.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:IL
Locality Name (eg, city) [:Elk Grove
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IBM Cleversafe
Organizational Unit Name (eg, section) [:Alliance Integration Team
```
Common Name (e.g. server FQDN or YOUR name) []:172.20.26.95
Email Address []:cleversafe-ait@wwpdl.vnet.ibm.com

Please enter the following 'extra' attributes to be sent with your certificate request
A challenge password []:
An optional company name []:

cleversafe@ait-pod1-lb-nginx:~$ ls -al 172.20.26.95.csr
-rw-rw-r-- 1 cleversafe cleversafe 1110 Apr 3 14:52
172.20.26.95.csr

Now that both the private key and the CSR is generated, the last step is to generate the self signed certificate using the private key and CSR:

cleversafe@ait-pod1-lb-nginx:~$ openssl x509 -req -days 730 -in ./172.20.26.95.csr -signkey ./172.20.26.95private.key -out ./172.20.26.95.crt
Signature ok
subject=/C=US/ST=IL/L=Elk Grove/O=IBM Cleversafe/OU=Alliance Integration Team/CN=172.20.26.95/emailAddress=cleversafe-ait@wwpdl.vnet.ibm.com
Getting Private key

cleversafe@ait-pod1-lb-nginx:~$ ls -al ./172.20.26.95.crt
-rw-rw-r-- 1 cleversafe cleversafe 1411 Apr 3 14:57 ./172.20.26.95.crt

Note, in this example, the generated self signed certificate is valid for 2 years. Depending on the requirements of your application, you may want to shorten that duration.

Now that the certificate file is generated, move it along with the private.key to the proper location. Note, this can be anywhere on the nginx server, so long as the nginx configuration file can reference it. As a best practice, NGINX recommends storing all SSL files in /etc/nginx/ssl to keep the root /etc/nginx directory clean. In this example, we chose to keep the self signed certificate in the parent directory of nginx configuration files:

root@ait-pod1-lb-nginx:/etc/nginx# cp /home/cleversafe/172.20.26.95.crt .
root@ait-pod1-lb-nginx:/etc/nginx# cp /home/cleversafe/172.20.26.95private.key .
root@ait-pod1-lb-nginx:/etc/nginx# ls -al 172.20.26.95private.key -rw-r--r-- 1 root root 1679 Apr 3 15:06 172.20.26.95private.keyoot@ait-pod1-lb-nginx:/etc/nginx# ls -al 172.20.26.95.crt -rw-r--r-- 1 root root 1411 Apr 3 15:06 172.20.26.95.crt
Please replace the /etc/nginx/sites-available/default file contents with the following, replacing the server IP addresses with your specific accesser IPs and the SSL certificate and private key with your appropriate certificates and keys:

```
upstream accessers {
    server 172.20.26.14 max_fails=5 fail_timeout=60s;
    server 172.20.26.13 max_fails=5 fail_timeout=60s;
    server 172.20.26.12 max_fails=5 fail_timeout=60s;
    server 172.20.26.11 max_fails=5 fail_timeout=60s;
    server 172.20.26.18 max_fails=5 fail_timeout=60s;
    server 172.20.26.17 max_fails=5 fail_timeout=60s;
    server 172.20.26.16 max_fails=5 fail_timeout=60s;
    server 172.20.26.15 max_fails=5 fail_timeout=60s;
}

server {
    listen 443 ssl;
    ssl_certificate /etc/nginx/172.20.26.95.crt;
    ssl_certificate_key /etc/nginx/172.20.26.95.private.key;
    ssl_session_tickets off;
    ssl_session_cache off;
    ssl_ciphers AES128-SHA256;
    ssl_prefer_server_ciphers on;
    client_body_buffer_size 1100M;
    client_max_body_size 1300M;
    keepalive_timeout 300s;
    keepalive_requests 1000000;

    server_name _;
    proxy_buffering off;
    access_log off; # disabled access log for best performance

    location / {
        proxy_pass http://backend;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $remote_addr;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
    }
}
```

b. Load balancer with SSL Termination on IBM COS Accessers(SSL Passthrough mode)

   Note, when running in SSL Passthrough mode, the IBM COS Access logs may not show client IP address, as the X-Forwarded* headers are not added. NGINX in SSL Passthrough mode functions as a TCP layer load balancer. We recommend using this mode only if you are using the SSE-C feature, which requires HTTPS.

To use this mode, we'll need to add one line to the master
/etc/nginx/nginx.conf at the highest level. Please add the following after the http block:

```
include /etc/nginx/conf.d/passthrough.conf;
```
Next create the following file: /etc/nginx/conf.d/passthrough.conf, with the following information, replacing the server IP addresses with your specific accesser IPs:

```plaintext
stream {
    upstream accessers {
        server 172.20.26.14:443;
        server 172.20.26.13:443;
        server 172.20.26.12:443;
        server 172.20.26.11:443;
        server 172.20.26.18:443;
        server 172.20.26.17:443;
        server 172.20.26.16:443;
        server 172.20.26.15:443;
    }
    server {
        listen 443;
        proxy_pass accessers;
    }
}
```

c. Load balancer with HTTP only and no SSL support.
This is less commonly used in production, but we do see a lot of HTTP only workflow in test labs. Please replace the /etc/nginx/sites-available/default file contents with the following, replacing the server IP addresses with your specific accesser IPs:

```plaintext
upstream accessers {
    server 172.20.26.14 max_fails=5 fail_timeout=60s;
    server 172.20.26.13 max_fails=5 fail_timeout=60s;
    server 172.20.26.12 max_fails=5 fail_timeout=60s;
    server 172.20.26.11 max_fails=5 fail_timeout=60s;
    server 172.20.26.18 max_fails=5 fail_timeout=60s;
    server 172.20.26.17 max_fails=5 fail_timeout=60s;
    server 172.20.26.16 max_fails=5 fail_timeout=60s;
    server 172.20.26.15 max_fails=5 fail_timeout=60s;
}

server {
    listen 80 default_server;

    #customize the following two parameters to suit your expected workflow. The maximum object that can be stored/retrieved in this example is ~1300MB.
    client_body_buffer_size 1100M;
    client_max_body_size 1300M;

    keepalive_timeout 300s;
    keepalive_requests 1000000;

    server_name _;
    proxy_buffering off;
    access_log off; #disabled access log for best performance

    location / {
        proxy_pass http://accessers;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $remote_addr;
    }
}```
d. Combination of HTTPS and HTTP load balancing.
   For a combination of HTTPS and HTTP load balancing, you just have to include the correct server block information from above. For both HTTPS and HTTP load balancing, you should have two server blocks. One server block for HTTP and one server block for HTTPS. For the HTTPS load balancing, you can choose between SSL Termination on NGINX or SSL Passthrough.

5. Tune the OS/Network Interface Card for best performance

For NGINX to perform optimally, the OS/NIC must also be tuned appropriately. We’ll start with the XL710 NIC. Note, the following example is specifically applicable to Intel’s XL710 Network Interface cards. For other NIC’s, refer to vendor documentation on how to adjust the ring buffer sizes. By default, the XL710 comes configured with both TX/RX ring buffers of 512:

```bash
sudo ethtool -G interfaceName rx 4096 tx 4096
```

Execute “sudo ethtool –G interfaceName rx 4096 tx 4096”, replacing `interfaceName` with the actual network interface(s) being used for load balancing. Repeat this on every single NIC being used for load balancing. Here’s an example of using an `interfaceName` of ens160:
You may want to add this to /etc/rc.local, so that the correct ring buffers settings are applied each time the host reboots.

Next, we want to disable IRQ balancing on the OS. There’s one defaults file that takes care of this for us. As the root user, please change the `ENABLED=’1’` line in /etc/default/irqbalance to `ENABLED=’0’`. Here’s an example of properly disabling IRQ balancing:

![Screenshot of command output]

This will require an OS reboot to take effect. However, before rebooting the OS, let’s also set up the server to properly affinitize all NIC interrupts in your server. If you have multiple NICs being used for load balancing, you’ll want to do this for each NIC. Start by downloading the `set_irq_affinity.sh` script and saving it in /etc/nginx. You may download this from: [https://gist.github.com/SaveTheRbtz/8875474](https://gist.github.com/SaveTheRbtz/8875474)

Be sure to make this script executable, by performing a “`chmod 755 /etc/nginx/set_irq_affinity.sh`”. You may try running this script followed by the network interface name being used. For our example interface name of ens3f1, you would run “`sudo /etc/nginx/set_irq_affinity.sh ens3f1`”. Repeat this for every single NIC you are using for load balancing traffic.

You may want to add this to /etc/rc.local, to make this automatically run each time the host reboots.

For clarity’s sake, here’s our example of /etc/rc.local that performs the tasks above for our particular server. Please note that depending on your NIC vendor, number of NICs, you will most likely have different settings than ours:
Now that you’ve properly optimized the NIC ring buffer parameters, and now that you’ve properly disabled IRQ balancing and affinitized NIC interrupts to CPU cores, we shall now tune the Linux kernel. These are the settings we found to have the optimal settings for high throughput object storage workflows:
The same sysctl tunables above, are presented below:

<table>
<thead>
<tr>
<th>sysctl tunable parameter</th>
<th>value(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>net.ipv4.tcp_tw_recycle</td>
<td>1</td>
</tr>
<tr>
<td>net.ipv4.tcp_tw_reuse</td>
<td>1</td>
</tr>
<tr>
<td>net.core.somaxconn</td>
<td>102400</td>
</tr>
<tr>
<td>net.core.netdev_max_backlog</td>
<td>250000</td>
</tr>
<tr>
<td>net.ipv4.tcp_low_latency</td>
<td>1</td>
</tr>
<tr>
<td>net.ipv4.ip_local_port_range</td>
<td>1024 65000</td>
</tr>
<tr>
<td>net.core.rmem_default</td>
<td>8388608</td>
</tr>
<tr>
<td>net.core.wmem_default</td>
<td>8388608</td>
</tr>
<tr>
<td>net.core.rmem_max</td>
<td>16777216</td>
</tr>
<tr>
<td>net.core.wmem_max</td>
<td>16777216</td>
</tr>
<tr>
<td>net.ipv4.tcp_rmem</td>
<td>4096 87380 16777216</td>
</tr>
<tr>
<td>net.ipv4.tcp_wmem</td>
<td>4096 65536 16777216</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>net.ipv4.tcp_timestamps</td>
<td>0</td>
</tr>
<tr>
<td>net.ipv4.tcp_sack</td>
<td>1</td>
</tr>
<tr>
<td>net.ipv4.tcp_congestion_control</td>
<td>cubic</td>
</tr>
<tr>
<td>net.ipv4.tcp_synack_retries</td>
<td>2</td>
</tr>
<tr>
<td>net.ipv4.tcp_syn_retries</td>
<td>2</td>
</tr>
<tr>
<td>net.ipv4.tcp_fin_timeout</td>
<td>15</td>
</tr>
<tr>
<td>net.ipv4.ip_nonlocal_bind</td>
<td>1</td>
</tr>
</tbody>
</table>

After the /etc/sysctl.conf file is updated with the optimized settings, you may execute “sysctl –p /etc/sysctl.conf”, to reload these new settings:

It is recommended to manually apply the sysctl.conf updates, just to check for any possible typos or errors. Once the manual sysctl reload is successful, you may reboot the OS just to verify that everything is properly set upon an OS restart.

6. Testing the NGINX/IBM COS Solution

Before commencing testing, be sure that NGINX is using the latest configuration. You may force NGINX to reload the configuration by executing the following as the root user: `nginx –s reload`.

You may now start your load testing, using the NGINX VIP as the endpoint for your applications.
Appendix B: Configuration Guide for NGINX Plus

The significant segments of deploying NGINX Plus load balancer together with IBM Cloud Object Storage are explained through the following steps taken.

1. Prepare IBM COS
2. Install base OS
3. Install NGINX Plus
4. Configure NGINX Plus
   a. Optionally configure SSL Termination
   b. Optionally configure SSL Passthrough
5. Tune the OS/Network Interface Card for best performance
   a. Optionally update kernel tunables
   b. Optionally increase NIC buffers
   c. Optionally disable IRQ balancing, and Affinitize interrupts to cores.

Assumptions

This guide assumes that existing IBM COS systems are operating, and with certain capabilities already configured.

IBM COS

- Storage pool
- COS Accesser pool

NGINX Plus

- Ubuntu 16.04.3 Server with root level access
- DNS resolver configured. If using FQDN instead of IP address for the accessers, there should be both Primary AND Secondary DNS servers configured. Alternatively, /etc/hosts can also be updated to eliminate DNS lookups. If the environment is relatively stable/static, using a list of IP addresses minimizes points of failure in the solution.
• Public Internet access for software updates and NGINX Plus package installation
• One or more dedicated NIC(s). For highest performance, NGINX Plus should be installed on a bare metal server with direct access to the NIC(s). If that isn’t an option, we’ve also seen great performance with NGINX Plus on an ESXi hypervisor, with the NIC(s) in PCI-Passthrough mode.

Prerequisites

IBM COS
• Tested Software Version
  o 3.10.2.96

NGINX Plus
• Tested Software Version
  o nginx/1.13.4 (nginx-plus-r13)
  o OpenSSL 1.0.2g
• Purchased NGINX Plus subscription
• E-mail containing links to your purchased nginx-repo.crt and nginx-repo.key files.

1. Prepare COS
   All the Accessers being used should belong to an Access Pool. You’ll need a list of the FQDNs and/or IP addresses of the accessers in the access pool. We will use this list of accesser IP’s when we configure NGINX.

<table>
<thead>
<tr>
<th>List of Accesser IPs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

2. Install Ubuntu 16.04.3 LTS Server OS
   Please install the base operating system required for NGINX operation.
   You may obtain the ISO package from: https://www.ubuntu.com/download/server
You will also want to configure a “Management IP” address for this server. You should also configure a Virtual IP address to be used as the load balanced endpoint. The Management IP and Virtual IP can be the same IP address, depending on the resources you have available. Optionally, if you are in two-arm configuration, you may also want to configure a Subnet IP.

Once the OS is installed, ssh into the server and execute the following two commands:

```
sudo apt-get update
sudo apt-get upgrade
```

You may be prompted for the root password during these commands. Please enter the root password for these commands to complete.

<table>
<thead>
<tr>
<th>Management IP (optional)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual IP</td>
<td></td>
</tr>
<tr>
<td>Subnet IP (optional)</td>
<td></td>
</tr>
</tbody>
</table>

3. **Install NGINX Plus**

Now that the base OS is installed, we can install NGINX Plus.

First, create the /etc/ssl/nginx directory, by executing:

```
sudo mkdir /etc/ssl/nginx
```

Next, obtain your nginx-repo.crt and nginx-repo.key from the NGINX Plus Customer Portal, and place them in the /etc/ssl/nginx directory you just created:

```
sudo cp nginx-repo.crt /etc/ssl/nginx
sudo cp nginx-repo.key /etc/ssl/nginx
```

Download the NGINX signing key from nginx.org and add it:

```
sudo wget http://nginx.org/keys/nginx_signing.key
sudo apt-key add nginx_signing.key
```

Install apt-utils package by executing:

```
sudo apt-get install apt-transport-https lsb-release ca-certificates
```

Install NGINX Plus Repository by executing:

```
printf "deb https://plus-pkgs.nginx.com/debian `lsb_release -cs` nginx-plus\n" | sudo tee /etc/apt/sources.list.d/nginx-plus.list
```

Download the “90nginx” file to /etc/apt/apt.conf.d:

```
sudo wget -q -O /etc/apt/apt.conf.d/90nginx https://cs.nginx.com/static/files/90nginx
```

Update the repository information:

```
sudo apt-get update
```
Finally, install the nginx-plus package:

```bash
sudo apt-get install -y nginx-plus
```

4. **Configure NGINX Plus load balancing/reverse proxy functionality.**

Now that NGINX Plus is working with the default static welcome page, we will configure NGINX Plus to accept requests on the Virtual IP and reverse proxy(load balance) those requests to IBM COS Accessers.

Obtain a root shell on the NGINX Plus server, by executing `sudo bash`:

```bash
sudo bash
```

Change directory to `/etc/nginx`:

```
root@it-pod1-nginxtmp:~# cd /etc/nginx
```

There’s a file in this directory called `nginx.conf`. This is a key high level configuration file. For the most part, the default settings are good. From our testing, we have found better values to use for general purpose object storage workflows. Ensure that the following values within the `http` block are uncommented and set as follows:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sendfile</code></td>
<td><code>on;</code></td>
</tr>
<tr>
<td><code>tcp_nopush</code></td>
<td><code>on;</code></td>
</tr>
<tr>
<td><code>tcp_nodelay</code></td>
<td><code>on;</code></td>
</tr>
<tr>
<td><code>keepalive_timeout</code></td>
<td><code>300;</code></td>
</tr>
<tr>
<td><code>ssl_prefer_server_ciphers</code></td>
<td><code>on;</code></td>
</tr>
</tbody>
</table>

Ensure the following values with the `http` block are commented out and set as follows:

```bash
## Commented out values
#gzip off;
#gzip_disable "msie6";
```

Ensure that the following values within the `events` block are uncommented and set as follows:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>worker_connections</code></td>
<td><code>10240;</code></td>
</tr>
<tr>
<td><code>accept_mutex</code></td>
<td><code>off;</code></td>
</tr>
<tr>
<td><code>multi_accept</code></td>
<td><code>off;</code></td>
</tr>
</tbody>
</table>

Ensure that the following values within the `events` block are commented out and set as follows:

```bash
# use epoll;
```

Finally, in the main section, we want to manually set the following:

```bash
worker_rlimit_nofile 10240;
```
Now that the master nginx.conf is optimized, we shall now configure the load balancer/reverse proxy specific portion. The configuration file this resides in is in /etc/nginx/conf.d/.

From a root shell, please execute “mv /etc/nginx/conf.d/default /etc/nginx/conf.d/default.orig” to backup the original NGINX Plus server configuration file. Please pick from one of the following configurations:

e. Load balancer with SSL Termination on NGINX Plus (recommended configuration)

This is the by far the more common mode, and is the recommended configuration we recommend for general purpose object storage workflows. By terminating SSL on the NGINX Plus node, you only need one certificate, no matter how many back end accessers are being used. When run in Highly Available mode, the same SSL certificate is used by both the Active and Standby instances. It is important to note by terminating SSL on NGINX Plus, unencrypted traffic does pass between the NGINX Plus load balancer and the IBM COS Accessers. If your client's policies prohibit unencrypted traffic, you may need to terminate SSL on the IBM COS Accessers. Also, if you are using the SSE-C feature, you will also need to terminate SSL on the IBM COS Accessers. Another advantage to terminating SSL on the NGINX Plus node is, that it may be useful for debugging/troubleshooting purposes. By being able to terminate SSL on NGINX Plus, and passing HTTP directly to the accessers, we’re able to packet capture in the clear, and inspect the HTTP headers.

We will be using self-signed certificates in this example. For production environments, you will definitely want to purchase certificates from a reputable CA.

As with any self signed certificates, the steps are:

d. Generate a private key
e. Generate a certificate signing request
f. Create a self-signed certificate using the private key and the CSR

First generate a private key:

cleversafe@ait-pod1-lb-nginx:~$ openssl genrsa -out 172.20.26.95private.key 2048
Generating RSA private key, 2048 bit long modulus
...............................................................+++ ............................................+++ e is 65537 (0x10001)
cleversafe@ait-pod1-lb-nginx:~$ ls -al 172.20.26.95private.key
-rw-rw-r-- 1 cleversafe cleversafe 1679 Apr 3 14:50
172.20.26.95private.key

Now that the private key is generated, we can just create a certificate signing
request using that private key:

cleversafe@ait-pod1-lb-nginx:~$ openssl req -new -key ./172.20.26.95private.key -out 172.20.26.95.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:IL
Locality Name (eg, city) []:Elk Grove
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IBM Cleversafe
Organizational Unit Name (eg, section) []:Alliance Integration Team
Common Name (e.g. server FQDN or YOUR name) []:172.20.26.95
Email Address []:cleversafe-ait@wwpdl.vnet.ibm.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

cleversafe@ait-pod1-lb-nginx:~$ ls -al 172.20.26.95.csr
-rw-rw-r-- 1 cleversafe cleversafe 1110 Apr 3 14:52
172.20.26.95.csr

Now that both the private key and the CSR is generated, the last step is to generate
the self signed certificate using the private key and CSR:

cleversafe@ait-pod1-lb-nginx:~$ openssl x509 -req -days 730 -in ./172.20.26.95.csr -signkey ./172.20.26.95private.key -out ./172.20.26.95.crt
Signature ok
subject=/C=US/ST=IL/L=Elk Grove/O=IBM Cleversafe/OU=Alliance Integration Team/CN=172.20.26.95/emailAddress=cleversafe-ait@wwpdl.vnet.ibm.com
Getting Private key

cleversafe@ait-pod1-lb-nginx:~$ ls -al ./172.20.26.95.crt
-rw-rw-r-- 1 cleversafe cleversafe 1411 Apr 3 14:57
./172.20.26.95.crt
Note, in this example, the generated self signed certificate is valid for 2 years. Depending on the requirements of your application, you may want to shorten that duration.

Now that the certificate file is generated, move it along with the private.key to the proper location. Note, this can be anywhere on the NGINX Plus server, so long as the NGINX Plus configuration file can reference it. As a best practice, NGINX recommends storing all SSL files in /etc/nginx/ssl to keep the root /etc/nginx directory clean. In this example, we chose to keep the self signed certificate in the parent directory of nginx configuration files:

```
root@ait-pod1-lb-nginx:/etc/nginx# cp /home/cleversafe/172.20.26.95.crt .
root@ait-pod1-lb-nginx:/etc/nginx# cp /home/cleversafe/172.20.26.95private.key .
root@ait-pod1-lb-nginx:/etc/nginx# ls -al
-rw-r---r-- 1 root root 1679 Apr 3 15:06 172.20.26.95private.key
root@ait-pod1-lb-nginx:/etc/nginx# ls -al
-rw-r---r-- 1 root root 1411 Apr 3 15:06 172.20.26.95.crt
```

Please replace the /etc/nginx/conf.d/default file contents with the following, replacing the server IP addresses with your specific accesser IPs and the SSL certificate and private key with your appropriate certificates and keys:

```nginx
upstream accessers {
    server 172.20.26.14 max_fails=5 fail_timeout=60s;
    server 172.20.26.13 max_fails=5 fail_timeout=60s;
    server 172.20.26.12 max_fails=5 fail_timeout=60s;
    server 172.20.26.11 max_fails=5 fail_timeout=60s;
    server 172.20.26.18 max_fails=5 fail_timeout=60s;
    server 172.20.26.17 max_fails=5 fail_timeout=60s;
    server 172.20.26.16 max_fails=5 fail_timeout=60s;
    server 172.20.26.15 max_fails=5 fail_timeout=60s;
}

server {
    listen 443 ssl;
    ssl_certificate /etc/nginx/172.20.26.95.crt;
    ssl_certificate_key /etc/nginx/172.20.26.95private.key;
    ssl_session_tickets off;
    ssl_session_cache off;
    ssl_ciphers AES128-SHA256;
    ssl_prefer_server_ciphers on;
    client_body_buffer_size 1100M;
    client_max_body_size 1300M;
    keepalive_timeout 300s;
    keepalive_requests 1000000;
    server_name _;
    proxy_buffering off;
```
access_log off; #disabled access log for best performance

location / {
    proxy_pass http://backend;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $remote_addr;
    proxy_http_version 1.1;
    proxy_set_header Connection "";
}
server 172.20.26.11 max_fails=5 fail_timeout=60s;
server 172.20.26.18 max_fails=5 fail_timeout=60s;
server 172.20.26.17 max_fails=5 fail_timeout=60s;
server 172.20.26.16 max_fails=5 fail_timeout=60s;
server 172.20.26.15 max_fails=5 fail_timeout=60s;
}

server {
  listen 80 default_server;

  #customize the following two parameters to suit your
  #expected workflow. The maximum object that can be
  #stored/retrieved in this example is ~1300MB.
  client_body_buffer_size 1100M;
  client_max_body_size 1300M;

  keepalive_timeout 300s;
  keepalive_requests 1000000;

  server_name _;
  proxy_buffering off;
  access_log off; #disabled access log for best
  performance

  location / {
    proxy_pass http://accessers;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $remote_addr;
    proxy_http_version 1.1;
    proxy_set_header Connection "";
  }
}

h. Combination of HTTPS and HTTP load balancing.
   For a combination of HTTPS and HTTP load balancing, you just have to
   include the correct server block information from above. For both HTTPS
   and HTTP load balancing, you should have two server blocks. One server
   block for HTTP and one server block for HTTPS. For the HTTPS load
   balancing, you can choose between SSL Termination on NGINX or SSL
   Passthrough.

5. Tune the OS/Network Interface Card for best performance

For NGINX Plus to perform optimally, the OS/NIC must also be tuned appropriately.
We’ll start with the XL710 NIC. Note, the following example is specifically applicable
 to Intel’s XL710 Network Interface cards. For other NIC’s, refer to vendor
documentation on how to adjust the ring buffer sizes. By default, the XL710 comes
configured with both TX/RX ring buffers of 512:
Execute “sudo ethtool –G interfaceName rx 4096 tx 4096”, replacing interfaceName with the actual network interface(s) being used for load balancing. Repeat this on every single NIC being used for load balancing. Here’s an example of using an interfaceName of ens160:

You may want to add this to /etc/rc.local, so that the correct ring buffers settings are applied each time the host reboots.

Next, we want to disable IRQ balancing on the OS. There’s one defaults file that takes care of this for us. As the root user, please change the ENABLED="1" line in /etc/default/irqbalance to ENABLED="0". Here’s an example of properly disabling IRQ balancing:

This will require an OS reboot to take effect. However, before rebooting the OS, let’s also set up the server to properly affinitize all NIC interrupts in your server. If you have multiple NICs being used for load balancing, you’ll want to do this for each NIC. Start by downloading the set_irq_affinity.sh script and saving it in /etc/nginx. You may download this from: https://gist.github.com/SaveTheRbtz/8875474

Be sure to make this script executable, by performing a “chmod 755 /etc/nginx/set_irq_affinity.sh”. You may try running this script followed by the network
interface name being used. For our example interface name of ens3f1, you would run
“sudo /etc/nginx/set_irq_affinity.sh ens3f1”. Repeat this for every single NIC you are
using for load balancing traffic.

You may want to add this to /etc/rc.local, to make this automatically run each time the
host reboots.

For clarity’s sake, here’s our example of /etc/rc.local that performs the tasks above for
our particular server. Please note that depending on your NIC vendor, number of
NICs, you will most likely have different settings than ours:

Now that you’ve properly optimized the NIC ring buffer parameters, and now that
you’ve properly disabled IRQ balancing and affinitized NIC interrupts to CPU cores,
we shall now tune the Linux kernel. These are the settings we found to have the
optimal settings for high throughput object storage workflows:
The same sysctl tunables above, are presented below:

<table>
<thead>
<tr>
<th>sysctl tunable parameter</th>
<th>value(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>net.ipv4.tcp_tw_recycle</td>
<td>1</td>
</tr>
<tr>
<td>net.ipv4.tcp_tw_reuse</td>
<td>1</td>
</tr>
<tr>
<td>net.core.somaxconn</td>
<td>102400</td>
</tr>
<tr>
<td>net.core.netdev_max_backlog</td>
<td>250000</td>
</tr>
<tr>
<td>net.ipv4.tcp_low_latency</td>
<td>1</td>
</tr>
<tr>
<td>net.ipv4.ip_local_port_range</td>
<td>1024 65000</td>
</tr>
<tr>
<td>net.ipv4.tcp_rmem</td>
<td>4096 87380 16777216</td>
</tr>
<tr>
<td>net.ipv4.tcp_wmem</td>
<td>4096 65536 16777216</td>
</tr>
<tr>
<td>net.core.rmem_default</td>
<td>8388608</td>
</tr>
<tr>
<td>net.core.wmem_default</td>
<td>8388608</td>
</tr>
<tr>
<td>net.core.rmem_max</td>
<td>16777216</td>
</tr>
<tr>
<td>net.core.wmem_max</td>
<td>16777216</td>
</tr>
<tr>
<td>net.ipv4.tcp_synack_retries</td>
<td>2</td>
</tr>
<tr>
<td>net.ipv4.tcp_syn_retries</td>
<td>2</td>
</tr>
<tr>
<td>net.ipv4.tcp_fin_timeout</td>
<td>15</td>
</tr>
</tbody>
</table>

#kernel.hostname = example.com
After the /etc/sysctl.conf file is updated with the optimized settings, you may execute “sysctl –p /etc/sysctl.conf”, to reload these new settings:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>net.ipv4.tcp_timestamps</td>
<td>0</td>
</tr>
<tr>
<td>net.ipv4.tcp_sack</td>
<td>1</td>
</tr>
<tr>
<td>net.ipv4.tcp_congestion_control</td>
<td>cubic</td>
</tr>
<tr>
<td>net.ipv4.tcp_synack_retries</td>
<td>2</td>
</tr>
<tr>
<td>net.ipv4.tcp_syn_retries</td>
<td>2</td>
</tr>
<tr>
<td>net.ipv4.tcp_fin_timeout</td>
<td>15</td>
</tr>
<tr>
<td>net.ipv4.ip_nonlocal_bind</td>
<td>1</td>
</tr>
</tbody>
</table>

It is recommended to manually apply the sysctl.conf updates, just to check for any possible typos or errors. Once the manual sysctl reload is successful, you may reboot the OS just to verify that everything is properly set upon an OS restart.

6. Testing the NGINX Plus/IBM COS Solution

Before commencing testing, be sure that NGINX Plus is using the latest configuration. You may force NGINX Plus to reload the configuration by executing the following as the root user: `nginx -s reload`.

You may now start your load testing, using the NGINX Plus VIP as the endpoint for your applications.
Appendix-C: Highly Available Pair Configuration Guide

The significant segments of deploying a highly available pair of NGINX Plus load balancers together with IBM Cloud Object Storage are explained through the following steps taken.

1. Prepare IBM COS
2. Install base OS on both active and standby nodes
3. Install NGINX Plus on both active and standby nodes
4. Install NGINX Plus add on modules on both active and standby nodes
5. Run nginx-ha-setup on both active and standby nodes
6. Configure and run nginx-sync to synchronize configuration files on both nodes

Assumptions

This guide assumes that existing IBM COS systems are operating, and with certain capabilities already configured.

IBM COS
- Storage pool
- COS Accesser pool

NGINX Plus
- Ubuntu 16.04.3 Server with root level access
- DNS resolver configured. If using FQDN instead of IP address for the accessers, there should be both Primary AND Secondary DNS servers configured. Alternatively, /etc/hosts can also be updated to eliminate DNS lookups. If the environment is relatively stable/static, using a list of IP addresses minimizes points of failure in the solution.
- Public Internet access for software updates and NGINX Plus package installation
- One or more dedicated NIC(s). For highest performance, NGINX Plus should be installed on a bare metal server with direct access to the NIC(s). If that isn’t an option, we’ve also seen great performance with NGINX Plus on an ESXi hypervisor, with the NIC(s) in PCI-Passthrough mode.
Prerequisites

IBM COS
- Tested Software Version
  - 3.10.2.96

NGINX Plus
- Tested Software Version
  - nginx/1.13.4 (nginx-plus-r13)
  - OpenSSL 1.0.2g
- Purchased NGINX Plus subscription
- E-mail containing links to your purchased nginx-repo.crt and nginx-repo.key files.

NGINX Plus includes out of the box support for running NGINX in a “HA Cluster”. The solution is based on keepalived, which itself uses an implementation of VRRP.

There are two packages to install to configure HA functionality.
- nginx-ha-keepalived
- nginx-sync

The nginx-sync package allows configuration synchronization between the servers in the cluster. With this package installed and properly configured, configuration changes made to the master server can be automatically pushed to the standby NGINX Plus instances.

You will also require an additional “Cluster IP” address for this feature. This “Cluster IP” address will be your highly available IBM COS endpoint. This will be the IP address your applications should use.

The first part of configuring a pair of HA NGINX Plus load balancers is to set up two identical NGINX Plus instances, each with distinct static IP addresses. Please follow Appendix-B to set up two identical NGINX Plus instances prior to proceeding with this configuration guide.

1. Shell into the first NGINX Plus instance, which we will designate as the “master” NGINX Plus instance. Install keepalived:
   ```sh
cleversafe@ait-pod1-ib-nginx-s$ sudo apt-get install nginx-ha-keepalived
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  iproute
The following NEW packages will be installed:
  iproute nginx-ha-keepalived
0 upgraded, 2 newly installed, 0 to remove and 4 not upgraded.
```
Need to get 142 kB of archives.
After this operation, 431 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y

Get:1 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 iproute all 1:4.3.0-1ubuntu3.16.04.1 [2,432 B]
Get:2 https://plus-pkgs.nginx.com/ubuntu xenial/nginx-plus amd64 nginx-ha-keepalived 1.3.4-1-xenial [140 kB]
Fetched 142 kB in 4s (28.7 kB/s)

Selecting previously unselected package iproute.
(Reading database ... 60067 files and directories currently installed.)
Preparing to unpack .../iproute_1%3a4.3.0-1ubuntu3.16.04.1_all ... 
Unpacking iproute (1:4.3.0-1ubuntu3.16.04.1) ...
Selecting previously unselected package nginx-ha-keepalived.
Preparing to unpack .../nginx-ha-keepalived_1.3.4-1-xenial_amd64.deb ...
Unpacking nginx-ha-keepalived (1.3.4-1-xenial) ...
Processing triggers for man-db (2.7.5-1) ...
Processing triggers for systemd (229-4ubuntu19) ...
Processing triggers for ureadahead (0.100.0-19) ...
Setting up iproute (1:4.3.0-1ubuntu3.16.04.1) ...
Setting up nginx-ha-keepalived (1.3.4-1-xenial) ...
Processing triggers for systemd (229-4ubuntu19) ...
Processing triggers for ureadahead (0.100.0-19) ...

2. While still shelled into the first NGINX Plus instance, install nginx-sync:

    cleversafe@ait-pod1-lb-nginx:~$ sudo apt-get install nginx-sync
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  nginx-sync
0 upgraded, 1 newly installed, 0 to remove and 15 not upgraded.
Need to get 13.1 kB of archives.
After this operation, 57.3 kB of additional disk space will be used.
Get:1 https://plus-pkgs.nginx.com/ubuntu xenial/nginx-plus amd64 nginx-sync all 1.0 [13.1 kB]
Fetched 13.1 kB in 0s (30.5 kB/s)
Selecting previously unselected package nginx-sync.
(Reading database ... 60991 files and directories currently installed.)
Preparing to unpack .../nginx-sync_1.0_all.deb ...
Unpacking nginx-sync (1.0) ...
Setting up nginx-sync (1.0) ...

3. Repeat steps 1 and 2 for the second NGINX Plus instance. The 2\textsuperscript{nd} NGINX Plus instance we will be designating as a “Backup” instance.

4. After the two requisite packages are installed on both the Master NGINX Plus instance as well as the Backup NGINX Plus instance, then all that is needed is to run a configuration setup script on both instances simultaneously. It is best to open up two shells, one shell into each instance, and run the procedure in parallel. For the following example in this configuration guide, we are going to be using the following configuration details. Please substitute our IP’s with your IP’s.

    • Master NGINX instance static IP address: 172.20.26.104
    • Backup NGINX instance static IP address: 172.20.26.105
    • NGINX Cluster IP static IP address: 172.20.26.106

5. In the master node shell, please execute “sudo nginx-ha-setup” and press enter to continue:
In the backup node shell, please execute “sudo nginx-ha-setup” and press enter to continue:

```
cleversafe@bit-podl-lb-nginx:~$ sudo nginx-ha-setup
Thank you for using NGINX Plus!

This script is intended for use with RHEL/CentOS/SLES/Debian/Ubuntu-based systems.
It will configure highly available NGINX Plus environment in Active/Passive pair.

NOTE: you will need the following in order to continue:
- 2 running systems (nodes) with static IP addresses
- one free IP address to use as Cluster IP endpoint

It is strongly recommended to run this script simultaneously on both nodes,
e.g. use two terminal windows and switch between them step by step.

It is recommended to run this script under screen(1) in order to allow
installation process to continue in case of unexpected session disconnect.
Press <Enter> to continue...
```

6. In the master node shell, please enter the internal management IP address when prompted. You will then enter the master NGINX Plus instance static IP address. Answer yes to “Use this address for internal cluster communication”. Here’s an example using our lab IP address:

```
Step 1: configuring internal management IP addresses.
In order to communicate with each other, both nodes must have at least one IP address.
The guessed primary IP of this node is: 172.20.26.104
Do you want to use this address for internal cluster communication? (y/n)
IP address of this host is set to: 172.20.26.104
Primary network interface: ens160

In the backup node shell, please enter the internal management IP address when prompted. You will enter the backup NGINX Plus instance static IP address. Here’s an example using our lab IP address:

```
Step 1: configuring internal management IP addresses.
In order to communicate with each other, both nodes must have at least one IP address.
The guessed primary IP of this node is: 172.20.26.105
Do you want to use this address for internal cluster communication? (y/n)
IP address of this host is set to: 172.20.26.105
Primary network interface: ens160
```

7. You will be then be prompted for the IP address of the 2\textsuperscript{nd} node. On the master NGINX Plus instance, you will be entering in the backup NGINX Plus instance static IP address as the 2\textsuperscript{nd} node, pressing enter to continue:
On the backup NGINX Plus instance, you will be entering in the master NGINX instance static IP address as the 2\textsuperscript{nd} node, pressing enter to continue:

8. The next step is creating the keepalived configuration. On both the master and backup NGINX Plus instances, you will be entering in the “Cluster IP address”. This will be the load balanced endpoint/IP address applications will use to access IBM COS.

9. Next, you will be choosing which node should have the MASTER role in this cluster. On the master NGINX Plus instance, we will be selecting 1, for master node:
On the backup NGINX Plus instance, we will be selecting 2, for backup node:

10. At this time, the NGINX Plus “Cluster IP” should be active, with the master NGINX Plus instance servicing all requests. You should be able to issue a curl request against the Cluster IP to verify successful operation.

You may also run “ip addr show interface”, to see which node is the master node. Here, our ait-pod1-lb-nginx is the master node, as it has the “Cluster IP assigned to it”:

And our ait-pod1-lb-nginx2 is the backup node, as the “Cluster IP isn’t active on it”:
11. Note, even if HTTPS/SSL was set up properly on each individual NGINX Plus instance prior to executing this procedure, new certificates will have to be generated with the correct “Cluster IP/FQDN” for HTTPS to work properly. If you are just load balancing HTTP traffic, you may skip to step x to sync NGINX Plus configuration files across both nodes.

12. To configure HTTPS for our Cluster IP, we’ll need to generate new certificates. As with any self signed certificates, the steps are:
   a. Generate a private key
   b. Generate a certificate signing request
   c. Create a self-signed certificate using the private key and the CSR.

13. Shell into the master NGINX Plus instance, create a directory called /etc/nginx/clusterIPssl, and change your current working directory to the newly created directory. Run the following to create your private key:

```
root@lit-pod-lb-nginx:~/etc/nginx/clusterIPssl# openssl genrsa -out private.key 2048
Generating RSA private key, 2048 bit long modulus
.........++++
...+     
root@lit-pod-lb-nginx:~/etc/nginx/clusterIPssl# ls -al private.key
-rw-------. root root 1679 Sep 7 14:33 private.key
```

14. Now that the private key is generated, we can just create a certificate signing request using that private key:

```
root@lit-pod-lb-nginx:~/etc/nginx/clusterIPssl# openssl req -new -key .private.key -out 1 72.20.26.106.csr
You are about to be asked to enter information that will be incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter ' ' the field will be left blank.

-----

Country Name (2 letter code) [AU]:US
State or Province Name (Full name) [Some-State]:IL
Locality Name (eg, city) []:IL
Organization Name (eg, company) []:EKA Grove
Organizational Unit Name (eg, section) []:Alliance Integration Team
Common Name (e.g. server FQDN or YOUR name) []:172.20.26.106
Email Address []:dlinus.IBM.com

Please enter the following 'extra' attributes to be sent with your certificate request
A challenge password []:
An optional company name []:
```

15. Now that both the private key and the CSR are generated, the last step is to generate the self signed certificate using the private key and the CSR:
Note, in this example, the generated self signed certificate is valid for 2 years. Depending on the requirements of your application, you may want to shorten that duration.

16. Now that our certificate file is generated, move it along with the private key to the proper location. Note, this can be anywhere on the server, so long as the NGINX Plus configuration file can reference it. As a best practice, NGINX recommends storing all SSL files in /etc/nginx/ssl to keep the root /etc/nginx directory clean. In this example, we chose to keep the self signed certificate and private key in the parent directory of NGINX Plus configuration files.

17. Once the certs are in the proper location, the next step is to be sure that NGINX Plus is listening on the proper SSL port, and that the NGINX Plus configuration file can find them. The following 3 lines can be added or modified to the server{} section of the NGINX Plus configuration file in /etc/nginx/conf.d/default.conf:

18. Now that our configuration file on the master NGINX Plus node supports the Cluster IP for both HTTP/HTTPS, we need to make sure the same configuration file(s) as well as SSL certificates (if you have it configured) are present on the backup NGINX Plus node as well. NGINX’s nginx-sync utility requires passwordless ssh as the root user between both master and root nodes. As the root user on both the master node, and backup node, execute “ssh-keygen” to generate and rsa key pair. Here’s an example of generating ssh keys on the master node:
19. By default, Ubuntu 16.04.x does not allow either root logins or passwordless root logins. Please change /etc/ssh/sshd_config on both the master and backup node to allow passwordless SSH. You will be need to change “PermitRootLogin” to be set to “without-password”.

Once the sshd_config file is updated correctly on both the master and backup NGINX Plus nodes, you'll need to restart sshd, by executing “service ssdh restart” on both the master and backup NGINX Plus nodes.

20. NGINX’s nginx-sync utility uses one configuration file located in the /etc directory. You will create a new file in /etc with the filename: nginx-sync.conf. This configuration file contains in the NODES variable, the FQDN or IP address of the master and backup node. It also contains a CONFPATHS variable that contains all the NGINX Plus configuration files you would like to push to the backup nodes. Here, we should include /etc/nginx.conf, the entire /etc/nginx/conf.d directory, as well as the Cluster IP SSL certificates if you are configuring this for HTTPS/SSL. Here's how our example looks:

21. Now that nginx-sync’s configuration file is correct, let’s execute nginx-sync as the root user from the master NGINX Plus node:
Note, this does require passwordless ssh set up as the root user in the previous step. Since we disabled password based logins, and only allowed key based logins, you will see the root login failure. Ignore that, as nginx-sync.sh will use passwordless SSH to rsync the configuration files.

22. Now that NGINX Plus's configuration files are identical across both master and backup NGINX Plus nodes, and with nginx-ha-keepalived also configured, the highly available NGINX Plus load balancer is ready for service. We recommend testing the failover in both directions with both HTTP and HTTPS workflows to ensure that your application requirements are met. In our testing both HTTP and HTTPS workflows failed over within 3-7 seconds.

Also note that once a node is designated an NGINX Plus master node, it will always take priority. Meaning, once the master node comes back, it'll become master node again, leaving the backup node as standby. So if you are testing failover by a rebooting the master node, you may see the master node regain control of servicing HTTP requests when it comes back in service.
References

The following reference material are available in the AIT Solutions Guide Box Folder for NGINX: https://ibm.box.com/s/z3yz89gp6rs7ewomxj4d1agey9nad7yr

As well as the AIT Lab Testing Documentation folder: https://ibm.box.com/s/yp7wq56fae7tcepmiw3gq2y4jbihz1w6

- **NGINX Performance Tuning – Tips and Tricks v.02**
- OpenSSLBenchmarks
- NGINX-SSL-Performance
- NGINX_OpenSSL Performance Study

The latest versions of NGINX’s documentation can always be found: http://nginx.org/en/docs/

The latest NGINX Plus installation procedure can be found: https://www.nginx.com/resources/admin-guide/installing-nginx-plus/

The latest NGINX Open Source Installation procedure can be found: https://www.nginx.com/resources/admin-guide/installing-nginx-open-source/

The latest NGINX administration guide for configuring High Availability can be found: https://www.nginx.com/resources/admin-guide/nginx-ha-keepalived/

The IBM COS OG object storage load generation tool can be found on IBM’s github: https://github.ibm.com/cs-tools/og/tree/1.4.0
Notices

This information was developed for products and services offered in the US. This material might be available from IBM® in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.
IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing IBM
Corporation
North Castle Drive, MD-NC119 Armonk,
NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data discussed herein is presented as derived under specific operating conditions. Actual results may vary.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.
If you are viewing this information softcopy, the photographs and color illustrations may not appear.

---

**Trademarks**

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at Copyright and trademark information at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

Accesser®, Cleversafe®, ClevOS™, Dispersed Storage®, dsNet®, IBM Cloud Object Storage Accesser®, IBM Cloud Object Storage Dedicated™, IBM Cloud Object Storage Insight™, IBM Cloud Object Storage Manager™, IBM Cloud Object Storage Slicestor®, IBM Cloud Object Storage Standard™, IBM Cloud Object Storage System™, IBM Cloud Object Storage Vault™, SecureSlice™, and Slicestor® are trademarks or registered trademarks of Cleversafe, an IBM Company and/or International Business Machines Corp.

Other product and service names might be trademarks of IBM or other companies.