Message Queue Performance Study on VMware Tanzu Kubernetes Grid

VMware Tanzu RabbitMQ 1.3 and Confluent Kafka

vmware

DELL Technologies

Table of Contents

Overview

Note: This solution provides an overview of the performance improvements in new version of Tanzu RabbitMQ and guidelines for running Apache Kafka running on VMware Tanzu® Kubernetes Grid™. The reference architecture applies to general vSphere and vSAN platforms.

Introduction

VMware Tanzu RabbitMQ 1.3 contains RabbitMQ 3.10 that was released on the 3rd of May 2022, with many new features and improvements.

Apache Kafka is an open-source distributed event streaming platform for high-performance data pipelines, streaming analytics. It's used as a popular message queue for distributed systems and is commonly used to stream data in the IoT use cases.

This paper provides the performance improvements of Tanzu RabbitMQ for Kubernetes 1.3 on VMware Tanzu Kubernetes Grid. And it covered the general functional testing and guideline for running Apache Kafka on Tanzu Kubernetes Grid.

Prerequisites

You must first set up a reference architecture environment for Tanzu Kubernetes Grid. And then, the material takes you through the steps to install VMware Tanzu RabbitMQ for Kubernetes.

Technology Overview

The technology components in this solution are:

- VMware Tanzu Kubernetes Grid Multi-Cloud (TKGm)
- Tanzu RabbitMQ for Kubernetes
- Apache Kafka
- Strimzi

VMware Tanzu Kubernetes Grid

VMware Tanzu Kubernetes Grid provides organizations with a consistent, upstream-compatible, regional Kubernetes substrate that is ready for end-user workloads and ecosystem integrations. You can deploy Tanzu Kubernetes Grid across software-defined datacenters (SDDC) and public cloud environments, including vSphere, Microsoft Azure, and Amazon EC2.

Tanzu Kubernetes Grid provides the services such as networking, authentication, ingress control, and logging that a production Kubernetes environment requires. It can simplify operations of large-scale, multi-cluster Kubernetes environments and keep your workloads properly isolated. Also, it automates lifecycle management to reduce your risk and shift your focus to more strategic work.

Tanzu RabbitMQ for Kubernetes

VMware Tanzu RabbitMQ for Kubernetes provides the building blocks for a cloud native messaging and streaming service that you can deploy on any Kubernetes cluster.

RabbitMQ stream is a new feature of both the open source RabbitMQ and commercial Tanzu RabbitMQ editions. It delivers some of the benefits of log structures, like fast data transfer and the ability to replay and reprocess messages, combined with the best Tanzu RabbitMQ features like flexible routing and command and response, all with a high degree of data safety.

See VMware Tanzu RabbitMQ 1.3 for detailed information.

Apache Kafka

Apache Kafka is a community distributed event streaming platform capable of handling trillions of events a day. Initially conceived as a messaging queue, Kafka is based on an abstraction of a distributed commit log. Since being created and open sourced by LinkedIn in 2011, Kafka has quickly evolved from messaging queue to a full-fledged event streaming platform.

A cluster of Kafka brokers handles delivery of messages.

A broker uses Apache ZooKeeper for storing configuration data and for cluster coordination. Before running Apache Kafka, an Apache ZooKeeper cluster has to be ready. Apache ZooKeeper is a core dependency for Kafka as it provides a cluster coordination service, storing and tracking the status of brokers and consumers. ZooKeeper is also used for controller election.

Strimzi

Strimzi provides a way to run an Apache Kafka cluster on Kubernetes in various deployment configurations. For development it's easy to set up a cluster in a few minutes. For production you can tailor the cluster to your needs, using features such as rack awareness to spread brokers across availability zones, and Kubernetes taints and tolerations to run Kafka on dedicated nodes. You can expose Kafka outside Kubernetes using NodePort, Load balancer, Ingress and OpenShift Routes, depending on your needs, and these are easily secured using TLS. The Operators provided with Strimzi are purpose-built with specialist operational

knowledge to effectively manage Kafka.

Configuration

Architecture

In our solution, we deployed the Tanzu RabbitMQ test environment using Tanzu Kubernetes Grid on a 4-node cluster. Firstly, we deployed the Tanzu Kubernetes Grid management cluster with Tanzu CLI by either an installer UI or a configuration file. Refer to Prepare to deploy Management clusters to vSphere for more details. After you deploy a management cluster to vSphere, use the Tanzu CLI to deploy the Tanzu Kubernetes Grid workload cluster. The Tanzu Kubernetes cluster is deployed through a configuration file with customized variables for cluster settings and scalability. Refer to Deploy Tanzu Kubernetes Clusters to vSphere for more details. To support the Tanzu RabbitMQ and Apache Kafka deployment, we recommend configuring a Tanzu Kubernetes Grid workload cluster with at least 3 nodes; each node is requested with a minimum value of 24 vCPU, 72GB memory, and 2TB disk space.

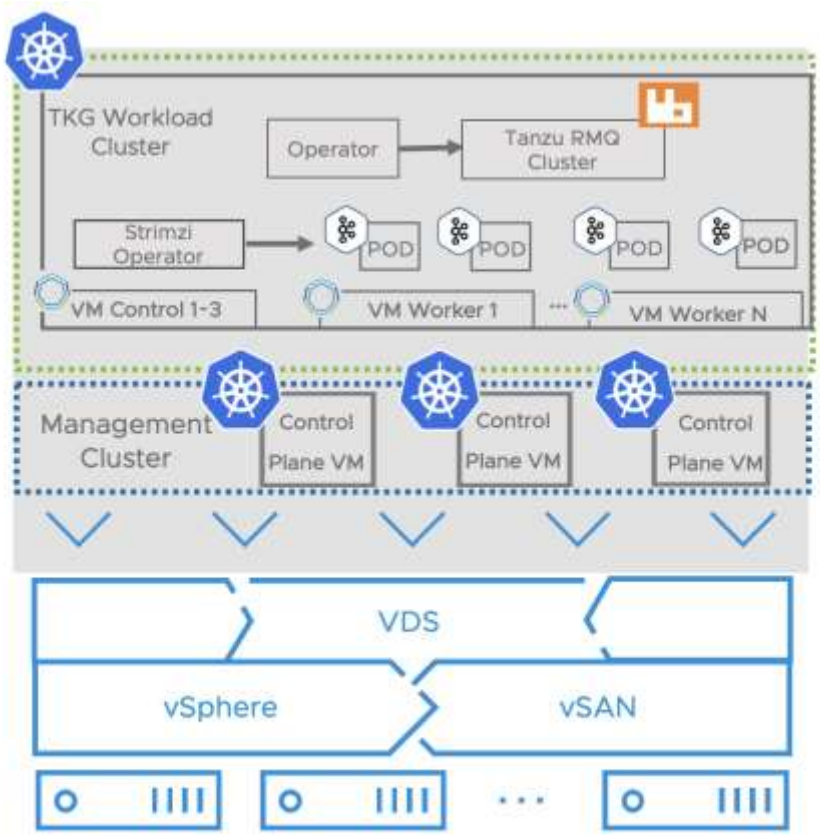


Figure 1: Tanzu RabbitMQ and Apache Kafka Running on Tanzu Kubernetes Grid

Now you can deploy VMware Tanzu RabbitMQ for Kubernetes to a TKG workload cluster. You can pull all the Tanzu RabbitMQ components of the package from the registry. Then you will install the package to install the Tanzu RabbitMQ Operator to your cluster. After the installation is completed, you can now create a RabbitMQ cluster via the configuration file.

We choose Strimzi operator to run an Apache Kafka cluster on TKG in customized deployment configurations. Strimzi provides a Helm chart to deploy the Cluster Operator. Apache Kafka components are provided for deployment to Kubernetes with the Strimzi distribution. The Cluster Operator watches for updates in the namespaces where the Kafka resources are deployed. You can now deploy a Kafka cluster via the configuration files. Strimzi also installs a ZooKeeper cluster and adds the necessary configuration to connect Kafka with ZooKeeper.

Hardware Resource

Table 1: Hardware Configuration

PROPERTY	SPECIFICATION
CPU	2 x Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz, 28 core each
RAM	480GB
Network adapter	2 x Intel 10Gb Ethernet Controller X550
Storage adapter	1 x Dell HBA330 Mini Adapter
Disks	Cache - 2 x 1.46TB NVMe Capacity - 8 x 1.75TB SSD

Software Resource

Table 2: Software Resources

SOFTWARE	VERSION	PURPOSE
VMware vSphere	7.0 U3c	VMware vSphere is a suite of products: vCenter Server and ESXi.
Tanzu RabbitMQ for Kubernetes	1.3	Based on the widely popular open source RabbitMQ messaging system, Tanzu RabbitMQ for Kubernetes is designed to work seamlessly with Tanzu and also run on any certified Kubernetes distribution.
Tanzu CLI	V0.11.6	Command line interface that allows deploying CNCF conformant Kubernetes clusters to vSphere and other cloud infrastructure.
Kubernetes	V1.22.9	Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.

Kubernetes OVA for Tanzu Kubernetes Grid	Ubuntu 20.04 and Kubernetes v1.22.9+vmware.1	A base image template for the Kubernetes Operating System of Tanzu Kubernetes Grid management and workload clusters.
RabbitMQ PerfTest	V2.18.0	A throughput testing tool that is based on the Java client and can be configured to simulate basic workloads and more advanced workloads as well.
RabbitMQ Stream Java Client	V0.7.0	A Java library to communicate with the RabbitMQ Stream Plugin. It contains a performance tool to allow creating and deleting streams as well as publishing to and consuming from these streams to test the performance.
Strimzi	V0.31.0	A way to run an Apache Kafka cluster on Kubernetes in various deployment configurations.
Apache Kafka	V3.2.1	An open-source event streaming platform, it aims to provide the user with a unified, high latency, high throughput platform for handling real-time data.

Table 3. VM Configuration

VM Role	vCPU	Memory (GB)	Storage (GB)	VM Count
Tanzu Kubernetes Grid Management cluster – Control Plane VM	2	4	20	1
Tanzu Kubernetes Grid Management cluster – Worker node VM	2	4	20	1
Tanzu Kubernetes Grid Workload cluster – Control Plane VM	4	32	120	3
Tanzu Kubernetes Grid Workload cluster – Worker node VM	24	72	2000	3

Application Validation

Introduction

This solution testing showcases Tanzu RabbitMQ running on Tanzu Kubernetes Grid for performance improvement and Apache Kafka running Tanzu Kubernetes Grid for performance validation.

Performance Testing Scope and Result

Tanzu RabbitMQ for Kubernetes 1.3 includes RabbitMQ 3.10 with many new features and improvements on Quorum Queue.

We used RabbitMQ PerfTest Tool to generate loads for the RabbitMQ cluster to compare quorum queues with the previous version. In the performance testing, we provisioned a cluster of three Tanzu RabbitMQ pods, each with 8 vCPUs, 16GB of RAM, and 2000GB storage.

See RabbitMQ PerfTest for more information about the performance tool,

Scenario 1: We used just 1 quorum queue with a fast publisher and consumer. Compared with the previous version, Tanzu RabbitMQ 1.3 improved 30% throughput. And we tested with message sizes of 10, 100, 1000 and 5000 bytes.

Figure 1. Messages published/s of 10, 100, 1000 and 5000 bytes

We ran the test with different message sizes:

- Each quorum queue published rate average at 55,634 message/s, consumed average at 55,628 message/s at small message size
- Each quorum queue published rate average at 15,334 message/s, consumed average at 15,207 message/s at large message size

Scenario 2: We published large million messages and then consumed all of them. It took ~130s to publish 10 million messages and ~170s to consume the messages.

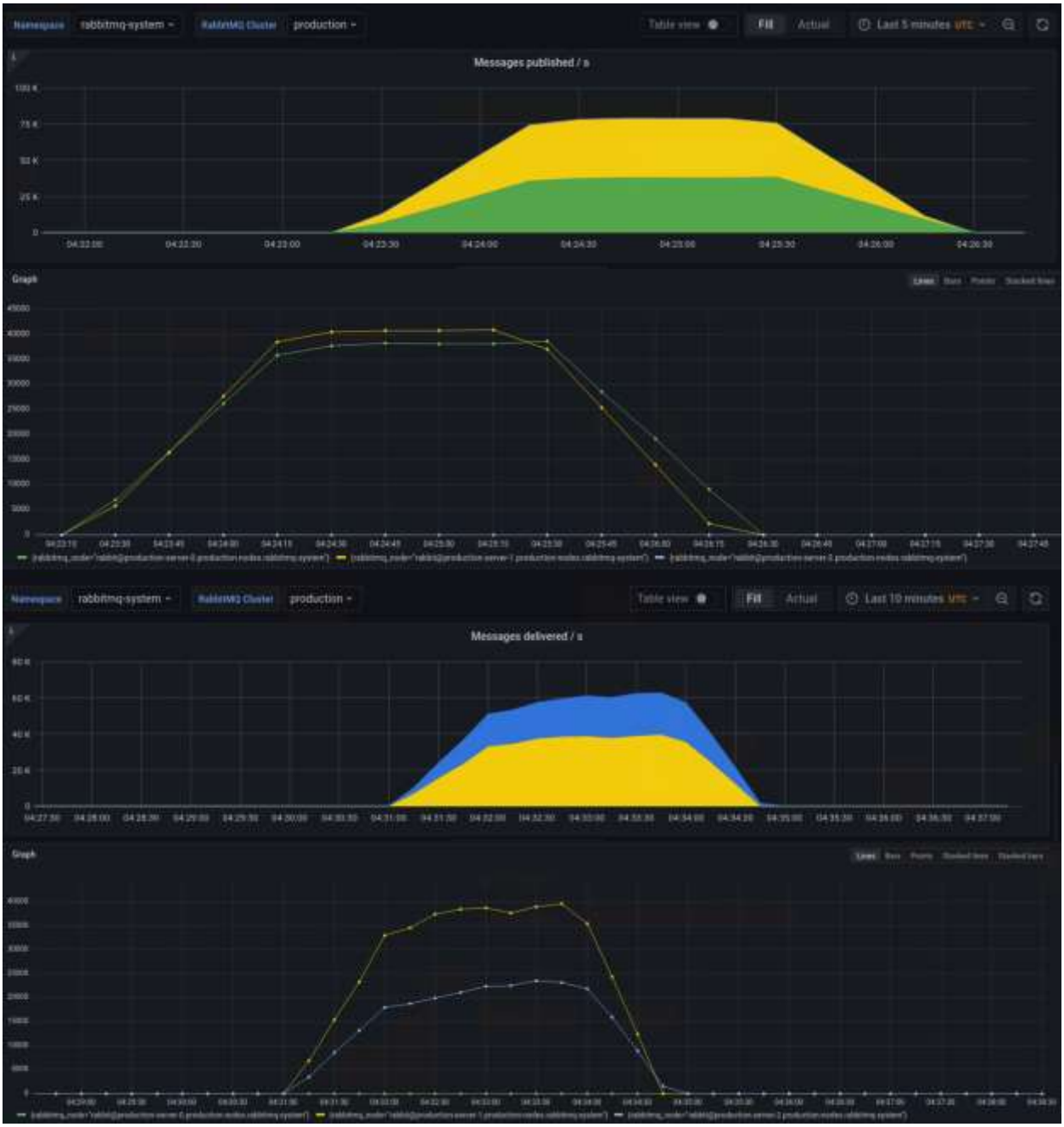


Figure 2. 10 million message published and consumed

Figure 3. Memory available before publishers blocked

During the process of publishing and consuming, Quorum queue used ~3GB memory. But when we published 50 million messages, the cluster hit a memory alarm and stopped due to wait for publisher confirms for too long.

Scenario 3: we tested Tanzu RabbitMQ streams with different number of streams. It didn't improve much than the previous version. The detailed published, consumer results are shown below:

- Single RabbitMQ streams published 976589 message/s, consumed 973953 msg/s
- Three (3) streams published 2475177 msg/s, consumed 2473692 msg/s

Application Validation

Introduction

This solution testing showcases Apache Kafka running on Tanzu Kubernetes Grid for performance validation.

Performance Testing Scope and Result

Apache Kafka is a community distributed event streaming platform capable of handling trillions of events a day. Initially conceived as a messaging queue, Kafka is based on an abstraction of a distributed commit log. Strimzi provides a way to run an Apache Kafka cluster on Kubernetes in various deployment configurations.

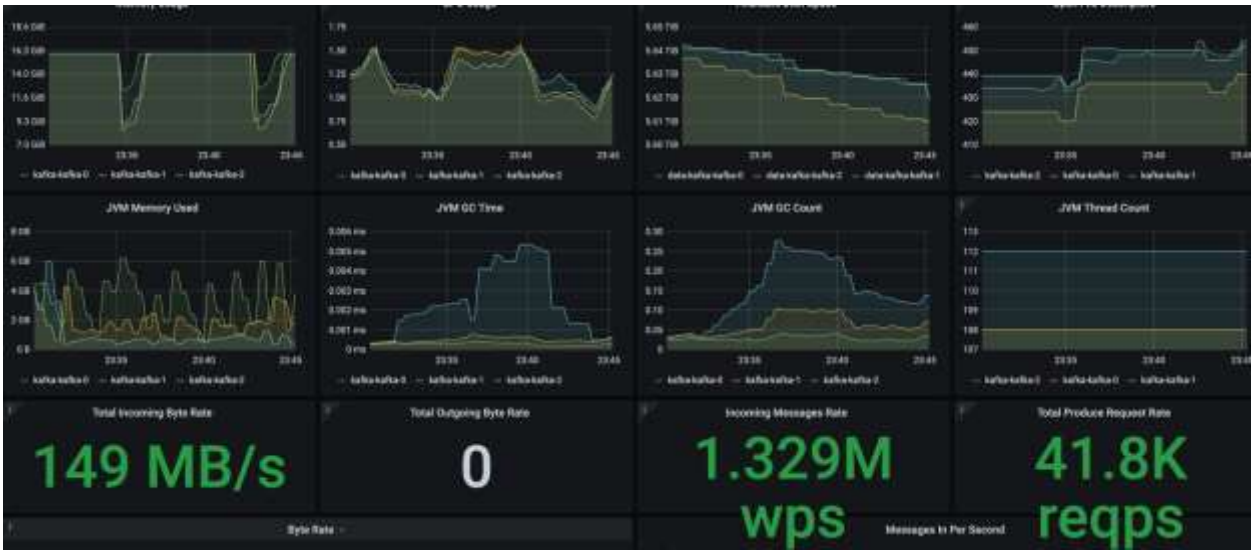
Strimzi provides a Helm chart to deploy the Cluster Operator. The Cluster Operator watches for updates in the namespaces where the Kafka resources are deployed. You can now deploy a Kafka cluster via the configuration files. Strimzi also installs a ZooKeeper cluster and adds the necessary configuration to connect Kafka with ZooKeeper. In configuration file, it can export the metrics that we can get the performance through Prometheus and Grafana.

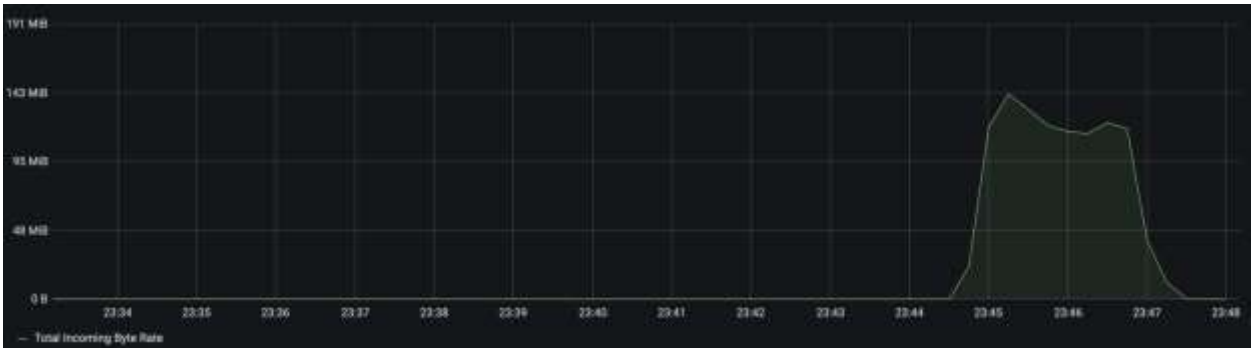
I allocated 3 broker pods with 8 vCPU and 32 GB RAM in container resource spec. I created the topic with 12 partitions and a replication factor of 1 or 3.

Scenario 1: We created a job with the configuration to achieve the maximum throughput. For single Producer throughput with no replication and 3 replicas, each broker runs the following producer-perf test command below:

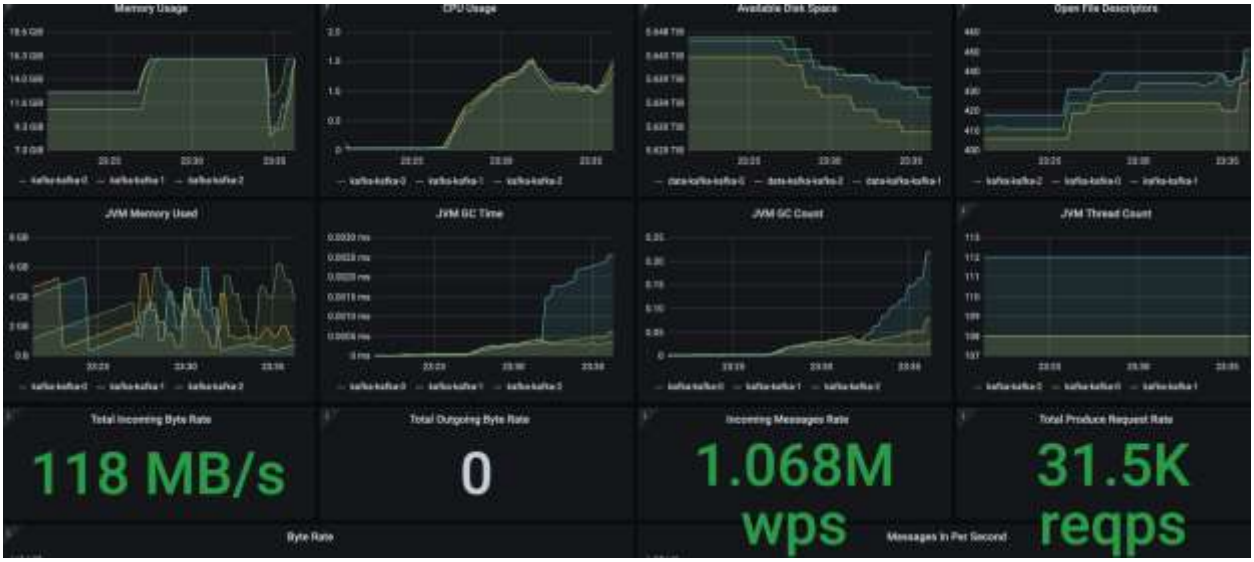
./kafka-producer-perf-test.sh --topic test_p12_r1 --producer-props bootstrap.servers=bootstrap.kafka.svc:9092 buffer.memory=67108864 batch.size=64000 acks=1 --record-size 100 --throughput -1 --num-records 50000000

I got an average of 492K records/s with 100 bytes per record, 46.9MB/s throughput and 1.13ms latency. For 3 producers, I can get the total peak throughput at 149MB/s.

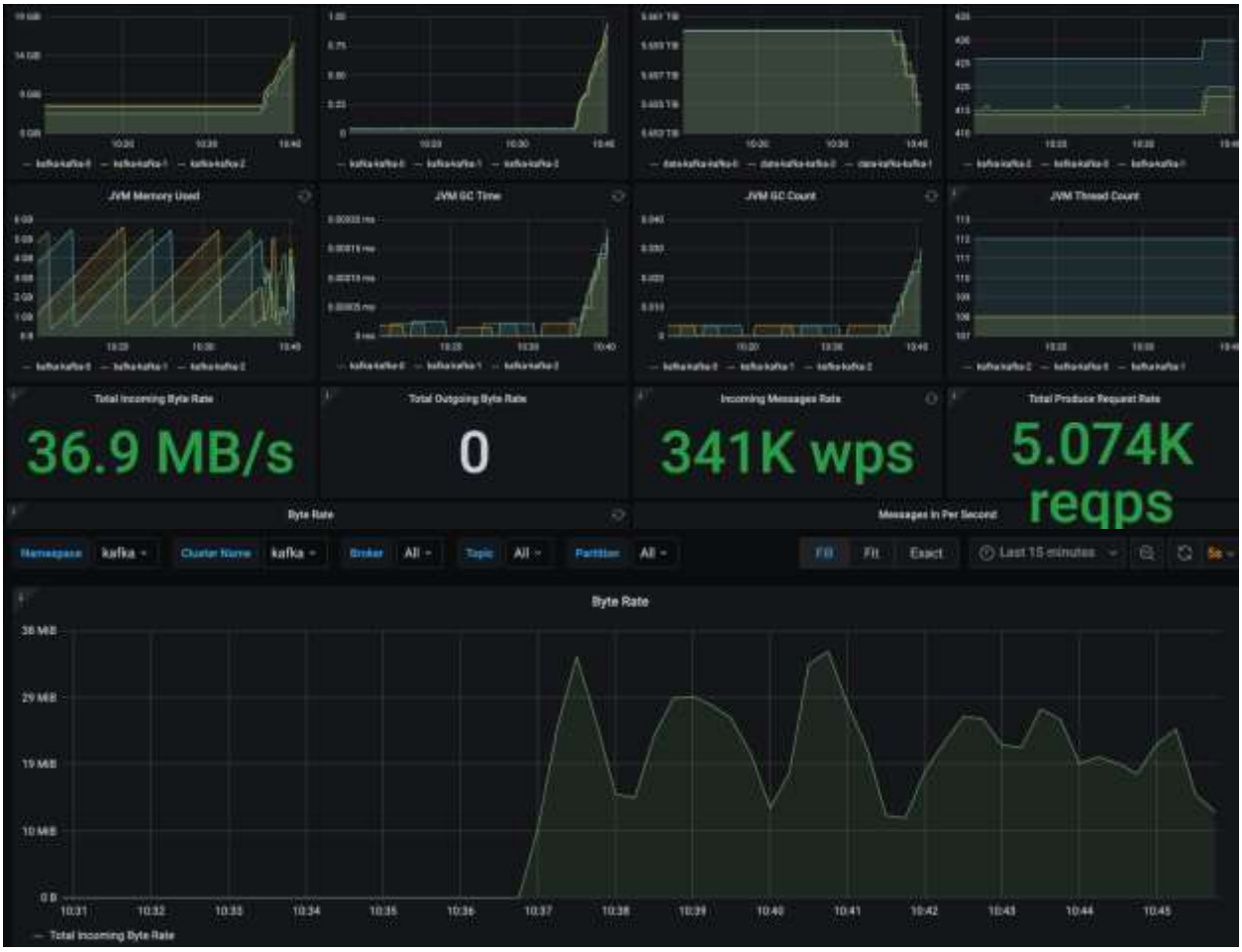




Then we created a topic with 3x replication and get the replica async. It can get the total peak throughput at 118MB/s.



Next, we created a topic with 3x replication and get the replica sync. It means all the replicas have been persisted before ACK the producer. We can see a little bit more than 3x performance reduction. That is related to the overhead of sending and receiving ACKs by the brokers.



We run the commands to consume all the messages,

```
./kafka-consumer-perf-test.sh --bootstrap-server localhost:9092 --messages 50000000 --topic test_p12_r3 --threads 1
```

We can see the throughput is bigger than the producer with no replica.

Apache Kafka is a stream processing and event-driven architectures, end to end latency for a message to traverse the pipeline from the producer through the system to the consumer. To test end-to-end latency, I run the below command:

```
sh kafka-run-class.sh kafka.tools.EndToEndLatency localhost:9092 test_p12_r3
10000 all 1000
```

```
175000  6.354546999999999
176000  1.458957
177000  1.4203610000000002
178000  1.42405
179000  1.115937
180000  1.518577
181000  1.516256
182000  1.540252
183000  1.47108
184000  1.920327
185000  1.540208
186000  1.459633
187000  1.9301
188000  1.088621
189000  1.572921
190000  1.260847
191000  1.466928
192000  1.562683
193000  1.396028
194000  1.452801
195000  1.471385
196000  1.594881
197000  1.101534
198000  1.463369
199000  1.527193
Avg latency: 2.7601 ms

Percentiles: 50th = 1, 99th = 14, 99.9th = 209
```

The topic test_p12_r3 was created with 12 partitions and RF=3, producer acks was set to all, the highest durability. I get an average of 3ms latency at 1KB message size.

Warm Standby Replication Validation

Introduction

This testing is a showcase of the warm standby replication scenario.

Testing Scope and Result

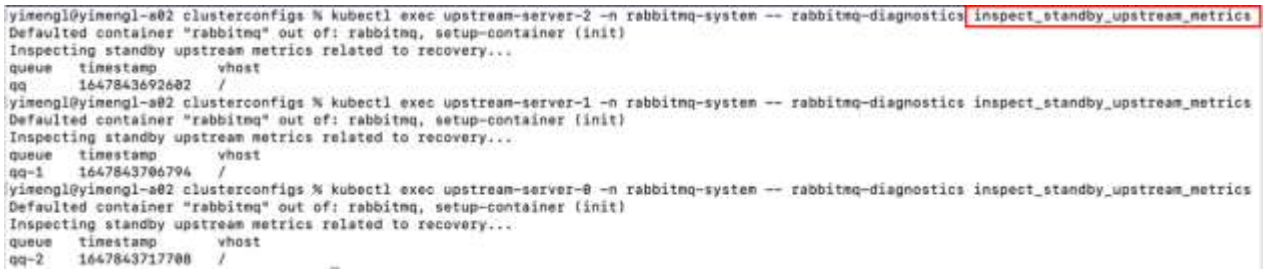
VMware Tanzu RabbitMQ supports continuous schema definition and message replication to a remote cluster, which makes it easy to run a standby cluster for disaster recovery. This feature is not available in the open source RabbitMQ distribution.

To achieve Tanzu RabbitMQ 's warm standby replication capability, perform the following steps:

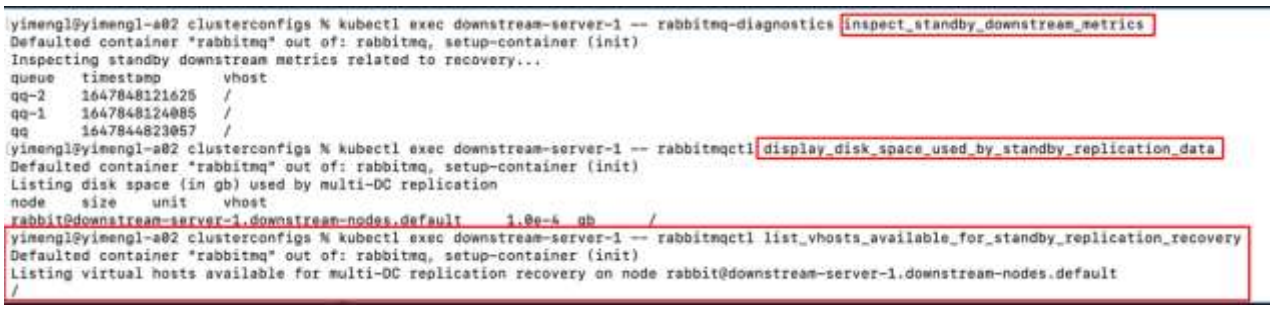
1. Set up an upstream and downstream RabbitMQ cluster with required plugins and configurations by following the manifest files.

2. Configure the upstream and downstream standby replication via standby offsite replication operator. The configuration files let the plugin know it should collect messages for all quorum queues in default vhost and connect RabbitMQ cluster downstream to RabbitMQ cluster at the endpoint — {EndPoint IP}:5552.

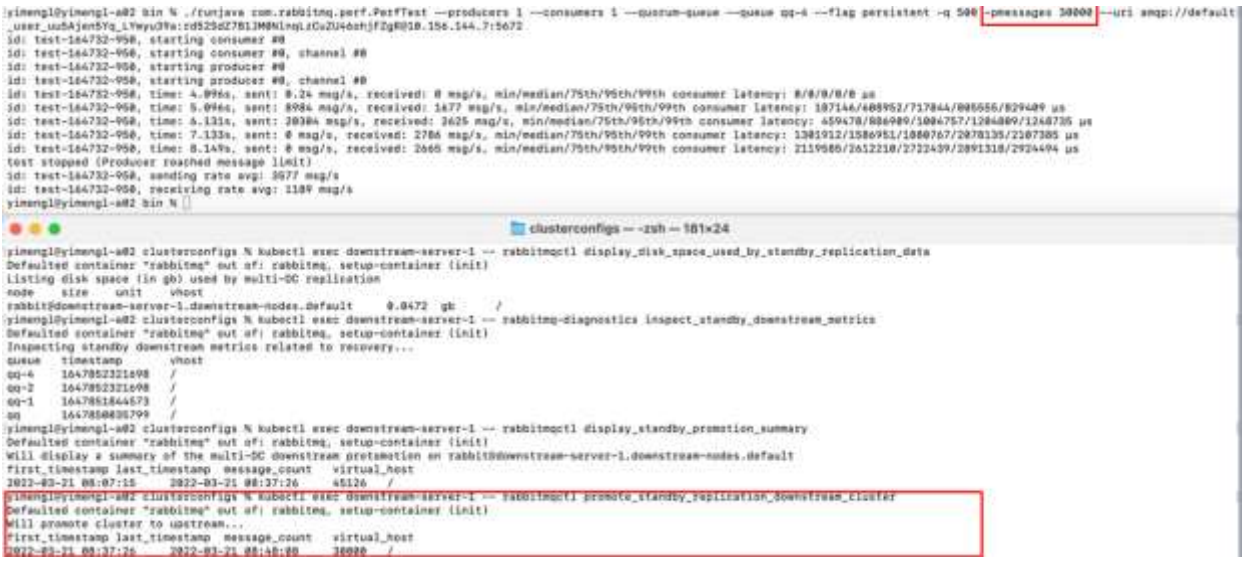
3. We created 3 queues used by the replication and ran the rabbitmq-diagnostics inspect_standby_upstream_metrics on three nodes of RabbitMQ cluster to verify how the replication is working on the upstream side. The figure shows the replication is spread across the three nodes of upstream cluster:



4. Verify the downstream replication is also working. The figure shows that the default vhost is responsible for replication on the downstream side.



Now we used PerfTool to publish 30,000 messages to a queue from upstream cluster and then replicated them to the downstream side. We could check the disk space used by the standby replication data. At this point, we supposed the upstream site went down and we initiated a recovery procedure to promote the downstream cluster to upstream. The figure shows that we promoted 30,000 messages on the downstream site.


yimengl@yimengl-a02 bin % ./runjava com.rabbitmq.perf.PerfTest --producers 1 --consumers 1 --quorum-queue --queue qq-4 --flag persistent -q 500 -pmessages 30000 --uri amqp://default
id: test-164732-950, starting consumer #0
id: test-164732-950, starting consumer #0, channel #0
id: test-164732-950, starting producer #0
id: test-164732-950, starting producer #0, channel #0
id: test-164732-950, time: 4.096s, sent: 0.24 msg/s, received: 0 msg/s, min/median/75th/95th/99th consumer latency: 0/0/0/0/0 µs
id: test-164732-950, time: 5.096s, sent: 8984 msg/s, received: 1677 msg/s, min/median/75th/95th/99th consumer latency: 107146/408952/717044/885656/829409 µs
id: test-164732-950, time: 6.131s, sent: 20304 msg/s, received: 2625 msg/s, min/median/75th/95th/99th consumer latency: 459478/886989/1004757/1284809/1248735 µs
id: test-164732-950, time: 7.133s, sent: 0 msg/s, received: 2786 msg/s, min/median/75th/95th/99th consumer latency: 1381912/1586951/1880767/2078135/2107385 µs
id: test-164732-950, time: 8.149s, sent: 0 msg/s, received: 2665 msg/s, min/median/75th/95th/99th consumer latency: 2119586/2612218/2722439/2891318/2924494 µs
test stopped (Producer reached message limit)
id: test-164732-950, sending rate avg: 3677 msg/s
id: test-164732-950, receiving rate avg: 1189 msg/s
yimengl@yimengl-a02 bin %
clusterconfigs — -zsh — 181×24
yimengl@yimengl-a02 clusterconfigs % kubectl exec downstream-server-1 -- rabbitmqctl display_disk_space_used_by_standby_replication_data
Defaulted container "rabbitmq" out of: rabbitmq, setup-container (init)
Listing disk space (in gb) used by multi-DC replication
node size unit vhost
rabbit@downstream-server-1.downstream-nodes.default 0.0472 gb /
yimengl@yimengl-a02 clusterconfigs % kubectl exec downstream-server-1 -- rabbitmq-diagnostics inspect_standby_downstream_metrics
Defaulted container "rabbitmq" out of: rabbitmq, setup-container (init)
Inspecting standby downstream metrics related to recovery...
queue timestamp vhost
qq-4 1647852321698 /
qq-2 1647852321698 /
qq-1 1647851844673 /
qq 1647850835799 /
yimengl@yimengl-a02 clusterconfigs % kubectl exec downstream-server-1 -- rabbitmqctl display_standby_promotion_summary
Defaulted container "rabbitmq" out of: rabbitmq, setup-container (init)
Will display a summary of the multi-DC downstream protomotion on rabbit@downstream-server-1.downstream-nodes.default
first_timestamp last_timestamp message_count virtual_host
2022-03-21 08:07:15 2022-03-21 08:37:26 45126 /
yimengl@yimengl-a02 clusterconfigs % kubectl exec downstream-server-1 -- rabbitmqctl promote_standby_replication_downstream_cluster
Defaulted container "rabbitmq" out of: rabbitmq, setup-container (init)
Will promote cluster to upstream...
first_timestamp last_timestamp message_count virtual_host
2022-03-21 08:37:26 2022-03-21 08:46:08 30000 /

Additional Resources

For more information about Tanzu RabbitMQ, you can explore the following resources:

- VMware vSphere
- VMware vSAN
- VMware Tanzu Kubernetes Grid
- VMware Tanzu RabbitMQ
- Running VMware Tanzu RabbitMQ on VMware Tanzu Kubernetes Grid
- RabbitMQ Stream Java Client
- Installing VMware Tanzu RabbitMQ for Kubernetes
- Solution Deployment and Configuration Files

About the Author and Contributors

Yimeng Liu, Senior Solutions Manager in the Workload Technical Marketing Team of the Cloud Infrastructure Big Group, wrote the original version of this paper. The following reviewers also contributed to the paper contents:

- *Ka Kit Wong, Staff Solutions Architect in VMware*
- *Myles Gray, Staff Solutions Architect in VMware*
- *Mark Xu, Senior Solutions Manager in VMware*
- *Michal Kuratczyk, Staff Engineer in VMware*
- *Catherine Xu, Workload Technical Marketing Manager in VMware*
- *Vic Dery, Senior Principal Engineer of VxRail Technical Marketing in Dell Technologies*

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com.
Copyright © 2022 VMware, Inc. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws. VMware products are covered by one or more patents listed at http://www.vmware.com/go/patents. VMware is a registered trademark or trademark of VMware, Inc. and its subsidiaries in the United States and other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies. Item No: vmw-wp-temp-word 2/19

vmware