

Vertica Eon Mode with VAST

Architecture and Configuration Guide

VERTICA

 VAST

Table of Contents

Introduction	3
About Vertica	3
About VAST	3
Vertica Deployment Architectures	4
Vertica in Eon Mode	4
VAST with Vertica	6
Assumptions	6
Speed	6
Data reduction	6
Simplicity	6
Sizing and Environment	7
Network and Load Balancing Environment	7
Storage Sizing	8
Vertica Hardware Requirements	10
Platform/OS Requirements	11
Prepare Communal Storage	12
Log into the CLI	12
Create Element Store View Policy	12
Create Element Store View	13
Create Vertica Bucket(s)	13
Install and Configure Vertica	15
Prepare the platform	15
Install Vertica Binaries	23
Running the Vertica Installer	24
Create an Eon Mode Database	25
Performance	27
Notes and Configuration	30
Configuration	30
Data Reduction	30
Notes and Observations	32

Introduction

In the document we combine the flexibility and power of Vertica Eon mode with the VAST Data Platform to create an unbeatable price/performance solution to meet any need. This paper covers the configuration of VAST to supply S3 storage to Vertica, the installation and configuration of Vertica in Eon mode up to the creation of your first database.

About Vertica

Vertica's SQL database is trusted by the world's leading data-driven companies, including Cerner, Etsy, Uber, and more to deliver speed, scale, and reliability on mission-critical analytics. Vertica is a database and query engine that leverages the power of cloud-based technologies, but can be deployed on-premises as well. SQL, Python, time series, geospatial, and machine learning analytics can help you unlock the true potential of your data no matter where it sits.

About VAST

By applying new thinking to how storage can be designed, scaled and how data can be efficiently managed, VAST Data has built an architecture with the principal focus of breaking the long standing trade-off that has existed between performance and capacity. We have introduced a new system that saves customers up to 90% vs. what they've paid for legacy flash technology.

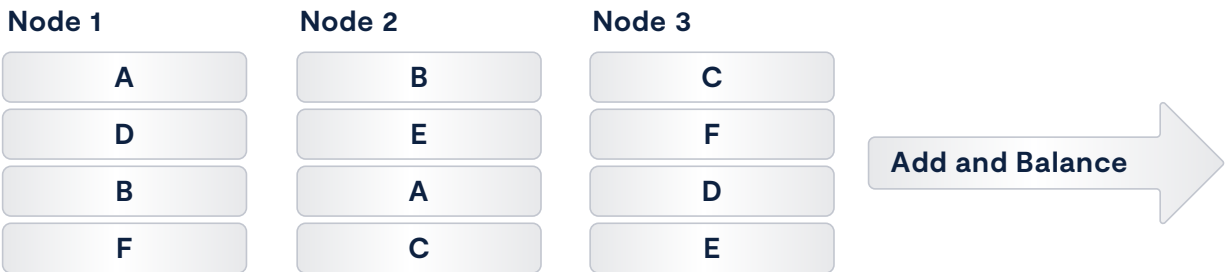
VAST's disaggregated, shared-everything architecture (DASE) is a purpose-built storage cluster architecture designed to scale to petabytes, give tier-1 all-flash performance, and sells for a price point equivalent to what customers pay for archive storage. By breaking the long-standing price/performance trade-off in storage, VAST has built a system architecture that's fast enough, scalable enough and affordable enough for all data and all applications.

Vertica Deployment Architectures

Vertica deploys in one of two modes. 1) **Enterprise Mode**, where data is resiliently stored on local storage in a shared-nothing architecture. 2) **Eon Mode** where all data is housed on a communal object store and staged to local file system storage for operations as needed.

Traditional Enterprise Vertica Deployment

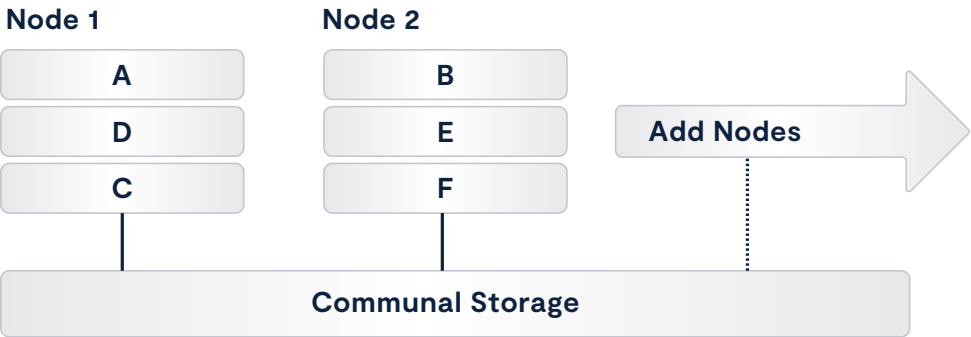
A Vertica deployment in Enterprise mode comprises multiple cluster nodes where data is distributed across nodes for performance and resilience. This architecture combines direct-attached storage and CPU together for data processing.



While this architecture comes with the benefit of keeping all data close to the CPUs, it creates a situation where CPU and storage are proportionally linked together. You cannot add CPU or storage capacity independently. This may be suitable for clusters with certain types of static tasking, but it won't be suitable for systems where additional types of analytics and/or data need to be added or changed over time. It is also worth mentioning that in enterprise mode, expansions, upgrades and other maintenance operations will involve activities that will degrade cluster performance for the duration of the operation.

Vertica in Eon Mode

To make Vertica more flexible for data-center and cloud deployments Eon mode was introduced. Eon mode decouples compute and storage by adding a communal object store that is accessible by all of the nodes in a Vertica cluster. The communal object store is used to populate fast local storage on the Vertica nodes as needed while all persistent data is housed in communal storage.



This architecture has several advantages:

- Eon mode relies on the VAST object store for extremely efficient data resilience and reduction techniques, eliminating the need for replication within Vertica.
- Decoupling storage and CPU allows you to add capacity or compute as needed by expanding the storage system or the Vertica cluster independently.
- Expand without affecting running workloads: adding nodes does not require a data rebalance that consumes I/O on the existing nodes.
- You can separate Vertica workloads by attaching additional Vertica subclusters to the communal object store.
- Ease and flexibility of maintaining Vertica clusters.

Eon mode appears to come with an obvious tradeoff: new nodes or changing workloads require that local storage be “rehydrated” from slower object storage. Intuition may tell you that this will result in slower performance over-all, however the opposite is usually true. And with the VAST Data Platform, access to communal storage is so fast that in some use cases local storage may even be obviated.

VAST with Vertica

The VAST Data Platform comes with several advantages, in general, and for Vertica Eon mode in particular:



Speed

Performance is the most pressing concern when considering the switch to Eon mode. VAST puts this concern to rest with high-performance across a wide range of object sizes with linear parallelism. This doesn't just impact Eon mode; Vertica includes tools that can access data directly on the object store, these will benefit as well.



Data reduction

All-flash can actually cost you less than spinning media since it enables VAST's advanced data reduction techniques. While the amount of this reduction will vary with your data, don't expect to see a "flash" price tag on the solution.



Simplicity

VAST's distributed shared-everything architecture and focus on quality enables a hands-off storage system with very low management overhead and enterprise-grade uptime.

Assumptions

This document makes a few assumption to keep it focused and relevant:

- The operating system platform for Vertica will be a recent Red Hat Enterprise Linux derivative supported by Vertica: RHEL, CentOS, Oracle Linux – all version 7 or 8. This document can still be used for supported Debian variants however some specific shell commands may differ.
- Target Vertica cluster is multi-node (single-node installations are possible but are usually used for development).
- Selected hardware is sufficient for your workload (see Hardware Requirements). This document will allow you to understand and implement any scale you want for both Vertica and VAST but ultimately it assumes you know what you need.

Sizing and Environment

Network and Load Balancing Environment

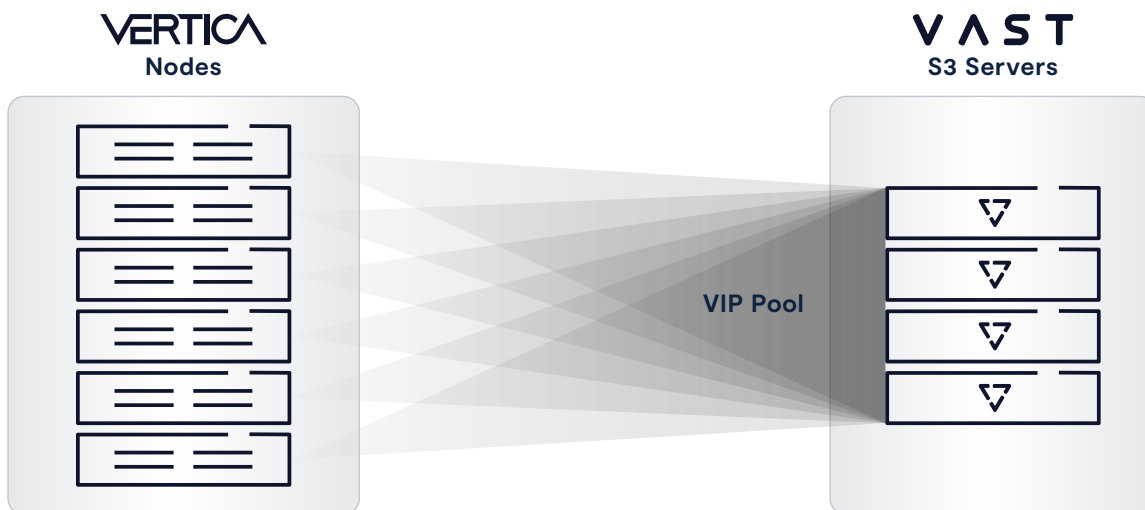
Network Environment

The switch gear connecting the Vertica nodes to the VAST Container servers (CNodes) should allow for no less than 40GbE connectivity per node throughout the system. The VAST cluster itself will be housed on 100GbE.

Load Balancing

All Eon mode Vertica nodes in the cluster will have access to a highly resilient and available object storage system. Object storage is fundamentally different from file system storage in that there are no mount points or file handles and all storage transactions are conducted via discrete, unrelated requests. This type of access makes it easy to optimize traffic to a distributed system with load balancing approaches.

There are many ways that you can load-balance for S3 storage. VAST's DNS delegation can be used to load-balance connections from Vertica nodes to VAST S3 endpoints. Environments that have sufficient networking gear and operate in a cloud-like fashion can use their own application layer load balancers to distribute S3 requests to the VAST endpoints.



Implementation materials for VAST DNS delegation load-balancing are included in the “Install and Configure Vertica” section of this paper.

Storage Sizing

Before considering hardware sizing for the Vertica cluster, and storage sizing for VAST, we need to determine how much storage we need.

Communal Storage Sizing (VAST)

If you know how much data you have, you can determine how much communal storage you'll need with Vertica's communal storage calculator which can be found for version 12 [here](#). For other versions, this calculator can be found at <https://vertica.com/docs> for your version under "Installation > Planning Your Installation > Planning Eon Mode Communal Storage".

In our install example we will be working with:

- a 240TB uncompressed data set
- assuming a compression ratio of 5:1 within Vertica
- allowing for 2 projections (two separate logical views of the data)

Using the storage calculator, we arrive at an on-disk size of 48TB compressed with a communal storage requirement of 200.33 TB.

NOTE

Sizing for VAST is discussed later in this document, however data housed in VAST will probably be smaller than 201.25 TB after data reduction.

Local Storage Sizing (Vertica)

Vertica recommends a minimum of 2T of local storage per node and up to roughly 10TB. The 10TB figure is a rule-of-thumb; the following calculators will call for exceeding that value somewhat at the thresholds, but the ballpark nature of the value holds true.

Vertica will store three classes of data in the local storage area:

- Depot storage: data staged from S3 to the local disks for executing queries
- Data storage: temporary tables and data from sort operators that spill to disk
- Catalog storage: storage that tracks shard information

By default, Vertica will allocate 60% of the local storage to the depot; the remaining 40% is shared by data and catalog storage.

In our example case, we are sizing for a compressed 48 TB working data set. To size for enough local storage per node, the calculation is pretty simple:

$$\frac{48 \text{ TB}}{6 \text{ Nodes}} \times 1.67 = 13.36 \text{ TB}$$

So roughly 14TB of local storage per node is considered optimal for the given data set, allowing for various overheads, data and catalog space.

Shard and Node Count

Shards are how Vertica distributes work throughout a cluster. A single dataset on a smaller cluster (<12 nodes) should probably have at least as many shards as there are nodes in the cluster. Vertica generally recommends a 2:1 ratio of shards to nodes, so in our six-node example, processing a 48TB (compressed) data set, we should probably distribute across 12 shards.

Shard count is a more complex topic than is implied here however. For instance, a 24 node cluster is capable of parallel processing a 12-shard database. It would simply be more efficient and eliminate some cross-talk if there were 24 shards. That said, increasing the shard count excessively will create a larger catalog which will also reduce efficiency.

In short, Vertica recommends the following (from the Vertica documentation [here](#)):

Cluster size	Data Size (compressed)	Shards	Nodes
Small	Up to 24TB	6	3
Medium	Up to 48TB	12	6
Large	Up to 96TB	24	12
Extra large	Up to 192TB	48	24

Table 1

Our data-set lies in the far end of the medium size cluster, calling for 6 nodes and 12 shards.

In summary, our sizing for processing a 240TB data set looks like this:

- 6 Vertica nodes
- 201.25 TB of communal storage required
- 14 TB of local storage each
- 12 database shards.

Vertica Hardware Requirements

CPU and System Memory

Your hardware will need to be sized according to workload. Optimize CPUs for price/performance (2 x gold class Intel processors) and do not stray below 256GB of RAM per node. Regarding memory:

- Vertica recommends 8-12 GB of RAM per core; the installer will issue a warning on install if memory is less than 1GB per logical processor.
- Make sure physical memory installation is optimized to take advantage of available memory channels. This will depend on the hardware vendor.

Local Storage and Cluster Sizing

Local storage sizing was covered above however how to fill that requirement was not. A couple of things to keep in mind:

- If using spinning media, at least 10 SAS disks in a RAID10 configuration should be used. Even so, the VAST system will likely outperform any spinning disk arrangement making the local depot an liability rather than an asset. You will want to use NVMe for local depots.
- Vertica’s “rule-of-thumb” is to store no more than ~10TB of compressed data per node. See “Storage Sizing” above.

Summary of this example system:

Nodes	6
CPUs	2 x Intel 6342 Processors
Memory	256GB
OS	1TB NVMe (can be read-intensive)
Depot	2 X 7TB NVMe, LVM, readahead=4MiB
Network	40GbE interface

Table 2

Platform/OS Requirements

Vertica supports several major Linux distributions for its most recent release (v12 at the time of this writing): RHEL/CentOS 7 and 8, SUSE 12 SP2 and higher, Debian 10, Ubuntu 14.04 and higher, a few others. The installation process is similar on all platforms. The examples in this document will assume a Red Hat based distribution – though the general steps will be the same for other platforms but the exact commands will not.

Install your platform of choice according to the distribution documentation with the following items accommodated:

- LVM should be used for OS partitions
- A swap area of at least 2GB must be present. Use size recommended by the installer if in doubt.
- OS file systems should be XFS or EXT4

Leave the device(s) intended for Vertica's local storage alone.

Prepare Communal Storage

This section assumes that your VAST installation is up and running. To make the procedure clear, we will use the CLI to configure the storage system.

Log into the CLI

- Establish a Secure Shell (SSH) connection to the management IP of any CNode in the cluster.
- For a first time connection, you can use the management VIP that was provided during installation. This is the IP used to connect to the VAST Cluster management server.
- In the VAST Web UI, you can find all CNode Management IPs listed in the **Management IP** column of the **CNodes** tab in the **Infrastructure** page.
- For the first time login, use the OS (Linux) user credentials for CNode.
[Refer to Managing VAST Cluster Passwords](#) for the OS (Linux) user credentials.

To run the CLI for the first time, log in with the factory default management user account:

```
/vast/data/vms.sh vcli -u admin -p {PASSWORD}
```

Create a User

Create a user using any number you like as the UID. This number is arbitrary; it is the POSIX UID when creating NFS exports for this user (which we won't be doing).

```
vcli: admin> user add --uid 42424 --name vertica --allow-create-bucket --allow-delete-bucket
```

Generate access/secret keys that we will need to access our bucket later. Users in VAST are referenced by UID:

```
vcli: admin> user generate-key --uid 42424
{'access_key': '[[ACCESS KEY STRING]]',
 'secret_key': '[[SECRET KEY STRING]]'}
```

Store these keys somewhere, we'll be using them shortly.

Create Element Store View Policy

Since VAST is a multi-protocol storage system with a single namespace, it is necessary to establish some rules with respect to how storage will behave when accessed via those protocols (NFS, SMB, S3, etc). Since Vertica expects a native S3 interface, we will create a policy where S3 is considered the primary protocol so it will not be constrained by the minimums imposed by other protocols. For instance, S3 supports key (path) lengths of 1024 UTF-8 characters. NFS minimums vary, but are usually in the low-hundreds of characters and NFS implementations generally do not recognize all UTF-8 characters.

Create a view policy with S3 being the primary protocol:

```
vcli: admin> viewpolicy create --name vertica --flavor S3_NATIVE --s3-visibility vertica --auth-source PROVIDERS
```

Create Element Store View

VAST uses an ultra-fast file-system-like hierarchy called an Element Store which is used to house the metadata for the protocols layered on top of it (NFS, SMB, S3). The element store can be understood as a top-level namespace that includes “directories” and “files” much like a standard file system. Creating a view in the element store is simply attaching a new storage export to a point in the hierarchy. This view can be an NFS or SMB share, or it can be a new S3 endpoint, which is what we’re creating right now:

```
vcli: admin> view create --path /vertica --protocols ENDPOINT --policy-id 87 --create-dir --bucket-creators vertica
```

Create Vertica Bucket(s)

At this point you should be able to connect to the cNode VIP pool with an S3 client and create a bucket. You may use whatever client you like; here we’ll use the AWS CLI on a nearby Linux/MacOS/Windows host:

- Download the client from [here](#)
- Install according to instructions
- Configure the client with the access and secret keys you created earlier. Using default

```
[user@localhost ~]$ aws configure --profile vertica
AWS Access Key ID [None]: [[ACCESS KEY]]
AWS Secret Access Key [None]: [[SECRET KEY]]
Default region name [None]: us-east-1
Default output format [None]:
[user@localhost ~]$
```

- Test connection using an IP address from your VIP pool (should return nothing without an error)

```
[user@localhost ~]$ aws --profile vertica \
  --endpoint http://[[VIP POOL IP]]/ s3 ls
[user@localhost ~]$
Create the bucket we’ll be using and list it
[user@localhost ~]$ aws --profile vertica \
  --endpoint http://[[VIP POOL IP]]/ s3 mb s3://vertica
[user@localhost ~]$ aws --profile vertica \
  --endpoint http://[[VIP POOL IP]] s3 ls
2022-08-15 20:18:11 vertica
[user@localhost ~]$
```

- Spot-check the bucket to make sure it's writable

```
[user@localhost ~]$ echo "This is a test" | aws --profile vertica \
--endpoint http://[[VIP POOL IP]] s3 cp - s3://vertica/test
[user@localhost ~]$ aws --profile vertica \
--endpoint http://[[VIP POOL IP]] \
s3 cp s3://vertica/test ./
[user@localhost ~]$ cat test
This is a test
[user@localhost ~]$
```

If all goes well, you have a working S3 bucket and you're ready to go.

Install and Configure Vertica

Vertica installation is relatively straight-forward:

- Install the binaries
- Run the cluster installer
- Create a database

The platform assessment tools that run with the cluster installer will direct you to additional documentation to correct and optimize the system according to its findings. More on that later.

NOTE

This guide summarizes the procedures specific to implementing a Vertica Eon mode cluster using VAST as communal storage. The complete generic Vertica documentation can be found [here](#).

NOTE

Vertica supports Red Hat Enterprise Linux 6 and most associated variants, however EL version 6 is not recommended since it went end-of-life in 2020.

NOTE

Cluster sizing is done and the following instructions are pursuant to the example sizing.

Prepare the platform

After OS installation you will need to prepare local storage for Vertica.

Note: all commands in this section are assumed to be run as root unless otherwise indicated.

Using LVM

Make sure the “lvm” command is available.

```
# which lvm || yum -y install lvm2
```

If using LVM to concatenate more than one device together as a single file system, prepare the volumes and file systems on each node. This example is for two NVMe devices (nvme1n1, nvme2n1).

```
# pvcreate /dev/nvme1n1 /dev/nvme2n1
```

Create the volume group:

```
# vgcreate vg_vertica /dev/nvme1n1 /dev/nvme2n1
```

Now create the logical volume using 100% of the volume group space. Note that Vertica does not support striping the logical volume across devices – a common performance enhancer for multi-device volumes. They do recommend using the read-ahead capability which we will configure for 4096K.

```
# lvcreate -r 4096K -l 100%FREE -n data vg_vertica
Logical volume "data" created.
```

Create the file system. Vertica expects a block size of 4096, which is the default for xfs and ext4:

```
# mkfs.xfs /dev/vg_vertica/data
[output suppressed]
```

Create an entry in /dev/fstab that looks like this:

```
/dev/vg_vertica/data /vertica/data xfs defaults 0 0
```

Create the mount-point, inform systemd of the fstab change and mount:

```
# mkdir -p /vertica/data
```

Not necessary on some platforms but won't hurt:

```
# systemctl daemon-reload
```

```
# mount -a
```

Not Using LVM

In this case you are using a single device, not using lvm to concatenate several devices together. Assumes device “nvme1n1”.

Create the file system directly on the device. Vertica expects a block size of 4096, which is the default for xfs and ext4. Remember to do this on all Vertica nodes.

```
# mkfs.xfs /dev/nvme1n1
[output suppressed]
```

Get the UUID value for the file system you just created

```
# blkid /dev/nvme1n1
/dev/sdb: UUID="521ecf08-b12a-40df-86a6-aaeac14acb05" BLOCK_SIZE="4096" TYPE="xfs"
```

Create an /dev/fstab entry using the UUID:

```
UUID=521ecf08-b12a-40df-86a6-aaeac14acb05 /vertica xfs defaults 0 0
```

Create the mount-point, inform systemd of the fstab change and mount:

```
# mkdir /vertica

Not necessary on some platforms but won't hurt:
# systemctl daemon-reload

# mount -a
```

Network configuration

Vertica can be installed on “flat” networks where all traffic traverses the same subnet. It is also possible to separate types of traffic on different networks. For example, you may want clients to access the database via one network while backend cluster traffic and access to communal storage traverses another. In this setup we will use a single network however you may use the “--control-network” option to the install script documented [here](#) to specify the desired network topology.

Ethernet bonding may be used on the Vertica hosts for link redundancy and possibly performance depending on switch configuration. Consult your OS platform documentation to configure for supported bonding types.

Disable Firewall

```
# systemctl disable firewalld --now
```

Operating System Tuning

The following tunings are necessary to complete the Vertica installation step.

NOTE

Many of these tunings are persisted using /etc/rc.local which is deprecated on current platforms. Red Hat variants version 8 and lower still support this mechanism. If you would like to use systemd to implement custom startup actions on EL 7+ rather than using rc.local, you can reference [this article](#).

Prepare rc.local

The rc.local file is used extensively in this section but it needs to be made executable for it to work:

```
# chmod +x /etc/rc.d/rc.local
# systemctl is-enabled rc-local.service
```

Disable tuned

The tuned service is used to automatically perform low-level platform tuning based on a high-level profile. Profiles like "latency-performance" will implement a set of low-level settings to maximize performance at the expense of power consumption, while "powersave" will implement settings to minimize power use.

Since the Vertica installer sets many of the same low-level values as tuned, we'll need to turn it off and let Vertica decide what it wants:

```
# systemctl disable tuned --now
```

Disk Read-Ahead

For each disk in the Vertica systems the read-ahead value should be set to at least 2048KB. In this example we have three devices we want to set: the root volume, the /var volume (both LVM volumes) and the vertica data disk.

```
# for disk in /dev/system/root /dev/system/var /dev/nvme1n1; do
    /sbin/blockdev --setra 4096 $disk
    echo "/sbin/blockdev --setra 4096 $disk" >> /etc/rc.local
done
```

I/O Scheduler

The scheduler is usually set correctly by the operating system however in some cases, such as when a RAID controller is used, it may do something unexpected. A simple rule of thumb is that if your Vertica data disks are flash, you will want the "noop" or "none" scheduler, if they are spinning disk, you'll want the "deadline" or "mq-deadline" scheduler. Under no situation will you want the "cfq" scheduler, though on some older Red Hat versions, "cfq" is the default.

EL 7 and lower use the "noop" and "deadline" schedulers.

Example assumes /dev/nvme1n1 (flash) or /dev/sdb (spinning) is the Vertica data disk

```
For flash media
# echo "noop" > /sys/block/nvme1n1/queue/scheduler
# echo 'echo "noop" > /sys/block/nvme1n1/queue/scheduler' >> /etc/rc.local

For spinning media
# echo "deadline" > /sys/block/sdb/queue/scheduler
# echo 'echo "deadline" > /sys/block/sdb/queue/scheduler' >> /etc/rc.local
```

EL 8 uses the "none" and "mq-deadline" schedulers.

Example assumes /dev/nvme1n1 (flash) or /dev/sdb (spinning) is the Vertica data disk

```
For flash media
# echo "none" > /sys/block/nvme1n1/queue/scheduler
# echo 'echo "none" > /sys/block/nvme1n1/queue/scheduler' >> /etc/rc.local

For spinning media
# echo "mq-deadline" > /sys/block/sdb/queue/scheduler
# echo 'echo "mq-deadline" > /sys/block/sdb/queue/scheduler' >> /etc/rc.local
```

Transparent Huge Pages

Transparent huge pages (THP) is a Linux kernel system that will automatically group memory together into large addressable blocks rather than the default 4096 byte blocks to more efficiently manage addressing in the Translation Lookaside Buffer (TLB). It can improve memory utilization but it can sometimes cause performance issues.

There are three cases for configuring THP for Vertica on Red Hat based platforms:

- EL 6: THP should be disabled or set to “madvice” to allow the system to choose
- EL 7: THP should be explicitly enabled
- EL 8: THP should be explicitly disabled

EL 6 Procedure

THP should be set to “madvice” or “never.” Setting to “never” is usually the safest bet. Check the current configuration:

```
# cat /sys/kernel/mm/transparent_hugepage/enabled
[always] madvice never
```

If the setting is something other than “never” you can update your rc.local to change it on boot:

```
# echo never > /sys/kernel/mm/transparent_hugepage/enabled
# cat << EOF >> /etc/rc.local
if test -f /sys/kernel/mm/transparent_hugepage/enabled; then
    echo never > /sys/kernel/mm/transparent_hugepage/enabled
fi
EOF
```

EL 7 Procedure

THP should be set to “always.” Check the current configuration:

```
# cat /sys/kernel/mm/transparent_hugepage/enabled
always madvice [never]
```

If the setting is something other than “always” you can update your rc.local to change it on boot:

```
# echo always > /sys/kernel/mm/transparent_hugepage/enabled
# cat << EOF >> /etc/rc.local
if test -f /sys/kernel/mm/transparent_hugepage/enabled; then
    echo always > /sys/kernel/mm/transparent_hugepage/enabled
fi
EOF
```

EL 8 Procedure

THP should be set to “never.” Check the current configuration:

```
# cat /sys/kernel/mm/transparent_hugepage/enabled
[always] madvice never
```

If the setting is something other than “never” you can update your rc.local to change it on boot:

```
# echo never > /sys/kernel/mm/transparent_hugepage/enabled
# cat << EOF >> /etc/rc.local
if test -f /sys/kernel/mm/transparent_hugepage/enabled; then
    echo never > /sys/kernel/mm/transparent_hugepage/enabled
fi
EOF
```

Swappiness

Kernel swappiness “controls the relative weight given to swapping out of runtime memory as opposed to dropping memory pages from the system page cache.” In effect, a larger value will make the system “swappier.”

Vertica recommends a value of “0” and it should be configured via an entry in /etc/sysctl.conf with the following line:

```
# echo 'vm.swappiness = 0' >> /etc/sysctl.d/60-vertica.conf
```

Make the configuration immediately active by running sysctl.

```
# sysctl -p /etc/sysctl.d/60-vertica.conf
```

NTP Configuration

It is imperative that all clocks in the Vertica cluster be synchronized both internally and with the rest of the outside world. Enterprise Linux distributions 7 and newer will prefer the use of chronyd while version 6 will use ntpd.

- Install an NTP client

```
EL 6
# yum install ntpd
# service ntpd start
EL 7 and 8
# yum install chrony
# systemctl enable chronyd --now
```

- Make sure valid, reachable NTP servers are configured:

```
EL 6
# edit /etc/ntpd.conf
[make sure “server” values are valid and reachable]
EL 7 and 8
# edit /etc/chrony.conf
[make sure “server” values are valid and reachable]
```

- **Test the configuration:**

```
EL 6
# service ntpd restart
[ wait 5 min ]
# ntpstat
synchronised to NTP server (132.163.97.1) at stratum 3
time correct to within 42 ms
polling server every 1024 s
```

```
EL 7 and 8
# systemctl restart chronyd
[ wait 5 min ]
# chronyd tracking
Reference ID      : 85F3EEA3 (ntp-b2.nict.go.jp)
Stratum           : 2
Ref time (UTC)    : Tue Aug 16 18:28:20 2022
System time       : 0.001767848 seconds slow of NTP time
Last offset       : -0.002855914 seconds
RMS offset        : 0.002855914 seconds
Frequency         : 47.937 ppm slow
Residual freq     : -0.000 ppm
Skew              : 33.422 ppm
Root delay        : 0.112360328 seconds
Root dispersion   : 0.001280128 seconds
Update interval   : 65.3 seconds
Leap status       : Normal
```

You should see a low value for “Stratum” – a value of 16 means the system is not synchronized. Check the server configuration and make sure UDP port 323 is accessible on the configured NTP servers.

SELinux

SELinux is not supported except in permissive mode. Since permissive mode incurs some overhead it is recommended that SELinux be disabled entirely. It is possible that some environments will want to maintain permissive mode to ensure logging of selinux violations.

To disable SELinux entirely (recommended):

```
# sed -i "s/^SELINUX=.*SELINUX=disabled/" /etc/selinux/config
```

A reboot of the system is required to disable selinux.

Otherwise, to set in permissive mode:

```
# sed -i "s/^SELINUX=.*SELINUX=permissive/" /etc/selinux/config
# setenforce 0
```

CPU Frequency Scaling

CPU frequency scaling should be disabled to maximize performance. This can be most reliably accomplished through the server's BIOS. These settings can be power-saving measures and some enable temporary overclocking, so it's best to consult your server's documentation for tuning recommendations. They will often provide profiles that you can select. In this case you'll want a "performance" profile.

Memory Fragmentation

All Red Hat variants need to disable huge page memory defragmentation.

EL 6

```
# cat << EOF >> /etc/rc.local
if test -f /sys/kernel/mm/redhat_transparent_hugepage/enabled; then
    echo never > /sys/kernel/mm/redhat_transparent_hugepage/defrag
fi
EOF
```

EL 7 and 8

```
# cat << EOF >> /etc/rc.local
if test -f /sys/kernel/mm/transparent_hugepage/enabled; then
    echo never > /sys/kernel/mm/transparent_hugepage/defrag
fi
EOF
```

Recommended Packages

Vertica support may ask for information from the system that will require the following packages:

```
EL 6

# yum install pstack mcelog sysstat dialog

EL 7 & 8

# yum install gdb mcelog sysstat dialog ncurses-compat-libs
```

At this point your systems should be prepared. Reboot all servers to ensure that they return to service properly configured. The Vertica installer will check the systems and report if any additional configuration is necessary.

Load Balancer Implementation

VAST's DNS delegation load-balancing is simple to set-up, however you must first configure your site's DNS environment to delegate requests to the VAST cluster for its own sub-domain. Refer to "DNS-Based VIP Distribution" in the [cluster administration guide](#) for your VAST version.

You will need to create and configure a DNS service if you will not be using an existing service or would like a separate service for Vertica. In the VAST UI, navigate to "Network Access → DNS" and click "Create DNS Service". If you already have a DNS service click the three-dot Actions button and select Edit. Make sure the following are done:

- Service name
- DNS Service IP
- DNS Service Suffix
- DNS Service Subnet CIDR
- **TTL must equal “0”**

The last item is important since it will tell clients that the address it gets back from a DNS query may be different every time. This will ensure that each of the Vertica nodes will get a rotated address on startup.

You may also want to establish a VIP pool that is dedicated to Vertica. You may do this using the UI by navigating to “Network Access → Virtual IP Pools”. Click “Create VIP Pool”. At a minimum you’ll need to set up:

- A Name
- A VIP pool domain name
- One or more IP ranges.

Whether you’re using an existing pool or creating a new one, you’ll need to know the DNS name for the pool which is shown under “DNS Service FQDN” when viewing or editing the VIP pool. This is used to direct S3 requests to the VIP pool.

Install Vertica Binaries

Install the GPG Key

Before you can install the RPM, you’ll need the Vertica GPG key installed. It can be found here:

<https://www.vertica.com/docs/GPGKey/RPM-GPG-KEY-VERTICA>

You may download and install the key via RPM:

```
# rpm --import https://www.vertica.com/docs/GPGKey/RPM-GPG-KEY-VERTICA
```

If you don’t have direct Internet access, you can download it to /tmp as a file and import that:

```
# rpm --import /tmp/RPM-GPG-KEY-VERTICA
```

Install the RPM

Get the current Vertica RPM from <https://www.vertica.com/> (log into support and download) and simply install the RPM package on all of the target Vertica hosts. This will make the installation tools available.

```
# rpm -Uvh vertica-12xxxx.rpm
```

The installer and associated scripts and binaries are now installed.

Running the Vertica Installer

Prepare SSH Keys

The Vertica installer is run from a single host in the cluster. It will use SSH to reach-out and perform initial configuration on all of the nodes. The easiest way to enable this is to establish an SSH keypair on the install host.

```
# ssh-keygen
[hit return when prompted for a password]
```

Distribute the public key to all target Vertica hosts. In this example we will use hostnames vertica-[1-6], but you may use IP addresses as well. If using hostnames make sure that DNS is properly configured on all target hosts or that all /etc/hosts files are properly configured.

Distribute the keys using ssh-copy-id. You'll be prompted for the root password for each host.

```
# for host in vertica-1 vertica-2 vertica-3 vertica-4 vertica-5 vertica-6
do
    ssh-copy-id $host
done
```

At this point you should be able to execute commands over ssh without a password:

```
# for host in vertica-1 vertica-2 vertica-3 vertica-4 vertica-5 vertica-6
do
    ssh $host hostname
done
```

Run the Installer

Running the installer will perform a number of platform configurations, inspect the system for proper implementation of the manual configuration, establish a database user and configure the Vertica cluster in preparation for the creation of a database.

```
# /opt/vertica/sbin/install_vertica \
--hosts vertica-1,vertica-2,vertica-3,vertica-4,vertica-5,vertica-6 \
--dba-user dbadmin
```

You will be prompted for a password for the dbadmin user that you're creating – supply it. This will be the UNIX account password that you can use to SSH into the system. All database and cluster management is done with this user after creation.

During inspection the system might issue warnings that will include an ID number and a link to documentation that addresses the issue. It will continue the installation process.

If the inspection encounters errors due to improper configuration, the installation will stop. Any error will be accompanied by an ID number and a link to documentation that will allow you to address the issue. If this guide was followed in full, there should be no critical issues.

Full documentation on the installation script can be found [here](#).

The Vertica cluster management tools are installed and you are ready to create an Eon Mode database.

Create an Eon Mode Database

Select the data paths

We provided for a properly sized local storage area on the Vertica nodes to be mounted at “/vertica/data” – set the permissions on this directory to allow the *dbadmin* user to access it.

```
# chown dbadmin:verticadba /vertica && chmod 755 /vertica
```

Since we’re using NVMe for the Eon mode depot we can also use it for temporary tables and the catalog. In this install we will explicitly set those paths during database creation:

```
/vertica/catalog  
/vertica/data  
/vertica/depot
```

Create the object storage authorization file

At this point you will need to switch to the *dbadmin* user and execute all subsequent commands as that user unless otherwise indicated.

```
# sudo -u dbadmin bash
```

You will need an authorization file that will house the endpoint and credentials information that Vertica will need to address the object storage. Placement of this file isn’t important since it is only used by the installer. You’ll need three pieces of information that should already be established:

- The endpoint name you configured when setting up the VIP pool, or the name used in your network load-balancing equipment (ex: <http://vertica.vastcluster.myorg.org>)
- Access key (ex: E8XNXNFZQ0OS8U3NYF50)
- Secret key (ex: OUI0nbWgcmW8AC7+GEFQusMF8kRor4rCrjaAJVop)

As the user “dbadmin” create the `auth_params.conf` file with the endpoint name and the keys for your S3 user.

```
% cat << EOF > ~/auth_params.conf  
awsauth = E8XNXNFZQ0OS8U3NYF50:OUI0nbWgcmW8AC7+GEFQusMF8kRor4rCrjaAJVop  
awsendpoint = vertica.vastcluster.myorg.org  
awsenablehttps = 0  
EOF
```

If you’re not using the standard HTTP/HTTPS port (either 80 and 443 respectively) then you may specify it in the endpoint (e.g.: `vertica.vastcluster.myorg.org:8181`).

Create the database

```
% admintools -t create_db -x ~/auth_params.conf \  
-c /vertica/catalog \  
-D /vertica/data \  
--depot-path /vertica/depot \  
--communal-storage-location s3://vertica \  
-s vertica-1,vertica-2,vertica-3,vertica-4,vertica-5,vertica-6 \  
--shard-count 12 \  
-d my_database
```

Congratulations, you now have an empty Eon Mode Vertica database ready for use.

Performance

Test Environment

Vertica performance was evaluated on a standard 3x3 VAST architecture.

VAST System Info

Enclosures (DBox)	3 x Lightspeed 2U NVMe enclosures
Front-ends (CNode)	3 x 4-node stateless container enclosures (12 nodes)
Network info	100GbE networking

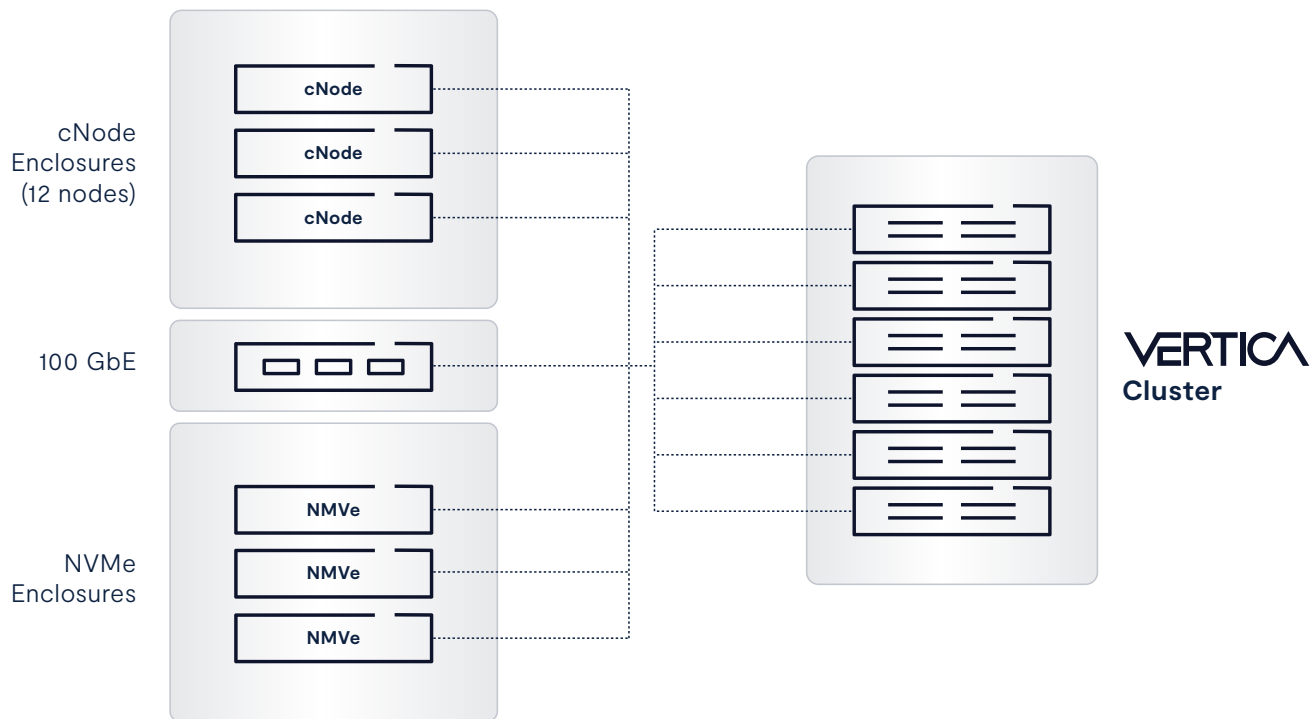
Table 3

Vertica Cluster Info

Node count	6
Processor	2x Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz
Memory	256GB
Network info	100GbE networking
Local data	NFS over RDMA (NVMe performance equivalent)

Table 4

Network Architecture



Benchmark Tests

A standardized 10TB sample database was used to evaluate the performance of the Eon Mode cluster using TPC-DS (decision support) benchmark queries: <https://www.tpc.org/tpcds/>. This suite, supplied by Vertica, tests the following:

- Load data: time to load data into Vertica from flat text files
- External table test: benchmarks tables that are backed directly by parquet files in S3 rather than in the native Vertica format
- Depot on: Queries executed on native Vertica tables using the local storage depot
- Depot off: Queries executed on native Vertica tables by reading directly from the object store

The depot on/off query tests were executed by 1 and 4 users concurrently.

Results are compared to a baseline 6-node Vertica install housed in AWS EC2 instances using flash EBS storage and AWS S3 services directly. Values are in minutes.

Test	AWS	VAST
Load data	181	174
External table bench	184	75
Depot on 1 User	143	108
Depot on 4 Users	340	180
Depot off 1 User	338	98
Depot off 4 Users	766	180

10TB Database Benchmark Timings



It's worth noting that the depot on/off figures for VAST are effectively identical for both concurrent and non concurrent user counts (1 and 4 users). This implies that data processing is the bottleneck and not communal storage in the depot-off tests. While these results do not necessarily conclude that a local depot isn't beneficial for your use-case, they do show that communal storage hosted on VAST holds its own against local NVMe-class storage during an aggressive benchmark that utilizes 100% of the CPU for hours.

Notes and Configuration

Configuration

Storage System

No optimizations were performed on the VAST test system.

Vertica

External table benchmark performance was increased by adding `rowGroupSizeMB=20` to the queries. This setting improves the efficiency of PUT operations on VAST as they relate to the MPU size chosen by Vertica. No other extended configuration or optimizations were deployed during benchmarking.

Data Reduction

A fundamental feature of a columnar data store, be it parquet data or Vertica's read-optimized store (ROS), is that columns can be efficiently encoded and compressed as the contained data is of identical type (character, numeric, binary, etc). This, as opposed to a traditional row-based store that will house several data types going across. This puts the onus of compression traditionally on the database application rather than the storage. That said, VAST can take things a step further.

Data Reduction in Vertica

Vertica uses a system of encoding and compression for column data: Encoding is used to optimize disk I/O and processing access for the target data type; compression is used for overall data reduction. Vertica allows you to set the encoding type and compression using the "ENCODING TYPE" parameter during table creation. If an encoding is not specified it will automatically choose what it believes to be an optimal setting based on the data type.

There are several encoding/compression formats available in Vertica that relate to character and binary data (where compression will yield significant results):

- GZIP_COMP
- BZIP_COMP
- LZO (3 flavors):
 - ZSTD_FAST_COMP
 - ZSTD_COMP
 - ZSTD_HIGH_COMP

Of these options, ZSTD_FAST_COMP will give you the best dedupe and compression in VAST because it is the weakest in Vertica. ZSTD_COMP encoding is auto-selected when no encoding is specified for character data. Our testing has shown only a small byte-count delta between ZSTD_COMP and ZSTD_FAST_COMP encodings so Vertica's defaults might already be sufficient for your use case. However if your database has a lot of character data you may see more

profound data reduction when explicitly setting ZSTD_FAST_COMP as your encoding as in this example:

```
CREATE TABLE public.char_data
(
    index int NOT NULL,
    chapter int,
    title char(255) NOT NULL ENCODING ZSTD_FAST_COMP,
    text varchar(65000) ENCODING ZSTD_FAST_COMP,

);
```

Data Reduction in VAST

The standard TPC-DS benchmark data (which includes some character data but is largely numeric) achieved a data reduction ratio of 1.2:1 (20%) in communal storage when using Vertica's default ZSTD_COMP for character data. Real-world databases which are heavy on character (or in some cases binary) data will exceed that ratio, especially if columns are explicitly created with "ENCODING ZSTD_FAST_COMP".

In summary:

- Vertica default encodings are close to optimal for VAST data reduction.
- ZSTD_FAST_COMP encoding can be explicitly set for character (and binary) data types to improve reduction on character-heavy databases
- Avoid ZSTD_HIGH_COMP, BZIP_COMP and GZIP_COMP encodings as they will consume more CPU on Vertica and will defeat the data reduction benefits of VAST. There's no upside to these encodings.

Notes and Observations

When Eon mode was introduced it was to obviate the need for data replication and to allow for simplified scaling & management. It was assumed to be a possible performance trade-off in exchange for better utilization. In Eon mode, users effectively stage data-sets from slow S3 storage to fast local RAID10 arrays or flash for processing. One risk related to this system is that it might complicate local depot storage sizing where users are uncertain about the amount of local storage they need to accommodate a workload – now and in the future.

This performance risk is only present under the assumption that communal S3 storage is slow. The assumption no longer holds true. VAST's S3 implementation was able to keep a powerful 6-host Vertica cluster processor-bound during 10TB TPC-DS benchmarking, bypassing local storage entirely.



For more information on the VAST Data Platform and how it can help you solve your application problems, reach out to us at hello@vastdata.com.