使用NetApp NFS 存储的 Apache Kafka 工作负载

NetApp artificial intelligence solutions

NetApp
August 18, 2025

目录

使用NetApp NFS 存储的 Apache Kafka 工作负载

TR-4947：使用NetApp NFS 存储的 Apache Kafka 工作负载 - 功能验证和性能

Shantanu Chakole、Karthikeyan Nagalingam 和 Joe Scott， NetApp

Kafka 是一个分布式发布-订阅消息系统，具有强大的队列，可以接受大量消息数据。使用 Kafka，应用程序可以非常快速地向主题写入和读取数据。由于其容错性和可扩展性，Kafka 通常被用在大数据领域，作为一种可靠的方式来快速提取和移动大量数据流。用例包括流处理、网站活动跟踪、指标收集和监控、日志聚合、实时分析等。

尽管 NFS 上的正常 Kafka 操作运行良好，但在 NFS 上运行的 Kafka 集群调整大小或重新分区期间，愚蠢的重命名问题会导致应用程序崩溃。这是一个重大问题，因为必须调整 Kafka 集群的大小或重新分区以实现负载平衡或维护目的。您可以找到更多详细信息 "此处"。

本文档描述了以下主题：

- 愚蠢的重命名问题和解决方案验证
- 降低 CPU 利用率以减少 I/O 等待时间
- 更快的 Kafka 代理恢复时间
- 云端和本地的性能

为什么使用 NFS 存储来存储 Kafka 工作负载？

生产应用程序中的 Kafka 工作负载可以在应用程序之间传输大量数据。这些数据保存并存储在 Kafka 集群中的 Kafka 代理节点中。 Kafka 还以可用性和并行性而闻名，它通过将主题分成多个分区，然后在整个集群中复制这些分区来实现。这最终意味着流经 Kafka 集群的大量数据通常会成倍增加。随着代理数量的变化，NFS 可以非常快速和轻松地重新平衡数据。对于大型环境，当代理数量发生变化时，跨 DAS 重新平衡数据非常耗时，并且在大多数 Kafka 环境中，代理数量经常发生变化。

其他好处包括：

- 到期。 NFS 是一种成熟的协议，这意味着它的实现、保护和使用的大多数方面都已被很好地理解。
- 打开。 NFS 是一个开放协议，其持续发展在互联网规范中被记录为一个自由开放的网络协议。
- 具有成本效益。 NFS 是一种低成本的网络文件共享解决方案，由于它使用现有的网络基础设施，因此易于设置。
- 集中管理。 NFS 的集中管理减少了单个用户系统上添加软件和磁盘空间的需要。
- 分布式。 NFS 可以用作分布式文件系统，减少对可移动介质存储设备的需求。

为什么选择NetApp来处理 Kafka 工作负载？

NetApp NFS 实施被认为是该协议的黄金标准，并被应用于无数企业 NAS 环境中。除了NetApp的信誉之外，它还提供以下优势：

- 可靠性和效率
- 可扩展性和性能
- 高可用性（ NetApp ONTAP集群中的 HA 合作伙伴）
- 数据保护
  - 灾难恢复（NetApp SnapMirror）。*您的网站瘫痪了，或者您想从另一个网站开始并从上次中断的地方继续。
  - 存储系统的可管理性（使用NetApp OnCommand进行管理）。
  - *负载平衡。*该集群允许您从托管在不同节点上的数据 LIF 访问不同的卷。
  - 无中断运行。 LIF 或卷移动对于 NFS 客户端来说是透明的。

NetApp针对 NFS 到 Kafka 工作负载重命名问题的解决方案

Kafka 的构建假设底层文件系统符合 POSIX 标准：例如 XFS 或 Ext4。当应用程序仍在使用文件时，Kafka 资源重新平衡会删除这些文件。符合 POSIX 标准的文件系统允许取消链接继续进行。但是，只有在对该文件的所有引用都消失后，它才会删除该文件。如果底层文件系统是网络连接的，那么 NFS 客户端会拦截取消链接调用并管理工作流程。由于正在取消链接的文件上有待打开的文件，因此 NFS 客户端会向 NFS 服务器发送重命名请求，并在最后一次关闭取消链接的文件时，对重命名的文件发出删除操作。此行为通常称为 NFS 愚蠢重命名，由 NFS 客户端精心策划。

由于这种行为，任何使用 NFSv3 服务器存储的 Kafka 代理都会遇到问题。但是，NFSv4.x 协议具有解决此问题的功能，它允许服务器负责打开的未链接的文件。支持此可选功能的 NFS 服务器在打开文件时将所有权能力传达给 NFS 客户端。当有待处理的打开操作时，NFS 客户端将停止取消链接管理，并允许服务器管理流程。尽管 NFSv4 规范提供了实施指南，但到目前为止，还没有任何已知的 NFS 服务器实施支持此可选功能。

为了解决这个愚蠢的重命名问题，需要对 NFS 服务器和 NFS 客户端进行以下更改：

- *对 NFS 客户端 (Linux) 的更改。*在打开文件时，NFS 服务器会响应一个标志，表明有能力处理打开文件的取消链接。 NFS 客户端的更改允许 NFS 服务器在存在标志的情况下处理取消链接。 NetApp已根据这些更改更新了开源 Linux NFS 客户端。更新后的 NFS 客户端现已在 RHEL8.7 和 RHEL9.1 中普遍可用。
- 对 **NFS** 服务器的更改。 NFS 服务器跟踪打开的情况。现在，服务器管理对现有打开文件的取消链接以匹配 POSIX 语义。当最后一个打开的文件关闭时，NFS 服务器将启动文件的实际删除，从而避免愚蠢的重命名过程。 ONTAP NFS 服务器在其最新版本ONTAP 9.12.1 中实现了此功能。

通过对 NFS 客户端和服务器进行上述更改，Kafka 可以安全地获得网络附加 NFS 存储的所有好处。

功能验证 - 愚蠢的重命名修复

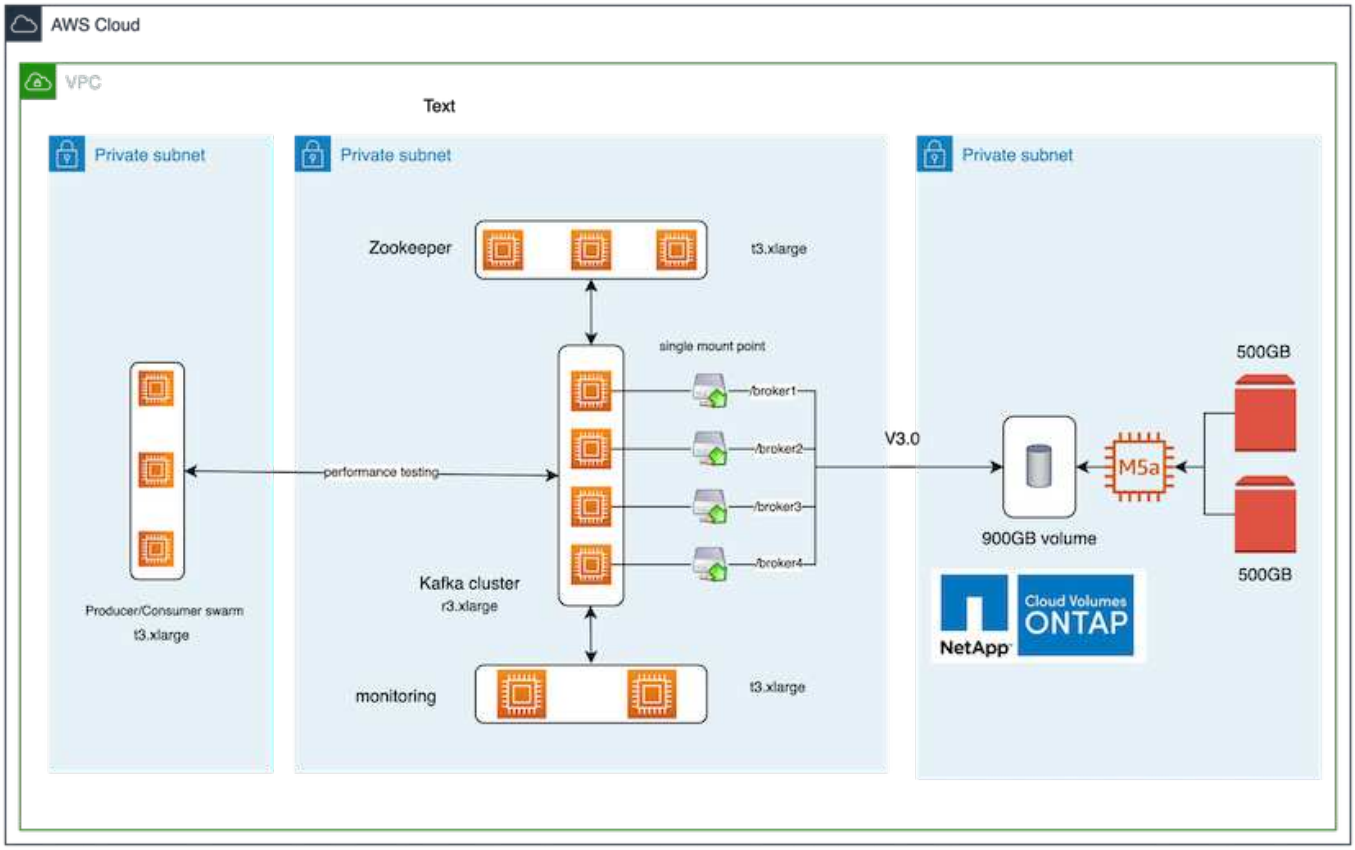
对于功能验证，我们表明，使用 NFSv3 挂载存储的 Kafka 集群无法执行分区重新分配等 Kafka 操作，而使用修复程序挂载在 NFSv4 上的另一个集群可以执行相同的操作而不会出现任何中断。

验证设置

该设置在 AWS 上运行。下表显示了用于验证的不同平台组件和环境配置。

平台组件	环境配置
Confluent 平台版本 7.2.1	• 3 个动物园管理员 – t3.xlarge • 4 个代理服务器 – r3.xlarge • 1 x Grafana – t3.xlarge • 1 x 控制中心 – t3.xlarge • 3 x 生产者/消费者
所有节点上的操作系统	RHEL8.7或更高版本
NetApp Cloud Volumes ONTAP实例	单节点实例 – M5.2xLarge

下图显示了该解决方案的架构配置。



建筑流程

- *计算。*我们使用了四节点 Kafka 集群，并在专用服务器上运行三节点 zookeeper 集合。
- *监控*我们使用两个节点来实现 Prometheus-Grafana 组合。
- *工作量*为了生成工作负载，我们使用了一个单独的三节点集群，该集群可以从该 Kafka 集群中生产和消费。

- *贮存。*我们使用了单节点NetApp Cloud Volumes ONTAP实例，该实例连接了两个 500GB GP2 AWS-EBS 卷。然后，这些卷通过 LIF 作为单个 NFSv4.1 卷公开给 Kafka 集群。

所有服务器都选择了 Kafka 的默认属性。对动物园管理员群也做了同样的事情。

测试方法

1. 更新 `-is-preserve-unlink-enabled true`到kafka卷，如下：

```
aws-shantanclastrecall-aws::*> volume create -vserver kafka_svm -volume
kafka_fg_vol01 -aggregate kafka_aggr -size 3500GB -state online -policy
kafka_policy -security-style unix -unix-permissions 0777 -junction-path
/kafka_fg_vol01 -type RW -is-preserve-unlink-enabled true
[Job 32] Job succeeded: Successful
```

2. 创建了两个类似的 Kafka 集群，但有以下区别：
   - *集群 1.*运行生产就绪ONTAP版本 9.12.1 的后端 NFS v4.1 服务器由NetApp CVO 实例托管。代理上安装了 RHEL 8.7/RHEL 9.1。
   - *集群 2.*后端 NFS 服务器是手动创建的通用 Linux NFSv3 服务器。
3. 在两个 Kafka 集群上都创建了一个演示主题。

   集群 1：

```
[root@ip-172-30-0-160 demo]# kafka-topics --bootstrap-server=172.30.0.160:9092,172.30.0.172:9092,172.30.0.188:90
92,172.30.0.123:9092 --describe --topic __a_demo_topic
Topic: __a_demo_topic   TopicId: 2ty29xfhQLq65HKsUQv-pg PartitionCount: 4       ReplicationFactor: 2    Configs:
 min.insync.replicas=1,segment.bytes=1073741824
        Topic: __a_demo_topic   Partition: 0    Leader: 4       Replicas: 4,1   Isr: 4,1        Offline:
        Topic: __a_demo_topic   Partition: 1    Leader: 2       Replicas: 2,4   Isr: 2,4        Offline:
        Topic: __a_demo_topic   Partition: 2    Leader: 3       Replicas: 3,2   Isr: 3,2        Offline:
        Topic: __a_demo_topic   Partition: 3    Leader: 1       Replicas: 1,3   Isr: 1,3        Offline:
```

   集群 2：

```
[root@ip-172-30-0-198 demo]# kafka-topics --bootstrap-server=172.30.0.198:9092,172.30.0.163:9092,172.30.0.221:90
92,172.30.0.204:9092 --describe --topic __a_demo_topic
Topic: __a_demo_topic   TopicId: AwQpsZTQShyeMIhaquCG3Q PartitionCount: 4       ReplicationFactor: 2    Configs:
 min.insync.replicas=1,segment.bytes=1073741824
        Topic: __a_demo_topic   Partition: 0    Leader: 2       Replicas: 2,3   Isr: 2,3        Offline:
        Topic: __a_demo_topic   Partition: 1    Leader: 3       Replicas: 3,1   Isr: 3,1        Offline:
        Topic: __a_demo_topic   Partition: 2    Leader: 1       Replicas: 1,4   Isr: 1,4        Offline:
        Topic: __a_demo_topic   Partition: 3    Leader: 4       Replicas: 4,2   Isr: 4,2        Offline:
```

4. 数据被加载到这两个集群新创建的主题中。这是使用默认 Kafka 包中的生产者性能测试工具包完成的：

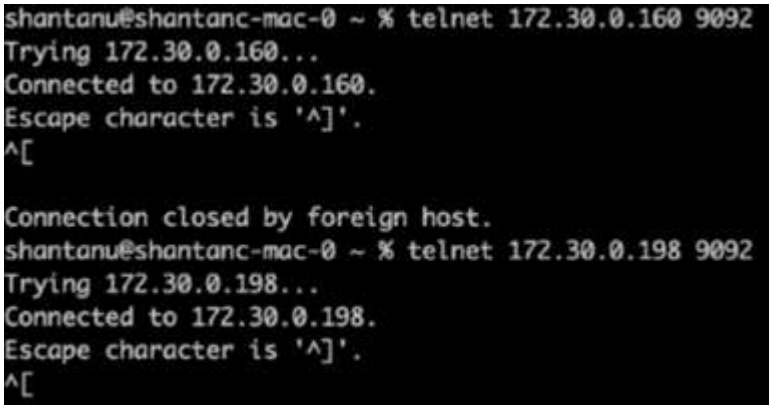
```
./kafka-producer-perf-test.sh --topic __a_demo_topic --throughput -1
--num-records 3000000 --record-size 1024 --producer-props acks=all
bootstrap.servers=172.30.0.160:9092,172.30.0.172:9092,172.30.0.188:9092,
172.30.0.123:9092
```

5. 使用 telnet 对每个集群的 broker-1 执行健康检查：

   ◦ 远程登录 `172.30.0.160 9092`

   ◦ 远程登录 `172.30.0.198 9092`

   下图显示了两个集群上的代理的成功健康检查：



6. 为了触发导致使用 NFSv3 存储卷的 Kafka 集群崩溃的故障条件，我们在两个集群上启动了分区重新分配过程。分区重新分配是使用 `kafka-reassign-partitions.sh`。具体过程如下：

   a. 为了重新分配 Kafka 集群中主题的分区，我们生成了建议的重新分配配置 JSON（对两个集群都执行了此操作）。

   ```
   kafka-reassign-partitions --bootstrap
   -server=172.30.0.160:9092,172.30.0.172:9092,172.30.0.188:9092,172.30.
   0.123:9092 --broker-list "1,2,3,4" --topics-to-move-json-file
   /tmp/topics.json --generate
   ```

   b. 生成的重新分配 JSON 随后保存在 `/tmp/reassignment- file.json`。

   c. 实际的分区重新分配过程由以下命令触发：

   ```
   kafka-reassign-partitions --bootstrap
   -server=172.30.0.198:9092,172.30.0.163:9092,172.30.0.221:9092,172.30.
   0.204:9092 --reassignment-json-file /tmp/reassignment-file.json
   -execute
   ```

7. 重新分配完成后几分钟，对代理进行的另一次健康检查显示，使用 NFSv3 存储卷的集群遇到了一个愚蠢的重命名问题并崩溃了，而使用已修复的NetApp ONTAP NFSv4.1 存储卷的集群 1 继续运行而没有任何中断。

```
shantanu@shantanc-mac-0 ~ % telnet 172.30.0.160 9092
Trying 172.30.0.160...
Connected to 172.30.0.160.
Escape character is '^]'.
^[

Connection closed by foreign host.
shantanu@shantanc-mac-0 ~ % telnet 172.30.0.198 9092
Trying 172.30.0.198...
telnet: connect to address 172.30.0.198: Connection refused
telnet: Unable to connect to remote host
```

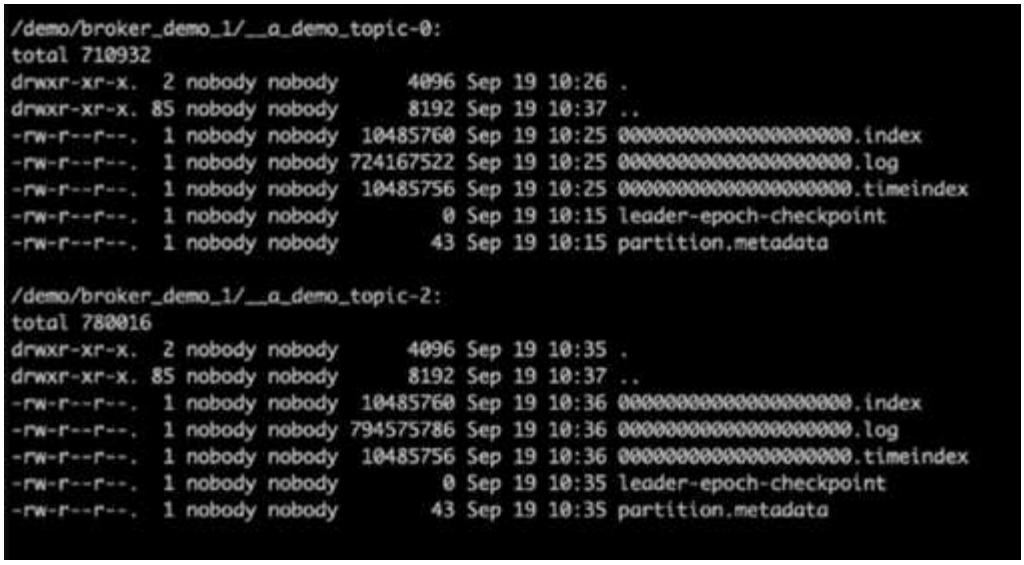
- Cluster1-Broker-1 处于活动状态。
- Cluster2-broker-1 已死亡。

8. 检查 Kafka 日志目录后，很明显，使用已修复的NetApp ONTAP NFSv4.1 存储卷的集群 1 具有干净的分区分配，而使用通用 NFSv3 存储的集群 2 由于愚蠢的重命名问题而没有，从而导致崩溃。下图显示了集群 2 的分区重新平衡，这导致了 NFSv3 存储上出现了一个愚蠢的重命名问题。

```
/demo/broker_demo_1/__a_demo_topic-1.b31a8dd60fd443b283ffda2ecca9c2b9-delete:
total 40
drwxr-xr-x.   2 nobody nobody  4096 Sep 19 10:37 .
drwxr-xr-x. 246 nobody nobody 32768 Sep 19 10:36 ..
-rw-r--r--.   1 nobody nobody     5 Sep 19 10:22 .nfs0000000025f9008400000045
-rw-r--r--.   1 nobody nobody     0 Sep 19 10:25 .nfs0000000025f91d6800000048

/demo/broker_demo_1/__a_demo_topic-2:
total 832592
drwxr-xr-x.   2 nobody nobody      4096 Sep 19 10:26 .
drwxr-xr-x. 246 nobody nobody     32768 Sep 19 10:36 ..
-rw-r--r--.   1 nobody nobody         5 Sep 19 10:22 .nfs0000000025f91d5500000046
-rw-r--r--.   1 nobody nobody         0 Sep 19 10:25 .nfs0000000025f91fce00000047
-rw-r--r--.   1 nobody nobody  10485760 Sep 19 10:24 00000000000000000000.index
-rw-r--r--.   1 nobody nobody 848113134 Sep 19 10:24 00000000000000000000.log
-rw-r--r--.   1 nobody nobody  10485756 Sep 19 10:24 00000000000000000000.timeindex
-rw-r--r--.   1 nobody nobody         0 Sep 19 10:16 leader-epoch-checkpoint
-rw-r--r--.   1 nobody nobody        43 Sep 19 10:16 partition.metadata
```

下图显示了使用NetApp NFSv4.1 存储对集群 1 进行干净的分区重新平衡。



为什么选择**NetApp NFS** 来支持 **Kafka** 工作负载?

既然有了针对 Kafka 的 NFS 存储中愚蠢重命名问题的解决方案，您就可以创建利用NetApp ONTAP存储来处理 Kafka 工作负载的强大部署。这不仅显著降低了运营开销，还为您的 Kafka 集群带来以下好处：

- *降低 Kafka 代理的 CPU 利用率。*使用分解的NetApp ONTAP存储将磁盘 I/O 操作与代理分离，从而减少其 CPU 占用。
- *更快的经纪人恢复时间。*由于分解的NetApp ONTAP存储在 Kafka 代理节点之间共享，因此与传统的 Kafka 部署相比，新的计算实例可以在很短的时间内随时替换损坏的代理，而无需重建数据。
- *存储效率。*由于应用程序的存储层现在是通过NetApp ONTAP进行配置的，因此客户可以利用ONTAP带来的所有存储效率优势，例如线内数据压缩、重复数据删除和压缩。

这些好处在本节我们将详细讨论的测试案例中得到了测试和验证。

降低 **Kafka** 代理的 **CPU** 利用率

我们发现，当我们在两个技术规格相同但存储技术不同的独立 Kafka 集群上运行类似的工作负载时，整体 CPU 利用率低于其 DAS 对应集群。当 Kafka 集群使用ONTAP存储时，不仅整体 CPU 利用率较低，而且 CPU 利用率的增加比基于 DAS 的 Kafka 集群表现出更平缓的梯度。

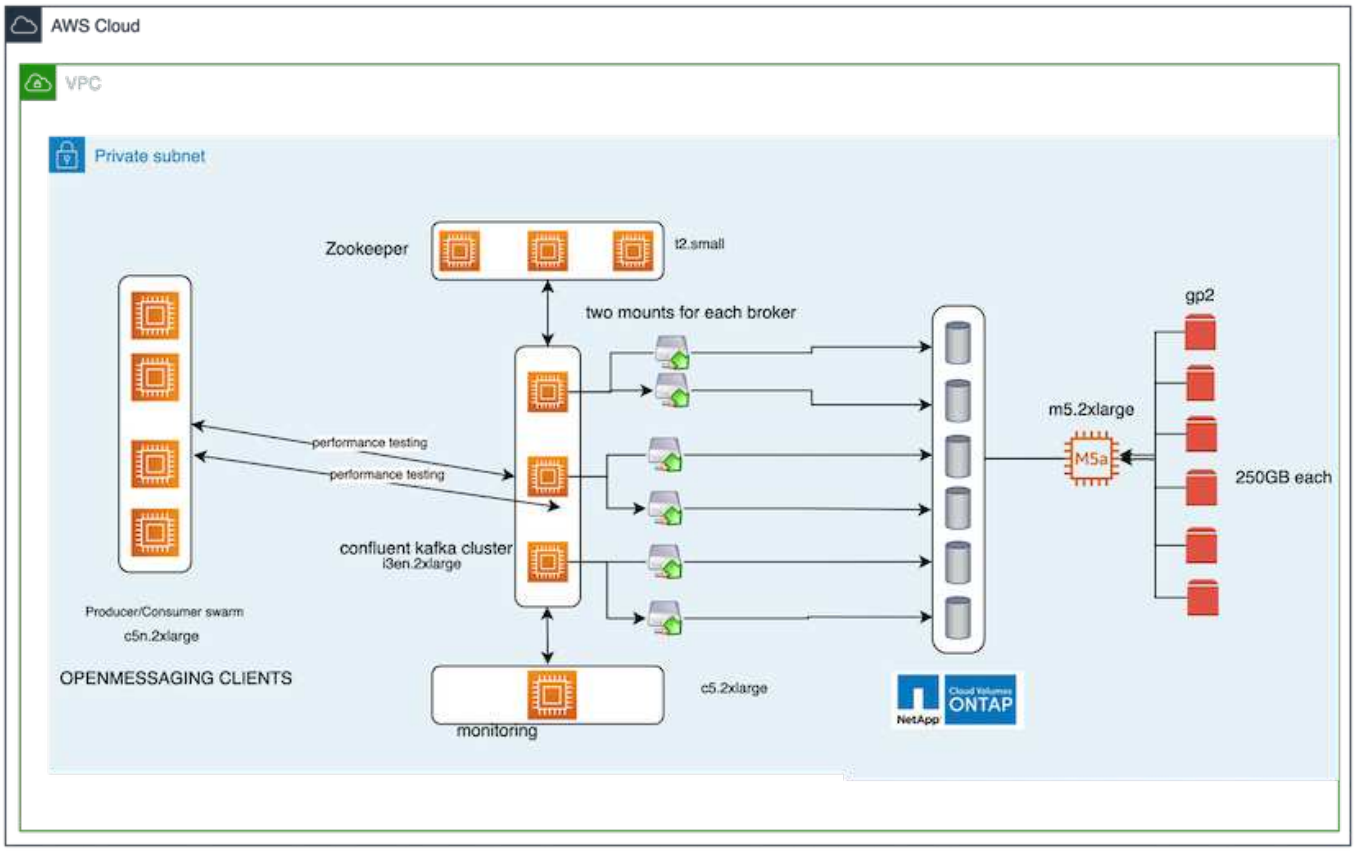
建筑设置

下表显示了用于演示降低 CPU 利用率的环境配置。

平台组件	环境配置
Kafka 3.2.3 基准测试工具：OpenMessaging	• 3 名动物园管理员 – t2.small • 3 个代理服务器 – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x 生产者/消费者 - c5n.2xlarge
所有节点上的操作系统	RHEL 8.7 或更高版本
NetApp Cloud Volumes ONTAP实例	单节点实例 – M5.2xLarge

基准测试工具

本测试用例中使用的基准测试工具是 "开放消息传递"框架。 OpenMessaging 与供应商无关且与语言无关；它为金融、电子商务、物联网和大数据提供行业指南；并有助于开发跨异构系统和平台的消息传递和流媒体应用程序。下图描述了 OpenMessaging 客户端与 Kafka 集群的交互。



- *计算。*我们使用了三节点 Kafka 集群，并在专用服务器上运行三节点 zookeeper 集合。每个代理通过专用 LIF 拥有两个 NFSv4.1 挂载点，指向NetApp CVO 实例上的单个卷。
- *监控*我们使用两个节点来实现 Prometheus-Grafana 组合。为了生成工作负载，我们有一个单独的三节点集群，可以从该 Kafka 集群中生产和消费。
- *贮存。*我们使用了单节点NetApp Cloud Volumes ONTAP实例，该实例上安装了六个 250GB GP2 AWS-EBS 卷。然后，这些卷通过专用 LIF 作为六个 NFSv4.1 卷公开给 Kafka 集群。
- *配置。*该测试用例中的两个可配置元素是 Kafka 代理和 OpenMessaging 工作负载。

- *经纪人配置*为 Kafka 代理选择了以下规格。我们对所有测量使用了 3 的重复因子，如下所示。

```
broker.id=1
advertised.listeners=PLAINTEXT://172.30.0.185:9092
log.dirs=/mnt/data-1
zookeeper.connect=172.30.0.13:2181,172.30.0.108:2181,172.30.0.253:2181
num.replica.fetchers=8
message.max.bytes=10485760
replica.fetch.max.bytes=10485760
num.network.threads=8
default.replication.factor=3
replica.lag.time.max.ms=100000000
replica.fetch.max.bytes=1048576
replica.fetch.wait.max.ms=500
num.replica.fetchers=1
replica.high.watermark.checkpoint.interval.ms=5000
fetch.purgatory.purge.interval.requests=1000
producer.purgatory.purge.interval.requests=1000
replica.socket.timeout.ms=30000
replica.socket.receive.buffer.bytes=65536
```

- OpenMessaging 基准 (OMB) 工作负载配置。*提供了以下规格。我们指定了目标生产率，如下所示。

```
name: 4 producer / 4 consumers on 1 topic
topics: 1
partitionsPerTopic: 100
messageSize: 1024
payloadFile: "payload/payload-1Kb.data"
subscriptionsPerTopic: 1
consumerPerSubscription: 4
producersPerTopic: 4
producerRate: 40000
consumerBacklogSizeGB: 0
testDurationMinutes: 5
```

测试方法

1. 创建了两个类似的集群，每个集群都有自己的一组基准集群群。
   - *集群 1.*基于NFS的Kafka集群。
   - *集群 2.*基于DAS的Kafka集群。
2. 使用 OpenMessaging 命令，每个集群上都会触发类似的工作负载。

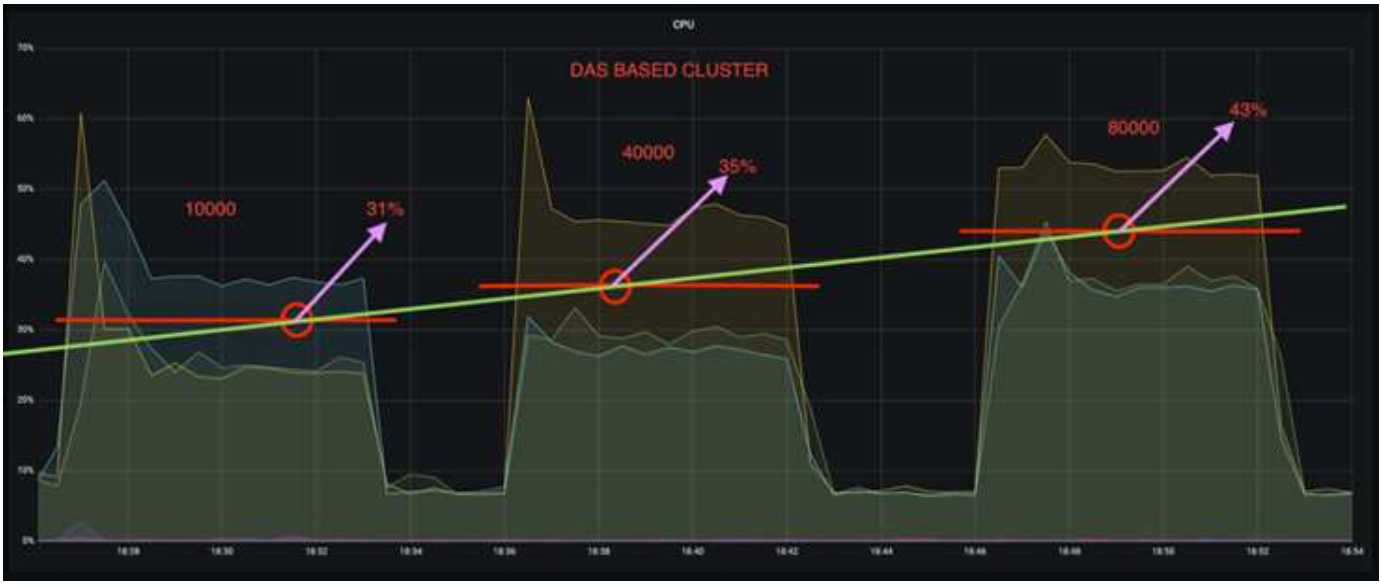
```
sudo bin/benchmark --drivers driver-kafka/kafka-group-all.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

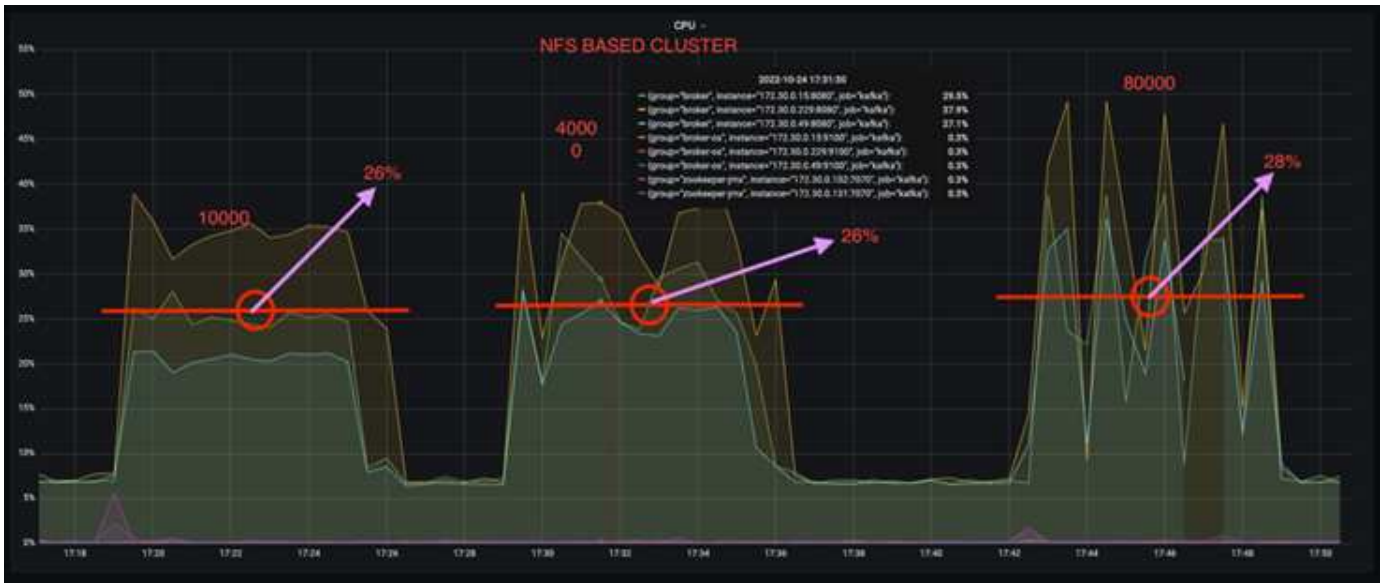
3. 生产率配置在四次迭代中增加，并使用 Grafana 记录 CPU 利用率。生产率设定为以下水平：
   - 10,000
   - 40,000
   - 80,000
   - 100,000

观察结果

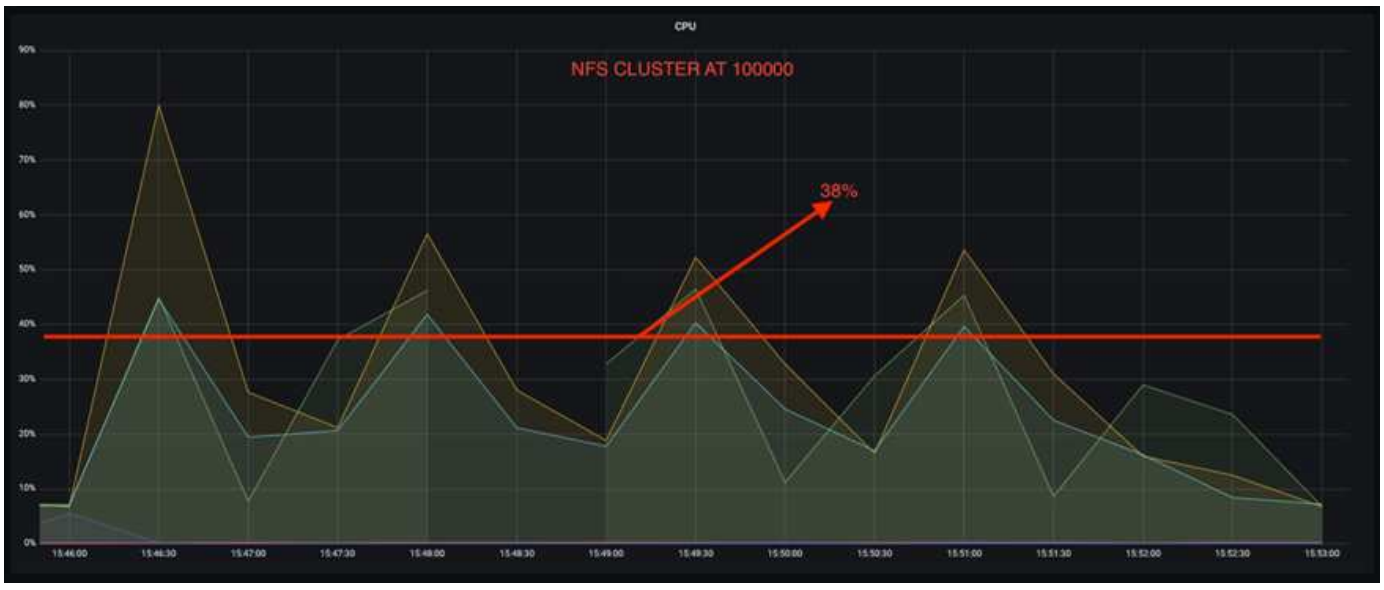
将NetApp NFS 存储与 Kafka 结合使用有两个主要好处：

- *您可以将 CPU 使用率降低近三分之一。*与 DAS SSD 相比，相似工作负载下 NFS 的总体 CPU 使用率较低；节省范围从较低生产率的 5% 到较高生产率的 32%。
- *在更高的生产率下，CPU 利用率漂移减少了三倍。*正如预期的那样，随着生产率的提高，CPU 利用率呈上升趋势。然而，使用 DAS 的 Kafka 代理的 CPU 利用率从较低生产率时的 31% 上升到较高生产率时的 70%，增幅为 39%。然而，有了 NFS 存储后端，CPU 利用率从 26% 上升到 38%，增加了 12%。





此外，在 100,000 条消息时，DAS 显示的 CPU 利用率比 NFS 集群更高。



更快的经纪人恢复

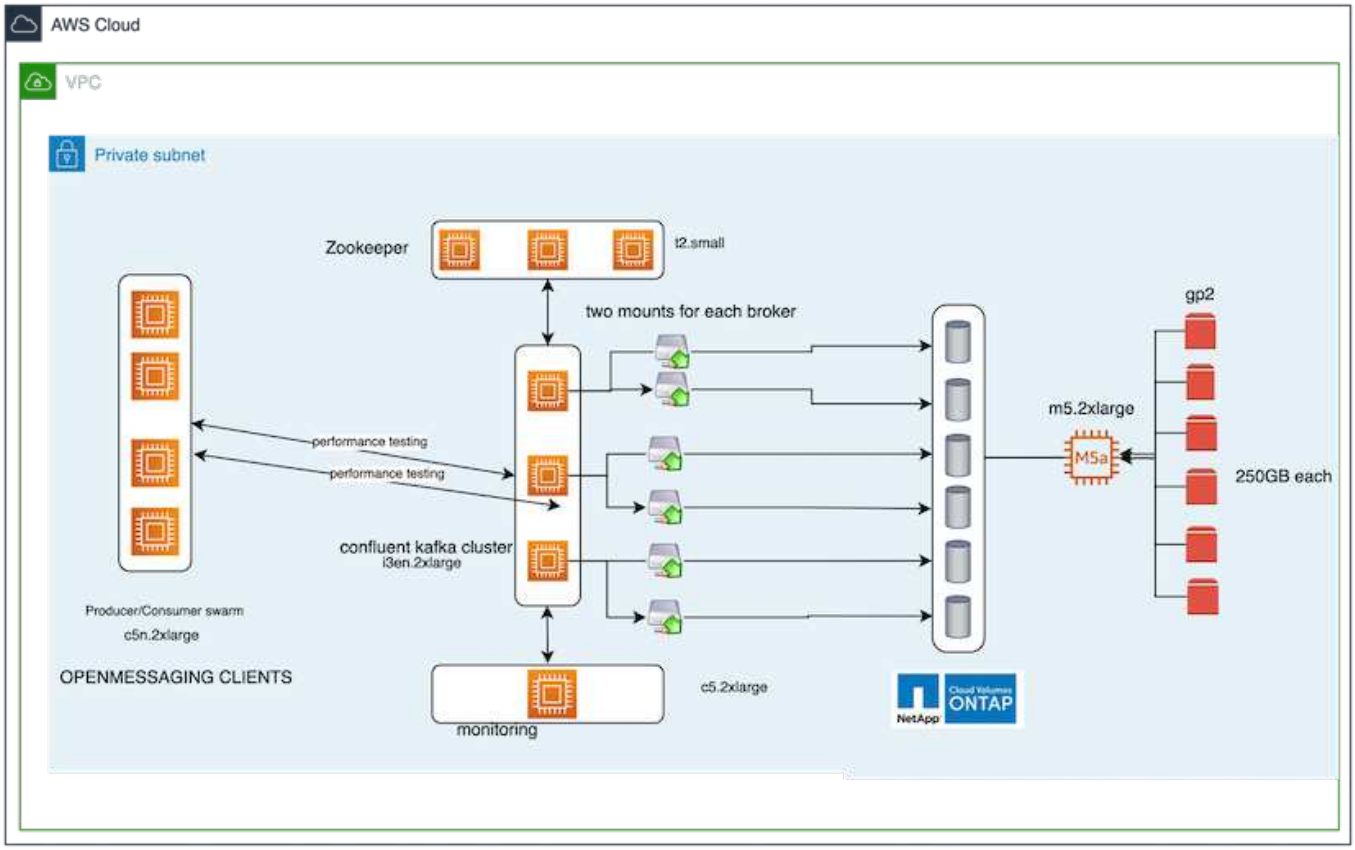
我们发现，当 Kafka 代理使用共享NetApp NFS 存储时，恢复速度更快。当 Kafka 集群中某个 Broker 崩溃时，可以用具有相同 Broker ID 的健康 Broker 替代该 Broker。在执行此测试用例时，我们发现，对于基于 DAS 的 Kafka 集群，集群会在新添加的健康 Broker 上重建数据，这非常耗时。对于基于NetApp NFS 的 Kafka 集群，替换代理将继续从以前的日志目录读取数据，并且恢复速度更快。

建筑设置

下表展示了使用NAS的Kafka集群的环境配置。

平台组件	环境配置
卡夫卡 3.2.3	• 3 名动物园管理员 – t2.small • 3 个代理服务器 – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x 生产者/消费者 - c5n.2xlarge • 1 x 备份 Kafka 节点 – i3en.2xlarge
所有节点上的操作系统	RHEL8.7 或更高版本
NetApp Cloud Volumes ONTAP实例	单节点实例 – M5.2xLarge

下图是基于NAS的Kafka集群架构图。



- *计算。*一个三节点 Kafka 集群，带有一个三节点 zookeeper 集合，在专用服务器上运行。每个代理通过专用 LIF 拥有两个指向NetApp CVO 实例上的单个卷的 NFS 挂载点。
- 监控 Prometheus-Grafana 组合的两个节点。为了生成工作负载，我们使用一个单独的三节点集群，该集群可以为该 Kafka 集群生产和消费。
- *贮存。*单节点NetApp Cloud Volumes ONTAP实例，实例上安装了六个 250GB GP2 AWS-EBS 卷。然后，这些卷通过专用 LIF 作为六个 NFS 卷公开给 Kafka 集群。
- *经纪人配置。*此测试用例中一个可配置元素是 Kafka 代理。为 Kafka 代理选择了以下规格。这 `replica.lag.time.mx.ms`设置为较高的值，因为这决定了特定节点从 ISR 列表中取出的速度。当您在坏节点和健康节点之间切换时，您不希望该代理 ID 被排除在 ISR 列表中。

```
broker.id=1
advertised.listeners=PLAINTEXT://172.30.0.185:9092
log.dirs=/mnt/data-1
zookeeper.connect=172.30.0.13:2181,172.30.0.108:2181,172.30.0.253:2181
num.replica.fetchers=8
message.max.bytes=10485760
replica.fetch.max.bytes=10485760
num.network.threads=8
default.replication.factor=3
replica.lag.time.max.ms=100000000
replica.fetch.max.bytes=1048576
replica.fetch.wait.max.ms=500
num.replica.fetchers=1
replica.high.watermark.checkpoint.interval.ms=5000
fetch.purgatory.purge.interval.requests=1000
producer.purgatory.purge.interval.requests=1000
replica.socket.timeout.ms=30000
replica.socket.receive.buffer.bytes=65536
```

测试方法

1. 创建了两个类似的集群：
   - 基于 EC2 的汇合集群。
   - 基于NetApp NFS 的汇合集群。
2. 创建了一个备用 Kafka 节点，其配置与原始 Kafka 集群中的节点相同。
3. 在每个集群上，都创建了一个示例主题，并且在每个代理上填充了大约 110GB 的数据。
   - 基于 **EC2** 的集群。 Kafka 代理数据目录映射到 `/mnt/data-2`（下图中 cluster1 的 Broker-1[左侧终端]）。
   - 基于**NetApp NFS** 的集群。 Kafka 代理数据目录安装在 NFS 点上 `/mnt/data`（下图中 cluster2 的 Broker-1【右侧终端】）。

4. 在每个集群中，Broker-1 被终止以触发失败的代理恢复过程。

5. 代理终止后，代理 IP 地址被分配作为备用代理的辅助 IP。这是必要的，因为 Kafka 集群中的代理通过以下方式标识：
   - *IP 地址。*通过将发生故障的代理 IP 重新分配给备用代理来进行分配。
   - *经纪人ID*这是在备用代理中配置的 `server.properties`。
6. 分配 IP 后，备用代理上启动了 Kafka 服务。
7. 过了一会儿，拉取服务器日志来检查在集群中的替换节点上构建数据所花费的时间。

观察结果

Kafka 代理的恢复速度几乎提高了 9 倍。与在 Kafka 集群中使用 DAS SSD 相比，使用NetApp NFS 共享存储时恢复故障代理节点所需的时间明显更快。对于 1TB 的主题数据，基于 DAS 的集群的恢复时间为 48 分钟，而基于NetApp-NFS 的 Kafka 集群的恢复时间则不到 5 分钟。

我们观察到基于 EC2 的集群花费 10 分钟在新代理节点上重建 110GB 数据，而基于 NFS 的集群在 3 分钟内完成恢复。我们还在日志中观察到，EC2 分区的消费者偏移量为 0，而在 NFS 集群上，消费者偏移量是从前一个代理获取的。

```
[2022-10-31 09:39:17,747] INFO [LogLoader partition=test-topic-51R3EWs-
0000-55, dir=/mnt/kafka-data/broker2] Reloading from producer snapshot and
rebuilding producer state from offset 583999 (kafka.log.UnifiedLog$)
[2022-10-31 08:55:55,170] INFO [LogLoader partition=test-topic-qbVsEZg-
0000-8, dir=/mnt/data-1] Loading producer state till offset 0 with message
format version 2 (kafka.log.UnifiedLog$)
```

基于**DAS**的集群

1. 备份节点于 08:55:53,730 启动。

```
[2022-10-31 08:55:53,661] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotia
[2022-10-31 08:55:53,727] INFO Registered signal handlers for TERM, INT, HUP (org
[2022-10-31 08:55:53,730] INFO starting (kafka.server.KafkaServer)
[2022-10-31 08:55:53,730] INFO Connecting to zookeeper on 172.30.0.17:2181,172.30
[2022-10-31 08:55:53,755] INFO [ZooKeeperClient Kafka server] Initializing a new
```

2. 数据重建过程于 09:05:24,860 结束。处理 110GB 的数据大约需要 10 分钟。

```
[2022-10-31 09:05:24,860] INFO [ReplicaFetcherManager on broker 1] Removed fetcher for
partitions HashSet(test-topic-qbVsEZg-0000-95, test-topic-qbVsEZg-0000-5,
test-topic-qbVsEZg-0000-41, test-topic-qbVsEZg-0000-23, test-topic-qbVsEZg-0000-11,
test-topic-qbVsEZg-0000-47, test-topic-qbVsEZg-0000-83, test-topic-qbVsEZg-0000-35,
test-topic-qbVsEZg-0000-89, test-topic-qbVsEZg-0000-71, test-topic-qbVsEZg-0000-53,
test-topic-qbVsEZg-0000-29, test-topic-qbVsEZg-0000-59, test-topic-qbVsEZg-0000-77,
test-topic-qbVsEZg-0000-65, test-topic-qbVsEZg-0000-17)
(kafka.server.ReplicaFetcherManager)
```

基于**NFS**的集群

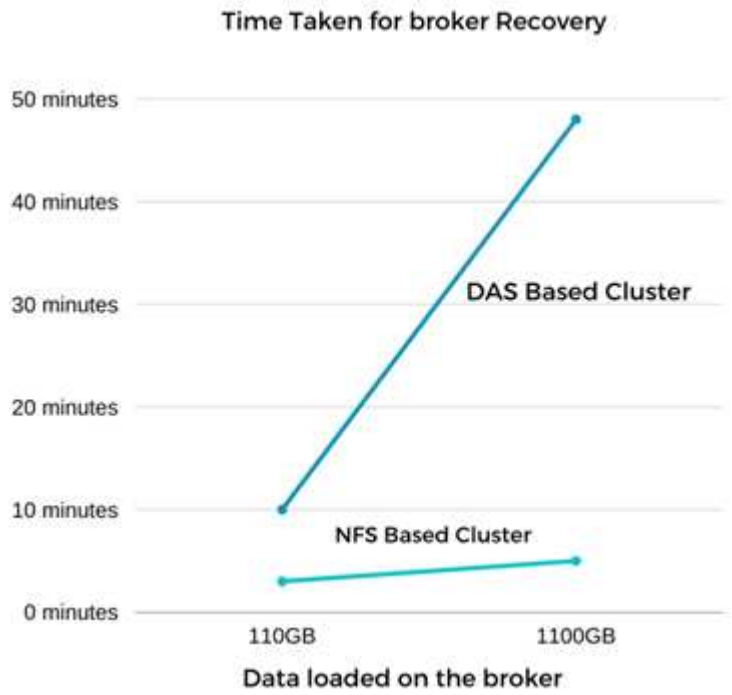
1. 备份节点于 09:39:17,213 启动。下面突出显示了起始日志条目。

```
[2022-10-31 09:39:17,142] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiati
[2022-10-31 09:39:17,211] INFO Registered signal handlers for TERM, INT, HUP (org.
[2022-10-31 09:39:17,213] INFO starting (kafka.server.KafkaServer)
[2022-10-31 09:39:17,214] INFO Connecting to zookeeper on 172.30.0.22:2181,172.30.
[2022-10-31 09:39:17,238] INFO [ZooKeeperClient Kafka server] Initializing a new s
[2022-10-31 09:39:17,244] INFO Client environment:zookeeper.version=3.6.3--6401e4a
[2022-10-31 09:39:17,244] INFO Client environment:host.name=ip-172-30-0-110.ec2.in
```

2. 数据重建过程于 09:42:29,115 结束。处理 110GB 的数据大约需要 3 分钟。

```
[2022-10-31 09:42:29,115] INFO [GroupMetadataManager brokerId=1] Finished loading offsets
and group metadata from __consumer_offsets-20 in 28478 milliseconds for epoch 3, of which
28478 milliseconds was spent in the scheduler.
(kafka.coordinator.group.GroupMetadataManager)
```

对包含约 1TB 数据的代理重复了测试，对于 DAS 大约需要 48 分钟，对于 NFS 大约需要 3 分钟。结果如下图所示。



存储效率

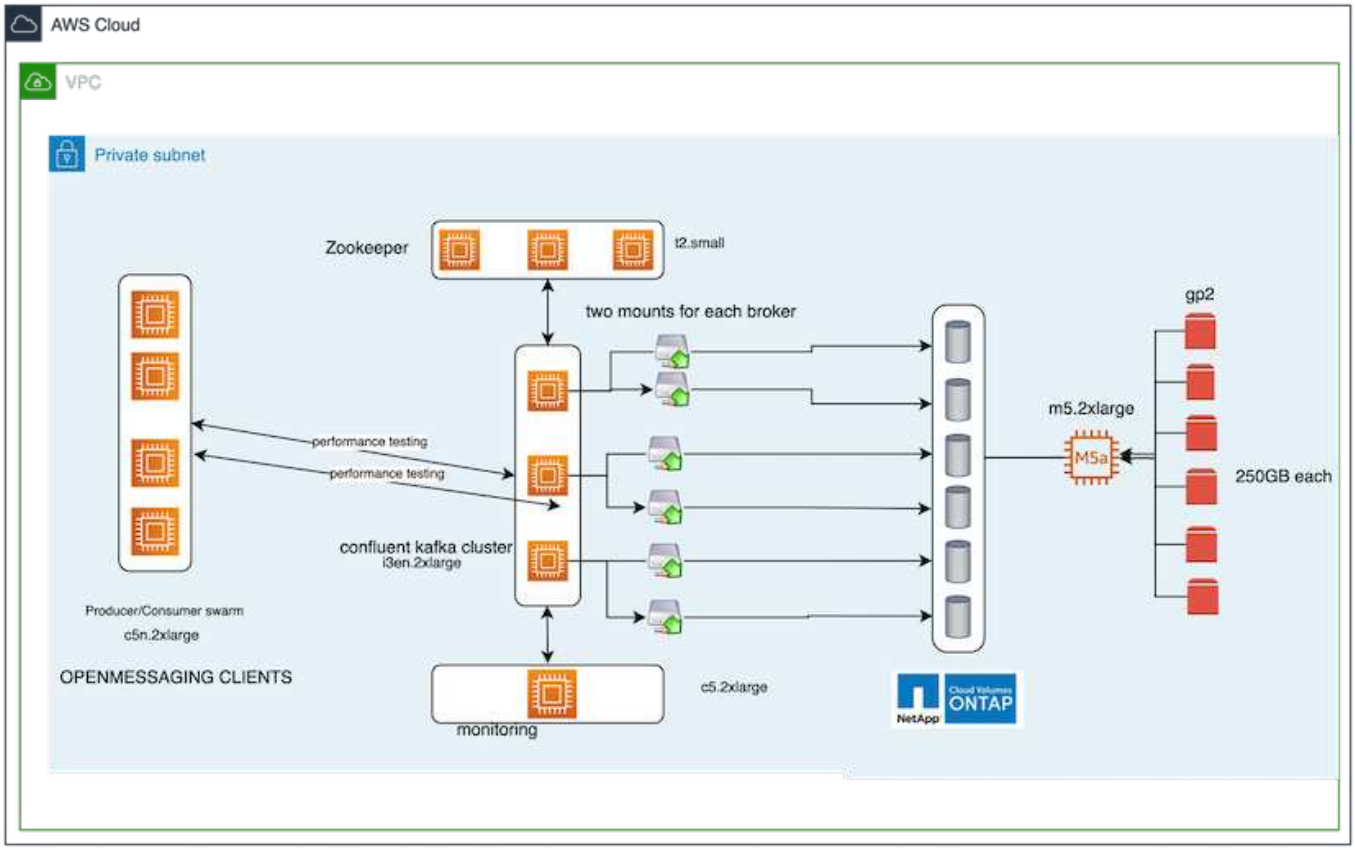
由于 Kafka 集群的存储层是通过NetApp ONTAP配置的，因此我们获得了ONTAP的所有存储效率功能。这是通过在Cloud Volumes ONTAP上配置 NFS 存储的 Kafka 集群上生成大量数据进行的测试。我们可以看到，由于ONTAP功能，空间显著减少。

建筑设置

下表展示了使用NAS的Kafka集群的环境配置。

平台组件	环境配置
卡夫卡 3.2.3	• 3 名动物园管理员 – t2.small • 3 个代理服务器 – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x 生产者/消费者 — c5n.2xlarge *
所有节点上的操作系统	RHEL8.7 或更高版本
NetApp Cloud Volumes ONTAP实例	单节点实例 – M5.2xLarge

下图是基于NAS的Kafka集群架构图。



- *计算。*我们使用了三节点 Kafka 集群，并在专用服务器上运行三节点 zookeeper 集合。每个代理通过专用 LIF 拥有两个指向NetApp CVO 实例上的单个卷的 NFS 挂载点。
- *监控*我们使用两个节点来实现 Prometheus-Grafana 组合。为了生成工作负载，我们使用了一个单独的三节点集群，该集群可以为该 Kafka 集群生产和消费。
- *贮存。*我们使用了单节点NetApp Cloud Volumes ONTAP实例，该实例上安装了六个 250GB GP2 AWS-EBS 卷。然后，这些卷通过专用 LIF 作为六个 NFS 卷公开给 Kafka 集群。
- *配置。*该测试用例中的可配置元素是 Kafka 代理。

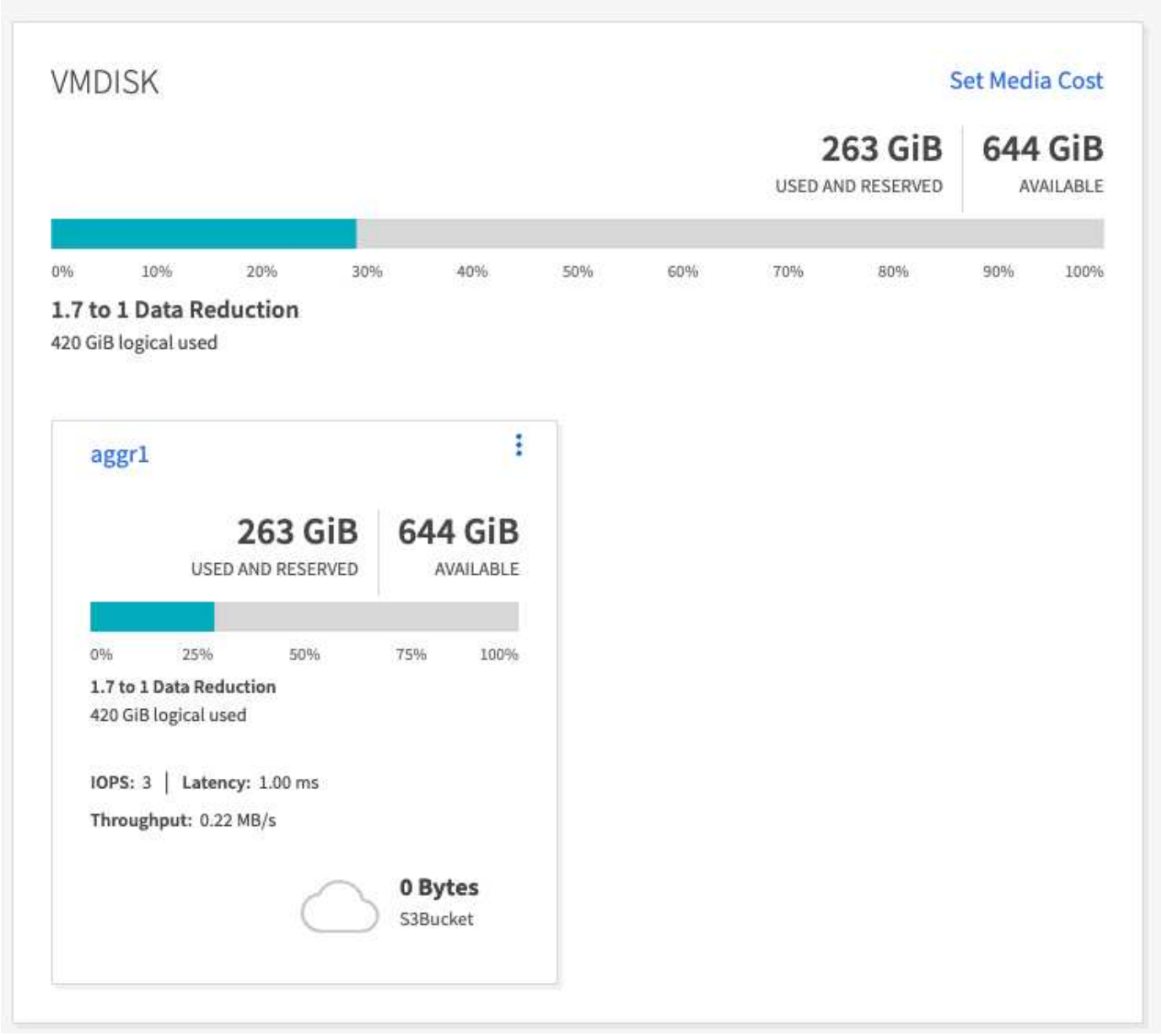
生产者端的压缩被关闭，从而使生产者能够产生高吞吐量。存储效率由计算层处理。

测试方法

1. 已按照上述规格配置了 Kafka 集群。
2. 在集群上，使用 OpenMessaging Benchmarking 工具产生了大约 350GB 的数据。
3. 工作负载完成后，使用ONTAP系统管理器和 CLI 收集存储效率统计数据。

观察结果

对于使用 OMB 工具生成的数据，我们发现空间节省了约 33%，存储效率比为 1.70:1。如下图所示，产生的数据所使用的逻辑空间为420.3GB，用于保存数据的物理空间为281.7GB。



```
shantanuCVOinstancenew::> df -h -S

Warning: The "-S" parameter is deprecated and may be removed in a future release. To show the efficiency ratio use "aggr show-efficiency"
         command.
Filesystem               used      total-saved    %total-saved    deduplicated    %deduplicated    compressed    %compressed  Vserver
/vol/vol0/             7319MB               0B              0%              0B              0%            0B            0%  shantanuCVOinstancenew-01
/vol/kafka_vol/         281GB            138GB             33%           138GB             33%            0B            0%  svm_shantanuCVOinstancenew
/vol/svm_shantanuCVOinstancenew_root/
                        660KB               0B              0%              0B              0%            0B            0%  svm_shantanuCVOinstancenew
3 entries were displayed.
```

```
                    Name of the Aggregate: aggr1
               Node where Aggregate Resides: shantanuCVOinstancenew-01
             Total Storage Efficiency Ratio: 1.70:1
Total Data Reduction Efficiency Ratio Without Snapshots: 1.70:1
Total Data Reduction Efficiency Ratio without snapshots and flexclones: 1.70:1
          Logical Space Used for All Volumes: 420.3GB
        Physical Space Used for All Volumes: 281.7GB
```

AWS 中的性能概述和验证

在NetApp NFS 上安装存储层的 Kafka 集群在 AWS 云中进行了性能基准测试。基准测试示例在以下章节中描述。

AWS 云中的 Kafka 与NetApp Cloud Volumes ONTAP （高可用性对和单节点）

对带有NetApp Cloud Volumes ONTAP （HA 对）的 Kafka 集群在 AWS 云中进行了性能基准测试。以下章节描述了此基准测试。

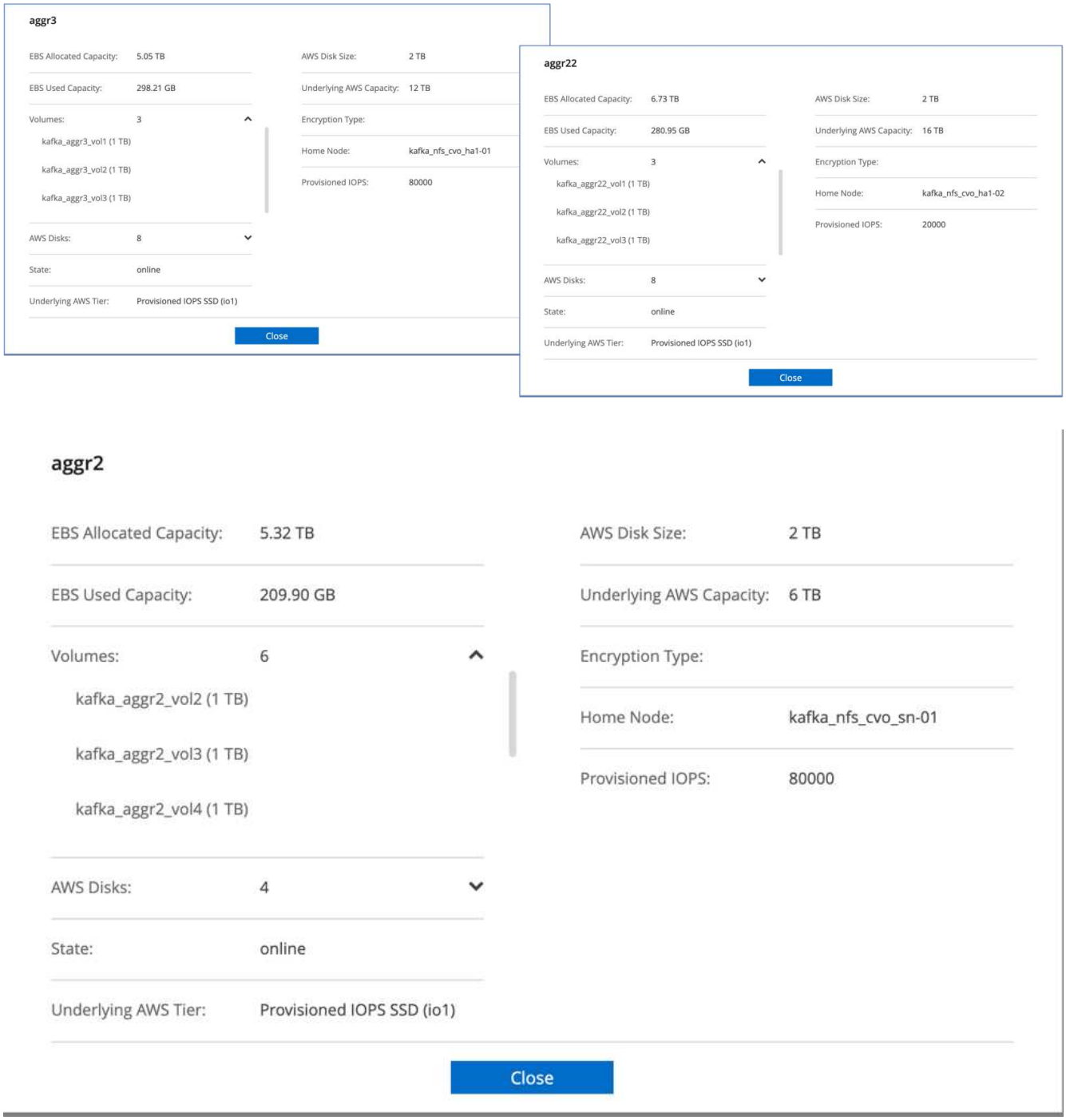
建筑设置

下表展示了使用NAS的Kafka集群的环境配置。

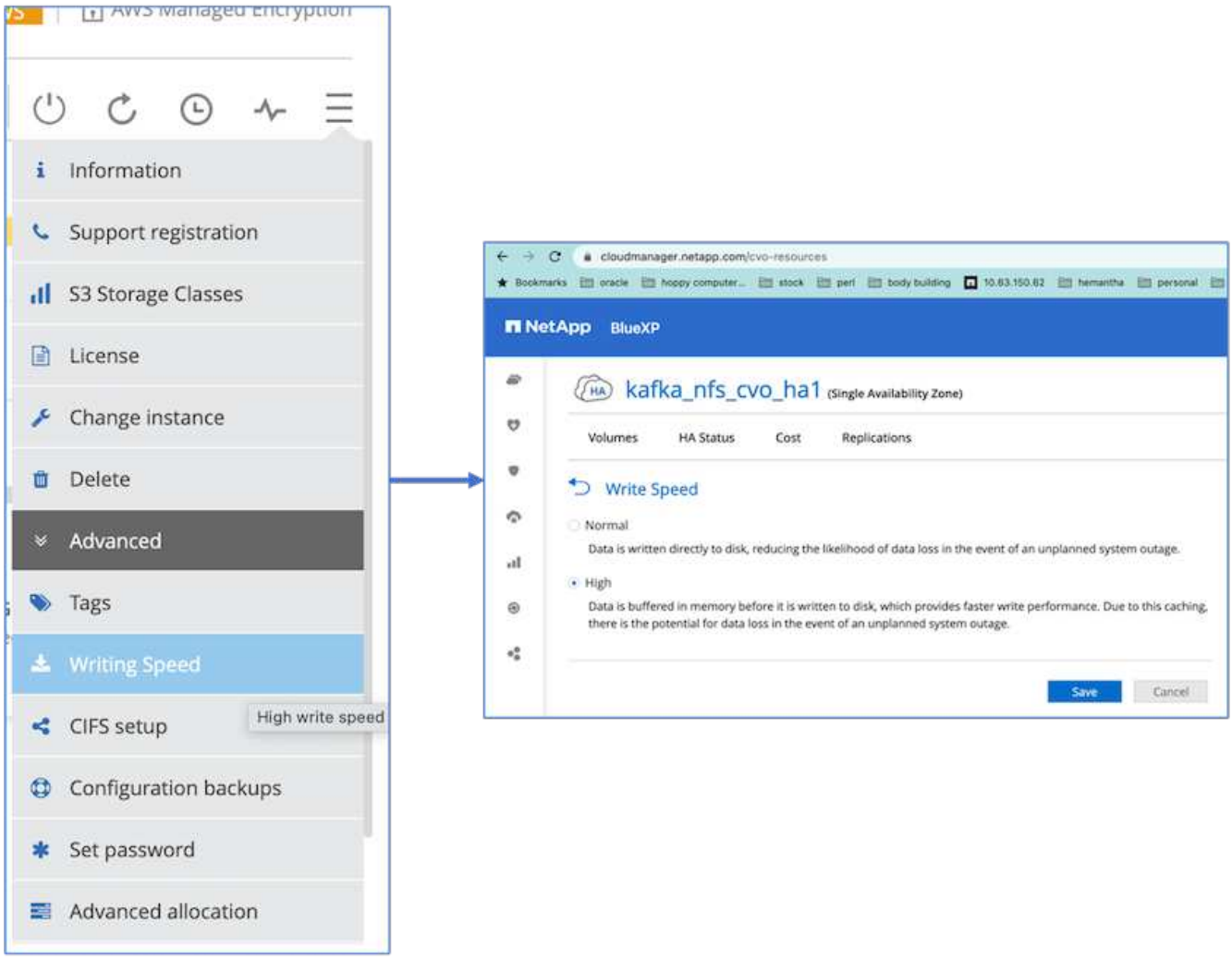
平台组件	环境配置
卡夫卡 3.2.3	• 3 名动物园管理员 – t2.small • 3 个代理服务器 – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x 生产者/消费者 — c5n.2xlarge *
所有节点上的操作系统	RHEL8.6
NetApp Cloud Volumes ONTAP实例	HA 对实例 – m5dn.12xLarge x 2node 单节点实例 - m5dn.12xLarge x 1 节点

NetApp集群卷**ONTAP**设置

1. 对于Cloud Volumes ONTAP HA 对，我们创建了两个聚合，每个存储控制器上的每个聚合上有三个卷。对于单个Cloud Volumes ONTAP节点，我们在一个聚合中创建六个卷。

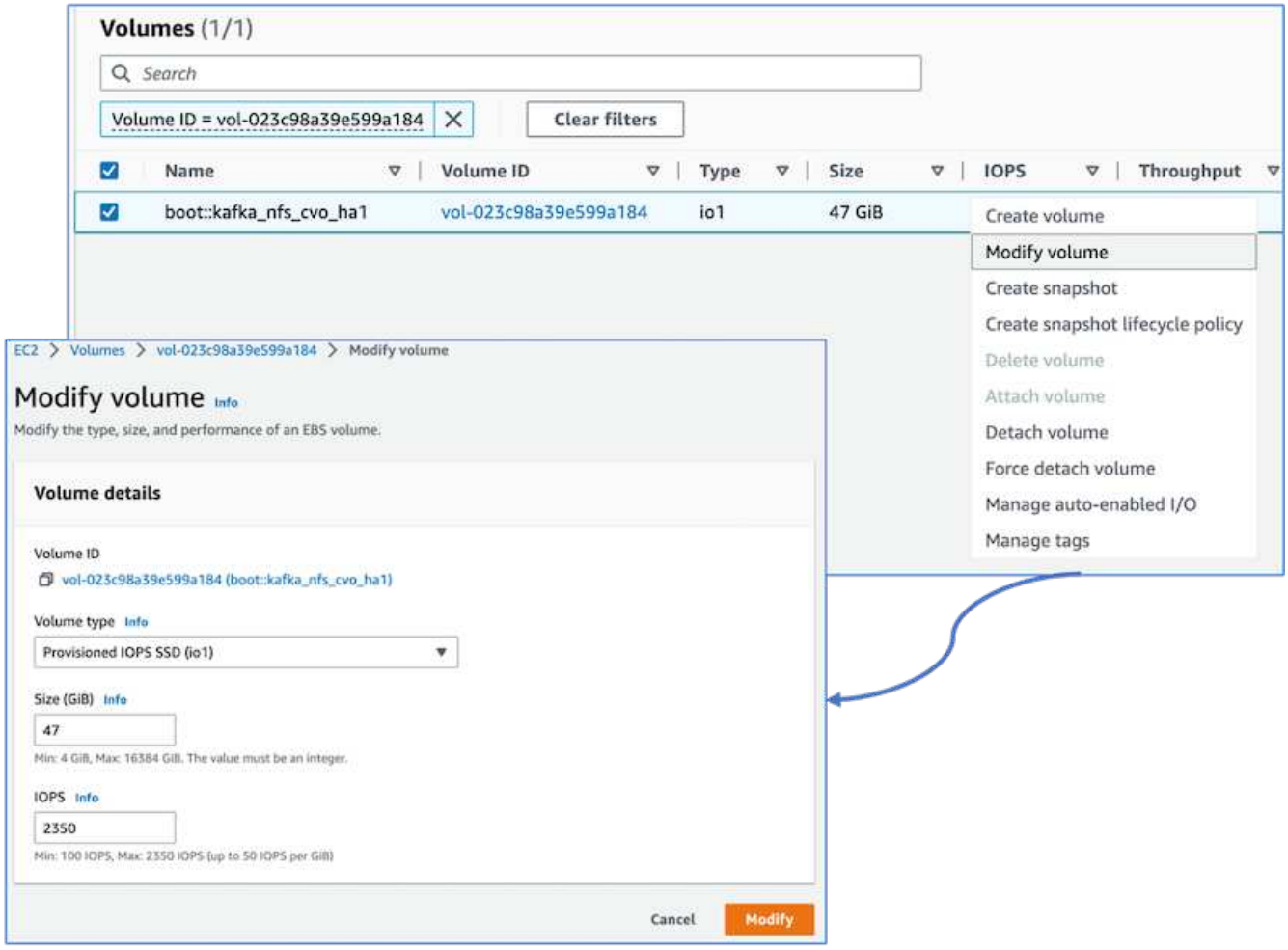


2. 为了实现更好的网络性能，我们为 HA 对和单个节点都启用了高速网络。



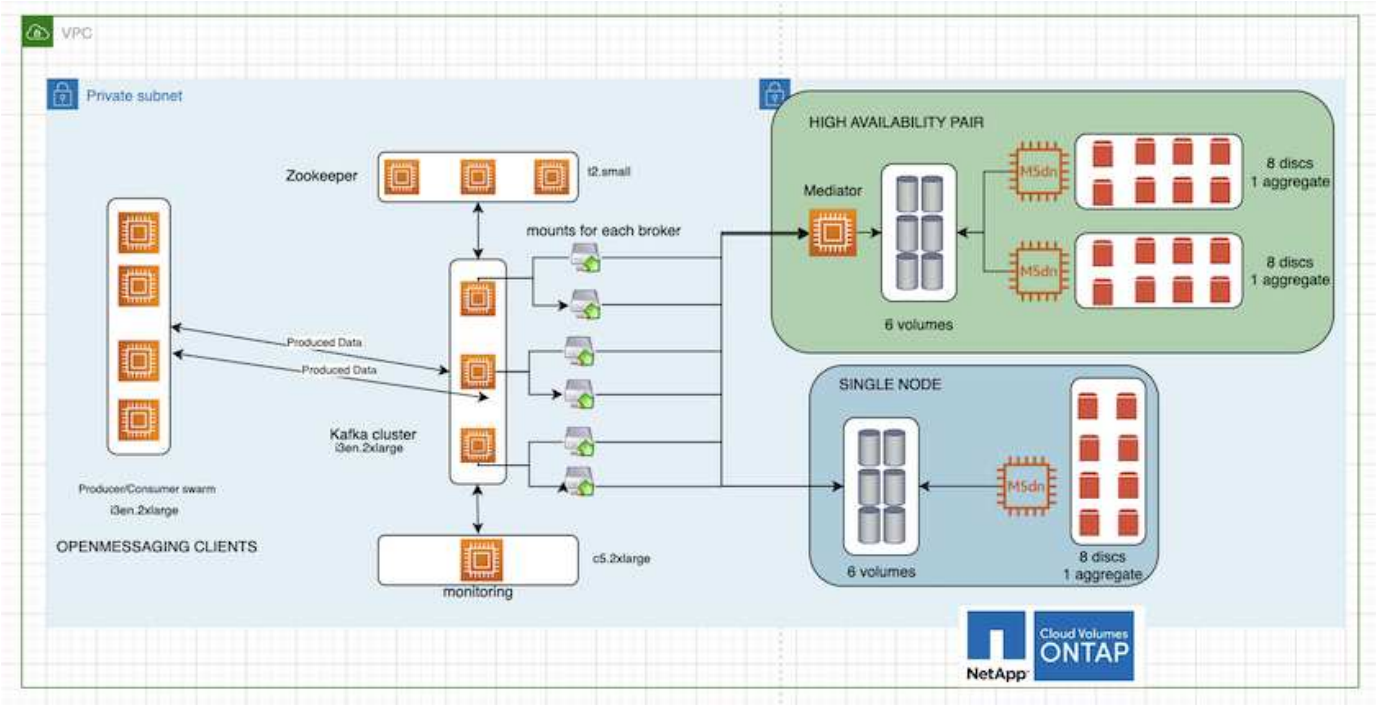
3. 我们注意到ONTAP NVRAM具有更多的 IOPS，因此我们将Cloud Volumes ONTAP根卷的 IOPS 更改为 2350。 Cloud Volumes ONTAP中的根卷磁盘大小为 47GB。以下ONTAP命令适用于 HA 对，相同步骤也适用于单个节点。

```
statistics start -object vnvram -instance vnvram -counter
backing_store_iops -sample-id sample_555
kafka_nfs_cvo_ha1::*> statistics show -sample-id sample_555
Object: vnvram
Instance: vnvram
Start-time: 1/18/2023 18:03:11
End-time: 1/18/2023 18:03:13
Elapsed-time: 2s
Scope: kafka_nfs_cvo_ha1-01
    Counter                                                     Value
    -------------------------------- --------------------------------
    backing_store_iops                                           1479
Object: vnvram
Instance: vnvram
Start-time: 1/18/2023 18:03:11
End-time: 1/18/2023 18:03:13
Elapsed-time: 2s
Scope: kafka_nfs_cvo_ha1-02
    Counter                                                     Value
    -------------------------------- --------------------------------
    backing_store_iops                                           1210
2 entries were displayed.
kafka_nfs_cvo_ha1::*>
```



下图是基于NAS的Kafka集群架构图。

- *计算。*我们使用了三节点 Kafka 集群，并在专用服务器上运行三节点 zookeeper 集合。每个代理通过专用 LIF 拥有两个指向Cloud Volumes ONTAP实例上的单个卷的 NFS 挂载点。
- *监控*我们使用两个节点来实现 Prometheus-Grafana 组合。为了生成工作负载，我们使用了一个单独的三节点集群，该集群可以为该 Kafka 集群生产和消费。
- *贮存。*我们使用了 HA 对 Cloud Volumes ONTAP实例，并在该实例上安装了一个 6TB GP3 AWS-EBS 卷。然后通过 NFS 挂载将该卷导出到 Kafka 代理。



OpenMessage 基准测试配置

1. 为了获得更好的 NFS 性能，我们需要在 NFS 服务器和 NFS 客户端之间建立更多的网络连接，可以使用 nconnect 创建。通过运行以下命令，使用 nconnect 选项将 NFS 卷挂载到代理节点上：

```
[root@ip-172-30-0-121 ~]# cat /etc/fstab
UUID=eaa1f38e-de0f-4ed5-a5b5-2fa9db43bb38/xfsdefaults00
/dev/nvme1n1 /mnt/data-1 xfs defaults,noatime,nodiscard 0 0
/dev/nvme2n1 /mnt/data-2 xfs defaults,noatime,nodiscard 0 0
172.30.0.233:/kafka_aggr3_vol1 /kafka_aggr3_vol1 nfs
defaults,nconnect=16 0 0
172.30.0.233:/kafka_aggr3_vol2 /kafka_aggr3_vol2 nfs
defaults,nconnect=16 0 0
172.30.0.233:/kafka_aggr3_vol3 /kafka_aggr3_vol3 nfs
defaults,nconnect=16 0 0
172.30.0.242:/kafka_aggr22_vol1 /kafka_aggr22_vol1 nfs
defaults,nconnect=16 0 0
172.30.0.242:/kafka_aggr22_vol2 /kafka_aggr22_vol2 nfs
defaults,nconnect=16 0 0
172.30.0.242:/kafka_aggr22_vol3 /kafka_aggr22_vol3 nfs
defaults,nconnect=16 0 0
[root@ip-172-30-0-121 ~]# mount -a
[root@ip-172-30-0-121 ~]# df -h
Filesystem                         Size  Used Avail Use% Mounted on
devtmpfs                            31G     0   31G   0% /dev
tmpfs                               31G  249M   31G   1% /run
tmpfs                               31G     0   31G   0% /sys/fs/cgroup
/dev/nvme0n1p2                      10G  2.8G  7.2G  28% /
/dev/nvme1n1                       2.3T  248G  2.1T  11% /mnt/data-1
/dev/nvme2n1                       2.3T  245G  2.1T  11% /mnt/data-2
172.30.0.233:/kafka_aggr3_vol1     1.0T   12G 1013G   2% /kafka_aggr3_vol1
172.30.0.233:/kafka_aggr3_vol2     1.0T  5.5G 1019G   1% /kafka_aggr3_vol2
172.30.0.233:/kafka_aggr3_vol3     1.0T  8.9G 1016G   1% /kafka_aggr3_vol3
172.30.0.242:/kafka_aggr22_vol1    1.0T  7.3G 1017G   1%
/kafka_aggr22_vol1
172.30.0.242:/kafka_aggr22_vol2    1.0T  6.9G 1018G   1%
/kafka_aggr22_vol2
172.30.0.242:/kafka_aggr22_vol3    1.0T  5.9G 1019G   1%
/kafka_aggr22_vol3
tmpfs                              6.2G     0  6.2G   0% /run/user/1000
[root@ip-172-30-0-121 ~]#
```

2. 检查Cloud Volumes ONTAP中的网络连接。以下ONTAP命令从单个Cloud Volumes ONTAP节点使用。相同的步骤适用于Cloud Volumes ONTAP HA 对。

```
Last login time: 1/20/2023 00:16:29
kafka_nfs_cvo_sn::> network connections active show -service nfs*
-fields remote-host
node               cid        vserver              remote-host
------------------ ---------- -------------------- ------------
```

```
kafka_nfs_cvo_sn-01 2315762628 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762629 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762630 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762631 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762632 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762633 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762634 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762635 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762636 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762637 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762639 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762640 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762641 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762642 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762643 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762644 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762645 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762646 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762647 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762648 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762649 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762650 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762651 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762652 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762653 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762656 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762657 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762658 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762659 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762660 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762661 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762662 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762663 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762664 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762665 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762666 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762667 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762668 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762669 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762670 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762671 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762672 svm_kafka_nfs_cvo_sn 172.30.0.72
kafka_nfs_cvo_sn-01 2315762673 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762674 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762676 svm_kafka_nfs_cvo_sn 172.30.0.121
kafka_nfs_cvo_sn-01 2315762677 svm_kafka_nfs_cvo_sn 172.30.0.223
```

```
kafka_nfs_cvo_sn-01 2315762678 svm_kafka_nfs_cvo_sn 172.30.0.223
kafka_nfs_cvo_sn-01 2315762679 svm_kafka_nfs_cvo_sn 172.30.0.223
48 entries were displayed.

kafka_nfs_cvo_sn::>
```

3. 我们使用以下 Kafka `server.properties`在Cloud Volumes ONTAP HA 对的所有 Kafka 代理中。这 `log.dirs`属性对于每个经纪人来说是不同的，其余属性对于经纪人来说是共同的。对于 broker1 来说， `log.dirs`值如下：

```
[root@ip-172-30-0-121 ~]# cat /opt/kafka/config/server.properties
broker.id=0
advertised.listeners=PLAINTEXT://172.30.0.121:9092
#log.dirs=/mnt/data-1/d1,/mnt/data-1/d2,/mnt/data-1/d3,/mnt/data-
2/d1,/mnt/data-2/d2,/mnt/data-2/d3
log.dirs=/kafka_aggr3_vol1/broker1,/kafka_aggr3_vol2/broker1,/kafka_aggr
3_vol3/broker1,/kafka_aggr22_vol1/broker1,/kafka_aggr22_vol2/broker1,/ka
fka_aggr22_vol3/broker1
zookeeper.connect=172.30.0.12:2181,172.30.0.30:2181,172.30.0.178:2181
num.network.threads=64
num.io.threads=64
socket.send.buffer.bytes=102400
socket.receive.buffer.bytes=102400
socket.request.max.bytes=104857600
num.partitions=1
num.recovery.threads.per.data.dir=1
offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1
replica.fetch.max.bytes=524288000
background.threads=20
num.replica.alter.log.dirs.threads=40
num.replica.fetchers=20
[root@ip-172-30-0-121 ~]#
```

- 对于 broker2， `log.dirs`属性值如下：

```
log.dirs=/kafka_aggr3_vol1/broker2,/kafka_aggr3_vol2/broker2,/kafka_a
ggr3_vol3/broker2,/kafka_aggr22_vol1/broker2,/kafka_aggr22_vol2/broke
r2,/kafka_aggr22_vol3/broker2
```

- 对于 broker3， `log.dirs`属性值如下：

```
log.dirs=/kafka_aggr3_vol1/broker3,/kafka_aggr3_vol2/broker3,/kafka_a
ggr3_vol3/broker3,/kafka_aggr22_vol1/broker3,/kafka_aggr22_vol2/broke
r3,/kafka_aggr22_vol3/broker3
```

4. 对于单个Cloud Volumes ONTAP节点，Kafka `servers.properties`与Cloud Volumes ONTAP HA 对相同，但`log.dirs`财产。
   - 对于 broker1 来说， `log.dirs`值如下：

```
log.dirs=/kafka_aggr2_vol1/broker1,/kafka_aggr2_vol2/broker1,/kafka_a
ggr2_vol3/broker1,/kafka_aggr2_vol4/broker1,/kafka_aggr2_vol5/broker1
,/kafka_aggr2_vol6/broker1
```

   - 对于 broker2， `log.dirs`值如下：

```
log.dirs=/kafka_aggr2_vol1/broker2,/kafka_aggr2_vol2/broker2,/kafka_a
ggr2_vol3/broker2,/kafka_aggr2_vol4/broker2,/kafka_aggr2_vol5/broker2
,/kafka_aggr2_vol6/broker2
```

   - 对于 broker3， `log.dirs`属性值如下：

```
log.dirs=/kafka_aggr2_vol1/broker3,/kafka_aggr2_vol2/broker3,/kafka_a
ggr2_vol3/broker3,/kafka_aggr2_vol4/broker3,/kafka_aggr2_vol5/broker3
,/kafka_aggr2_vol6/broker3
```

5. OMB 中的工作负载配置有以下属性： `(/opt/benchmark/workloads/1-topic-100-partitions-1kb.yaml)`。

```
topics: 4
partitionsPerTopic: 100
messageSize: 32768
useRandomizedPayloads: true
randomBytesRatio: 0.5
randomizedPayloadPoolSize: 100
subscriptionsPerTopic: 1
consumerPerSubscription: 80
producersPerTopic: 40
producerRate: 1000000
consumerBacklogSizeGB: 0
testDurationMinutes: 5
```

这 `messageSize`根据每个用例可能会有所不同。在我们的性能测试中，我们使用了 3K。

我们使用了来自 OMB 的两个不同的驱动程序（Sync 或 Throughput）来在 Kafka 集群上生成工作负载。

- 用于同步驱动程序属性的 yaml 文件如下 (/opt/benchmark/driver- kafka/kafka-sync.yaml)：

```
name: Kafka
driverClass:
io.openmessaging.benchmark.driver.kafka.KafkaBenchmarkDriver
# Kafka client-specific configuration
replicationFactor: 3
topicConfig: |
  min.insync.replicas=2
  flush.messages=1
  flush.ms=0
commonConfig: |

bootstrap.servers=172.30.0.121:9092,172.30.0.72:9092,172.30.0.223:909
2
producerConfig: |
  acks=all
  linger.ms=1
  batch.size=1048576
consumerConfig: |
  auto.offset.reset=earliest
  enable.auto.commit=false
  max.partition.fetch.bytes=10485760
```

- 用于吞吐量驱动程序属性的 yaml 文件如下 (/opt/benchmark/driver- kafka/kafka-throughput.yaml)：

```
name: Kafka
driverClass:
io.openmessaging.benchmark.driver.kafka.KafkaBenchmarkDriver
# Kafka client-specific configuration
replicationFactor: 3
topicConfig: |
  min.insync.replicas=2
commonConfig: |

bootstrap.servers=172.30.0.121:9092,172.30.0.72:9092,172.30.0.223:909
2
  default.api.timeout.ms=1200000
  request.timeout.ms=1200000
producerConfig: |
  acks=all
  linger.ms=1
  batch.size=1048576
consumerConfig: |
  auto.offset.reset=earliest
  enable.auto.commit=false
  max.partition.fetch.bytes=10485760
```

测试方法

1. 根据上面描述的规范，使用 Terraform 和 Ansible 配置了 Kafka 集群。 Terraform 用于使用 AWS 实例为 Kafka 集群构建基础设施，Ansible 在其上构建 Kafka 集群。
2. 使用上面描述的工作负载配置和同步驱动程序触发了 OMB 工作负载。

```
Sudo bin/benchmark -drivers driver-kafka/kafka- sync.yaml workloads/1-
topic-100-partitions-1kb.yaml
```

3. 使用具有相同工作负载配置的吞吐量驱动程序触发了另一个工作负载。

```
sudo bin/benchmark -drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

观察结果

使用两种不同类型的驱动程序来生成工作负载，以对在 NFS 上运行的 Kafka 实例的性能进行基准测试。驱动程序之间的区别在于日志刷新属性。

对于Cloud Volumes ONTAP HA 对：

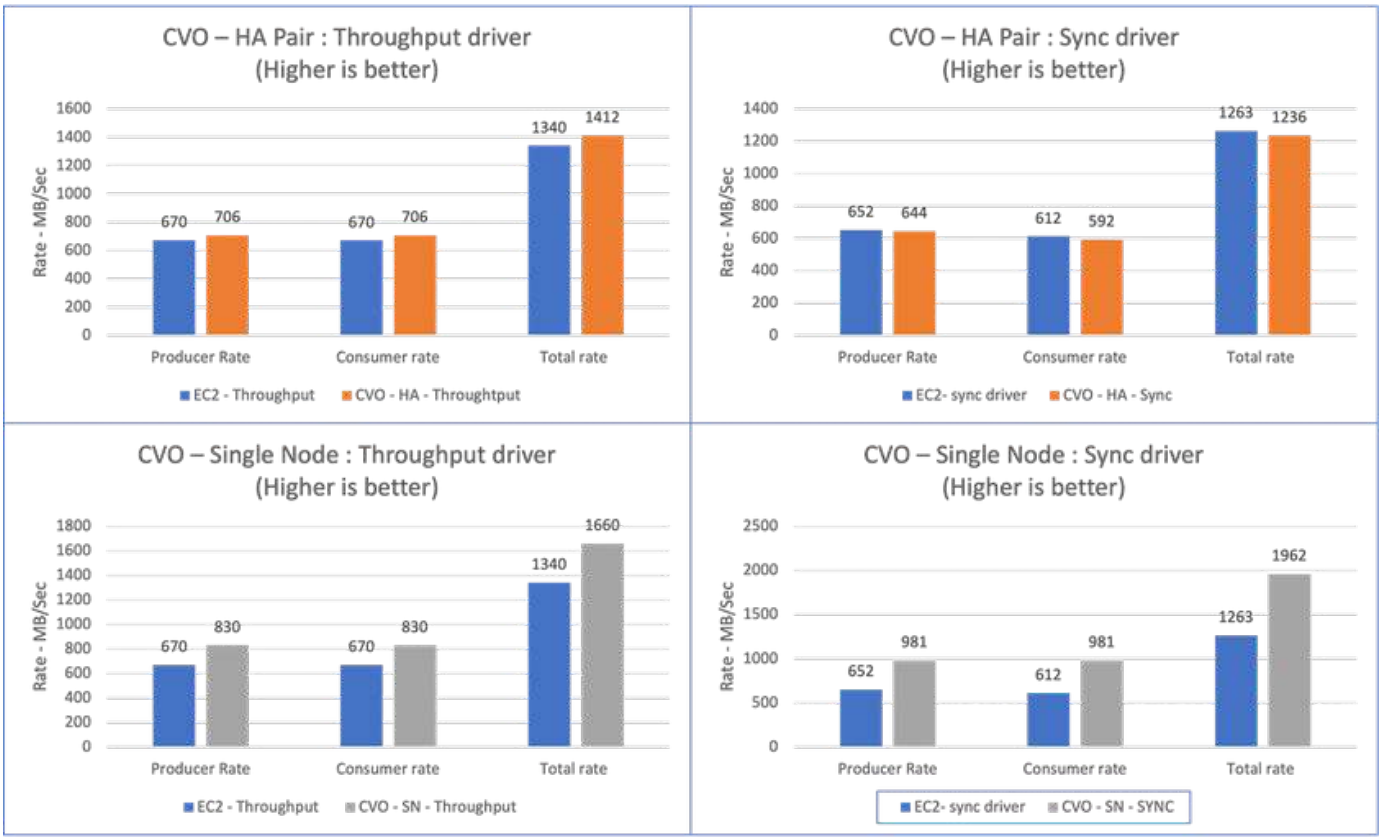
- 同步驱动程序持续产生的总吞吐量：~1236 MBps。
- 吞吐量驱动程序产生的总吞吐量：峰值~1412 MBps。

对于单个Cloud Volumes ONTAP节点：

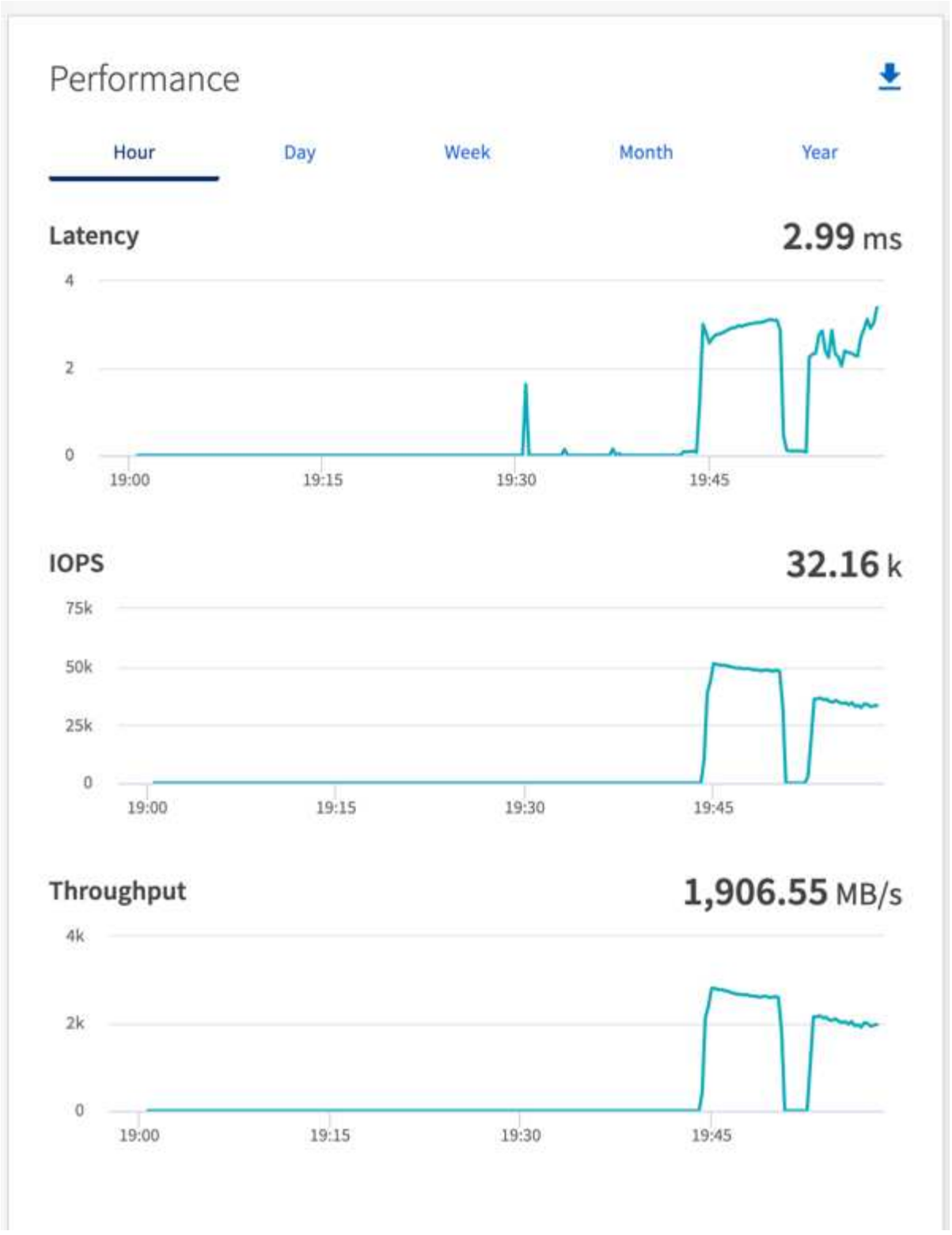
- 同步驱动程序持续产生的总吞吐量：~ 1962MBps。
- 吞吐量驱动程序产生的总吞吐量：峰值~1660MBps

由于日志会立即刷新到磁盘，因此同步驱动程序可以生成一致的吞吐量，而由于日志会批量提交到磁盘，因此吞吐量驱动程序会产生突发性的吞吐量。

这些吞吐量数字是针对给定的 AWS 配置生成的。对于更高的性能要求，可以扩大实例类型并进一步调整以获得更好的吞吐量数字。总吞吐量或总速率是生产者和消费者速率的组合。



在执行吞吐量或同步驱动程序基准测试时，请务必检查存储吞吐量。



AWS FSx ONTAP中的性能概述和验证

在NetApp NFS 上安装存储层的 Kafka 集群在 AWS FSx ONTAP中进行了性能基准测试。基准测试示例在以下章节中描述。

AWS FSx ONTAP中的 Apache Kafka

网络文件系统 (NFS) 是一种广泛用于存储大量数据的网络文件系统。在大多数组织中，数据越来越多地由 Apache Kafka 等流应用程序生成。这些工作负载需要可扩展性、低延迟以及具有现代存储功能的强大数据提取架构。为了实现实时分析并提供可操作的见解，需要精心设计且高性能的基础设施。

Kafka 在设计上与 POSIX 兼容的文件系统协同工作，并依赖该文件系统来处理文件操作，但是在 NFSv3 文件系统上存储数据时，Kafka 代理 NFS 客户端可以以不同于 XFS 或 Ext4 等本地文件系统的方式解释文件操作。一个常见的例子是 NFS Silly 重命名，它导致 Kafka 代理在扩展集群和重新分配分区时失败。为了应对这一挑战，NetApp更新了开源 Linux NFS 客户端，这些更改现在已在 RHEL8.7、RHEL9.1 中普遍可用，并且从当前 FSx ONTAP版本ONTAP 9.12.1 开始受支持。

Amazon FSx ONTAP在云中提供完全托管、可扩展且高性能的 NFS 文件系统。 FSx ONTAP上的 Kafka 数据可以扩展以处理大量数据并确保容错能力。 NFS 为关键和敏感数据集提供集中存储管理和数据保护。

这些增强功能使 AWS 客户能够在 AWS 计算服务上运行 Kafka 工作负载时利用 FSx ONTAP 。这些好处是：* 降低 CPU 利用率以减少 I/O 等待时间* 更快的 Kafka 代理恢复时间。 * 可靠性和效率。 * 可扩展性和性能。 * 多可用区域可用性。 * 数据保护。

AWS FSx ONTAP中的性能概述和验证

在NetApp NFS 上安装存储层的 Kafka 集群在 AWS 云中进行了性能基准测试。基准测试示例在以下章节中描述。

AWS FSx ONTAP中的 Kafka

使用 AWS FSx ONTAP 的Kafka 集群在 AWS 云中进行了性能基准测试。以下章节描述了此基准测试。

建筑设置

下表显示了使用 AWS FSx ONTAP 的Kafka 集群的环境配置。

平台组件	环境配置
卡夫卡 3.2.3	• 3 名动物园管理员 – t2.small • 3 个代理服务器 – i3en.2xlarge • 1 x Grafana – c5n.2xlarge • 4 x 生产者/消费者 — c5n.2xlarge *
所有节点上的操作系统	RHEL8.6
AWS FSx ONTAP	多可用区，吞吐量 4GB/秒，IOPS 160000

NetApp FSx ONTAP设置

1. 在我们的初步测试中，我们创建了一个容量为 2TB、IOP 为 40000 的 FSx ONTAP文件系统，吞吐量为 2GB/秒。

```
[root@ip-172-31-33-69 ~]# aws fsx create-file-system --region us-east-2
--storage-capacity 2048 --subnet-ids <desired subnet 1> subnet-<desired
subnet 2> --file-system-type ONTAP --ontap-configuration
DeploymentType=MULTI_AZ_HA_1,ThroughputCapacity=2048,PreferredSubnetId=<
desired primary subnet>,FsxAdminPassword=<new
password>,DiskIopsConfiguration="{Mode=USER_PROVISIONED,Iops=40000"}
```

在我们的示例中，我们通过 AWS CLI 部署 FSx ONTAP 。您将需要根据需要在您的环境中进一步自定义命令。 FSx ONTAP还可以通过 AWS 控制台进行部署和管理，从而通过更少的命令行输入获得更轻松、更简化的部署体验。

文档在 FSx ONTAP中，我们测试区域（US-East-1）中 2GB/秒吞吐量文件系统可实现的最大 IOPS 为 80,000 iops。 FSx ONTAP文件系统的最大总 iops 为 160,000 iops，需要 4GB/秒的吞吐量部署才能实现，我们将在本文档的后面进行演示。

有关 FSx ONTAP性能规格的更多信息，请随时访问此处的 AWS FSx ONTAP文档：https://docs.aws.amazon.com/fsx/latest/ONTAPGuide/performance.html 。

FSx“create-file-system”的详细命令行语法可以在这里找到： https://docs.aws.amazon.com/cli/latest/reference/fsx/create-file-system.html

例如，您可以指定特定的 KMS 密钥，而不是在未指定 KMS 密钥时使用的默认 AWS FSx 主密钥。

2. 在创建 FSx ONTAP文件系统时，请按照如下方式描述您的文件系统，等待 JSON 返回中的“LifeCycle”状态变为“AVAILABLE”：

```
[root@ip-172-31-33-69 ~]# aws fsx describe-file-systems  --region us-
east-1 --file-system-ids fs-02ff04bab5ce01c7c
```

3. 通过使用 fsxadmin 用户登录 FSx ONTAP SSH 来验证凭据：Fsxadmin 是 FSx ONTAP文件系统创建时的默认管理员帐户。 fsxadmin 的密码是首次在 AWS 控制台或使用 AWS CLI 创建文件系统时配置的密码，如我们在步骤 1 中完成的那样。

```
[root@ip-172-31-33-69 ~]# ssh fsxadmin@198.19.250.244
The authenticity of host '198.19.250.244 (198.19.250.244)' can't be
established.
ED25519 key fingerprint is
SHA256:mgCyRXJfWRc2d/jOjFbMBsUcYOWjxoIky0ltHvVDL/Y.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '198.19.250.244' (ED25519) to the list of
known hosts.
(fsxadmin@198.19.250.244) Password:

This is your first recorded login.
```

4. 验证您的凭据后，在 FSx ONTAP文件系统上创建存储虚拟机

```
[root@ip-172-31-33-69 ~]# aws fsx --region us-east-1 create-storage-
virtual-machine --name svmkafkatest --file-system-id fs-
02ff04bab5ce01c7c
```

存储虚拟机 (SVM) 是一个独立的文件服务器，具有自己的管理凭据和端点，用于管理和访问 FSx ONTAP卷中的数据，并提供 FSx ONTAP多租户。

5. 配置好主存储虚拟机后，通过 SSH 进入新创建的 FSx ONTAP文件系统，并使用以下示例命令在存储虚拟机中创建卷，同样，我们为此验证创建 6 个卷。根据我们的验证，保留默认成分（8）或更少的成分将为 kafka 提供更好的性能。

```
FsxId02ff04bab5ce01c7c::*> volume create -volume kafkafsxN1 -state
online -policy default -unix-permissions ---rwxr-xr-x -junction-active
true -type RW -snapshot-policy none  -junction-path /kafkafsxN1 -aggr
-list aggr1
```

6. 我们的测试需要额外的容量。将卷的大小扩展至 2TB 并安装在连接路径上。

```
FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN1 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN1" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN2 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN2" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN3 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN3" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN4 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN4" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN5 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN5" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume size -volume kafkafsxN6 -new-size +2TB
vol size: Volume "svmkafkatest:kafkafsxN6" size set to 2.10t.

FsxId02ff04bab5ce01c7c::*> volume show -vserver svmkafkatest -volume *
Vserver   Volume       Aggregate    State      Type       Size
Available Used%
--------- ------------ ------------ ---------- ---- ----------
---------- -----
svmkafkatest
          kafkafsxN1   -            online     RW       2.10TB
```

```
1.99TB    0%
svmkafkatest
          kafkafsxN2   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN3   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN4   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN5   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          kafkafsxN6   -            online     RW       2.10TB
1.99TB    0%
svmkafkatest
          svmkafkatest_root
                       aggr1        online     RW          1GB
968.1MB    0%
7 entries were displayed.

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN1 -junction
-path /kafkafsxN1

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN2 -junction
-path /kafkafsxN2

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN3 -junction
-path /kafkafsxN3

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN4 -junction
-path /kafkafsxN4

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN5 -junction
-path /kafkafsxN5

FsxId02ff04bab5ce01c7c::*> volume mount -volume kafkafsxN6 -junction
-path /kafkafsxN6
```

在 FSx ONTAP中，卷可以进行精简配置。在我们的示例中，扩展卷的总容量超过了文件系统的总容量，因此我们需要扩展文件系统的总容量以解锁额外的预配置卷容量，我们将在下一步中演示这一点。

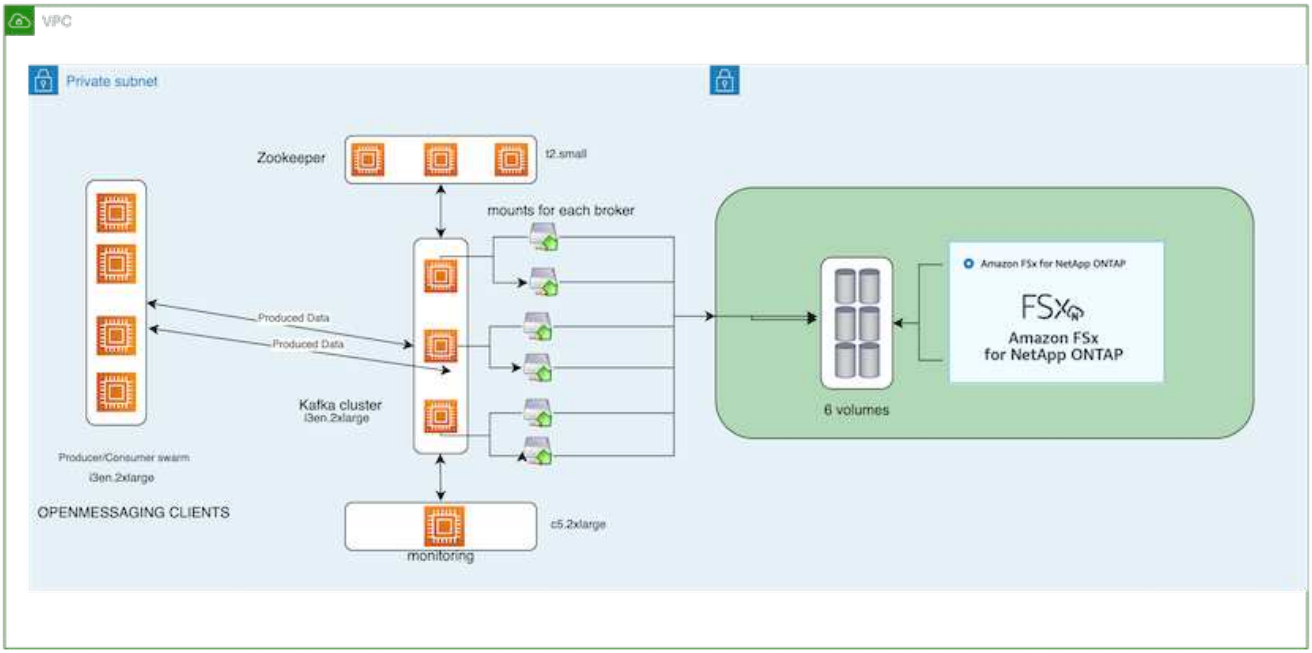
7. 接下来，为了提高性能和容量，我们将 FSx ONTAP吞吐容量从 2GB/秒扩展到 4GB/秒，IOPS 扩展到 160000，容量扩展到 5 TB

```
[root@ip-172-31-33-69 ~]# aws fsx update-file-system --region us-east-1
--storage-capacity 5120 --ontap-configuration
'ThroughputCapacity=4096,DiskIopsConfiguration={Mode=USER_PROVISIONED,Io
ps=160000}' --file-system-id fs-02ff04bab5ce01c7c
```

FSx“update-file-system”的详细命令行语法可以在这里找到
：https://docs.aws.amazon.com/cli/latest/reference/fsx/update-file-system.html[]

8. FSx ONTAP卷通过 nconnect 和 Kafka 代理中的默认选项进行挂载

下图展示了我们基于 FSx ONTAP 的Kafka 集群的最终架构：



- 计算。我们使用了三节点 Kafka 集群，并在专用服务器上运行三节点 zookeeper 集合。每个代理有六个 NFS 挂载点，分别指向 FSx ONTAP实例上的六个卷。
- 监控。我们使用两个节点来实现 Prometheus-Grafana 组合。为了生成工作负载，我们使用了一个单独的三节点集群，该集群可以为该 Kafka 集群生产和消费。
- 贮存。我们使用了安装了六个 2TB 卷的 FSx ONTAP 。然后将该卷导出到具有 NFS 挂载的 Kafka 代理。FSx ONTAP卷在 Kafka 代理中安装了 16 个 nconnect 会话和默认选项。

OpenMessage 基准测试配置。

我们使用了与NetApp Cloud Volumes ONTAP相同的配置，其详细信息请参见此处 - xref:./data-analytics/kafka-nfs-performance-overview-and-validation-in-aws.html#architectural-setup

测试方法

1. 根据上面描述的规范，使用 terraform 和 ansible 配置了 Kafka 集群。 Terraform 用于使用 AWS 实例为 Kafka 集群构建基础设施，并且 ansible 在其上构建 Kafka 集群。
2. 使用上面描述的工作负载配置和同步驱动程序触发了 OMB 工作负载。

```
sudo bin/benchmark -drivers driver-kafka/kafka-sync.yaml workloads/1-
topic-100-partitions-1kb.yaml
```

3. 使用具有相同工作负载配置的吞吐量驱动程序触发了另一个工作负载。

```
sudo bin/benchmark -drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

观察结果

使用两种不同类型的驱动程序来生成工作负载，以对在 NFS 上运行的 Kafka 实例的性能进行基准测试。驱动程序之间的区别在于日志刷新属性。

对于 Kafka 复制因子 1 和 FSx ONTAP：

- 同步驱动程序持续产生的总吞吐量：~ 3218 MBps，峰值性能约为 3652 MBps。
- 吞吐量驱动程序持续产生的总吞吐量：~ 3679 MBps，峰值性能为 ~ 3908 MBps。

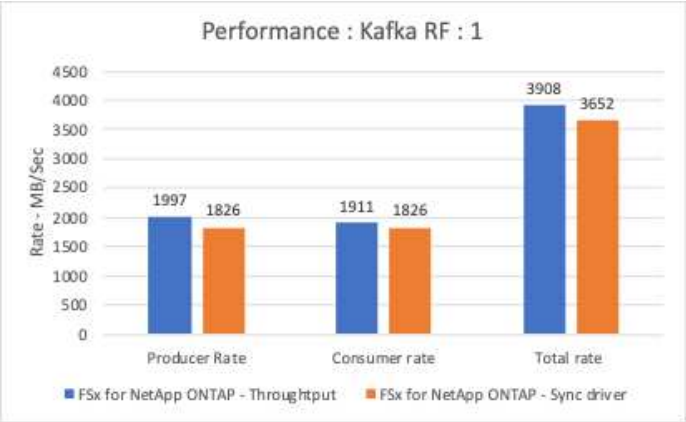
对于复制因子为 3 且具有 FSx ONTAP 的Kafka：

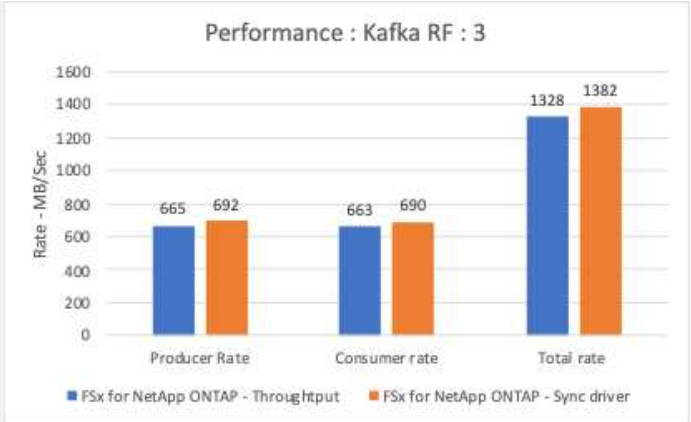
- 同步驱动程序持续产生的总吞吐量：~ 1252 MBps，峰值性能约为 1382 MBps。
- 吞吐量驱动程序持续产生的总吞吐量：~ 1218 MBps，峰值性能约为 1328 MBps。

在 Kafka 复制因子 3 中，读写操作在 FSx ONTAP上发生了三次，在 Kafka 复制因子 1 中，读写操作在 FSx ONTAP上发生了一次，因此在两种验证中，我们都能够达到 4GB/秒的最大吞吐量。

由于日志会立即刷新到磁盘，因此同步驱动程序可以生成一致的吞吐量，而由于日志会批量提交到磁盘，因此吞吐量驱动程序会产生突发性的吞吐量。

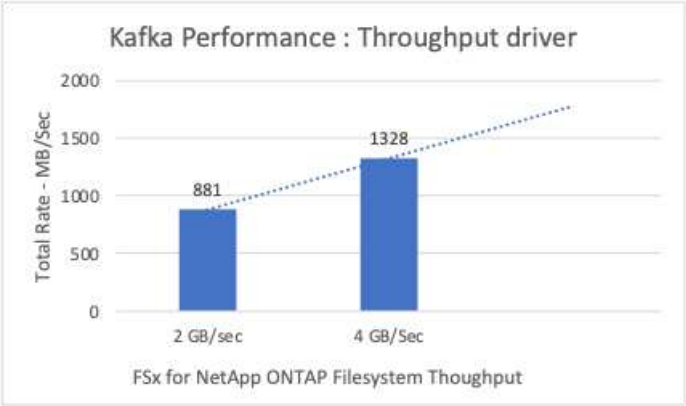
这些吞吐量数字是针对给定的 AWS 配置生成的。对于更高的性能要求，可以扩大实例类型并进一步调整以获得更好的吞吐量数字。总吞吐量或总速率是生产者和消费者速率的组合。

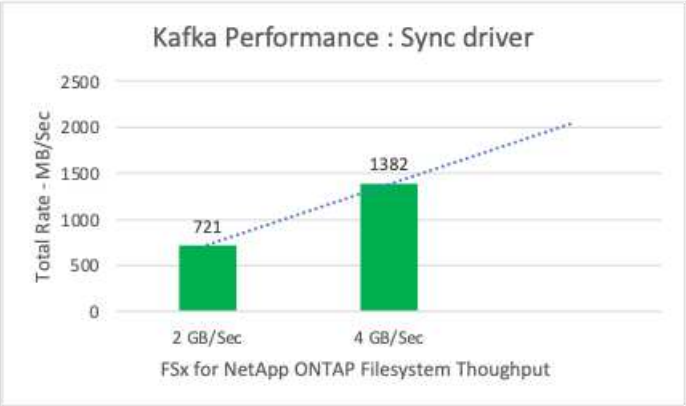




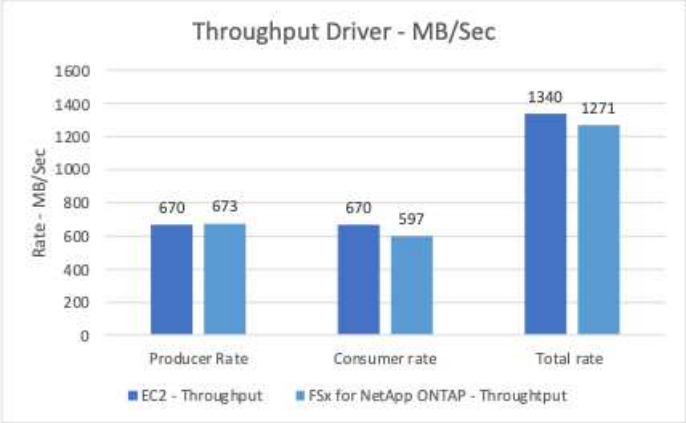
下图显示了 Kafka 复制因子 3 的 2GB/秒 FSx ONTAP和 4GB/秒性能。复制因子 3 在 FSx ONTAP存储上执行三次读写操作。吞吐量驱动程序的总速率为 881 MB/秒，在 2GB/秒 FSx ONTAP文件系统上以大约 2.64 GB/秒的速度读取和写入 Kafka 操作，吞吐量驱动程序的总速率为 1328 MB/秒，以大约 3.98 GB/秒的速度读取和写入

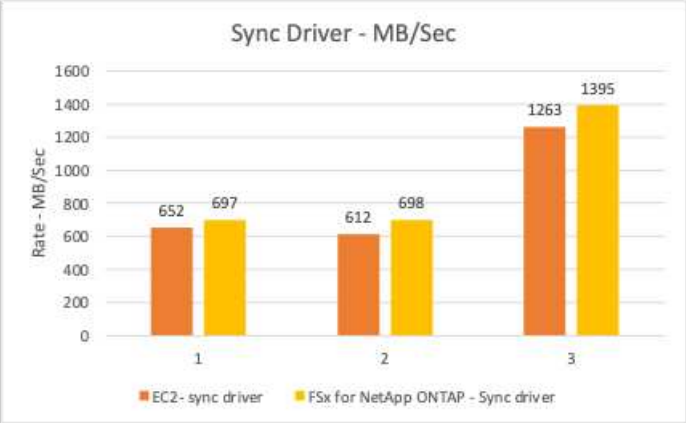
kafka 操作。 Kafka 性能是线性的，并且基于 FSx ONTAP吞吐量可扩展。





下图显示了 EC2 实例与 FSx ONTAP之间的性能（Kafka 复制因子：3）

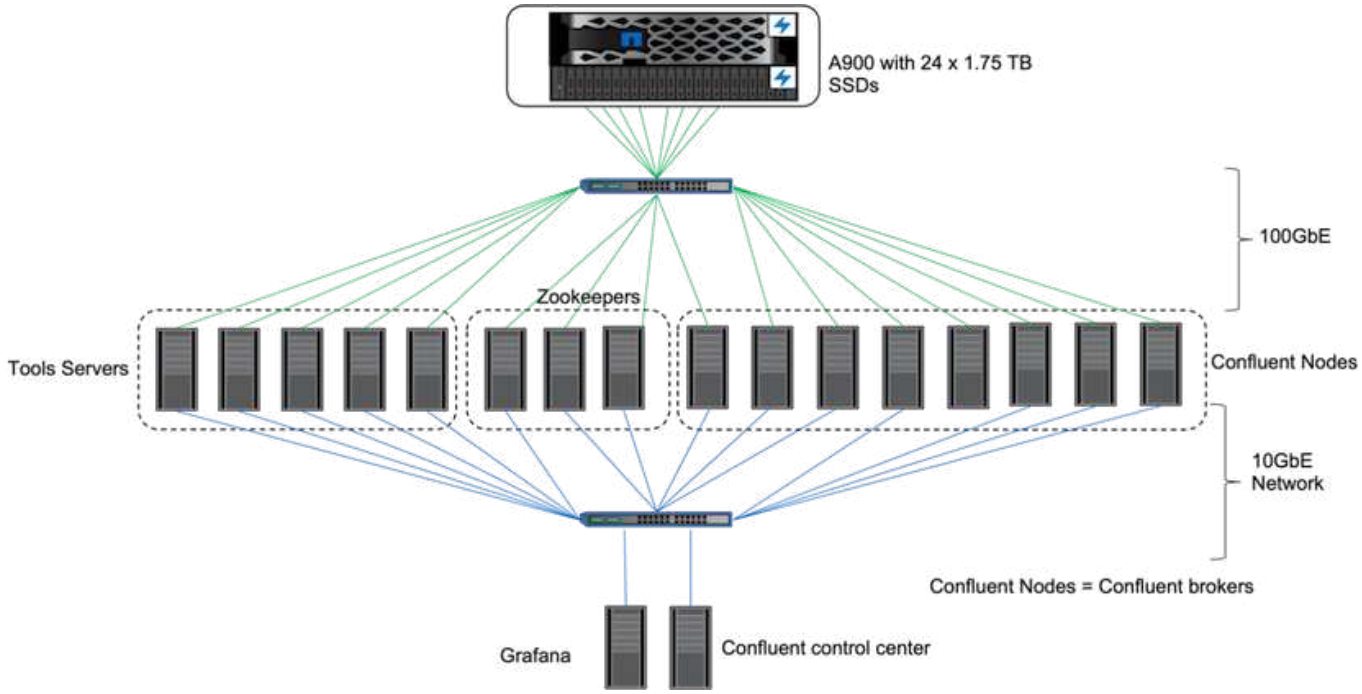




本地AFF A900的性能概述和验证

在本地，我们使用带有ONTAP 9.12.1RC1 的NetApp AFF A900存储控制器来验证 Kafka 集群的性能和扩展性。我们使用了与之前的ONTAP和AFF分层存储最佳实践相同的测试平台。

我们使用 Confluent Kafka 6.2.0 来评估AFF A900。该集群有八个代理节点和三个 zookeeper 节点。为了进行性能测试，我们使用了五个 OMB 工作节点。



存储配置

我们使用NetApp FlexGroups 实例为日志目录提供单一命名空间，从而简化恢复和配置。我们使用 NFSv4.1 和 pNFS 来提供对日志段数据的直接路径访问。

客户端调优

每个客户端使用以下命令挂载FlexGroup实例。

```
mount -t nfs -o vers=4.1,nconnect=16 172.30.0.121:/kafka_vol01
/data/kafka_vol01
```

此外，我们增加了 `max_session_slots``从默认 `64`到 `180。这与ONTAP中的默认会话槽限制相匹配。

Kafka 代理调优

为了最大限度地提高测试系统的吞吐量，我们显著增加了某些关键线程池的默认参数。对于大多数配置，我们建议遵循 Confluent Kafka 最佳实践。此调整用于最大化存储的未完成 I/O 的并发性。这些参数可以进行调整以匹配您的代理的计算资源和存储属性。

```
num.io.threads=96
num.network.threads=96
background.threads=20
num.replica.alter.log.dirs.threads=40
num.replica.fetchers=20
queued.max.requests=2000
```

工作负载生成器测试方法

我们对吞吐量驱动程序和主题配置使用了与云测试相同的 OMB 配置。

1. 使用 Ansible 在AFF集群上配置了FlexGroup实例。

```
---
- name: Set up kafka broker processes
  hosts: localhost
  vars:
    ntap_hostname: 'hostname'
    ntap_username: 'user'
    ntap_password: 'password'
    size: 10
    size_unit: tb
    vserver: vs1
    state: present
    https: true
    export_policy: default
    volumes:
      - name: kafka_fg_vol01
        aggr: ["aggr1_a", "aggr2_a", "aggr1_b", "aggr2_b"]
        path: /kafka_fg_vol01
  tasks:
    - name: Edit volumes
      netapp.ontap.na_ontap_volume:
        state: "{{ state }}"
        name: "{{ item.name }}"
        aggr_list: "{{ item.aggr }}"
        aggr_list_multiplier: 8
        size: "{{ size }}"
        size_unit: "{{ size_unit }}"
        vserver: "{{ vserver }}"
        snapshot_policy: none
        export_policy: default
        junction_path: "{{ item.path }}"
        qos_policy_group: none
        wait_for_completion: True
        hostname: "{{ ntap_hostname }}"
        username: "{{ ntap_username }}"
        password: "{{ ntap_password }}"
        https: "{{ https }}"
        validate_certs: false
      connection: local
      with_items: "{{ volumes }}"
```

2. ONTAP SVM 上已启用 pNFS。

```
vserver modify -vserver vs1 -v4.1-pnfs enabled -tcp-max-xfer-size 262144
```

3. 工作负载由吞吐量驱动程序触发，使用与Cloud Volumes ONTAP相同的工作负载配置。请参阅“[稳态性能] “以下。工作负载使用的复制因子为 3，这意味着在 NFS 中维护了三个日志段副本。

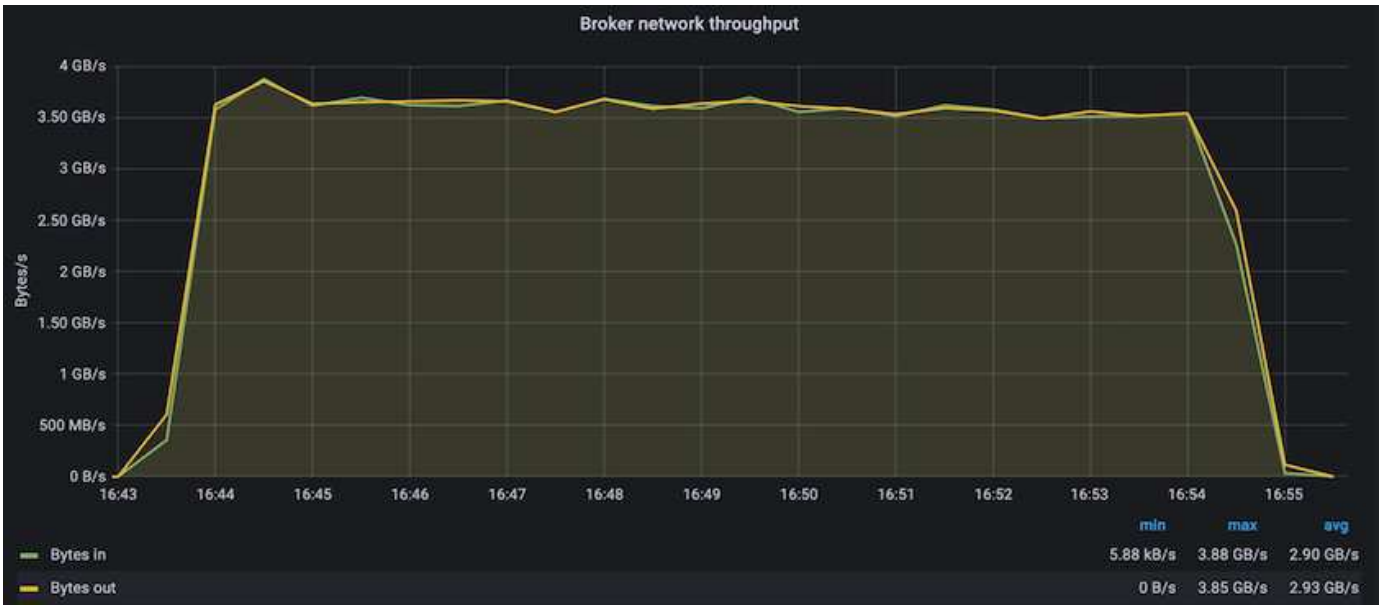
```
sudo bin/benchmark --drivers driver-kafka/kafka-throughput.yaml
workloads/1-topic-100-partitions-1kb.yaml
```

4. 最后，我们使用积压完成了测量，以衡量消费者赶上最新消息的能力。 OMB 通过在测量开始时暂停消费者来构建积压。这会产生三个不同的阶段：积压创建（仅生产者的流量）、积压消耗（消费者密集的阶段，其中消费者追赶主题中错过的事件）和稳定状态。请参阅“[极致性能和探索存储极限] ”了解更多信息。

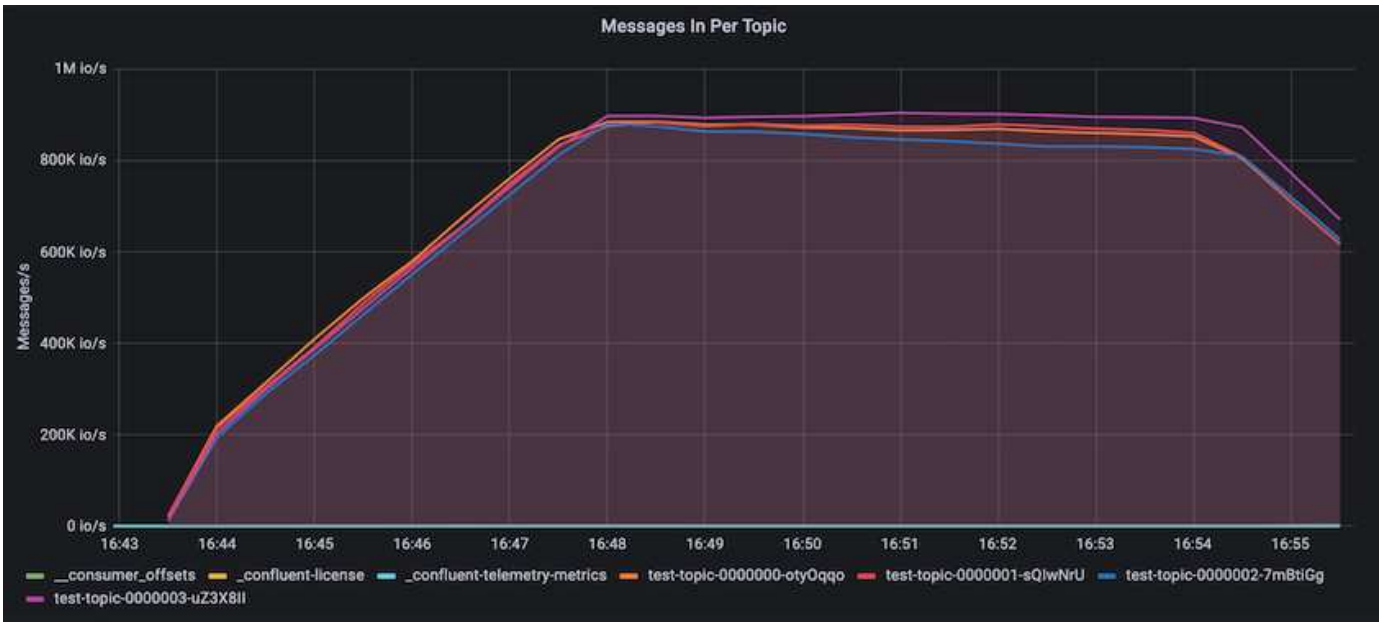
稳态性能

我们使用 OpenMessaging Benchmark 评估了AFF A900 ，以提供与 AWS 中的Cloud Volumes ONTAP和 AWS 中的 DAS 类似的比较。所有性能值代表生产者和消费者级别的 Kafka 集群吞吐量。

Confluent Kafka 和AFF A900的稳定状态性能为生产者和消费者实现了超过 3.4GBps 的平均吞吐量。 Kafka 集群中的消息超过 340 万条。通过以每秒字节数为单位可视化 BrokerTopicMetrics 的持续吞吐量，我们可以看到AFF A900支持的出色稳定状态性能和流量。



这与按主题传递的消息的视图非常一致。下图提供了按主题的细分情况。在测试的配置中，我们看到四个主题中每个主题有近 90 万条消息。

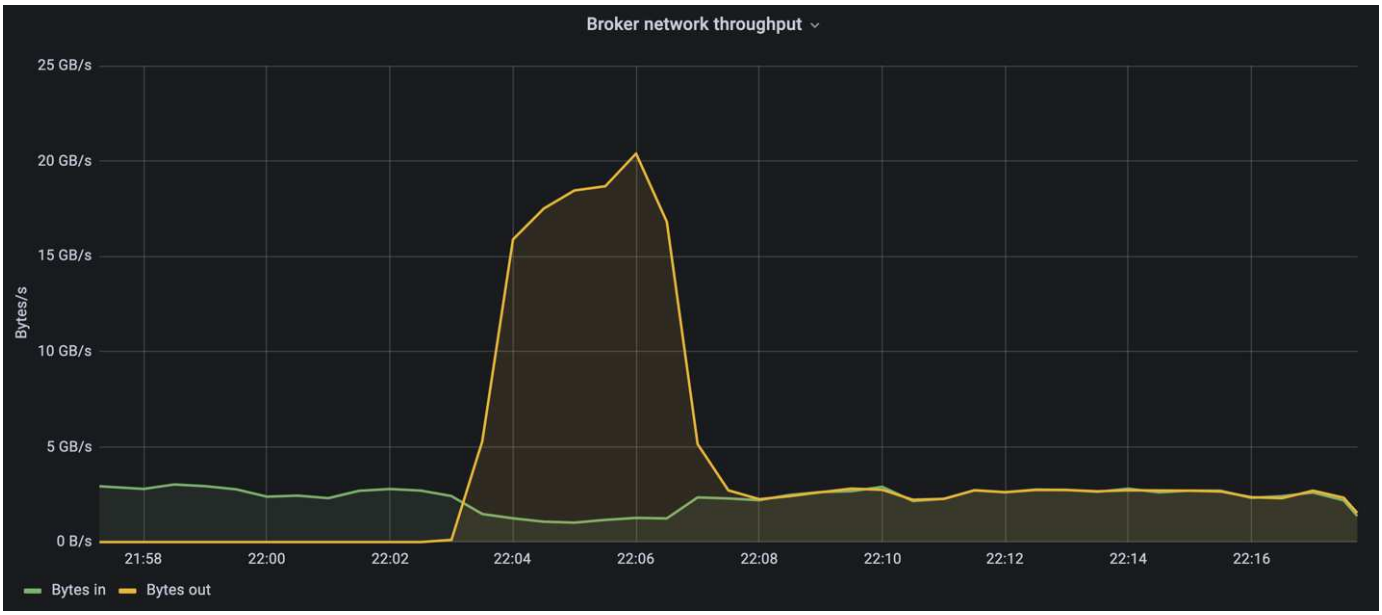


极致性能和探索存储极限

对于AFF，我们还使用积压功能对 OMB 进行了测试。当 Kafka 集群中积累了大量事件积压时，积压功能会暂停消费者订阅。在此阶段，仅发生生产者流量，从而生成提交到日志的事件。这最接近地模拟了批处理或离线分析工作流程；在这些工作流程中，消费者订阅已启动，并且必须读取已从代理缓存中逐出的历史数据。

为了了解此配置中存储对消费者吞吐量的限制，我们测量了仅生产者阶段，以了解 A900 可以吸收多少写入流量。请参阅下一节“[尺寸指南] “来了解如何利用这些数据。

在本次测量的仅生产者部分，我们看到了高峰值吞吐量，突破了 A900 性能的极限（当其他代理资源未饱和地为生产者和消费者流量提供服务时）。



我们将此测量的消息大小增加到 16k，以限制每个消息的开销并最大限度地提高 NFS 挂载点的存储吞吐量。

```
messageSize: 16384
consumerBacklogSizeGB: 4096
```

Confluent Kafka 集群实现了 4.03GBps 的峰值生产者吞吐量。

```
18:12:23.833 [main] INFO WorkloadGenerator - Pub rate 257759.2 msg/s /
4027.5 MB/s | Pub err     0.0 err/s …
```

OMB 完成填充事件积压日志后，消费者流量重新启动。在积压工作消耗的测量过程中，我们观察到所有主题的峰值消费者吞吐量超过 20GBps。存储 OMB 日志数据的 NFS 卷的组合吞吐量接近 ~30GBps。

尺寸指南

亚马逊网络服务提供 "尺码指南"用于 Kafka 集群大小调整和扩展。

此大小提供了一个有用的公式来确定 Kafka 集群的存储吞吐量要求：

对于复制因子为 r 的 tcluster 集群产生的聚合吞吐量，代理存储收到的吞吐量如下：

```
t[storage] = t[cluster]/#brokers + t[cluster]/#brokers * (r-1)
           = t[cluster]/#brokers * r
```

这可以进一步简化：

```
max(t[cluster]) <= max(t[storage]) * #brokers/r
```

使用此公式，您可以根据 Kafka 热层需求选择合适的ONTAP平台。

下表解释了具有不同复制因子的 A900 的预期生产者吞吐量：

复制因子	生产者吞吐量 **(GPps)**
3（测量）	3.4
2	5.1
1	10.2

结束语

NetApp针对愚蠢重命名问题的解决方案为以前与 NFS 不兼容的工作负载提供了一种简单、廉价且集中管理的存储形式。

这种新模式使客户能够创建更易于管理的 Kafka 集群，这些集群更易于迁移和镜像，以实现灾难恢复和数据保护。我们还看到 NFS 提供了其他好处，例如降低 CPU 利用率和加快恢复时间、显著提高存储效率以及通

过NetApp ONTAP实现更好的性能。

在哪里可以找到更多信息

要了解有关本文档中描述的信息的更多信息，请查看以下文档和/或网站：

- 什么是 Apache Kafka?

  "https://www.confluent.io/what-is-apache-kafka/"

- 什么是愚蠢的重命名?

  "https://linux-nfs.org/wiki/index.php/Server-side_silly_rename"

- ONATP 用于流媒体应用程序。

  "https://www.netapp.com/blog/ontap-ready-for-streaming-applications/"

- NetApp产品文档

  "https://www.netapp.com/support-and-training/documentation/"

- 什么是 NFS?

  "https://en.wikipedia.org/wiki/Network_File_System"

- 什么是 Kafka 分区重新分配?

  "https://docs.cloudera.com/runtime/7.2.10/kafka-managing/topics/kafka-manage-cli-reassign-overview.html"

- 什么是 OpenMessaging 基准?

  "https://openmessaging.cloud/"

- 如何迁移 Kafka 代理?

  "https://medium.com/@sanchitbansal26/how-to-migrate-kafka-cluster-with-no-downtime-58c216129058"

- 如何使用 Prometheus 监控 Kafka 代理?

  https://www.confluent.io/blog/monitor-kafka-clusters-with-prometheus-grafana-and-confluent/

- Apache Kafka 托管平台

  https://www.instaclustr.com/platform/managed-apache-kafka/

- 支持 Apache Kafka

  https://www.instaclustr.com/support-solutions/kafka-support/

- Apache Kafka 咨询服务

https://www.instaclustr.com/services/consulting/