Running Redis on VMware Tanzu

Kubernetes Grid 2

vmware

Table of Contents

Overview

Introduction

Redis is the open source in-memory data structure store used by millions of developers as a database, cache, streaming engine, and message broker. Redis has built-in replication, transactions and various levels of on-disk persistence and provides high availability and automatic partitioning with Redis Cluster. Redis Enterprise Software is a self-managed data platform that unlocks the full potential of Redis at enterprise scale. Redis Enterprise for Kubernetes provides a simple way to get a Redis Enterprise Cluster on. It leverages the Redis Kubernetes Operator to reliably manage application in an extensible, modular way while automating the deployment of clusters and databases.

This paper provides the guidelines of Redis Enterprise Software for Kubernetes on VMware Tanzu® Kubernetes Grid™ 2. And it covers the generic functional, performance testing, and use case demonstration.

Prerequisites

You must firstly set up a reference architecture environment for Tanzu Kubernetes Grid 2. And then, the material takes you through the steps to install Redis Enterprise Software for Kubernetes.

Technology Component

The technology components in this solution are:

- VMware Tanzu Kubernetes Grid 2.0
- Redis Enterprise Software
- Benchmark tools

VMware Tanzu Kubernetes Grid 2

VMware Tanzu Kubernetes Grid provides organizations with a consistent, upstream-compatible Kubernetes substrate that is ready for end-user workloads and ecosystem integrations. It uses a new API called ClusterClass that defines a common cluster configuration for different infrastructure providers. Tanzu Kubernetes Grid 2 deploys clusters using an opinionated configuration of Kubernetes open-source software that is supported by VMware, so that you do not have to build a Kubernetes environment by yourself, it also provides packaged services such as networking, authentication, ingress control, and logging that are required for production Kubernetes environments.

Tanzu Kubernetes Grid 2 supports two types of deployment models: supervisor deployment and standalone management cluster deployment. Supervisor deployment allows you to create and operate workload clusters natively in vSphere with Tanzu and leverage many vSphere features. Standalone management cluster deployment allows you to create workload clusters on vSphere 6.7, 7, and 8 without supervisor, or on AWS.

Redis Enterprise Software

Redis is a popular NoSQL database and an in-memory data store supporting multiple abstract data structures. These structures include strings, lists, hashes, sets, and streams. Redis Enterprise is a self-managed and enterprise-grade version of Redis. It maintains the simplicity and high performance of Redis, while adding many enterprise-grade capabilities including:

- Linear scalability
- High availability
- Geo-replicated, active-active data distribution
- Predictable performance

Redis Enterprise Software is the on-premises distribution of Redis Enterprise. You can deploy and manage a Redis Enterprise Software cluster wherever you like.

A Redis cluster is a set of Redis instances that automatically shards data across nodes. Using a cluster gives users the ability to split their datasets among nodes and keep running the database even when some nodes fail.

Benchmark Tools

Redis-benchmark: It is a quick way to get some figures and evaluate the performance of a Redis instance on a given hardware. It simulates running by *N* clients while at the same time sending *M* total queries. Redis supports pipelining, so it is possible to send multiple commands at once, a feature often exploited by real-world applications.

Memtier_benchmark: It is a command line utility for load generation and benchmark NoSQL key-value databases. It supports both Redis and Memcache protocols, read-write ratio, random payload, random or ranged key expiration and Redis cluster mode.

Solution Configuration

Architecture

In our solution, we deployed Redis Enterprise Software for Kubernetes test environment using Tanzu Kubernetes Grid on a four-node vSphere cluster.

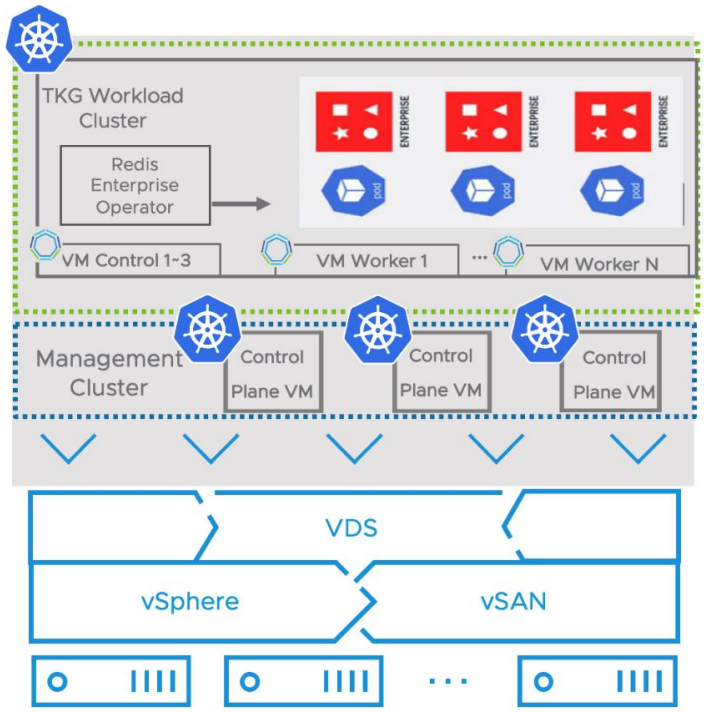

Figure 1. Redis Enterprise Software Running on Tanzu Kubernetes Grid

Tanzu Kubernetes Grid and Redis Configuration

We performed the following procedures for Tanzu Kubernetes Grid and Redis configuration:

1. We deployed the Tanzu management cluster with Tanzu CLI by an installer UI, you can also deploy the cluster with a configuration file. Refer to deploying Tanzu standalone management clusters for more details.

2. After deploying a Tanzu Kubernetes Grid management cluster in vSphere, we used the Tanzu CLI to deploy the Tanzu Kubernetes Grid workload cluster. You could deploy a class-based Tanzu workload cluster from a cluster configuration file, or you can also deploy a plan-based cluster or Tanzu Kubernetes Grid Cluster. Refer to Create Workload Clusters for more details. To deploy a Redis Cluster on Tanzu Kubernetes Grid, we recommend configuring a Tanzu Kubernetes Grid workload cluster with at least three worker nodes; each node is requested with a minimum value of 24 vCPU, 72GB memory, and 2TB disk space.

3. Then we deployed a Redis cluster to the Tanzu Kubernetes Grid workload cluster. You can either use Bitnami's Redis Helm Chart or Redis Enterprise Operator to deploy the Redis cluster. The operator allows Redis to deploy and manage the lifecycle of a Redis Enterprise Cluster in customized options. Each Redis Enterprise Database can also scale horizontally across multiple Tanzu Kubernetes Grid cluster worker nodes for sharding or replication, which ensures high availability with Redis Enterprise replicas.

   Sharding is a type of database partitioning that separates large databases into smaller, faster, and more easily managed parts. To create a sharded cluster, you need to first specify the number of shards, and data will automatically be sharded or be divided into groups and placed on optimal nodes.

   The Redis Enterprise Cluster node can be deployed in one of the following four types:

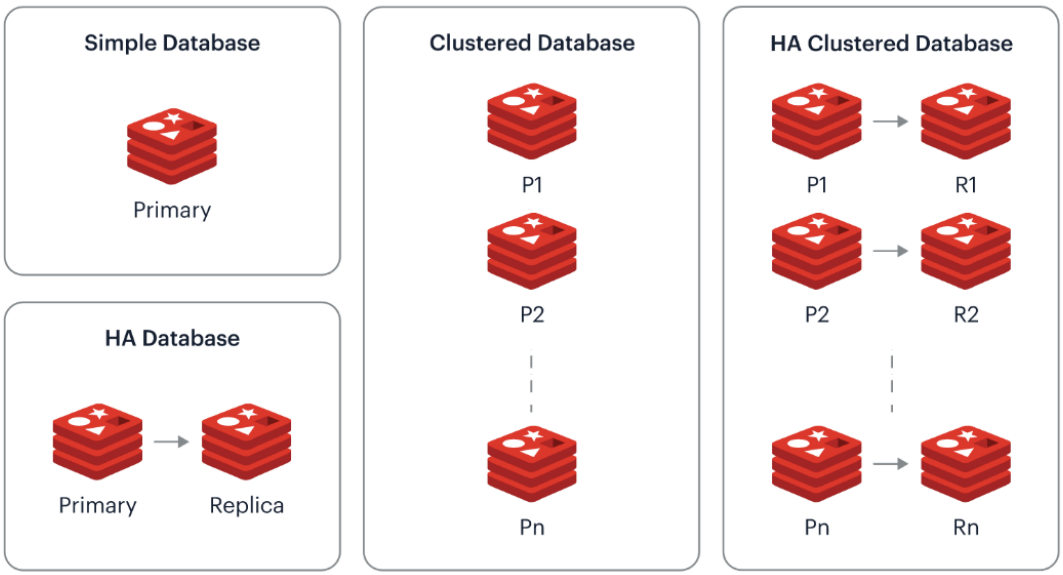

Figure 2. Redis Architectures

- Simple database: one single primary shard
- HA (High Availability) database: one pair of primary and replica shards
- Clustered database: multiple primary shards, each managing a subset of the dataset
- HA clustered database: multiple pairs of primary/replica shards

You could also use Bitnami to deploy Redis on Kubernetes. Bitnami supports two deployment options for Redis: HA clustered database and a Clustered database.

- If your workload demands high read throughput, using the HA cluster helps offload the read operations to the

replicas. In a failure case, it can promote a replica to primary.

- Redis cluster also supports high availability with each primary connected to one or more replicas. When the primary node crashes, one of the replicas will be promoted to the primary node.

In the production environment, you need to pay attention to the persistence to disk. It provides two persistence options:

- Redis Database (RDB): point-in-time snapshots.
- Append only File (AOF): logs of each Redis operation.

It is also possible to combine both options above, but it is important to understand the tradeoffs of the two options.

RDB is a compact snapshot optimized for a typical backup operation. RDB backup operation has minimal impact on Redis performance. In disaster recovery scenarios, RDB boots up faster than AOF because the file size is smaller. However, because RDB is a point-in-time snapshot, it will lose the updated or new created data after the last taken snapshot if failure occurs.

AOF keeps a log of each operation and is more durable than RDB as it can be configured to fancy on every second or write operation. You can have different fsync policies: no fsync at all, fsync every second, fsync at every query. With the default policy of fsync every second, write performance is still great. fsync is performed using a background thread. In the event of an outage, AOF can run through the log and replay each operation. Redis can also automatically and safely rewrite the AOF in the background if it gets too big. The tradeoff of AOF is file size and speed. With the replication turned on, sometimes the replicas cannot sync with the primary node fast enough to retrieve all the data. AOF can also be much slower than RDB with the fsync policy. Redis Helm Chart enables AOF and disables RDB by default.

Table 1. Comparison between RDB and AOF

	APPEND-ONLY FILE (AOF)	SNAPSHOT (RDB)
Resource Intensity	Higher	Lower
Recovery Time	Longer	Shorter
Capacity Requirement	Higher	Lower
Durability	Longer	Shorter

Hardware Resource

We validated the solution with a four-node vSphere cluster. Table 1 shows the specification of each node.

Table 2. 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 3. Software Resources

SOFTWARE	VERSION	PURPOSE
VMware vSphere	8.0	VMware vSphere is a suite of products: vCenter Server and ESXi.
Redis Enterprise Software	6.4.2	Redis is a popular NoSQL database and an in-memory data store supporting multiple abstract data structures. Redis Enterprise is a self-managed and enterprise-grade version of Redis.
Tanzu CLI	V0.28.0	Command line interface that allows deploying CNCF conformant Kubernetes clusters to vSphere and other cloud infrastructure.
Kubernetes	V1.24.9	Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.
Kubernetes OVA for Tanzu Kubernetes Grid	Photon v1.24.9+vmware.1	A base image template for the Kubernetes operating system of Tanzu Kubernetes Grid management and workload clusters.

VM Configuration

Table 4. 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

Networking Configuration

Tanzu Kubernetes Grid is configured with Virtual Distributed Scheduler (VDS). MetalLB (see installation instructions) provides load balancing for external services.

Application Deployment and Validation

This solution testing highlights the deployment, performance, and high availability use cases with Redis Enterprise Software running on Tanzu Kubernetes Grid 2.

Redis Enterprise Cluster Deployment

You can either use Bitnami's Redis Helm Chart or Redis Enterprise Operator to deploy Redis on the Tanzu Kubernetes Grid workload cluster. Redis provides a Kubernetes Operator that deploys and manages a Redis Enterprise Cluster. Operator allows Redis to deploy and manage the lifecycle of a Redis Enterprise Cluster in customized options.

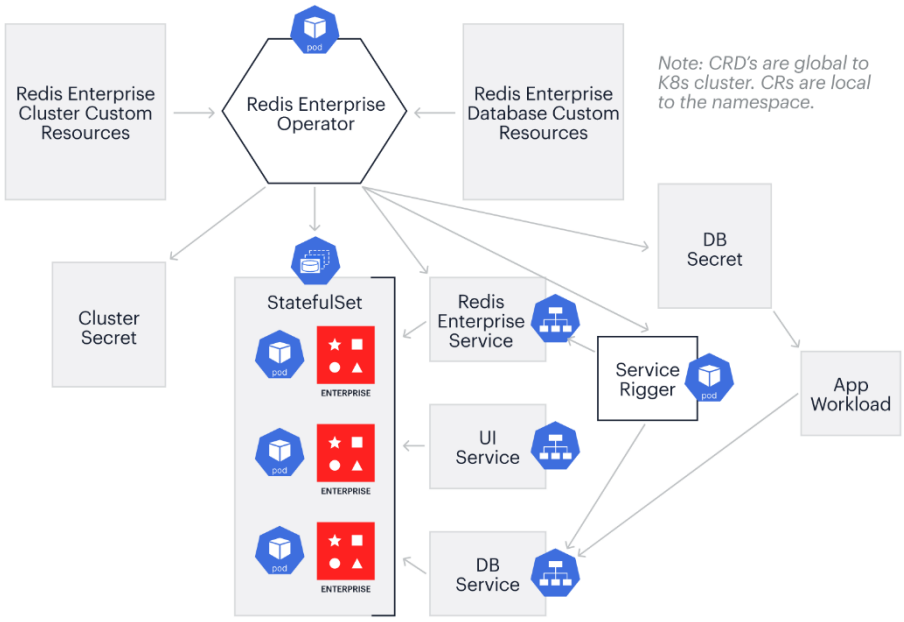

Figure 3. Redis Cluster Deployment with Redis Enterprise Operator

Perform the following steps for Redis Enterprise Cluster deployment:

1. Deploy the operator bundle in your REC namespace:

   kubectl apply -f https://raw.githubusercontent.com/RedisLabs/redis-enterprise-k8s-docs/$VERSION/bundle.yaml

2. Check the operator deployment to verify whether it is running:

   kubectl get deployment redis-enterprise-operator

3. Create a Redis Enterprise Cluster. To scale up node compute resource (the default value of CPU and memory is 2 cores and 4GB), you can add node resources to the spec section of the cluster. Redis Enterprise Software supports

using Redis on Flash, which extends node memory to use both RAM and flash storage. SSDs (solid state drives) can store infrequently used (warm) values while your keys and frequently used (hot) values are still stored in RAM. This improves performance and lowers costs for large datasets. Before creating Redis cluster or database, you should provision the flash storage as local persistent volumes with a unique name StorageClass. For more information, see Use Redis on Flash on Kubernetes. To deploy a Redis Enterprise Cluster with Redis Enterprise Operator, the spec should include a *persistentSpec* section. Persistence storage is a requirement for production deployments, refer to the YAML file for more information.

4. Create Redis Enterprise Database with the customized resource. You could also configure the memory size and the replica of a database. Wait for the status of database to be active and the database is ready to use then.

Redis Enterprise Cluster Performance Validation

We used built-in redis-benchmark and memtier_benchmark tool to generate loads for Redis. In the performance testing, we provisioned a cluster of three Redis instances, each instance was configured with 4 vCPUs and 24GB of RAM.

Scenario 1: We used redis-benchmark to compare Redis Enterprise Database with 1/2 sharding with or without replicas. We performed the following command:

```
redislabs@rec-0:/etc/opt/redislabs$ redis-benchmark -h redb -p 10377 -a xxx -q -n 10000000 -c 100 -P 50 -r 100000000000 set key:rand_int rand_int
```

The command options created 100 parallel connections to run 10,000,000 SET requests using random keys to the Redis server. The pipeline size was set to 50.

Table 4. Performance with Redis Database Enabling or Disabling Shard and Replica

	Mode	Throughput	Latency
Case 1	Redis Clustering disabled	1095650.25 requests per second	P50=4.159 ms
Case 2	Redis Clustering enabled shard count=2	902771.56 requests per second	P50=2.767 ms
Case 3	Redis Clustering disabled Replicas count=2	1070663.75 requests per second	P50=4.159 ms

Comparing to case 1, we can see the throughput result in case 3 was a little bit lower because Redis automatically handles the part of data sync between the two instances. However, we got similar performance result with high availability function in case 3.

Comparing to case 1, we added shards using the online method in case 2, the latency was improved with a little degradation in throughput because scaling is a compute-intensive operation. Redis cluster resharding online process continues to serve requests through the scaling operation.

Scenario 2: We used the memtier_benchmark tool to operate a performance benchmark of Redis Enterprise Software. It is recommended to run memtier_benchmark on a separate node that is not part of the cluster being tested. If you run it on a cluster node, note that it affects the performance of both the cluster and the memtier_benchmark tool.

- Write-to-read ratios of 5:95 (5% write and 95% read).
- Use the object size of 1,000 bytes to simulate with 50 concurrent clients:
  - Database persistence is kept to default.
  - Replication is disabled.
  - One database instance is used, which is composed of 4 vCPUs and 24GB of RAM.

```
memtier_benchmark -s 100.96.3.15 -p 12250 -a redb -t 5 -c 10 -d 1000 --test-time 300 --run-count 3 --key-maximum=1000000 --command="set __key__ __data__" --command-ratio=5 --command="get __key__" --command-ratio=95 --hide-histogram
```

You would get the output as follows:

```
5         Threads
10        Connections per thread
300       Seconds

BEST RUN RESULTS
============================================
Type         Ops/sec       Latency       KB/sec
--------------------------------------------
Sets          2727.35       0.91500      2790.97
Gets         51813.59       0.91500     52769.01
Totals       54540.95       0.91500     55559.98

WORST RUN RESULTS
============================================
Type         Ops/sec       Latency       KB/sec
--------------------------------------------
Sets          2666.19       0.93400      2728.38
Gets         50649.64       0.93600     51583.58
Totals       53315.83       0.93600     54311.96

AGGREGATED AVERAGE RESULTS (3 runs)
============================================
Type         Ops/sec       Latency       KB/sec
--------------------------------------------
Sets          2688.83       0.92800      2751.55
Gets         51080.37       0.92833     52022.26
Totals       53769.20       0.92833     54773.81
```

You can observe the overall database CPU from the following metric diagram.

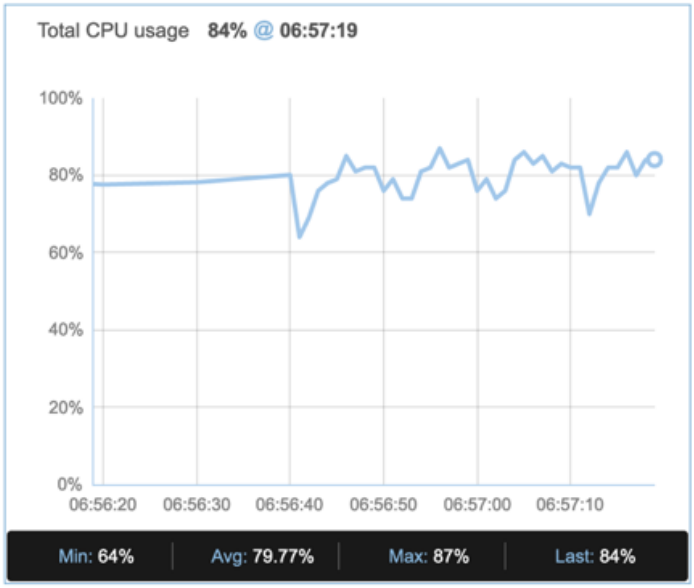

Figure 4. Redis Database CPU Usage Metric

Memtier_benchmark has several configuration options that can be tuned to better emulate the workload on your Redis server, in addition to offering cluster support. To simulate real-world environments, memtier_benchmark will test for GET and SET requests on a ratio of 5:95 or 1:10. This is more representative of a common web application using Redis as a database or cache. Thus, we recommend using the Memtier_benchmark tool for performance measuring in customer real cases.

Scenario 3: We tested the performance of a Redis on Flash (RoF) enabled database. Firstly, we configured a database instance with 4 vCPU and 24GB RAM to serve RoF data. RoF always keeps the Redis keys and Redis dictionary in RAM and the additional RAM is required for storing hot values. For these tests, 30% RAM was configured as an optimal value.

Table 5. Performance Results with RoF Enabled or Disabled

Database	SET Throughput	SET Latency	GET Throughput	GET Latency
RoF enabled	40854.18 requests per second	P50=6.447 ms	40107.49 requests per second	P50=6.551 ms
RoF disabled	39598.63 requests per second	P50=6.719 ms	40005.76 requests per second	P50=6.727 ms

Additionally, we compared the performance of an RoF enabled database with or without replication. We noticed an average throughput of:

- Around 255,497 ops/sec with the average latency of 1.9ms when testing without replication (for example, four primary shards)
- Around 188,226 ops/sec with the average latency of 2.65ms when testing with enabled replication (for example, four primary and two replica shards)

With replication enabled, your dataset is replicated to a replica shard, which is constantly synchronized with the primary shard. We can see from the results that with replication enabled, the throughput was lower and the latency was a bit higher. However, the data was still in normal state and the database replication ensured high availability.

Use Case Demonstration

In addition to the Redis performance validation, we also introduced the use case of integrating Redis into Kafka data pipelines by using the Kafka connector. This session demonstrates the use case scenario validation.

You can use the source connector to subscribe the Redis Pub/Sub records and write the messages to Kafka. And use the sink connector that consumes Kafka records in a Redis command format and applies them to Redis.

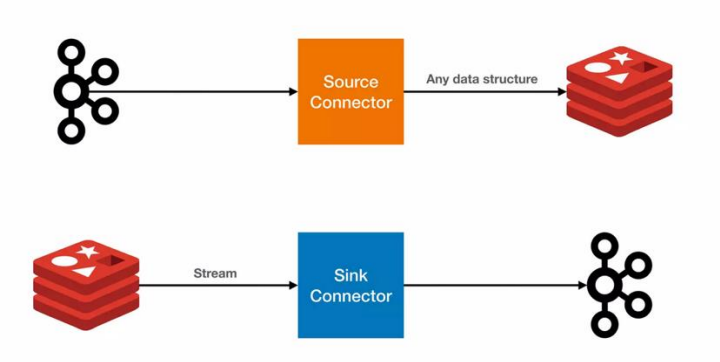

Figure 5. Kafka Connecting with Redis

Send a request to Kafka Connector API to configure it to use kafka-connect-redis. We would listen to all key space and key notifications in Redis.

- Source: Set up Kafka topic listener and create Redis events such as SET and SADD commands and from the Kafka client to catch up to the records that have already been there.
- Sink: Write records to Kafka, allowing you to use payload JSON format including schema and actual data, in a new terminal to confirm records in Redis.

We connected Kafka to Redis database with the following procedures:

1. We set up Kafka-connect-redis pods against Redis and Kafka cluster (they are on the same cluster) on Tanzu Kubernetes Grid by following the YAML file. We used the following JSON to sink-connector.json and used the curl command to post the configuration to one of the Kafka workers record serialization format to make a POST request against the Kafka Connect REST API.

   ```
   curl -X POST -H "content-type: application/json" -d @/Users/yimengl/sink-connector.json http://localhost:8083/connectors
   ```

   ```
   {"name":"demo-redis-sink-connector","config":{"connector.class":"io.github.kafkaconnectredis.sink.RedisSinkConnector","key.converter":"org.apache.kafka.connect.json.JsonConverter","value.converter":"org.apache.kafka.connect.json.JsonConverter","tasks.max":"1","topics":"redis.commands","redis.uri":"redis://IEPfIr0eLF7UsfwrIlzy80yUaBG258j9@redis-cluster","redis.cluster.enabled":"true","name":"demo-redis-sink-connector"},"tasks":[],"type":"sink"}%
   ```

2. After the connector for Redis was up and running, we wrote some records to Kafka and watched the data to be displayed in Redis. We opened the console producer and configured it to write records to the redis.commands topic:

   ```
   kafka-console-producer --bootstrap-server kafka-broker-0.kafka-broker:9092 --topic redis.commands
   ```

   We wrote a Redis SET command record with schema and wrote the actual data called “payload” in the JSON object:

   ```
   >{"payload":{"key":"{user.1}.username","value":"watermelon1"},"schema":{"name":"io.github.kafkaconnectredis.RedisSetCommand","type":"struct","fields":[{"field":"key","type":"string","optional":false},{"field":"value","type":"string","optional":false},{"field":"expiration","type":"struct","fields":[{"field":"type","type":"string","optional":false},{"field":"time","type":"int64","optional":true}],"optional":true},{"field":"condition","type":"string","optional":true}]}}
   ```

3. We ran a redis-client pod to connect to the Redis cluster to check whether the connector was working.

```
100.96.3.69:6379> GET {user.1}.username

-> Redirected to slot [9645] located at 100.96.1.36:6379

"watermelon1"
```

We can build a real-time inventory system using Kafka connector to seamlessly integrate to transform the data from the write-optimized to read-optimized data model and allow Redis Enterprise to serve it in real-time. Kafka acting as the real-time data pipeline, gathers and distributes events from several different sources such as the warehouse, the order management system, and the sales forecasting system. The pipeline provides information to the inventory manager.

Redis Enterprise is the in-memory database that enables real-time data access and maintains the inventory state with instant changes in the merchandise being tracked. The inventory status is then sent back to Kafka, who distributes this information to marketing, stores, and fulfillment. Together, Kafka and Redis Enterprise ensure the inventory is tracked and communicated in real time throughout the organization; they provide real-time data access with a real-time data pipeline.

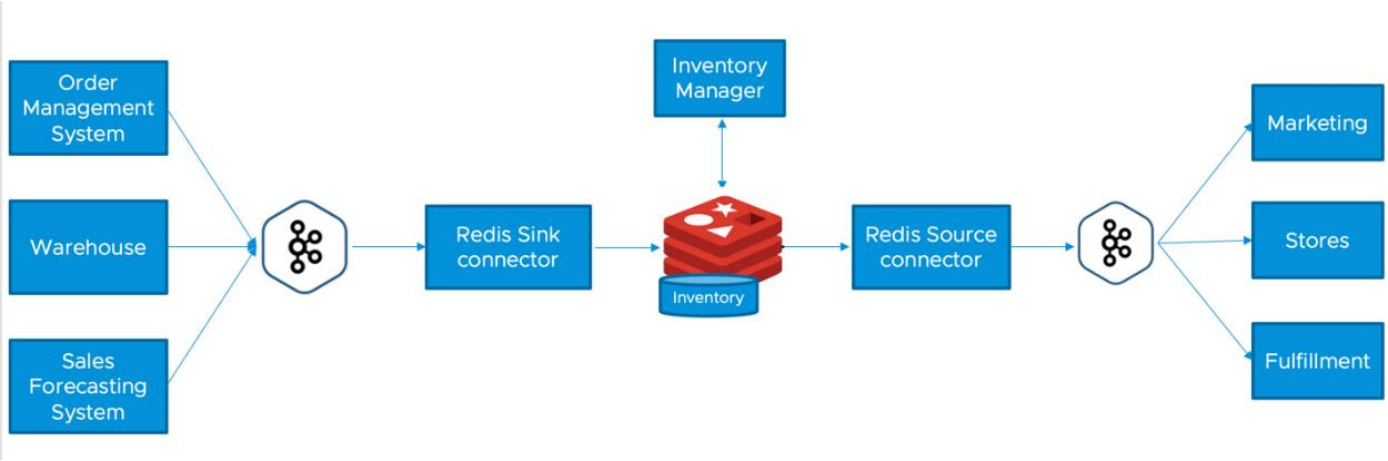

Figure 6. Real-time Inventory System via Kafka and Redis

High Availability Validation

Redis Enterprise Software comes with several features that make your data more durable and accessible, which can help protect your data in case of failures or outages. This session illustrates the Redis high availability scenario validation.

Replication in Redis follows a simple primary-replica model where the replication happens in one direction, from the primary to one or multiple replicas. Data is only written to the primary instance and replicas are kept in sync. To create a database replica, you can configure in the *replicaSources* section of the RDB.

```
apiVersion: v1

kind: Secret
```

```
metadata:
  name: my-replica-source
stringData:
  url: replica-source-url # "UI > database > configuration > Press the button Get Replica of source URL" in the administrative UI
```

Then create a database replica using the YAML file. After the replica database is active, the replica will try to sync with the primary database. Replicas are read-only, which means that you can configure your clients to read from them, but you cannot write data to them. If you need additional read throughput, you can configure your Redis client to read from replicas as well as from your primary node.

When a Redis Enterprise Cluster loses contact with more than half of its nodes either because of failed node or network split, the cluster stops responding to client connections. When this happens, you must recover the cluster to restore the connections. The Redis Enterprise for Kubernetes automates these recovery steps, but the cluster must be deployed with persistence. For the cluster recovery, you could set the *clusterRecovery* flag to true and wait for the cluster to be recovered until it is in the 'Running' state. And then restore a failed database from backup, AOF, or snapshot files.

You could create a snapshot of the data in source cluster and then restore it in a new cluster with the new parameters:

1. Copy the dump file from the Redis node for backup.
2. Before starting the restore process, to determine whether AOF is enabled. AOF tracks each write operation to the Redis database. Because we tried to restore from a point-in-time backup, we do not want Redis to re-create the operations stored in its AOF file. Modify the *appendonly* parameter to 'No'.
3. Start the new Redis cluster to mount the PVC with the recovery files to the new cluster nodes.
4. Recover the database using *rladmin* CLI.

An active-active architecture is a data resiliency architecture that distributes the database information over multiple data centers via independent and geographically distributed clusters and nodes. It is a network of separate processing nodes, each having access to a common replicated database so that all nodes can participate in a common application ensuring local low latency with each region being able to run in isolation. Redis Enterprise active-active databases provide read and write access to the same dataset from different Kubernetes clusters.

Clustering and replication are used together in active-active databases to distribute multiple copies of the dataset across multiple nodes and clusters. As a result, a node or cluster is less likely to become a single point of failure. If a primary node or a primary shard fails, a replica is automatically promoted to the primary mode. To avoid having one node holding all copies of certain data, the replica HA feature (enabled by default) automatically migrates replica shards to available nodes. In this case, there would be no data loss and the database is still accessible during the node or shard failure.

Best Practice

The following recommendations provide the best practices and sizing guidance to run Redis Enterprise Clusters on Tanzu Kubernetes Grid.

- Tanzu Kubernetes Grid:
  - Use the latest OVA image to create Tanzu Kubernetes Grid management and workload cluster that runs Redis Enterprise Cluster.
  - Customize and pre-allocate Performance Best Practices for Kubernetes with VMware Tanzu for sizing guidance for Tanzu Kubernetes Grid.
  - For multiple vSphere clusters managed all managed by the same vCenter server in a single datacenter, Tanzu Kubernetes clusters can be deployed applications across clusters using zones to achieve high availability for workloads.
- vSAN Storage:
  - vSAN supports the dynamic persistent volume provisioning with Storage Policy Based Management (SPBM).
  - For single vSAN cluster, we can deploy Redis without replication and the data resiliency can be protected by vSAN. vSphere Zone provide a way to create high-available TKG cluster. If you are provisioning a TKG cluster across vSphere Zones, for multiple vSAN clusters, we can deploy Redis with replication and set FTT=0 for vSAN, so the data will be protected by Redis itself.
  - Enable vSAN Trim/Unmap to allow space reclamation for persistent volumes.
- Redis Enterprise Software for Kubernetes:
  - It is recommended to deploy a Redis Enterprise Cluster with Redis Enterprise Operator, the spec should include a *persistentSpec* section. Persistence storage for snapshot and AOF files is a requirement for production deployments. We recommend that you use the flash-based storage for the persistent volume.
  - In a production environment, you must have enough resources to handle the load on the database and recover it from failures:
    i. At least three nodes are required to support a reliable, high available deployment that handles process failure and node failure. And it must be an odd number of nodes.
    ii. Redis Enterprise Software can run multiple Redis processes or shards on the same core without significant performance degradation. It is recommended to configure at least eight cores, 30GB RAM per node. Make sure the utilized CPU load of the cluster nodes is under 80% of the CPU, 30% of the RAM is available on each node at any given time.
    iii. It is recommended to run over 10Gb interface network used for processing application requests, inter-cluster communication, and storage access.
  - Database clustering allows spread the load over multiple servers. When the dataset is large enough (more than 25GB) or performing CPU-intensive operations, it is recommended to enable clustering to create multiple shards of the database and spread the requests and load across nodes.

- The location of primary and replica shards on the cluster nodes can affect the database and node performance. The shard placement policy helps to maintain optimal performance and resiliency:
    i. Dense shard placement policy is recommended for Redis on RAM databases to optimize memory resources.
    ii. Sparse shard placement policy is recommended for Redis on Flash databases to optimize disk resources.
- It is recommended that all active-active databases use replication for the best inter-cluster synchronization performance. In addition, you can enable replica HA to ensure that the replica shards are high available for this synchronization.

Conclusion

Being a successful open source in-memory data structure store, Redis is commonly used as a database, cache, and message broker. Developers use Redis for its versatility and simplicity. Low cognitive load makes development fast and efficiency. Redis is especially popular on cloud native platforms to achieve portability, scalability, and monitoring check.

Redis Enterprise Software integrates seamlessly on Tanzu Kubernetes Grid. This solution validates building and running real-time Redis workloads on Tanzu Kubernetes Grid, including solution deployment, performance validation, Redis connecting with Kafka use case and Redis high availability scenario. IT administrators can enable fast application deployment, achieve decent performance, ensure high availability governance, and lower TCO expenditure via this solution.

Additional Resources

For more information about Redis Enterprise Software and VMware Tanzu Kubernetes Grid, you can explore the following resources:

- VMware vSphere
- VMware vSAN
- VMware Tanzu Kubernetes Grid
- Redis Enterprise Software for Kubernetes

About the Author and Contributors

Yimeng Liu, Senior Technical Marketing 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:

- Chen Wei, Director of the Workload Technical Marketing Team in VMware
- Catherine Xu, Senior Manager of the Workload Technical Marketing Team in VMware

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com.

vmware®