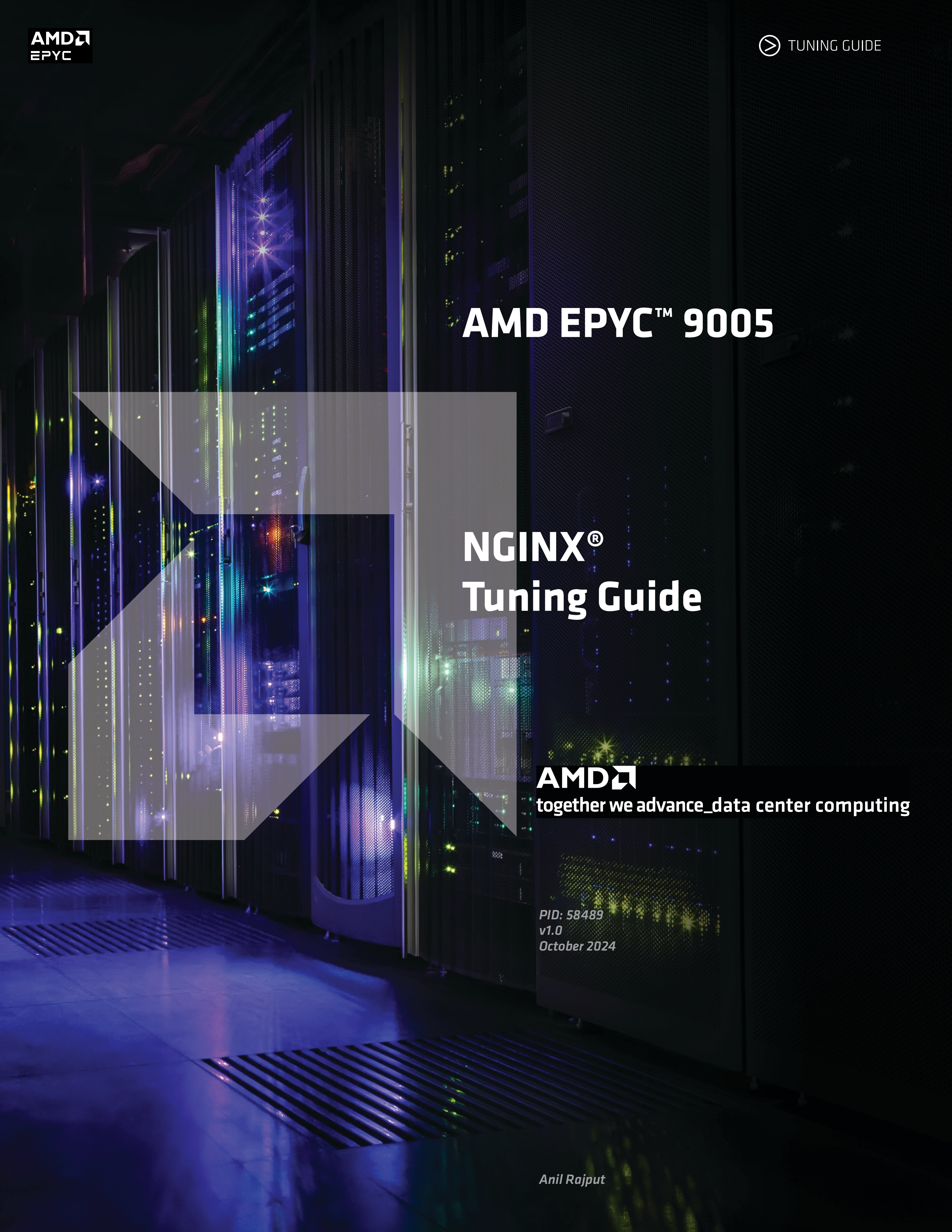
TUNING GUIDE



AMD EPYC™ 9005

NGINX® Tuning Guide

AMD
together we advance_data center computing

PID: 58489
v1.0
October 2024

Anil Rajput

DATE	VERSION	CHANGES
June, 2024	0.1	Initial NDA release
October, 2024	1.0	Initial public release

AUDIENCE

This document is intended for a technical audience such as NGINX® application architects, production deployment, and performance engineering teams with a server configuration background who have:

- Admin access to the server's management interface (BMC).
- Familiarity with the server's management interface.
- Admin OS access.
- Familiarity with the OS-specific configuration, monitoring, and troubleshooting tools.

NGINX®
TUNING GUIDE

CONTENTS

Chapter 1: Introduction

This tuning guide details parameters that can optimize performance on servers powered by AMD EPYC™ 9005 Series Processors. OEM-default hardware and BIOS configurations are optimized for a wide range of workloads, and specific changes are often needed to provide optimal performance and scaling for specific workloads on all OS platforms. This tuning guide describes the following settings to optimize performance:

- Hardware configurations
- BIOS settings
- Network performance tuning
- OS kernel parameters
- NGINX performance tuning

1.1 - About NGINX®

NGINX® is a high-performance, open-source web server designed for both web content delivery and modern cloud architectures across a wide range of use cases such as HTTP, reverse proxy, mail, and a generic TCP/UDP proxy server. More than 400 million websites worldwide use NGINX web server and application delivery solutions to deliver their content quickly, reliably, and securely. NGINX's modular event-driven architecture can provide more predictable performance under high loads while maintaining low latencies and responsiveness. It handles requests using an asynchronous event-driven approach instead of threads. NGINX can scale to hundreds of thousands of concurrent connections on modern hardware.

Simple packaging, configurability, and documents make NGINX easy to architect and deploy in on-premises, hybrid, or cloud environments using virtualization and containers with or without orchestration frameworks. Applying best practices for tuning the hardware platform, network configuration, operating system, and applications helps achieve optimal performance and full utilization. A balanced, fully-tuned platform should deliver superior performance compared to a reference system.

New platforms often feature changes to BIOS, cores, NUMA, memory population, network, OS, and even applications. The many changes can complicate identifying optimal configurations for a given platform. Performance benchmarks can help assess basic platform capabilities and help guide instance sizing, scaling up and scaling out, memory allocation, etc. One well-known test uses the `wrk` HTTP benchmarking tool along with the NGINX web server delivering web content. `wrk` data collection methodology and results allow users to understand all aspects of a given platform's web server performance. This tuning guide uses `wrk` and NGINX to validate most of the BIOS, OS, network, memory, NUMA placement and application-level parameter recommendations.

1.2 - Important Reading

Please be sure to read the following guides (available from the AMD Documentation Hub), which contain important foundational information about 5th Gen AMD EPYC processors:

- *AMD EPYC™ 9005 Processor Architecture Overview*
- *BIOS & Workload Tuning Guide for AMD EPYC™ 9005 Series Processors*
- *Memory Population Guidelines for AMD EPYC™ 9005 Series Processors*

Chapter 3: Hardware Configurations

3.1 - Memory Configurations

Proper memory subsystem configuration is important for achieving optimum performance testing. All I/O uses data transfers into or out of memory, and the I/O bandwidth can never exceed the memory subsystem limits. Achieving the maximum memory bandwidth on modern CPUs requires populating at least one DIMM in every DDR channel. Servers powered by AMD EPYC 9005 Series Processors include up to 12 DDR5 memory channels. AMD recommends populating all memory channels per socket, with every channel having the same speed and capacity DIMMs for optimal performance. Please see Memory Population Guidelines for AMD Family 19h Models 10h–1F for more details.

3.2 - Characteristics of AMD EPYC 9005 Platforms

The CCDs connect to memory, I/O, and each other through the I/O Die (IOD). Each CCD connects to the IOD via a dedicated high-speed Global Memory Interconnect (GMI) link. The IOD also contains memory channels, PCIe® Gen5 lanes, and Infinity Fabric links. All dies (or chiplets) interconnect with each other via AMD's Infinity Fabric Technology.

3.3 - Recommended BIOS Settings

AMD provides default BIOS setting guidance to all OEMs and provides additional guidance for specific applications and workloads. Please see the *BIOS & Workload Tuning Guide for AMD EPYC™ 9005 Series Processor Based Servers* (available from the AMD Documentation Hub) for additional tuning information.

THIS PAGE INTENTIONALLY LEFT BLANK.

Chapter 4: NGINX Architecture and Use Cases

AMD recommends deploying NGINX 1.22 or later for optimal performance, but some production deployments may use earlier versions for various reasons. AMD based the recommendations in this Tuning Guide on running NGINX as a web server, but most of them also apply when NGINX is deployed for reverse proxying, caching, load balancing, media streaming, etc.

4.1 - NGINX Architecture

NGINX is a web server that offers high performance, scalability, and stability. NGINX can also function as a proxy server for email (IMAP, POP3, and SMTP) and as a reverse proxy and load balancer for HTTP, TCP, and UDP servers. NGINX handles multiple connections within a single process. It includes:

- A Primary process.
- Worker processes that handle the connections.
- Helper processes that manage the on-disk content cache.

NGINX can interleave multiple connections within a single Worker process and can switch from connection to connection almost instantaneously as new network traffic arrives, which allows handling heavy loads with minimal latency.

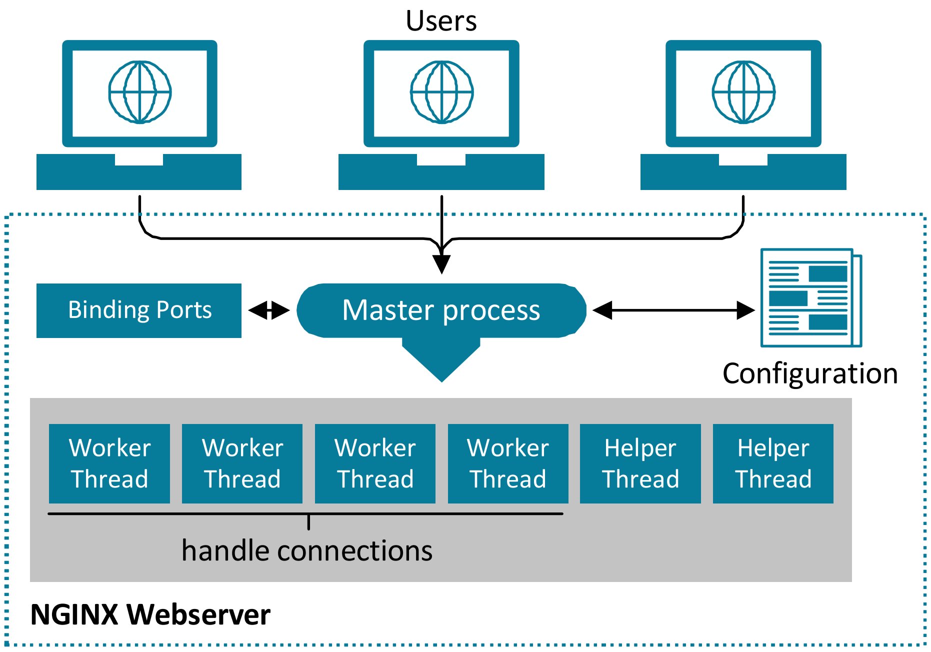


Figure 4-1: NGINX web server architecture

4.2 - NGINX as a Web Server

The `wrk` benchmark tool tests NGINX web server performance. `wrk` is lightweight and can easily handle multiple NGINX web server instances configured to deliver static or dynamic content ranging in size from ~1K bytes to many megabytes per request. It also provides an easy way to evaluate the impact of various deployment options such as compression, encryption, or local caching.

NGINX instance uses platform resources such as CPU cores and memory, cache, and network to respond to the client machine based on requests and assigned roles. This requires a balanced production environment. Here is a simple characterization of systems powered by AMD EPYC 9005 Series Processors based on findings gained by using `wrk`:

- Small (~1 KB) static and dynamic content requests require more compute and memory and scale well as more CPU cores are allocated to NGINX instance.
- Requests for large (>1 MB) static pages easily saturate a network bandwidth of ~100Gbps for most systems and will benefit from network-related optimizations and tunings.

Other important considerations that can significantly improve both throughput and response time include:

- Sizing NGINX instance for scale up and scale out.
- Platform-specific optimizations and tunings.

Chapter 4: NGINX Tuning Methods and Recommendations

This chapter describes the methodology and recommendations for tuning and optimizing NGINX instances for various deployments. A systematic approach follows these steps:

1. Number of vCPU cores per NGINX instance for scale up.
2. Scale out and optimal alignment with NUMA nodes.
3. Network configuration.
4. System configuration.
5. NGINX parameters.
6. NGINX builds using the latest GCC compiler.
7. Orchestration and container settings.

4.1 - Number of vCPU per NGINX Instance for Scale Up

The web server's ability to handle requests depends on factors such as (but not limited to) the size of the content to be fetched, the processing capacity of the instance, and the number of incoming connections. You can adjust `wrk` to model production environment behavior and use those initial findings to determine both instance size and scale out behavior. 8 vCPU per instance delivers good performance, and 16 vCPU gives optimal performance when testing with `wrk`.

NGINX utilizes a hybrid disk-memory cache where all cached items are loaded into memory. For instances that require large caches, starting an instance with more cores (>8 vCPU up to 64 vCPU, depending on your specific AMD EPYC 9005 Series Processor) will open additional memory bandwidth to the NGINX instance.

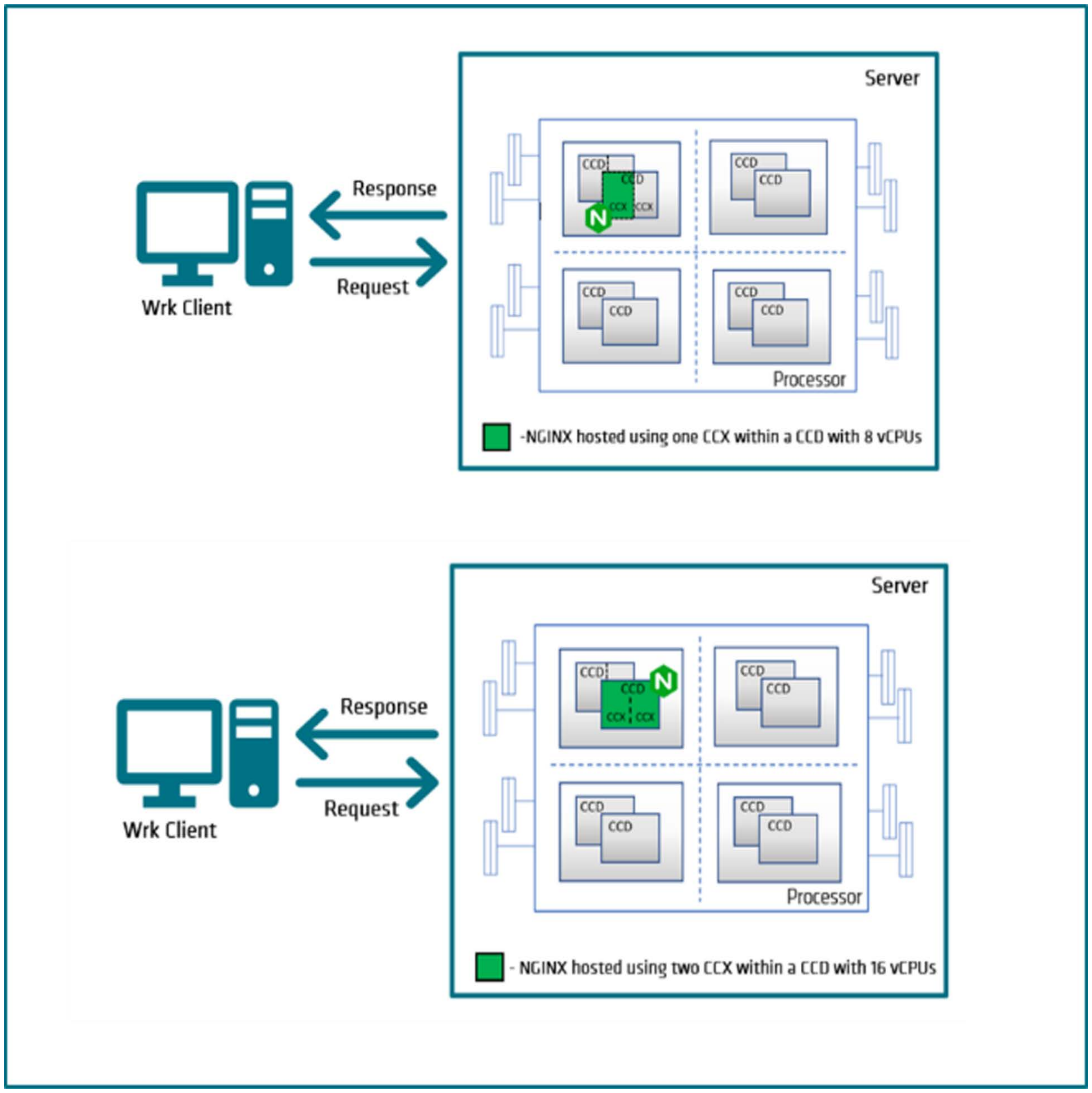

Server
Response
Request
Wrk Client
CCD
CCX
Processor
-NGINX hosted using one CCX within a CCD with 8 vCPUs
Server
Response
Request
Wrk Client
CCD
CCX
Processor
- NGINX hosted using two CCX within a CCD with 16 vCPUs

Figure 4-1: NGINX Scale-up solution examples of alignment with AMD EPYC 9005 Series Processors

Instances that are either cloud-based or that use VMs on-premises scale up well from 8vCPUs to 16 vCPUs and 32 vCPUs unless network bandwidth becomes a bottleneck. See “Network Configuration” on page xiii for important network tuning information.

Requests with small or dynamic content scale well even for larger instances with greater than 64 vCPUs if network bandwidth is not a bottleneck. IOMMU must be set correctly because it plays a very important role in scaling beyond 32 vCPUs. See “IOMMU Settings” on page xvi for more details.

4.2 - Scale out and Optimal Alignment with CCX/CCD and NUMA Nodes

Scale out strategy is the next step after determining the optimal number of vCPUs per NGINX instance. This involves a rough approximation of the overall scale out performance, deciding to align and pin instances to CCD and NUMA nodes, network, system, and NGINX parameters settings, and monitoring both CPU and network resources for bottlenecks.

Scale-out performance should be close to linear for cloud deployments where each additional NGINX instance will likely use a different node. On-premise deployments should also scale well unless network bandwidth or other shared resources become a bottleneck. Again, monitor and evaluate CPU and network utilization.

The ability to pin NGINX instances to a given NUMA node and aligning within CCD boundaries becomes critical for consistent performance as additional NGINX instances are deployed. Failing to do this allows the vCPUs of a VM to be allocated to any CPU cores, which often lowers throughput and increases response time. Worst- case scenarios saw up to 20% performance difference when comparing 32 vCPU NGINX instances that perfectly aligned and pinned versus instances split evenly between two sockets with 16 vCPUs per socket. AMD observed this performance difference in both cloud and bare metal VMs and recommends adopting allocation policies that avoid splitting a single instance across the sockets.

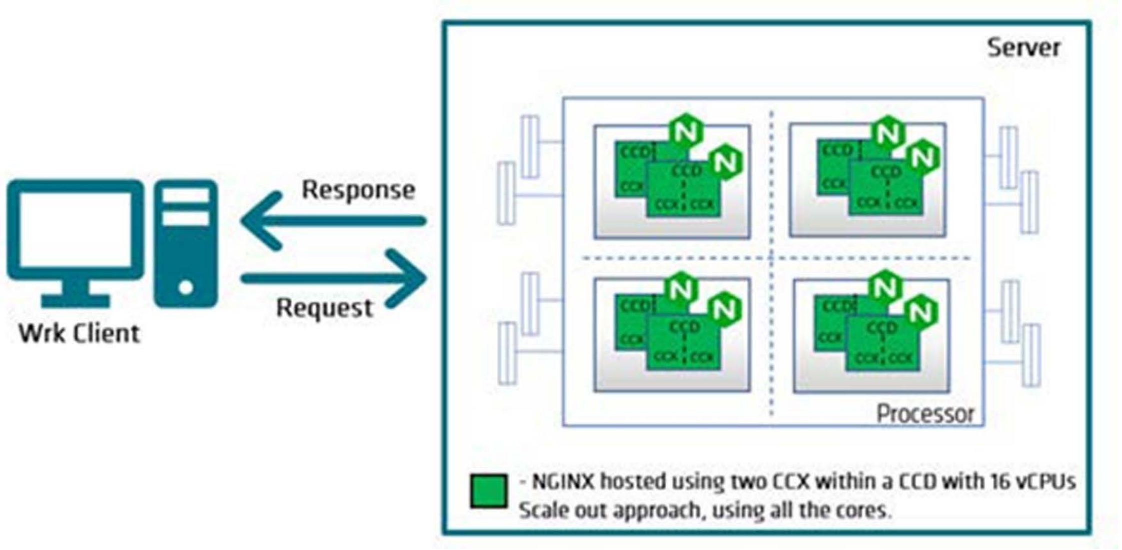

Server
Response
Request
Wrk Client
CCD
CCX
Processor
- NGINX hosted using two CCX within a CCD with 16 vCPUs
Scale out approach, using all the cores.

Figure 4-2: Sample NGINX scale out examples of alignment with AMD EPYC 9005 Series Processor CCDs

4.3 - Optimal CCX/CCD Alignment

The AMD EPYC 9005 Series Processor architecture has multiple CCDs in each socket (1P), and each CCD has a single CCX unit. Each CCX has up to 8C/16T (Zen5, or "Classic") or 16C/32T (Zen5c, or "Dense") with each core having its own L1i, L1d, and L2 (1MB) caches. Each CCX also includes an L3 cache.

L3 cache is shared across all cores of a CCX. A thread on one CCD runs optimally when it accesses memory from the memory channels connected to the same CCD, and it therefore best to ensure that (for example) each NGINX instance runs on a single CCD with memory allocations/accesses occurring from the channels connected to that CCD.

Using the AMD EPYC 9005 Series Processor architecture (2P, 192C/384T) as an example, having either one NGINX instance per CCD (8C/16T) or 2 CCDs in the same quadrant(16C/32T) will perform better than having one NGINX instance with 8C/16C split between 2 CCDs from different quadrants or sockets. AMD further recommends keeping the vCPU size of NGINX VMs within one NUMA node.

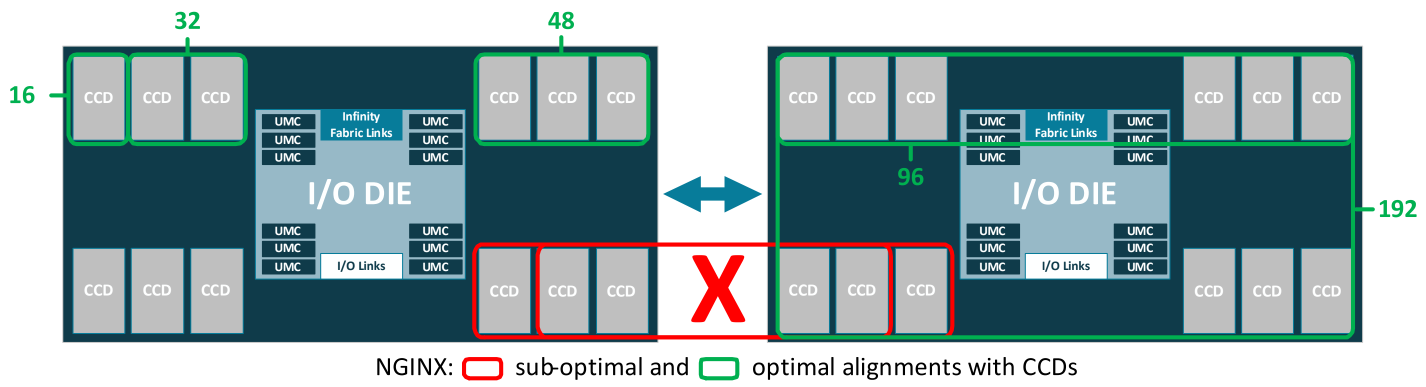


Figure 4-3: Example of pinning different # of vCPUs per NGINX aligned to CCX/CCD/NUMA nodes

AMD recommends pinning instances within a NUMA node while hosting/creating instances but does not recommend doing this via the application-level `worker_cpu_affinity` option. The following sample configuration lowers performance by increasing the time a process spends waiting for a free CPU.

```
worker_processes     4;
worker_cpu_affinnity 0001 0010 0100 1000;
```

Note: AMD recommends maximizing performance by pinning NGINX instances or VMs or containers deploying NGINX to NUMA nodes at minimum, and also aligning them to CCD boundaries.

4.3.1 - BIOS NPS Settings by NGINX Instance Size

AMD EPYC 9005 Series Processors include Node Per Socket (NPS) settings of NPS1, NPS2, and NPS4 which create 1, 2, and 4 NUMA nodes per socket, respectively. Table 4-1 shows the recommended NPS settings for a 96-core processor based on NGINX instance sizes:

vCPUs per NGINX instance	NPS setting in BIOS
32 or less	NPS4
96	NPS2
192 or more	NPS1

Table 4-1: Recommended optimal NPS settings based on vCPUs per NGINX instance

Determine the optimal NPS setting for AMD EPYC 9005 Series Processors with fewer than 192 vCPUs based on the number of vCPUs per NGINX instance matching as closely as possible to the number of vCPUs in a NUMA node or that best fit within a NUMA node.

Note: If deployment restrictions prevent you from pinning VMs or NGINX instances, then NPS1 will deliver the most consistent performance because memory accesses within each socket will all be uniform. This is the best trade-off for this situation.

4.4 - Network Configuration

Network configuration is one of the most important components in any NGINX deployment. Begin network configuration by applying the relevant recommendations from the *Linux® Network Tuning Guide for AMD EPYC 9005 Series Processors* (available from AMD EPYC Tuning Guides), which also includes additional recommendations for deployments that use an orchestration framework and containers. AMD recommends continually monitoring and recording network and disk I/O in both test and production environments.

4.4.1 - NIC Interrupt Handling

Having a number of interrupt queues that are equal to or less than the CPU cores per NUMA node optimizes performance. Assigning multiple queues to a single core risks thrashing the interrupt handler by swapping between queues. AMD recommends using a single queue per core.

There are three types of interrupt queues:

- Receive (RX)
- Transmit (TX)
- Combined. This uses a single queue to handle both RX and TX interrupts.

Note: Some vendors have separate RX and TX queues while others only implement combined queues.

Use `lscpu` output to determine the number of cores per NUMA node:

```
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Address sizes:                   52 bits physical, 57 bits virtual
Byte Order:                      Little Endian
CPU(s):                          384
On-line CPU(s) list:             0-383
Vendor ID:                       AuthenticAMD
Model name:                      AMD EPYC 9654 96-Core Processor
CPU family:                      25
```

```
Model:                           17
Thread(s) per core:              2
Core(s) per socket:              96
Socket(s):                       2
Stepping:                        1
Frequency boost:                 enabled
CPU max MHz:                     2400.0000
CPU min MHz:                     1500.0000
BogoMIPS:                        4800.15
Flags:                           fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm constant_tsc rep_good nopl
nonstop_tsc cpuid extd_apicid aperfmperf rapl pni pclmulqdq monitor ssse3 fma cx16 pcid sse4_1 sse4_2
x2apic movbe popcnt aes xsave avx f16c rdrand lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a
misalignsse 3dnowprefetch osvw ibs skinit wdt tce topoext perfctr_core perfctr_nb bpext perfctr_llc mwaitx
cpb cat_l3 cdp_l3 invpcid_single hw_pstate ssbd mba ibrs ibpb stibp vmmcall fsgsbase bmi1 avx2 smep bmi2
erms invpcid cqm rdt_a avx512f avx512dq rdseed adx smap avx512ifma clflushopt clwb avx512cd sha_ni avx512bw
avx512vl xsaveopt xsavec xgetbv1 xsaves cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local avx512_bf16
clzero irperf xsaveerptr rdpru wbnoinvd amd_ppin arat npt lbrv svm_lock nrip_save tsc_scale vmcb_clean
flushbyasid decodeassists pausefilter pfthreshold avic v_vmsave_vmload vgif v_spec_ctrl avx512vbmi umip
pku ospke avx512_vbmi2 gfni vaes vpclmulqdq avx512_vnni avx512_bitalg avx512_vpopcntdq la57 rdpid
overflow_recov succor smca fsrm flush_l1d
Virtualization:                  AMD-V
L1d cache:                       6 MiB (192 instances)
L1i cache:                       6 MiB (192 instances)
L2 cache:                        192 MiB (192 instances)
L3 cache:                        768 MiB (24 instances)
NUMA node(s):                    24
NUMA node0 CPU(s):               0-7,192-199
NUMA node1 CPU(s):               24-31,216-223
NUMA node2 CPU(s):               48-55,240-247
NUMA node3 CPU(s):               72-79,264-271
NUMA node4 CPU(s):               8-15,200-207
NUMA node5 CPU(s):               32-39,224-231
NUMA node6 CPU(s):               56-63,248-255
NUMA node7 CPU(s):               80-87,272-279
NUMA node8 CPU(s):               16-23,208-215
NUMA node9 CPU(s):               40-47,232-239
NUMA node10 CPU(s):              64-71,256-263
NUMA node11 CPU(s):              88-95,280-287
NUMA node12 CPU(s):              96-103,288-295
NUMA node13 CPU(s):              120-127,312-319
NUMA node14 CPU(s):              144-151,336-343
NUMA node15 CPU(s):              168-175,360-367
NUMA node16 CPU(s):              104-111,296-303
NUMA node17 CPU(s):              128-135,320-327
NUMA node18 CPU(s):              152-159,344-351
NUMA node19 CPU(s):              176-183,368-375
NUMA node20 CPU(s):              112-119,304-311
NUMA node21 CPU(s):              136-143,328-335
NUMA node22 CPU(s):              160-167,352-359
NUMA node23 CPU(s):              184-191,376-383
Vulnerability Itlb multihit:     Not affected
Vulnerability L1tf:              Not affected
Vulnerability Mds:               Not affected
Vulnerability Meltdown:          Not affected
Vulnerability Mmio stale data:   Not affected
Vulnerability Retbleed:          Not affected
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Retpolines, IBPB conditional, IBRS_FW, STIBP always-on, RSB
filling, PBRSB-eIBRS Not affected
Vulnerability Srbds:             Not affected
Vulnerability Tsx async abort:   Not affected
```

The preceding shows the physical core numbers followed by the logical core numbers. For instance, the highlighted row for NUMA node 2 shows eight physical cores 48–55 and logical cores 240– 247. In this example, eight total interrupt queues is the best setting. If then NIC combines the RX and TX interrupts, then execute the following command:

```
root@testsystem:~# ethtool -L enp33s0f0 combined 8
```

Evaluate the throughput vs. latency trade-off for dual-socket systems powered by AMD EPYC 9005 Series Processors based on your production requirements. Splitting the interrupts across NUMA-nodes will maximize throughput, while limiting the interrupts to a single node will reduce latency.

4.4.2 - RX and TX Ring Sizes

The NIC ring values represent the number of buffers that a NIC uses to DMA data into system memory. Having more buffers available queues more work for processing during a single interrupt. The `ethtool` utility allows you to check the current and maximum NIC ring sizes:

```
root@testsystem:~ethtool -g enp33s0f0
Ring parameters for enp33s0f0:
Pre-set maximums:
RX:            8192
RX Mini:       0
RX Jumbo:      0
TX:            8192
Current hardware settings:
RX:            512
RX Mini:       0
RX Jumbo:      0
TX:            512
```

The silicon and driver in the adapter shown in this example allow a maximum ring size of 8192, but it is currently set to 512, or only 1/16th of the maximum allowed. Use `ethtool` to boost network performance by setting the maximum allowable ring size:

```
root@testsystem:~# ethtool -G enp33s0f0 rx 8192 tx 8192
```

Older kernels or drivers do not support byte queue limits, and bigger values might bloat the transmit (TX) interrupt queue buffers. AMD recommends using the latest kernel and driver versions for optimal performance.

Enabling interrupt coalescing will avoid interrupt throttling, and enabling it may add latency. If you need higher throughput vs. latency, then enable interrupt coalescing. Keep the default setting for latency-sensitive applications.

4.4.3 - Preferred I/O and PCIe Relaxed Ordering

3rd Gen and prior AMD EPYC processors included the **Preferred I/O** and **Relaxed Ordering** settings that helped optimize network and disk I/O performance. 5th Gen AMD EPYC processors (9xx5 models) include architectural enhancements that deliver optimal network and disk I/O performance by default without the need for either of these features.

4.4.4 - IOMMU Settings

The Linux kernel is constantly being updated. You can obtain official mainline releases that typically include the latest features from The Linux Foundation at http://kernel.org*. Common enterprise-grade Linux distributions rarely use the latest mainline kernel.

```
# vi /etc/default/grub
add "iommu=pt" to the GRUB_CMDLINE_LINUX_DEFAULT line
GRUB_CMDLINE_LINUX_DEFAULT="splash=silent resume=/dev/disk/by-path/pci- 0000:03:00.0-scsi-0:0:0:0-part4
mitigations=auto iommu=pt quiet"
# grub2-mkconfig -o /boot/grub2/grub.cfg # reboot (required to take effect)
```

AMD testing observed a scaling bottleneck when deploying NGINX instances with more than 32 vCPUs within a single host OS Ubuntu 18.04 image with kernel 4.20 and `wrk` requesting static 1 KB content. Updating to kernel 5.5 that includes IOMMU-optimized patches allowed NGINX instances up to 256 vCPUs to exhibit expected scaling performance. You may also try deploying a guest OS image for each scale out 32 vCPUs NGINX instance. AMD strongly recommends using Linux kernel 5.20 or newer with IOMMU optimized patches.

4.5 - System Configurations

4.5.1 - Simultaneous Multi-Threading (SMT)

Enabling SMT depends on your application. AMD testing saw improved performance with SMT=ON for static-content requests of ~1 KB when using NGINX as a web server. AMD recommends enabling SMT in BIOS.

4.5.2 - Max File Open

NGINX allows users between 1024 (soft limit) and 4096 (hard limit) open files by default. You can increase this to 65535 for a NGINX worker. Make this change in nginx.service by adding the `LimitNOFILE` option with a value.

```
#nproc – number of processes
#nofile – number of file descriptors
*     soft     nproc     32768
*     hard     nproc     65535
*     soft     nofile    32768
*     hard     nofile    65535
root  soft     nproc     32768
root  hard     nproc     65535
root  soft     nofile    32768
root  hard     nofile    65535
```

4.5.3 - Configure Limits

`/etc/security/limits.conf` contains the system limits for various resources. Setting these values is important to make `sysctl .conf` parameters work for the specified values.

4.5.4 - Sysctl Configuration

`Sysctl` modifies kernel parameters at runtime which helps boost performance at the OS level. "Sample /etc/sysctl1.conf File" on page xxiii has a sample `/etc/sysctl.conf` that was tested to scale up to 64 vCPUs in VM environments and up to 384 vCPUs in bare metal environments.

4.6 - NGINX Parameters

NGINX documentation incl0udes many parameters that can significantly impact throughput, response time, and system resource utilization. AMD validated most of the recommendations in this section for NGINX as a web server for static and dynamic content using `wrk` as a load generator. Be sure to verify the impact of these parameters in your production environment.

The easiest way to do this is to create test scripts that are optimized for performance so as to obtain stable and predictable results and separate test scripts that match production requirements. For example, the keepalive tuning, disable access log, and other settings are idea for test scripts. They will not provide absolute performance figures that apply directly to a production environment, but they indicate whether the setting is likely to have a significant impact.

4.6.1 - Application Parameters Matching sysctl

Configuring all OS-level tunings, and then make sure that the application is aware of resource availability by enabling application-level parameters such as `worker_rlimit_nofile` which is equal to the number of file descriptors, `open_file_cache`, `open_file_cache_valid`, `open_file_cache_min_uses`, `open_file_cache_errors` and `keepalive_requests`.

4.6.2 - Worker_processes

It is important to adjust `worker_processes` based on the use case when assigning more vCPUs to a NGINX instance. It is common practice to run 1 worker process per vCPU. There are times when you may want to increase this number, such as when the worker processes must do a lot of disk I/O.

```
worker_processes number | auto; Default: worker_processes 1;
```

4.6.3 - Worker_connections

Each worker process can handle 512 simultaneous processes by default, but most systems have enough resources to support a larger number. The appropriate setting depends on the use case, content size, etc., and should be evaluated via testing.

```
worker_connections number; Default: worker_connections 512;
```

4.6.4 - Access Logging

NGINX writes client request information in the access log right after processing the request. Enabling default access logging causes very poor scaling beyond 4 VCPUs NGINX instance because multiple threads are attempt to write to the same log file, which results in `osq_lock()` dominating the profile. You can also try enabling conditional access logging to reduce the locking issue. AMD recommends disabling access logging to evaluate the best possible performance. Enable it later to test with a logging infrastructure that can support both scale up and scale out NGINX instances.

```
 access_log path [format [buffer=size] [gzip[=level]] [flush=time] [if=condition]];
access_log off;
```

```
Default: access_log logs/access.log combined;
```

4.6.5 - Sendfile and tcp_nopush

Enabling the `sendfile()` directive eliminates the step of copying the data into the buffer and enables direct copying data from one file descriptor to another. Enabling `sendfile()` improves performance for request with large content size (>1MB) and results in a small performance loss for requests with small content sizes. AMD recommends setting `sendfile_max_chunk` to equal to the request content size. If requests could be a mix of various content sizes, then set it to the typical average request size.

Use the `tcp_nopush` directive together with the `sendfile on;directive` to enable NGINX to send HTTP response headers in one packet right after `sendfile()` obtains the chunk of data.

```
sendfile on | off; Default: sendfile off;
sendfile_max_chunk size; Default: sendfile_max_chunk 0;
tcp_nopush on | off; Default: tcp_nopush off;
```

4.6.6 - Keepalive

NGINX enables `keepalive` by default. Keepalive connections can have a major impact on performance by reducing the CPU and network overhead needed to open and close connections. Production environment needs may restrict keeping connections alive. AMD recommends exploring solutions that can enhance security and performance.

```
keepalive_requests number; Default: keepalive_requests 100;
keepalive_timeout timeout [header_timeout]; Default: keepalive_timeout 75s;
keepalive_disable none | browser ...; Default: keepalive_disable msie6;
```

4.6.7 - HTTP vs. HTTPS

NGINX web server setups evaluating the impact of HTTP vs. HTTPS traffic found minimal performance and latency impact when comparing HTTP and HTTPS requests being sent by the wrk tool.

4.6.8 - Caching and Compression

Enabling caching on NGINX web server will boost performance. Most of the time, serving stale content while re-validating in the background will enhance user performance and engagement. Set the caching limit and duration by analyzing the load on the server. Make sure that there is sufficient memory available to store the hot cached content in the OS page cache when caching is enabled.

NGINX supports Gzip compression. Compression decreases file sizes and accelerates the transfer rate from the server to the client, which can boost application performance.

4.7 - NGINX Build with the Latest GCC Compiler Using Znver5 Optimizations

AMD used `znver5` to enable "Zen 5" support in GNU Compiler Collection (GCC) versions 13 and later.

4.8 - Orchestration and Container Settings

Production NGINX deployments often use orchestration on all types of infrastructure such as bare- metal, containers, VMs, and public, private clouds, and hybrid clouds. Orchestration frameworks such as Kubernetes are most prevalent for automating deployment, scaling, and managing containerized applications.

AMD used a test setup created using Kubernetes for orchestration and Docker NGINX containers. NGINX web server often has suboptimal performance settings related to container overlay, networking, file systems, and health monitoring. Here are some references for overlay network configurations and performance comparison among the popular ones in use:

- Cluster Networking*
- Networking Overview*

Here is a sample deployment file for NGINX on K8S with no CNI:

```
kind: DaemonSet
metadata:
  name: web-server-payload
  namespace: nginx-ingress
spec:
  selector:
    matchLabels:
      app: web-server-payload
  template:
    metadata:
      labels:
        app: web-server-payload
spec:
  hostNetwork: true
  containers:
    - name: web-server-payload
    image: mynginx:latest
    imagePullPolicy: IfNotPresent
    ports:
      -  containerPort: 80
      -  containerPort: 443
  nodeSelector:
    name: webserver
```

Note: Developers and administrators should be aware there is a known issue with NGINX ignoring the `cgroup` resource requests and limits created by Kubernetes and will use all CPUs on the host by default. A possible workaround is to set the number of worker processes parameter to the desired number of CPUs in the NGINIX configuration file.

Pinning `containerd` to certain cores can reduce overhead. Docker, K8S, and other container platforms use `containerd` to abstract away `syscalls` or OS-specific functionality to run containers on Linux, Windows, and other operating systems. For example:

$ sudo v`im /usr/lib/systemd/system/containerd.service`

`[Service]`

`CPUAffinity=`24

THIS PAGE INTENTIONALLY LEFT BLANK.

Chapter 5: References and Setup

This tuning guide focuses on NGINX-related optimizations and tunings, but a complete system requires tuning all modules. See the AMD Documentation Hub for the complete list of available tuning guides, and be sure to use the latest version applicable to your AMD EPYC processor model. Some specific tuning guides you may want to look at (available from the AMD Documentation Hub) include:

- *Linux® Network Tuning Guide for AMD EPYC™ 9005 Series Processor Based Servers*
- *Microsoft® Windows® Network Tuning Guide for AMD EPYC™ 9005 Series Processor Based Servers*
- *BIOS & Workload Tuning Guide for AMD EPYC™ 9005 Series Processor Based Servers*

5.1 - NGINX Setup for Cloud VMs

See the following resources for cloud VM instances based on AMD EPYC 9005 Series Processors with Ubuntu 20.04:

- **wrk:** https://github.com/wg/wrk*
- **NGINX:** http://nginx.org/en/download.html*

5.2 - NGINX Setup for Bare Metal

See the following resources for bare metal VM instances:

- **wrk:** https://github.com/wg/wrk*
- **NGINX:** http://nginx.org/en/download.html*
- **OS:** Ubuntu 20.04 with kernel 5.13

5.3 - NGINX Compilation with GCC

See the following resources for NGINX compiled using the latest GCC 12:

- **NGINX:** http://nginx.org/en/download.html*
- **Compiler:** Latest GCC 12 to build NGINX 1.22 version with Znver4, O3 optimization flags

5.4 - Orchestration and Container Setup

See the following resources for NGINX using Kubernetes to orchestrate Docker containers:

- **K8S env:** OS: 5.3.0-64-generic #58-Ubuntu SMP Fri Jul 10 19:33:51 UTC 2020 x86_64 kubeadm version: &version.Info{Major:"1", Minor:"19", GitVersion:"v1.19.0"
- **Docker version:** 19.03.12, build 48a66213fe
- **Official NGINX container image pulled from Docker Hub:** 1.19.2

Chapter 6: Sample /etc/sysctl1.conf File

This example presents a sample `/etc/sysctl.conf` that scales well up to 64 vCPUs in VM environments and up to 256 vCPUs in bare metal environments:

```
# /etc/sysctl.conf
########## Kernel##############
# Controls the System Request debugging functionality of the kernel kernel.sysrq = 0

# Controls whether core dumps will append the PID to the core #filename.
Useful for debugging multi-threaded applications kernel.core_uses_pid = 1
# increase system file descriptor limit fs.file-max = 65535

#Allow for more PIDs kernel.pid_max = 65536

#Enable ExecShield protection kernel.exec-shield = 1
kernel.randomize_va_space = 1


kernel.printk = 4 4 1 7
kernel.panic = 10
kernel.shmmax = 4294967296
kernel.shmall = 4194304
kernel.msgmnb = 65536
kernel.msgmax = 65536


########## Swap ##############
vm.swappiness = 10
vm.vfs_cache_pressure = 50
vm.dirty_ratio = 80

########## IPv4 networking ##############
# Controls IP packet forwarding net.ipv4.ip_forward = 0

# Controls source route verification net.ipv4.conf.default.rp_filter = 1

# Do not accept source routing net.ipv4.conf.default.accept_source_route = 0

# Send redirects, if router, but this is just server net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.default.send_redirects = 0

# Accept packets with SRR option? No net.ipv4.conf.all.accept_source_route = 0

# Accept Redirects? No, this is not router net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.all.secure_redirects = 0

# Log packets with impossible addresses to kernel log? yes net.ipv4.conf.all.log_martians = 1
net.ipv4.conf.default.log_martians = 1
net.ipv4.conf.default.accept_source_route = 0
net.ipv4.conf.default.accept_redirects = 0
net.ipv4.conf.default.secure_redirects = 0

# Ignore all ICMP ECHO and TIMESTAMP requests sent to it via broadcast/multicast
net.ipv4.icmp_echo_ignore_broadcasts = 1

# Turn on protection for bad icmp error messages net.ipv4.icmp_ignore_bogus_error_responses = 1
# Prevent against the common 'syn flood attack' net.ipv4.tcp_syncookies = 1

# Controls the use of TCP syncookies net.ipv4.tcp_synack_retries = 2

# Enable source validation by reversed path, as specified in RFC1812 net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1

# TCP and memory optimization
# increase TCP max buffer size setable using setsockopt() net.ipv4.tcp_rmem = 4096 87380 8388608
net.ipv4.tcp_wmem = 4096 87380 8388608

# increase Linux auto tuning TCP buffer limits net.core.rmem_max = 8388608
net.core.wmem_max = 8388608
net.core.netdev_max_backlog = 5000
net.ipv4.tcp_window_scaling = 1

#Increase system IP port limits net.ipv4.ip_local_port_range = 2000 65000
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 0
net.ipv4.tcp_fin_timeout = 60
net.ipv4.tcp_slow_start_after_idle = 0
net.ipv4.tcp_mtu_probing = 1

# TCP Fast Open net.ipv4.tcp_fastopen = 3

net.ipv4.tcp_congestion_control = htcp

########## IPv4 networking ends ##############

########## IPv6 networking start ##############

# Number of Router Solicitations to send until assuming no routers are present.
# This is host and not router net.ipv6.conf.default.router_solicitations = 0

# Accept Router Preference in RA? net.ipv6.conf.default.accept_ra_rtr_pref = 0

# Learn Prefix Information in Router Advertisement net.ipv6.conf.default.accept_ra_pinfo = 0

# Setting controls whether the system will accept Hop Limit settings # from a router advertisement
net.ipv6.conf.default.accept_ra_defrtr = 0


#router advertisements can cause the system to assign a global unicast #address to an interface
net.ipv6.conf.default.autoconf = 0


#how many neighbor solicitations to send out per address? net.ipv6.conf.default.dad_transmits = 0

# How many global unicast IPv6 addresses can be assigned to each interface?
net.ipv6.conf.default.max_addresses = 1

########## IPv6 networking ends ##############
```

Chapter 7: Resources

- NGINX on AMD powered Amazon C5a instances
- NGINX on AWS powered Amazon M5a instances
- NGINX on AMD powered Azure Eav4 instances
- NGINX on AMD powered Google Cloud N2D confidential VMs
- NGINX on AMD powered GCP N2D standard VMs

NGINX® Tuning Guide for AMD EPYC™ 9005 Processors

PID: 58489

Anil Rajput