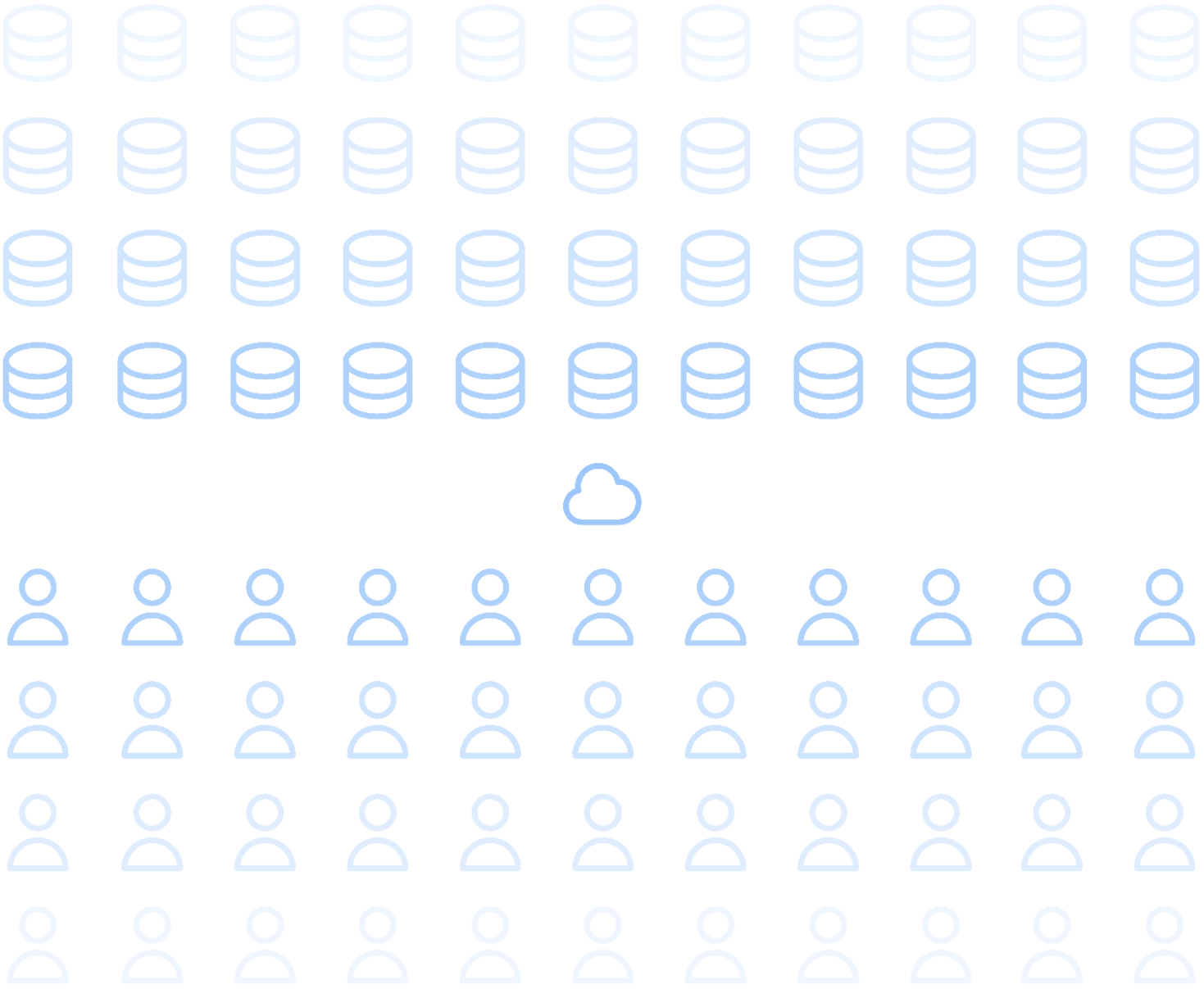
Multi-Tenancy and Isolation: Scaling Real-Time Data Across Teams with Apache Pulsar



StreamNative

Table of contents

Introduction to Pulsar isolation

In this eBook, we'll discuss three approaches for implementing isolation in Apache Pulsar, and when to use each approach. Before we dive into the details of Pulsar's isolation capabilities, let's look at why businesses benefit from isolation, isolation in monolithic and multi-tenant architectures, and how Pulsar provides multi-tenancy.

Why businesses benefit from isolation

Today's organizations manage large volumes of data, and often need to closely manage data access and isolate and control data storage when processing sensitive user data. For businesses in heavily regulated industries such as finance and healthcare, the ability to secure data in compliance with industry regulations is not only required, but doing so efficiently can be a competitive advantage. Other large organizations that have

multiple services or products must be able to isolate resources between those services to maintain SLAs during high-traffic periods.

Software systems that provide built-in options for resource isolation give organizations a flexible way to manage data access and meet security requirements efficiently, without needing to design and implement their own isolation mechanisms. The type of resource isolation an organizations implements will impact the level of data security that is achievable, as well as have implications on the data infrastructure and cost.

Businesses with strict requirements for data security and privacy can opt for full isolation of resources, while other organizations can choose to share resources for a simpler and more cost-effective solution.

Isolation in monolithic and multi-tenant architectures

The majority of existing streaming and messaging technologies have monolithic architectures (like Apache Kafka). In a monolithic architecture, the resources used for storage and compute are tightly coupled and must be scaled together. Mononlithic architectures also impact the options for how data and compute can be isolated, as the coupling of compute and storage is rigid.

The most practical option for any isolation in a monotholithic architecture is often full physical isolation, where a dedicated cluster of compute and storage resources must be assigned to each tenant. Since each tenant requires a dedicated cluster, organizations must manage multiple clusters. This multi-cluster approach can increase operational costs and complexity, and lead to unintentional data silos or data duplicates.

Conversely, decoupled compute and storage enable multi-tenant architectures and a range of isolation approaches, with isolation occuring at a logical or physical level. An organization can add new tenants without affecting existing tenant data security and leverage established cluster infrastructure resources. Flexible, decoupled compute and storage combined with multi-tenancy offer the lowest operational costs and complexity in addition to maximizing usage of cluster resources.

Isolation in Pulsar

Apache Pulsar is a cloud-native streaming and messaging platform that enables organizations to build scalable, reliable applications in elastic cloud environments. Pulsar has a multi-tenant architecture, giving organizations greater flexibility when it comes to achieving isolation.

In Pulsar's multi-layer architecture, each layer is scalable, distributed, and decoupled from the other layers. This architecture provides the ability to isolate tenants to different resource pools, including broker isolation and bookie isolation. Pulsar's decoupled multi-tenant architecture enables organizations to:

- Achieve different levels of isolation for individual teams.
- Securely deploy applications in a shared environment, ensuring better resource utilization across teams and simplifying infrastructural complexity.

Overview of Pulsar's built-in isolation features

Pulsar has built-in features to isolate workloads. Designing a cluster with either a shared or separate storage layer provides a wide range of isolation approaches. Let's briefly walk through those built-in isolation features and cluster design options.

Namespace isolation policies and broker groups
Namespace isolation policies ensure a namespace and all of the topics will run on brokers that match a given naming pattern, and other namespaces will avoid scheduling to these brokers. This means grouping brokers with a common naming pattern. In general, Pulsar will do this isolation as a best effort. If a group of brokers fails, the system will prioritize availability by moving these topics onto any available brokers.

Bookie affinity groups
Bookie affinity groups are analogous to namespace isolation policies, but apply to the bookie storage nodes and ensure all topic data in a given namespace is written to a particular set of bookies. If this affinity group becomes unavailable, other bookies will be used to store data.

Using bookie affinity groups and namespace isolation policies, cluster operators can provide strong logical isolation, with best-effort "soft" physical isolation. This contrasts with using Pulsar's architectural components to achieve a strong isolation guarantee.

Shared BookKeeper storage
Pulsar's decoupled compute and storage makes it possible to share BookKeeper storage across multiple Pulsar clusters. As a result, the user achieves hard physical isolation in the compute layer while allowing for efficient usage of storage resources.

Using multiple clusters with shared configuration store
Pulsar's built-in configuration store allows you to easily manage and configure an entire Pulsar Instance of multiple clusters. These clusters can share the same set of tenants

and namespaces, with tenants and namespaces able to replicate data between clusters or isolate to an individual cluster. This allows for teams to have multiple physical clusters that can be managed as a single global unit.

Three approaches to isolation in Pulsar

Pulsar offers three approaches for isolation to meet organizations' needs. We've categorized the approaches for isolation based on which resources an architecture shares: Shared Nothing, Shared Something, and Shared Everything.

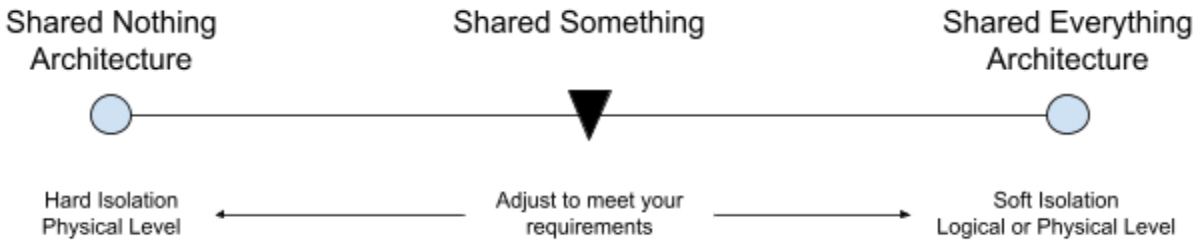


Figure 1.1 - Options for achieving isolation, dependent on which resources are shared

Below we will provide a brief overview of each, then dive further into the three approaches in the following chapters.

1. **Shared Everything: A shared bookie and brokers yield the lowest operational complexity.** Pulsar's built-in multi-tenancy provides separation of access and storage for each tenant at a logical level, with the option of best-effort physical isolation when using namespace isolation policies and bookie affinity groups. As with any shared resources, quotas must be set to ensure no single tenant negatively impacts another.

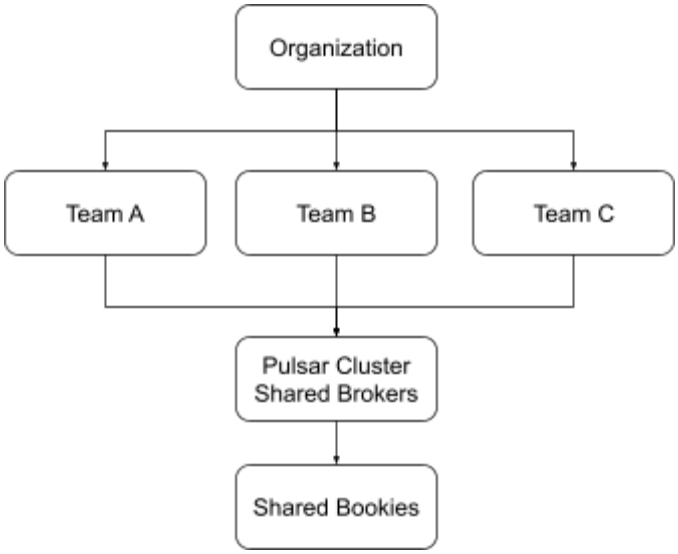


Figure 1.2 - Shared Everything: Multi-tenant option serves many teams with shared compute (brokers) and storage (BookKeeper) resource

2. **Shared Something: Shared bookies optimize storage allocation while providing separate brokers.** This option physically isolates end users from one another at the API level while leveraging a shared storage resource. Advantages of the Shared Something option include high utilization of storage and low operational complexity. Quotas must be set to ensure no single tenant negatively impacts another.

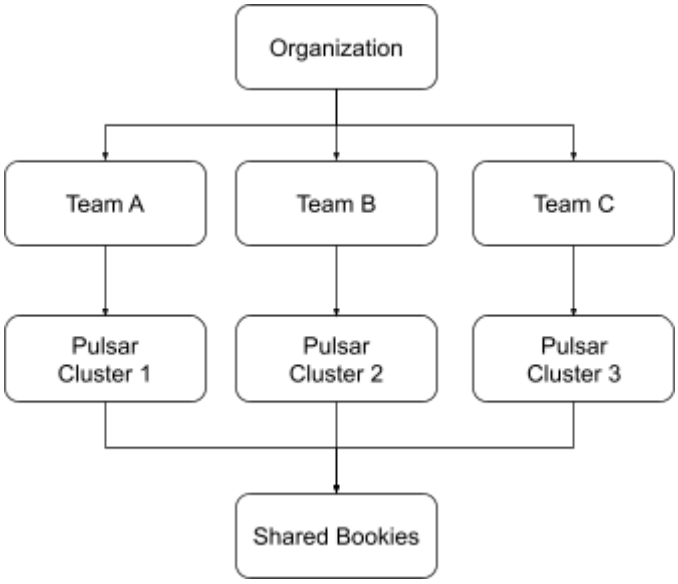


Figure 1.3 - Shared Something: Multi-tenant option serves one team per compute cluster (brokers) with shared storage (BookKeeper) resource

3. **Shared Nothing: Separate brokers with a dedicated BookKeeper offer the highest level of isolation.** The Shared Nothing option is ideal for storing highly sensitive data with physical isolation requirements at both the storage and compute tiers. No resources are shared between tenants, and individual Pulsar clusters can be optimized to meet specific processing needs. It is important to note that while the resources are isolated, the management of Pulsar objects (namespaces, topics, etc.) is done globally. This option guarantees resource availability, although it also increases operational complexity.

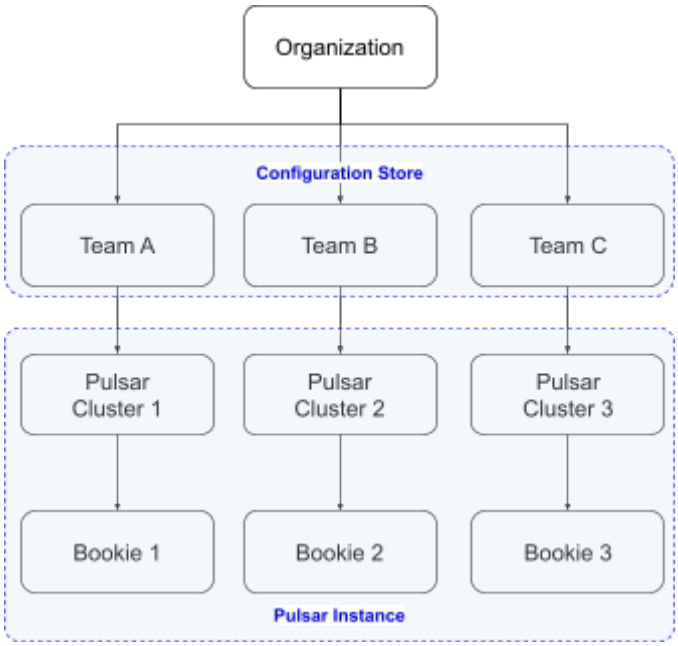


Figure 1.4 - Shared Nothing: Single-Tenant option serves one team per cluster and storage resource, with a dedicated broker and BookKeeper

Now that you understand the merits of each approach, let's compare them:

Isolation in Pulsar	**Shared Nothing** Nothing is shared	**Shared Something** Shared Bookies	**Shared Everything** Shared Broker & Bookies
Broker Compute	Separate	Separate	Shared
Bookies Storage	Separate	Shared	Shared
Broker Compute Utilization	⭐	⭐⭐	⭐⭐
Bookie Compute Utilization	⭐	⭐⭐⭐	⭐⭐
Bookie Storage Utilization	⭐	⭐⭐⭐	⭐⭐

Quota Guarantees	⭐⭐⭐	⭐⭐	⭐⭐
Operational Complexity	High	Medium	Low

Approach 1: Use a Shared Everything architecture to achieve isolation

This chapter details how to use a single cluster to achieve broker and bookie isolation, which is the most flexible approach to isolation in Pulsar. This more traditional approach takes advantage of Pulsar's built-in multi-tenancy and removes the need to manage multiple broker and bookie clusters. Organizations who use the Shared Everything approach benefit from high storage and compute utilization while minimizing operational complexity.

How the Shared Everything architecture works

Figure 2.1 demonstrates the deployment of a single Pulsar cluster to achieve isolation.

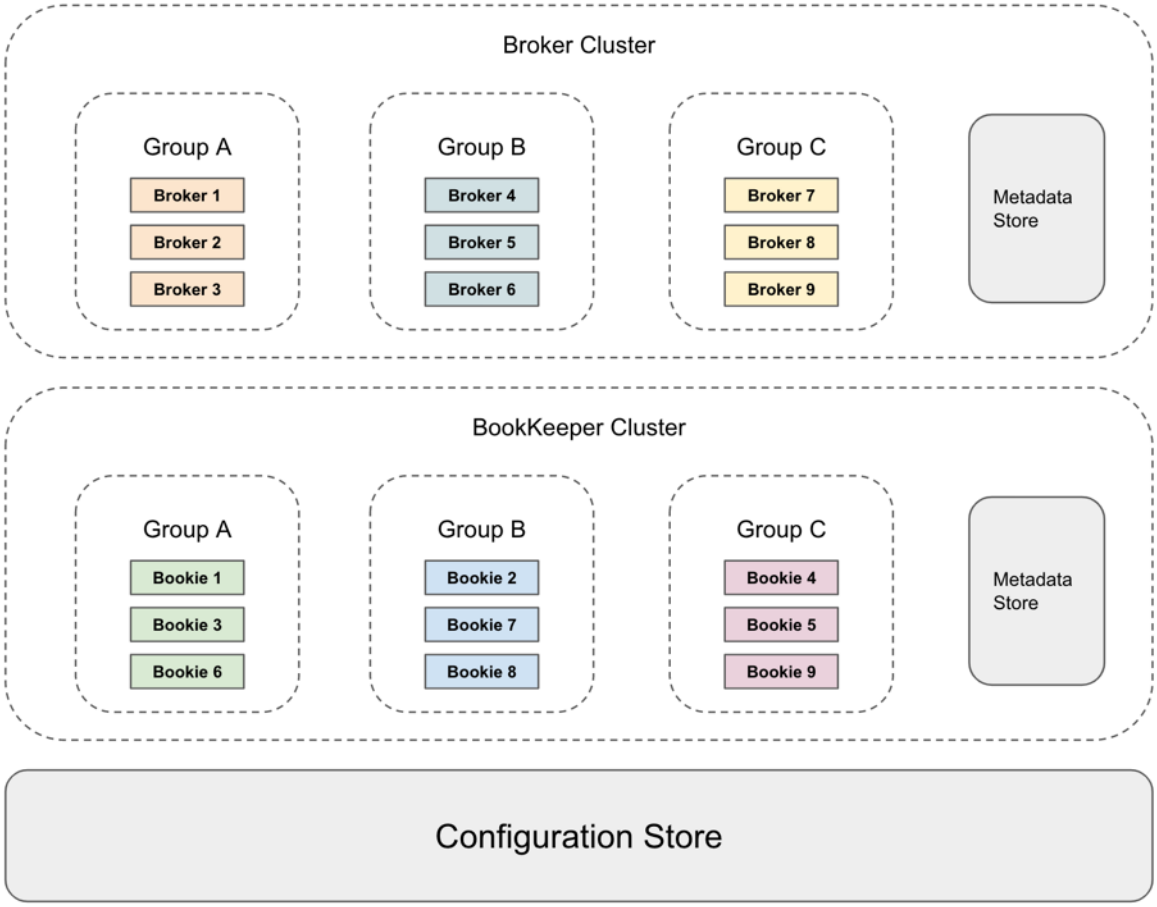


Figure 2.1 - Deployment of single Pulsar cluster

Here are some key points to understand how this deployment works:

- The Pulsar cluster exposes its service through a DNS entry point and makes sure a client can access the cluster through the DNS entry point. From the client side, the client can use the Pulsar URL that the Pulsar cluster exposes as the service URL.
- Broker isolation is achieved by different broker isolation groups (Pulsar assigns the topic to the broker under the specific broker isolation). For how to set broker isolation groups, see here.
- Storage isolation is achieved by different bookie affinity groups. For how to set bookie affinity groups, see here.

How to migrate a namespace

If you want to migrate the message service of the namespace to another broker cluster, you need to change the cluster for the namespace.

If you want to migrate the namespace to another bookie affinity group, you need to change the bookie affinity group. To learn how to set a bookie affinity group, see here. Since the BookKeeper cluster is a shared resource across all broker clusters, there is no need to copy data to another BookKeeper cluster.

Scale nodes up or down

Broker

The following key points should be considered when scaling brokers:

- Scaling up: Specify the broker isolation group for the newly added broker using the primary or secondary group.
- Scaling down: Make sure the broker isolation group has enough brokers.

Bookie

The following key points should be considered when scaling bookies:

- Scaling up: Specify the bookie affinity group for the newly added bookies.
- Scaling down: Make sure the bookie affinity group has enough bookies. For how to set bookie affinity groups, see here.

How to deploy a shared Pulsar cluster and a shared BookKeeper cluster

Below we provide a step-by-step tutorial on how to deploy a shared Pulsar cluster and a shared BookKeeper cluster.

Preparation

In this tutorial we use the `docker-compose` to establish a Pulsar cluster. First, we need to install the docker environment.

Note: This tutorial is based on docker `20.10.10`*, docker-compose* `1.29.2`*, and MacOS* `12.3.1`*.*

1. Get the docker-compose configuration files.
```
git clone https://github.com/gaoran10/pulsar-docker-compose
cd pulsar-docker-compose
```

2. Start the cluster.
```
docker-compose up
```

3. Check the pods.
```
docker-compose ps
   Name                  Command                State
Ports
-------------------------------------------------------------------
-----------------------------------------
bk1           bash -c export dbStorage_w ...   Up
bk2           bash -c export dbStorage_w ...   Up
bk3           bash -c export dbStorage_w ...   Up
bk4           bash -c export dbStorage_w ...   Up
broker1       bash -c bin/apply-config-f ...   Up
broker2       bash -c bin/apply-config-f ...   Up
broker3       bash -c bin/apply-config-f ...   Up
proxy1        bash -c bin/apply-config-f ...   Up
0.0.0.0:6650->6650/tcp, 0.0.0.0:8080->8080/tcp
pulsar-init   bin/init-cluster.sh              Exit 0
zk1           bash -c bin/apply-config-f ...   Up
```

After the cluster initiation completes, we can begin setting the broker isolation policy.

Broker isolation

1. Download a Pulsar release package to execute the pulsar-admin command.
```
wget
https://archive.apache.org/dist/pulsar/pulsar-2.10.0/apache-pul
sar-2.10.0-bin.tar.gz
tar -txvf apache-pulsar-2.10.0-bin.tar.gz
// we can execute pulsar-admin command in this directory
cd apache-pulsar-2.10.0
```

2. Get the broker list.
```
bin/pulsar-admin brokers list test
# output
"broker1:8080"
"broker2:8080"
"broker3:8080"
```

3. Create a namespace.
```
bin/pulsar-admin namespaces create public/ns-isolation
bin/pulsar-admin namespaces set-retention -s 1G -t 3d
public/ns-isolation
```

4. Set the namespace isolation policy.
```
bin/pulsar-admin ns-isolation-policy set \
--auto-failover-policy-type min_available \
--auto-failover-policy-params min_limit=1,usage_threshold=80 \
--namespaces public/ns-isolation \
--primary "broker1:*" \
--secondary "broker2:*" \
test ns-broker-isolation
```

5. Get the namespace isolation policies.
```
bin/pulsar-admin ns-isolation-policy list test
# output
ns-broker-isolation
NamespaceIsolationDataImpl(namespaces=[public/ns-isolation],
primary=[broker1:*], secondary=[broker2:*],
autoFailoverPolicy=AutoFailoverPolicyDataImpl(policyType=min_av
ailable, parameters={min_limit=1, usage_threshold=80}))
```

6. Create a partitioned topic.
```
bin/pulsar-admin topics create-partitioned-topic -p 10
public/ns-isolation/t1
```

7. Do a partitioned lookup.
```
bin/pulsar-admin topics partitioned-lookup
public/ns-isolation/t1
# output
persistent://public/ns-isolation/t1-partition-0
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-1
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-2
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-3
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-4
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-5
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-6
```

```
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-7
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-8
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-9
pulsar://broker1:6650
```

8. Stop broker1.

```
${DOCKER_COMPOSE_HOME}/docker-compose stop broker1
# output
Stopping broker1 ... done
```

9. Check the partitioned lookup.

   After broker1 stops, the topics will be owned by secondary broker `broker2:*`.

```
bin/pulsar-admin topics partitioned-lookup
public/ns-isolation/t1
# output
persistent://public/ns-isolation/t1-partition-0
pulsar://broker2:6650
persistent://public/ns-isolation/t1-partition-1
pulsar://broker2:6650
persistent://public/ns-isolation/t1-partition-2
pulsar://broker2:6650
persistent://public/ns-isolation/t1-partition-3
pulsar://broker2:6650
persistent://public/ns-isolation/t1-partition-4
pulsar://broker2:6650
persistent://public/ns-isolation/t1-partition-5
pulsar://broker2:6650
persistent://public/ns-isolation/t1-partition-6
pulsar://broker2:6650
persistent://public/ns-isolation/t1-partition-7
pulsar://broker2:6650
persistent://public/ns-isolation/t1-partition-8
pulsar://broker2:6650
persistent://public/ns-isolation/t1-partition-9
pulsar://broker2:6650
```

10. Stop broker2.

```
${DOCKER_COMPOSE_HOME}/docker-compose stop broker2
# output
```

```
Stopping broker2 ... done
```

11. Check the partitioned lookup.

    After stopping broker2, there are no available brokers for namespace `public/ns-isolation-broker`.

    ```
    bin/pulsar-admin topics partitioned-lookup
    public/ns-isolation/t1
    # output
    HTTP 503 Service Unavailable

    Reason: javax.ws.rs.ServiceUnavailableException: HTTP 503
    Service Unavailable
    ```

12. Restart broker1 and broker2.

    ```
    ${DOCKER_COMPOSE_HOME}/docker-compose start broker1
    # output
    Starting broker1 ... done

    ${DOCKER_COMPOSE_HOME}/docker-compose start broker2
    # output
    Starting broker2 ... done
    ```

Migrate the namespace between brokers

Because the Pulsar broker is stateless, we can migrate the namespace between broker groups by simply changing the namespace isolation policy.

1. Check the namespace isolation policies.

   ```
   bin/pulsar-admin ns-isolation-policy list test
   # output
   ns-broker-isolation
   NamespaceIsolationDataImpl(namespaces=[public/ns-isolation],
   primary=[broker1:*], secondary=[broker2:*],
   autoFailoverPolicy=AutoFailoverPolicyDataImpl(policyType=min_av
   ailable, parameters={min_limit=1, usage_threshold=80}))
   ```

   We could find that the primary and secondary brokers of the namespace `public/ns-isolation` are `broker1:*` and `broker2:*`.

2. Check the topic partitioned lookup results.

   ```
   bin/pulsar-admin topics partitioned-lookup
   public/ns-isolation/t1
   ```

```
# output
persistent://public/ns-isolation/t1-partition-0
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-1
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-2
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-3
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-4
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-5
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-6
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-7
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-8
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-9
pulsar://broker1:6650
```

3. Modify a new namespace isolation policy.

```
bin/pulsar-admin ns-isolation-policy set \
--auto-failover-policy-type min_available \
--auto-failover-policy-params min_limit=1,usage_threshold=80 \
--namespaces public/ns-isolation \
--primary "broker3:*" \
--secondary "broker2:*" \
test ns-broker-isolation
```

4. Check the namespace isolation policy.

```
bin/pulsar-admin ns-isolation-policy list test
# output
ns-broker-isolation
NamespaceIsolationDataImpl(namespaces=[public/ns-isolation],
primary=[broker3:*], secondary=[broker2:*],
autoFailoverPolicy=AutoFailoverPolicyDataImpl(policyType=min_av
ailable, parameters={min_limit=1, usage_threshold=80}))
```

5. Unload the namespace to make the namespace isolation policy take effect.

```
bin/pulsar-admin namespaces unload public/ns-isolation
```

6. Check the partitioned lookup.

   We could find that topics are already owned by the primary broker (broker3).

```
bin/pulsar-admin topics partitioned-lookup
public/ns-isolation/t1
# output
persistent://public/ns-isolation/t1-partition-0
pulsar://broker3:6650
persistent://public/ns-isolation/t1-partition-1
pulsar://broker3:6650
persistent://public/ns-isolation/t1-partition-2
pulsar://broker3:6650
persistent://public/ns-isolation/t1-partition-3
pulsar://broker3:6650
persistent://public/ns-isolation/t1-partition-4
pulsar://broker3:6650
persistent://public/ns-isolation/t1-partition-5
pulsar://broker3:6650
persistent://public/ns-isolation/t1-partition-6
pulsar://broker3:6650
persistent://public/ns-isolation/t1-partition-7
pulsar://broker3:6650
persistent://public/ns-isolation/t1-partition-8
pulsar://broker3:6650
persistent://public/ns-isolation/t1-partition-9
pulsar://broker3:6650
```

Scale up and down brokers

Scale up

1. Start broker4.

   Add broker4 configurations in the docker-compose file.

```
  broker4:
    hostname: broker4
    container_name: broker4
    image: apachepulsar/pulsar:latest
    restart: on-failure
    command: >
      bash -c "bin/apply-config-from-env.py conf/broker.conf &&
\
               bin/apply-config-from-env.py conf/pulsar_env.sh
```

```
&& \
               bin/watch-znode.py -z $$zookeeperServers -p
/initialized-$$clusterName -w && \
               exec bin/pulsar broker"
    environment:
      clusterName: test
      zookeeperServers: zk1:2181
      configurationStore: zk1:2181
      webSocketServiceEnabled: "false"
      functionsWorkerEnabled: "false"
      managedLedgerMaxEntriesPerLedger: 100
      managedLedgerMinLedgerRolloverTimeMinutes: 0
    volumes:
      -
./apply-config-from-env.py:/pulsar/bin/apply-config-from-env.py
    depends_on:
      - zk1
      - pulsar-init
      - bk1
      - bk2
      - bk3
      - bk4
    networks:
      pulsar:
```

Start broker4.

```
${DOCKER_COMPOSE_HOME}/docker-compose create
# output
zk1 is up-to-date
bk1 is up-to-date
bk2 is up-to-date
bk3 is up-to-date
broker1 is up-to-date
broker2 is up-to-date
broker3 is up-to-date
Creating broker4 ... done
proxy1 is up-to-date

${DOCKER_COMPOSE_HOME}/docker-compose start broker4
# output
Starting broker4 ... done
```

2. Check the broker list.

```
bin/pulsar-admin brokers list test
# output
```

```
broker4:8080
broker1:8080
broker2:8080
broker3:8080
```

3. Set a namespace isolation policy.

```
bin/pulsar-admin ns-isolation-policy set \
--auto-failover-policy-type min_available \
--auto-failover-policy-params min_limit=1,usage_threshold=80 \
--namespaces public/ns-isolation \
--primary "broker1:*,broker4:*" \
--secondary "broker2:*" \
test ns-broker-isolation
```

4. Get the namespace isolation policies.

```
bin/pulsar-admin ns-isolation-policy list test
# output
ns-broker-isolation
NamespaceIsolationDataImpl(namespaces=[public/ns-isolation],
primary=[broker1:*, broker4:*], secondary=[broker2:*],
autoFailoverPolicy=AutoFailoverPolicyDataImpl(policyType=min_av
ailable, parameters={min_limit=1, usage_threshold=80}))
```

5. Unload the namespace.

```
bin/pulsar-admin namespaces unload public/ns-isolation
```

6. Check the partitioned lookup.

   The topic should be owned by broker1 and broker4.

```
bin/pulsar-admin topics partitioned-lookup
public/ns-isolation/t1
# output
persistent://public/ns-isolation/t1-partition-0
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-1
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-2
pulsar://broker4:6650
persistent://public/ns-isolation/t1-partition-3
pulsar://broker4:6650
persistent://public/ns-isolation/t1-partition-4
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-5
pulsar://broker1:6650
```

```
persistent://public/ns-isolation/t1-partition-6
pulsar://broker4:6650
persistent://public/ns-isolation/t1-partition-7
pulsar://broker4:6650
persistent://public/ns-isolation/t1-partition-8
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-9
pulsar://broker1:6650
```

Scale down

1. Remove broker4 from the namespace isolation policy.
```
bin/pulsar-admin ns-isolation-policy set \
--auto-failover-policy-type min_available \
--auto-failover-policy-params min_limit=1,usage_threshold=80 \
--namespaces public/ns-isolation \
--primary "broker1:*" \
--secondary "broker2:*" \
test ns-broker-isolation
```

2. Check the namespace isolation policy.
```
bin/pulsar-admin ns-isolation-policy list test
# output
ns-broker-isolation
NamespaceIsolationDataImpl(namespaces=[public/ns-isolation],
primary=[broker1:*], secondary=[broker2:*],
autoFailoverPolicy=AutoFailoverPolicyDataImpl(policyType=min_av
ailable, parameters={min_limit=1, usage_threshold=80}))
```

3. Stop broker4.
```
${DOCKER_COMPOSE_HOME}/docker-compose stop broker4
# output
Stopping broker4 ... done
```

4. Check the broker list.
```
bin/pulsar-admin brokers list test
# output
broker1:8080
broker2:8080
broker3:8080
```

5. Check the partitioned lookup.
```
bin/pulsar-admin topics partitioned-lookup
public/ns-isolation/t1
# output
```

```
persistent://public/ns-isolation/t1-partition-0
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-1
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-2
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-3
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-4
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-5
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-6
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-7
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-8
pulsar://broker1:6650
persistent://public/ns-isolation/t1-partition-9
pulsar://broker1:6650
```

BookKeeper isolation

1. Get the bookie list.

```
bin/pulsar-admin bookies list-bookies
# output
{
  "bookies" : [ {
    "bookieId" : "bk2:3181"
  }, {
    "bookieId" : "bk4:3181"
  }, {
    "bookieId" : "bk3:3181"
  }, {
    "bookieId" : "bk1:3181"
  } ]
}
```

2. Set the bookie rack.

   The default value of the configuration `bookkeeperClientRackawarePolicyEnabled` is true, so the `RackawareEnsemblePlacementPolicy` is the default bookie isolation policy, we'll set the rack name like this `/rack`.

```
bin/pulsar-admin bookies set-bookie-rack \
--bookie bk1:3181 \
```

```
--hostname bk1:3181 \
--group group1 \
--rack /rack1

bin/pulsar-admin bookies set-bookie-rack \
--bookie bk3:3181 \
--hostname bk3:3181 \
--group group1 \
--rack /rack1

bin/pulsar-admin bookies set-bookie-rack \
--bookie bk2:3181 \
--hostname bk2:3181 \
--group group2 \
--rack /rack2

bin/pulsar-admin bookies set-bookie-rack \
--bookie bk4:3181 \
--hostname bk4:3181 \
--group group2 \
--rack /rack2
```

3. Check the bookie racks placement.

```
bin/pulsar-admin bookies racks-placement
group1    {bk1:3181=BookieInfoImpl(rack=/rack1,
hostname=bk1:3181), bk3:3181=BookieInfoImpl(rack=/rack1,
hostname=bk3:3181)}
group2    {bk2:3181=BookieInfoImpl(rack=/rack2,
hostname=bk2:3181), bk4:3181=BookieInfoImpl(rack=/rack2,
hostname=bk4:3181)}
```

4. Set the bookie affinity group for the namespace.

```
bin/pulsar-admin namespaces set-bookie-affinity-group
public/ns-isolation \
--primary-group group1 \
--secondary-group group2
```

5. Check the namespace affinity group.

```
bin/pulsar-admin namespaces get-bookie-affinity-group
public/ns-isolation
{
  "bookkeeperAffinityGroupPrimary" : "group1",
  "bookkeeperAffinityGroupSecondary" : "group2"
```

```
}
```

6. Produce messages to the topic.
```
bin/pulsar-client produce -m 'hello' -n 500
public/ns-isolation/t2
```

7. Get internal stats of the topic.
```
bin/pulsar-admin topics stats-internal public/ns-isolation/t2 |
grep ledgerId | tail -n 6
    "ledgerId" : 0,
    "ledgerId" : 1,
    "ledgerId" : 2,
    "ledgerId" : 3,
    "ledgerId" : 4,
    "ledgerId" : -1,
```

8. Check ledger ensembles for the ledgers [0, 1, 2, 3, 4].
```
# execute these commands in the node bk1
${DOCKER_COMPOSE_HOME}/docker-compose exec bk1 /bin/bash

bin/bookkeeper shell ledgermetadata -ledgerid 0
# check ensembles
ensembles={0=[bk1:3181, bk3:3181]}

bin/bookkeeper shell ledgermetadata -ledgerid 1
# check ensembles
ensembles={0=[bk3:3181, bk1:3181]}

bin/bookkeeper shell ledgermetadata -ledgerid 2
# check ensembles
ensembles={0=[bk1:3181, bk3:3181]}

bin/bookkeeper shell ledgermetadata -ledgerid 3
# check ensembles
ensembles={0=[bk1:3181, bk3:3181]}

bin/bookkeeper shell ledgermetadata -ledgerid 4
# check ensembles
ensembles={0=[bk1:3181, bk3:3181]}
```

9. Stop bookie1.
```
${DOCKER_COMPOSE_HOME}/docker-compose stop bk1
```

10. Produce messages to the topic.

```
bin/pulsar-client produce -m 'hello' -n 500
public/ns-isolation/t2
```

11. Check ledger metadata.

```
bin/pulsar-admin topics stats-internal public/ns-isolation/t2 |
grep ledgerId | tail -n 6
    "ledgerId" : 5,
    "ledgerId" : 6,
    "ledgerId" : 7,
    "ledgerId" : 8,
    "ledgerId" : 9,
    "ledgerId" : -1,
```

Check ledger metadata for the newly added ledgers [5,6,7,8,9]. Because bookie1 is not usable and the configuration `bookkeeperClientEnforceMinNumRacksPerWriteQuorum` is false, we should find that the secondary bookies are used. Bookie3 is in the primary group so bookie3 is always used.

```
# execute these commands in the node bk2
${DOCKER_COMPOSE_HOME}/docker-compose exec bk2 /bin/bash

bin/bookkeeper shell ledgermetadata -ledgerid 5
# check ensembles
ensembles={0=[bk4:3181, bk3:3181]}

bin/bookkeeper shell ledgermetadata -ledgerid 6
# check ensembles
ensembles={0=[bk3:3181, bk2:3181]}

bin/bookkeeper shell ledgermetadata -ledgerid 7
# check ensembles
ensembles={0=[bk2:3181, bk3:3181]}

bin/bookkeeper shell ledgermetadata -ledgerid 8
# check ensembles
ensembles={0=[bk3:3181, bk2:3181]}

bin/bookkeeper shell ledgermetadata -ledgerid 9
# check ensembles
ensembles={0=[bk3:3181, bk2:3181]}
```

Restart bk1

```
${DOCKER_COMPOSE_HOME}/docker-compose start bk1
```

Migrate bookie affinity group

1. Check the bookie affinity group.
```
bin/pulsar-admin namespaces get-bookie-affinity-group
public/ns-isolation
{
  "bookkeeperAffinityGroupPrimary" : "group1",
  "bookkeeperAffinityGroupSecondary" : "group2"
}
```

2. Modify the bookie affinity group of the namespace.
```
bin/pulsar-admin namespaces set-bookie-affinity-group
public/ns-isolation \
--primary-group group2
```

3. Check the bookie affinity group.
```
bin/pulsar-admin namespaces get-bookie-affinity-group
public/ns-isolation
{
  "bookkeeperAffinityGroupPrimary" : "group2"
}
```

4. Unload the namespace.
```
bin/pulsar-admin namespaces unload public/ns-isolation
```

5. Produce messages.
```
bin/pulsar-client produce -m 'hello' -n 500
public/ns-isolation/t2
```

6. Check the ensemble's bookies for newly created ledgers.
```
bin/pulsar-admin topics stats-internal public/ns-isolation/t2 |
grep ledgerId | tail -n 6
    "ledgerId" : 12,
    "ledgerId" : 13,
    "ledgerId" : 14,
    "ledgerId" : 15,
    "ledgerId" : 16,
    "ledgerId" : -1,
```

7. Check ledger metadata for the newly added ledgers [12, 13, 14, 15, 16].
```
# execute these commands in the node bk2
```

```
${DOCKER_COMPOSE_HOME}/docker-compose exec bk2 /bin/bash

bin/bookkeeper shell ledgermetadata -ledgerid 12
# check ensembles
ensembles={0=[bk4:3181, bk2:3181]}

bin/bookkeeper shell ledgermetadata -ledgerid 13
# check ensembles
ensembles={0=[bk4:3181, bk2:3181]}

bin/bookkeeper shell ledgermetadata -ledgerid 14
# check ensembles
ensembles={0=[bk4:3181, bk2:3181]}

bin/bookkeeper shell ledgermetadata -ledgerid 15
# check ensembles
ensembles={0=[bk4:3181, bk2:3181]}

bin/bookkeeper shell ledgermetadata -ledgerid 16
# check ensembles
ensembles={0=[bk2:3181, bk4:3181]}
```

Scale up and down bookies

Scale up

1. Add the following configuration in the docker-compose file.

```
  bk5:
    hostname: bk5
    container_name: bk5
    image: apachepulsar/pulsar:latest
    command: >
      bash -c "export
dbStorage_writeCacheMaxSizeMb="${dbStorage_writeCacheMaxSizeMb:
-16}" && \
               export
dbStorage_readAheadCacheMaxSizeMb="${dbStorage_readAheadCacheMa
xSizeMb:-16}" && \
               bin/apply-config-from-env.py
conf/bookkeeper.conf && \
               bin/apply-config-from-env.py conf/pulsar_env.sh
&& \
               bin/watch-znode.py -z $$zkServers -p
```

```
/initialized-$$clusterName -w && \
                exec bin/pulsar bookie"
    environment:
      clusterName: test
      zkServers: zk1:2181
      numAddWorkerThreads: 8
      useHostNameAsBookieID: "true"
    volumes:
      -
./apply-config-from-env.py:/pulsar/bin/apply-config-from-env.py
    depends_on:
      - zk1
      - pulsar-init
    networks:
      pulsar:
```

2. Start bookie5.

```
${DOCKER_COMPOSE_HOME}/docker-compose create
${DOCKER_COMPOSE_HOME}/docker-compose start bk5
```

3. Check the readable and writable bookie list. Because bookie1 is stopped, there should be 4 bookies.

```
# execute this command in bk2
${DOCKER_COMPOSE_HOME}/docker-compose exec bk2 bin/bookkeeper
shell listbookies -rw

ReadWrite Bookies :
BookieID:bk2:3181, IP:192.168.32.5, Port:3181, Hostname:bk2
BookieID:bk4:3181, IP:192.168.32.7, Port:3181, Hostname:bk4
BookieID:bk3:3181, IP:192.168.32.6, Port:3181, Hostname:bk3
BookieID:bk1:3181, IP:192.168.32.4, Port:3181, Hostname:bk1
BookieID:bk5:3181, IP:192.168.32.9, Port:3181, Hostname:bk5
```

4. Add the newly added bookie node to the primary group.

```
bin/pulsar-admin bookies set-bookie-rack \
--bookie bk5:3181 \
--hostname bk5:3181 \
--group group2 \
--rack /rack2
```

5. Check the bookie racks placement.

```
bin/pulsar-admin bookies racks-placement
group1    {bk1:3181=BookieInfoImpl(rack=/rack1,
hostname=bk1:3181), bk3:3181=BookieInfoImpl(rack=/rack1,
```

```
hostname=bk3:3181)}
group2    {bk2:3181=BookieInfoImpl(rack=/rack2,
hostname=bk2:3181), bk4:3181=BookieInfoImpl(rack=/rack2,
hostname=bk4:3181), bk5:3181=BookieInfoImpl(rack=/rack2,
hostname=bk5:3181)}
```

6. Unload the namespace.
   ```
   bin/pulsar-admin namespaces unload public/ns-isolation
   ```

7. Produce messages to a new topic.
   ```
   bin/pulsar-client produce -m 'hello' -n 500
   public/ns-isolation/t2
   ```

8. Check the newly added ledger of the topic.
   ```
   bin/pulsar-admin topics stats-internal public/ns-isolation/t2 |
   grep ledgerId | tail -n 6
       "ledgerId" : 17,
       "ledgerId" : 20,
       "ledgerId" : 21,
       "ledgerId" : 22,
       "ledgerId" : 23,
       "ledgerId" : -1,
   ```

   Verify ledger ensembles, we could find that the new created ledgers are all wrote to primary group, because there are enough rw bookies.
   ```
   # execute these commands in the node bk1
   ${DOCKER_COMPOSE_HOME}/docker-compose exec bk2 /bin/bash

   bin/bookkeeper shell ledgermetadata -ledgerid 17
   # check ensembles
   ensembles={0=[bk5:3181, bk2:3181]}

   bin/bookkeeper shell ledgermetadata -ledgerid 20
   # check ensembles
   ensembles={0=[bk2:3181, bk4:3181]}

   bin/bookkeeper shell ledgermetadata -ledgerid 21
   # check ensembles
   ensembles={0=[bk5:3181, bk4:3181]}

   bin/bookkeeper shell ledgermetadata -ledgerid 22
   # check ensembles
   ensembles={0=[bk5:3181, bk4:3181]}
   ```

```
bin/bookkeeper shell ledgermetadata -ledgerid 23
# check ensembles
ensembles={0=[bk2:3181, bk4:3181]}
```

Scale down

1. Check the placement of the racks.
   ```
   bin/pulsar-admin bookies racks-placement
   group1    {bk1:3181=BookieInfoImpl(rack=/rack1,
   hostname=bk1:3181), bk3:3181=BookieInfoImpl(rack=/rack1,
   hostname=bk3:3181)}
   group2    {bk2:3181=BookieInfoImpl(rack=/rack2,
   hostname=bk2:3181), bk4:3181=BookieInfoImpl(rack=/rack2,
   hostname=bk4:3181), bk5:3181=BookieInfoImpl(rack=/rack2,
   hostname=bk5:3181)}
   ```

2. Delete the bookie from the affinity bookie group.
   ```
   bin/pulsar-admin bookies delete-bookie-rack -b bk5:3181
   ```

3. Check if there are under-replicated ledgers, which should be expected because we deleted a bookie.
   ```
   # execute these commands in the node bk2
   ${DOCKER_COMPOSE_HOME}/docker-compose exec bk2 bin/bookkeeper
   shell listunderreplicated
   ```

4. Stop the bookie.
   ```
   ${DOCKER_COMPOSE_HOME}/docker-compose stop bk5
   ```

5. Decommission the bookie.
   ```
   ${DOCKER_COMPOSE_HOME}/docker-compose exec bk2 bin/bookkeeper
   shell decommissionbookie -bookieid bk5:3181
   ```

6. Check ledgers in the decommissioned bookie.
   ```
   ${DOCKER_COMPOSE_HOME}/docker-compose exec bk2 bin/bookkeeper
   shell listledgers -bookieid bk5:3181
   ```

7. List the bookies.
   ```
   ${DOCKER_COMPOSE_HOME}/docker-compose exec bk2 bin/bookkeeper
   shell listbookies -rw
   ReadWrite Bookies :
   BookieID:bk2:3181, IP:192.168.48.5, Port:3181, Hostname:bk2
   BookieID:bk4:3181, IP:192.168.48.7, Port:3181, Hostname:bk4
   ```

```
BookieID:bk3:3181, IP:192.168.48.6, Port:3181, Hostname:bk3
BookieID:bk1:3181, IP:192.168.48.4, Port:3181, Hostname:bk1
```

Approach 2: Use a Shared Something architecture to achieve isolation

This chapter details how to achieve isolation with a shared BookKeeper cluster across multiple broker clusters. This approach isolates end users from one another while allowing customized authentication methods based on use case. With a shared resource, storage utilization and operation cost can be amortized.

How the Shared Something architecture works

Figure 3.1 demonstrates the deployment of a shared BookKeeper cluster to achieve isolation.

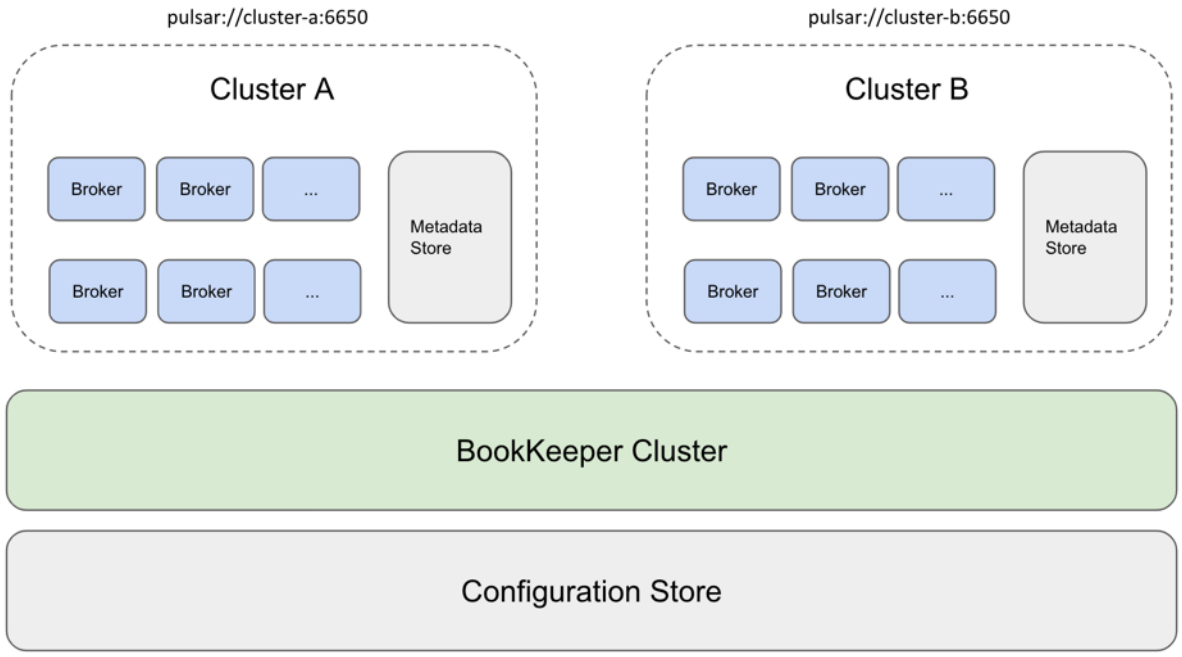


Figure 3.1 - Deployment of shared BookKeeper cluster

Here are some key points to understand how this deployment works:

- Each Pulsar cluster exposes its service through a DNS entry point and makes sure a client can access the cluster through the DNS entry point. From the client side, the client can use one or multiple Pulsar URLs that the Pulsar cluster exposes as the service URL.
- Each Pulsar cluster has one or multiple brokers.
- Each Pulsar cluster has one metadata store.
- Separate Pulsar clusters use a shared BookKeeper cluster.
- Pulsar's hierarchical resource management provides a solid foundation for isolation. In this approach, if you want to achieve namespace isolation, you need to specify a cluster for a namespace. The cluster must be in the allowed cluster list of the tenant. Topics under the namespace are assigned to this cluster. For how to set a cluster for a namespace, see here. For how to manage Pulsar clusters, see here.

As shown in Figure 3.2, the storage isolation is achieved by different bookie affinity groups:

- All bookie isolation groups use a shared BookKeeper cluster and a metadata store.
- Each bookie isolation group has one or several bookies.
- You can specify a primary or secondary group (one or several) for a namespace. Topics under the namespace are created on the bookies in the primary group firstly and then created on the bookies in the secondary group. For how to set bookie affinity groups, see here.

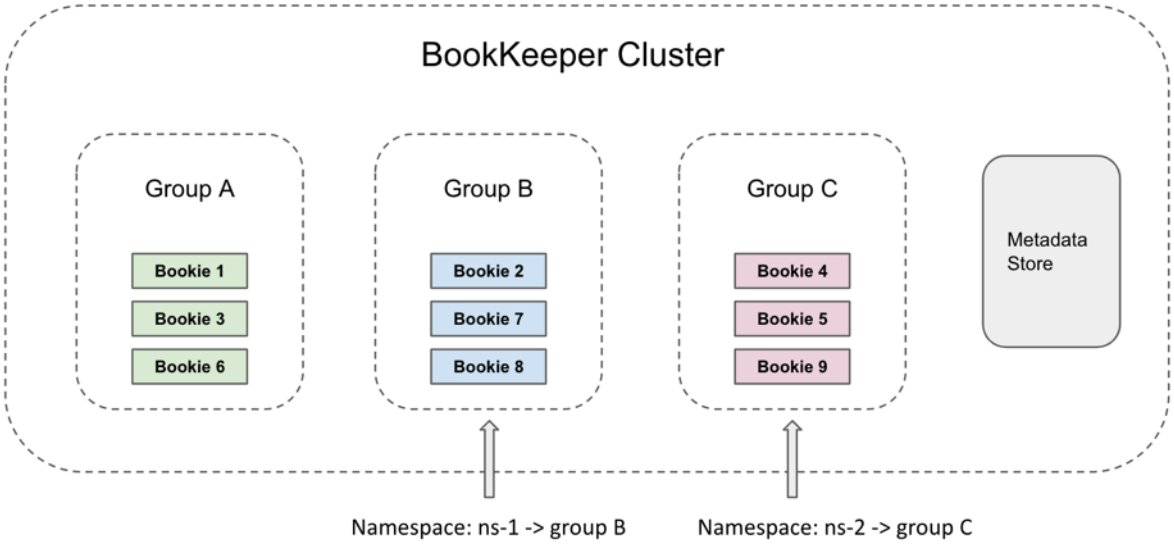


Figure 3.2 - Storage isolation is achieved by different bookie affinity groups

Note: See the previous chapter for details on how to migrate a namespace and what to consider when scaling nodes up or down.

How to deploy Pulsar clusters that share a BookKeeper cluster

We will deploy two Pulsar clusters that share one BookKeeper cluster following the steps below:

1. Set up the shared BookKeeper cluster
2. Deploy two Pulsar clusters that share one BookKeeper cluster
3. Verify isolation namespace by bookie affinity group
4. Scale up and down bookies

1. Set up the shared BookKeeper cluster

First, we set up the shared BookKeeper cluster on a computer that has an 8-core CPU and 16GB memory.

Note: All metadata services (ZooKeeper services) are single nodes. We don't discuss this in detail in this blog.

2. Deploy two Pulsar clusters that share one BookKeeper cluster

1. Download the latest binary Pulsar package. Currently, this would be the [`2.8.1` package].

2. Unzip the binary compression package.
   ```
   tar -zxvf apache-pulsar-2.8.1-bin.tar.gz
   ```

3. Prepare the following cluster directories. Change the configuration of each directory as instructed in the table below.

   Use the current directory as `PULSAR_HOME` and create the following topology of directories.
   ```
   cp -r apache-pulsar-2.8.1 configuration-store2
   mkdir -p bk-cluster
   cp -r apache-pulsar-2.8.1 bk-cluster/bk1
   cp -r apache-pulsar-2.8.1 bk-cluster/bk2
   ```

```
cp -r apache-pulsar-2.8.1 bk-cluster/bk3
cp -r apache-pulsar-2.8.1 bk-cluster/bk4
mkdir -p cluster1
cp -r apache-pulsar-2.8.1 cluster1/zk1
cp -r apache-pulsar-2.8.1 cluster1/broker1
mkdir -p cluster2
cp -r apache-pulsar-2.8.1 cluster2/zk1
cp -r apache-pulsar-2.8.1 cluster2/broker1
```

The directories' topology is outlined below.

- PULSAR_HOME
    - configuration-store

    - bk-cluster
        - bk1
        - bk2
        - bk3
        - bk4
        - bk5

    - cluster1
        - zk1
        - broker1

    - cluster2
        - zk1
        - Broker1

Component	Changed configurations
configuration-store	clientPort=2181 admin.serverPort=9991
bk-cluster/bk1	bookiePort=3181 allowLoopback=true zkServers=localhost:2181 httpServerPort=8011
bk-cluster/bk2	bookiePort=3182 allowLoopback=true zkServers=localhost:2181 httpServerPort=8012

bk-cluster/bk3	bookiePort=3183 allowLoopback=true zkServers=localhost:2181 httpServerPort=8013
bk-cluster/bk4	bookiePort=3184 allowLoopback=true zkServers=localhost:2181 httpServerPort=8014
bk-cluster/bk5	bookiePort=3185 allowLoopback=true zkServers=localhost:2181 httpServerPort=8015
cluster1/zk1	clientPort=2182 admin.serverPort=9992
cluster1/broker1	zookeeperServers=localhost:2182 configurationStoreServers=localhost:2181 brokerServicePort=6650 webServicePort=8080 bookkeeperMetadataServiceUri=zk://localhost:2181/ledgers managedLedgerMaxEntriesPerLedger=100 managedLedgerMinLedgerRolloverTimeMinutes=0
cluster2/zk1	clientPort=2183 admin.serverPort=9993
cluster2/broker1	zookeeperServers=localhost:2183 configurationStoreServers=localhost:2181 brokerServicePort=6651 webServicePort=8081 bookkeeperMetadataServiceUri=zk://localhost:2181/ledgers managedLedgerMaxEntriesPerLedger=100 managedLedgerMinLedgerRolloverTimeMinutes=0

4. Start and initialize the configuration store and the metadata store.

```
$PULSAR_HOME/configuration-store/bin/pulsar-daemon start configuration-store
$PULSAR_HOME/cluster1/zk1/bin/pulsar-daemon start zookeeper
$PULSAR_HOME/cluster2/zk1/bin/pulsar-daemon start zookeeper
```

```
$PULSAR_HOME/configuration-store/bin/pulsar
initialize-cluster-metadata \
--cluster cluster1 \
--zookeeper localhost:2182 \
--configuration-store localhost:2181 \
--web-service-url http://localhost:8080/ \
--broker-service-url pulsar://localhost:6650/

./configuration-store/bin/pulsar initialize-cluster-metadata \
--cluster cluster2 \
--zookeeper localhost:2183 \
--configuration-store localhost:2181 \
--web-service-url http://localhost:8081/ \
--broker-service-url pulsar://localhost:6651/
```

5. Initialize the BookKeeper metadata and start the bookie cluster.

```
$PULSAR_HOME/bk-cluster/bk1/bin/bookkeeper shell metaformat

$PULSAR_HOME/bk-cluster/bk1/bin/pulsar-daemon start bookie
$PULSAR_HOME/bk-cluster/bk2/bin/pulsar-daemon start bookie
$PULSAR_HOME/bk-cluster/bk3/bin/pulsar-daemon start bookie
$PULSAR_HOME/bk-cluster/bk4/bin/pulsar-daemon start bookie
```

6. Start brokers in cluster1 and cluster2.

```
$PULSAR_HOME/cluster1/broker1/bin/pulsar-daemon start broker
$PULSAR_HOME/cluster2/broker1/bin/pulsar-daemon start broker
```

7. Check brokers.

```
$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8080 brokers list cluster1
"localhost:8080"
$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8081 brokers list cluster2
"localhost:8081"
```

8. Check the bookie list for cluster1 and cluster2. As shown below, they share the bookie cluster.

```
$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8080 bookies list-bookies
{
  "bookies" : [ {
    "bookieId" : "127.0.0.1:3181"
  }, {
    "bookieId" : "127.0.0.1:3182"
```

```
  }, {
    "bookieId" : "127.0.0.1:3183"
  }, {
    "bookieId" : "127.0.0.1:3184"
  } ]
}
$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8081 bookies list-bookies
{
  "bookies" : [ {
    "bookieId" : "127.0.0.1:3181"
  }, {
    "bookieId" : "127.0.0.1:3182"
  }, {
    "bookieId" : "127.0.0.1:3183"
  }, {
    "bookieId" : "127.0.0.1:3184"
  } ]
}
```

Bookie Rack Placement

In order to archive resource isolation, we need to split the 4 bookie nodes into 2 resource groups.

1. Set the bookie rack for cluster1.

```
$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8080 bookies set-bookie-rack \
--bookie 127.0.0.1:3181 \
--hostname 127.0.0.1:3181 \
--group group-bookie1 \
--rack rack1

$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8080 bookies set-bookie-rack \
--bookie 127.0.0.1:3182 \
--hostname 127.0.0.1:3182 \
--group group-bookie1 \
--rack rack1

$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8080 bookies set-bookie-rack \
--bookie 127.0.0.1:3183 \
--hostname 127.0.0.1:3183 \
--group group-bookie2 \
```

```
--rack rack2

$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8080 bookies set-bookie-rack \
--bookie 127.0.0.1:3184 \
--hostname 127.0.0.1:3184 \
--group group-bookie2 \
--rack rack2
```

2. Check bookie racks placement in cluster1.

```
$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8080 bookies racks-placement
"group-bookie1    {127.0.0.1:3181=BookieInfoImpl(rack=rack1,
hostname=127.0.0.1:3181),
127.0.0.1:3182=BookieInfoImpl(rack=rack1,
hostname=127.0.0.1:3182)}"
"group-bookie2    {127.0.0.1:3183=BookieInfoImpl(rack=rack2,
hostname=127.0.0.1:3183),
127.0.0.1:3184=BookieInfoImpl(rack=rack2,
hostname=127.0.0.1:3184)}"
```

3. Set bookie racks for cluster2.

```
$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8081 bookies set-bookie-rack \
--bookie 127.0.0.1:3181 \
--hostname 127.0.0.1:3181 \
--group group-bookie1 \
--rack rack1

$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8081 bookies set-bookie-rack \
--bookie 127.0.0.1:3182 \
--hostname 127.0.0.1:3182 \
--group group-bookie1 \
--rack rack1

$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8081 bookies set-bookie-rack \
--bookie 127.0.0.1:3183 \
--hostname 127.0.0.1:3183 \
--group group-bookie2 \
--rack rack2

$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
```

```
http://localhost:8081 bookies set-bookie-rack \
--bookie 127.0.0.1:3184 \
--hostname 127.0.0.1:3184 \
--group group-bookie2 \
--rack rack2
```

4. Check bookie racks placement in cluster2.
```
$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8081 bookies racks-placement
"group-bookie1    {127.0.0.1:3181=BookieInfoImpl(rack=rack1,
hostname=127.0.0.1:3181),
127.0.0.1:3182=BookieInfoImpl(rack=rack1,
hostname=127.0.0.1:3182)}"
"group-bookie2    {127.0.0.1:3183=BookieInfoImpl(rack=rack2,
hostname=127.0.0.1:3183),
127.0.0.1:3184=BookieInfoImpl(rack=rack2,
hostname=127.0.0.1:3184)}"
```

3. Verify isolation namespace by bookie affinity group

Now that we have everything configured, let's verify namespace isolation by the bookie affinity group setting.

1. Create a namespace in cluster1.
```
$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8080 namespaces create -b 30 -c cluster1
public/c1-ns1
```

2. Set a bookie affinity group for the namespace.
```
$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8080 namespaces set-bookie-affinity-group
public/c1-ns1 \
--primary-group group-bookie1
```

3. Check the bookie affinity group of the namespace.
```
$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8080 namespaces get-bookie-affinity-group
public/c1-ns1
```

4. Produce some messages to a topic of the namespace `public/c1-ns1`.
```
# set retention for namespace `public/c1-ns1` to avoid messages
were deleted automatically
cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8080 namespaces set-retention -s 1g -t 3d
```

```
public/c1-ns1
$PULSAR_HOME/cluster1/broker1/bin/pulsar-client --url
pulsar://localhost:6650 produce -m 'hello' -n 300
public/c1-ns1/t1
```

5. Check the internal stats of the topic.

```
$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8080 topics stats-internal public/c1-ns1/t1
```

We should get a list of the ledgers in the topic. In this case it is ledgers 0, 2, and 3.

```
  "ledgers" : [ {
    "ledgerId" : 0,
    "entries" : 100,
    "size" : 5400,
    "offloaded" : false,
    "underReplicated" : false
  }, {
    "ledgerId" : 2,
    "entries" : 100,
    "size" : 5616,
    "offloaded" : false,
    "underReplicated" : false
  }, {
    "ledgerId" : 3,
    "entries" : 100,
    "size" : 5700,
    "offloaded" : false,
    "underReplicated" : false
  } ]
```

Check the ensembles for each of the ledgers to confirm that the ledger was written to bookies that are part of `group-bookie1`.

```
$PULSAR_HOME/bk-cluster/bk1/bin/bookkeeper shell ledgermetadata
-ledgerid 0
# check ensembles
ensembles={0=[127.0.0.1:3181, 127.0.0.1:3182]}

$PULSAR_HOME/bk-cluster/bk1/bin/bookkeeper shell ledgermetadata
-ledgerid 2
# check ensembles
ensembles={0=[127.0.0.1:3182, 127.0.0.1:3181]}

$PULSAR_HOME/bk-cluster/bk1/bin/bookkeeper shell ledgermetadata
```

```
-ledgerid 3
# check ensembles
ensembles={0=[127.0.0.1:3182, 127.0.0.1:3181]}
```

6. Repeat these steps in cluster2 so that we can isolate cluster1's namespaces from cluster2's.

Migrate Namespace

Migrate Bookie Affinity Group

Now that we have verified namespace isolation, if the bookie group hasn't enough space, we could migrate the bookie affinity group to a namespace.

1. Modify the bookie affinity group of the namespace.
```
$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8080 namespaces set-bookie-affinity-group
public/c1-ns1 --primary-group group-bookie2
```

2. Unload the namespace to make the bookie affinity group change take effect.
```
$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8080 namespaces unload public/c1-ns1
```

3. Produce messages to the topic `public/c1-ns1/t1` again.
```
$PULSAR_HOME/cluster1/broker1/bin/pulsar-client --url
pulsar://localhost:6650  produce -m 'hello' -n 300
public/c1-ns1/t1
```

4. Check ensembles for new added ledgers. We should see that a new ledger was already added in `group-bookie2`.
```
$PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
http://localhost:8080 topics stats-internal public/c1-ns1/t1
```

```
  "ledgers" : [ {
    "ledgerId" : 0,
    "entries" : 100,
    "size" : 5400,
    "offloaded" : false,
    "underReplicated" : false
  }, {
    "ledgerId" : 2,
    "entries" : 100,
    "size" : 5616,
```

```
    "offloaded" : false,
    "underReplicated" : false
  }, {
    "ledgerId" : 3,
    "entries" : 100,
    "size" : 5700,
    "offloaded" : false,
    "underReplicated" : false
  }, {
    "ledgerId" : 15,
    "entries" : 100,
    "size" : 5400,
    "offloaded" : false,
    "underReplicated" : false
  }, {
    "ledgerId" : 16,
    "entries" : 100,
    "size" : 5616,
    "offloaded" : false,
    "underReplicated" : false
  }, {
    "ledgerId" : 17,
    "entries" : 100,
    "size" : 5700,
    "offloaded" : false,
    "underReplicated" : false
  }]
```

Let’s check the ensembles for new added ledgers (15, 16, 17) to confirm that the ledger was written to bookies that are part of `group-bookie2`.

```
$PULSAR_HOME/bk-cluster/bk1/bin/bookkeeper shell ledgermetadata
-ledgerid 15
# check ensembles
ensembles={0=[127.0.0.1:3184, 127.0.0.1:3183]}

$PULSAR_HOME/bk-cluster/bk1/bin/bookkeeper shell ledgermetadata
-ledgerid 16
# check ensembles
ensembles={0=[127.0.0.1:3183, 127.0.0.1:3184]}

$PULSAR_HOME/bk-cluster/bk1/bin/bookkeeper shell ledgermetadata
-ledgerid 17
```

```
# check ensembles
ensembles={0=[127.0.0.1:3183, 127.0.0.1:3184]}
```

4. Scale up and down bookies

Eventually our data volume will grow beyond the capacity of our BookKeeper cluster, and we will need to scale up the number of bookies. In this section we will show you how to add a new bookie and assign it to an existing bookie affinity group.

Scale up

1. Start a new bookie node `bk-5`.
   ```
   cp -r apache-pulsar-2.8.1
   bk-cluster/bk5$PULSAR_HOME/bk-cluster//bk-cluster/bk5/bin/pulsa
   r-daemon start bookie
   ```

2. Add the newly added bookie node to `group-bookie1`.
   ```
   $PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
   http://localhost:8080 bookies set-bookie-rack \
   --bookie 127.0.0.1:3185 \
   --hostname 127.0.0.1:3185 \
   --group group-bookie2 \
   --rack rack2
   ```

3. Check bookie racks placement.
   ```
   $PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
   http://localhost:8080  bookies racks-placement
   "group-bookie1    {127.0.0.1:3181=BookieInfoImpl(rack=rack1,
   hostname=127.0.0.1:3181),
   127.0.0.1:3182=BookieInfoImpl(rack=rack1,
   hostname=127.0.0.1:3182)}"
   "group-bookie2    {127.0.0.1:3183=BookieInfoImpl(rack=rack2,
   hostname=127.0.0.1:3183),
   127.0.0.1:3184=BookieInfoImpl(rack=rack2,
   hostname=127.0.0.1:3184),
   127.0.0.1:3185=BookieInfoImpl(rack=rack2,
   hostname=127.0.0.1:3185)}"
   ```

4. Unload namespace `public/c1-ns1` to make the bookie affinity group change take effect.
   ```
   $PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
   http://localhost:8080 namespaces unload public/c1-ns1
   ```

5. Produce some messages to the topic `public/c1-ns1/t1` again.
   ```
   $PULSAR_HOME/cluster1/bin/pulsar-client --url
   pulsar://localhost:6650 produce -m 'hello' -n 300
   public/c1-ns1/t1
   ```

6. Check the newly added ledger of the topic `public/c1-ns1/t1`.
   ```
   $PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
   http://localhost:8080 topics stats-internal public/c1-ns1/t1

   $PULSAR_HOME/bk-cluster/bk1/bin/bookkeeper shell ledgermetadata
   -ledgerid ledgerid
   ```

We can see that the newly added ledger now exists in the newly added bookie node.

Scale down

In a distributed system, it is not uncommon for an individual component to fail. In this section, we will simulate the failure of one of the bookies and demonstrate that the shared BookKeeper cluster is able to tolerate the failure event. You could also refer to [this doc](https://bookkeeper.apache.org/docs/4.14.0/admin/decomission/) for a detailed example.

1. Make sure there are enough bookies in the affinity group.

   For example, if the configuration `managedLedgerDefaultEnsembleSize` of the broker is 2, then after we scale down the bookies we should have at least 2 bookies belonging to the affinity group.

   We can check the bookie rack placement.
   ```
   $PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
   http://localhost:8080 bookies racks-placement
   ```

2. Delete the bookie from the affinity bookie group.
   ```
   $PULSAR_HOME/cluster1/broker1/bin/pulsar-admin --admin-url
   http://localhost:8080 bookies delete-bookie-rack -b
   127.0.0.1:3185
   ```

3. Check if there are under-replicated ledgers, which should be expected given the fact that we have deleted a bookie.
   ```
   $PULSAR_HOME/bk-cluster/bk1/bin/bookkeeper shell
   listunderreplicated
   ```

4. Stop the bookie.
   ```
   $PULSAR_HOME/bk-cluster/bk5/bin/pulsar-daemon stop bookie
   ```

5. Decommission the bookie.
   ```
   $PULSAR_HOME/bk-cluster/bk1/bin/bookkeeper shell
   decommissionbookie -bookieid 127.0.0.1:3185
   ```

6. Check ledgers in the decommissioned bookie.
   ```
   $PULSAR_HOME/bk-cluster/bk1/bin/bookkeeper shell listledgers
   -bookieid 127.0.0.1:3185
   ```

7. List the bookies.
   ```
   $PULSAR_HOME/bk-cluster/bk1/bin/bookkeeper shell listbookies
   -rw -h
   ```

Approach 3: Use a Shared Nothing architecture to achieve isolation

This chapter details how to create multiple, separate Pulsar clusters for isolation of resources. Because this approach segregates resources and does not share storage or local ZooKeeper with other clusters, it provides the highest level of isolation.

This approach is ideal for critical workloads (such as billing and ads) and applications with a high level of security separation or with strict resource availability guarantee requirements. Prioritizing the highest isolation level leads to lower utilization of compute and storage, since each Pulsar cluster has its dedicated BookKeeper cluster.

How the Shared Nothing architecture works

Figure 4.1 demonstrates the deployment of separate Pulsar clusters to achieve isolation.

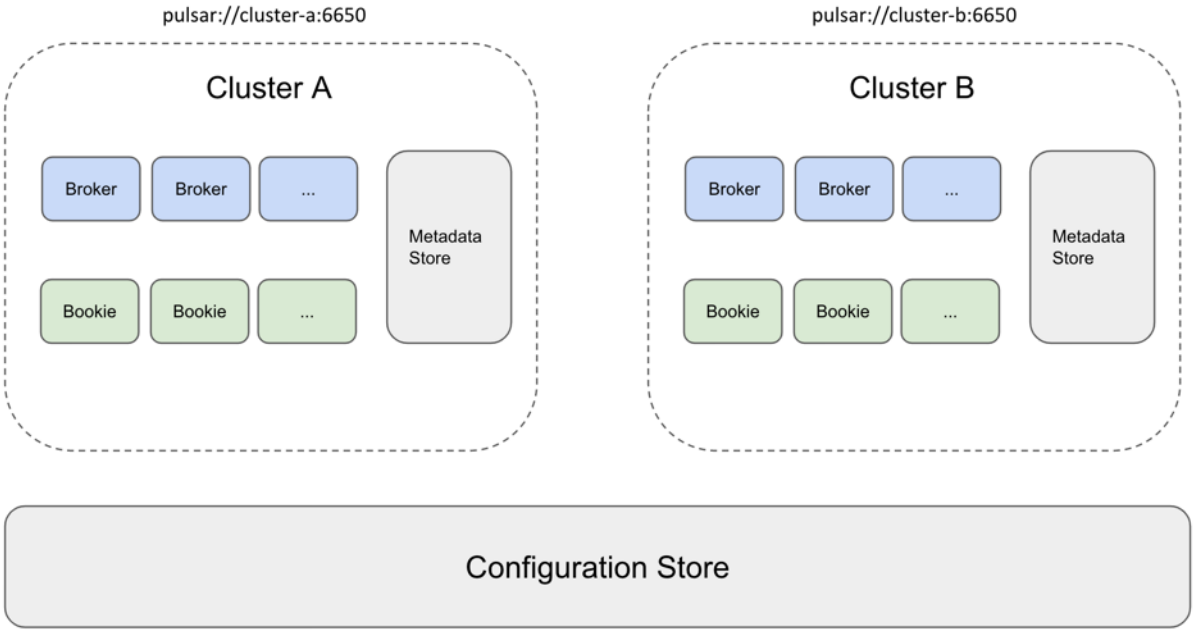


Figure 4.1 - Deployment of separate Pulsar clusters

Here are some key points to understand how this deployment works:

- Each Pulsar cluster exposes its service through a DNS entry point and makes sure a client can access the cluster through the DNS entry point. From the client side, the client can use one or multiple Pulsar URLs that the Pulsar cluster exposes as the service URL.
- Each Pulsar cluster has one or multiple brokers and bookies.
- Each Pulsar cluster has one metadata store.
- Metadata store can be separated into Pulsar metadata store and BookKeeper metadata store. While the metadata store in this guide refers to these two concepts rather than distinguish them.
- Separate Pulsar clusters use a shared configuration store.
- Pulsar's hierarchical resource management provides a solid foundation for isolation. In this approach, if you want to achieve namespace isolation, you need to specify a cluster for a namespace. The cluster must be in the allowed cluster list of the tenant. Topics under the namespace are assigned to this cluster. For how to set a cluster for a namespace, see here. For how to manage Pulsar clusters, see here.

How to deploy separate Pulsar clusters

To deploy separate Pulsar clusters, you'll need to take the following steps:

1. Deploy two separate Pulsar clusters
2. Verify data isolation of clusters
3. Synchronize and migrate data between clusters (optional)
4. Scale up and down nodes (optional)

1. Deploy two separate Pulsar clusters

Technical note: The examples in this chapter are developed on a macOS (version 11.2.3, memory 8G). Software requirement: Java 8.

First, deploy two clusters, each of which should support the following services:

- 1 ZooKeeper
- 1 Bookie
- 1 Broker

The following are the details of the two Pulsar clusters you will deploy:

Pulsar cluster 1		**Pulsar cluster 2**	
ZooKeeper 1	localhost:2181 web-service-url localhost:8080 web-service-url-tls localhost:8443 broker-service-url localhost:6650 broker-service-url-tls localhost:6651	ZooKeeper 1	localhost:2186 web-service-url localhost:8081 web-service-url-tls localhost:8444 broker-service-url localhost:6660 broker-service-url-tls localhost:6661
Bookie1	prometheusStatsHttpPort=8002 httpServerPort=8002	Bookie1	prometheusStatsHttpPort=8003 httpServerPort=8003 bookiePort=3182

	(The default value of bookiePort is 3181, no need for manual configuration.)		
Broker1	zookeeperServers=127.0.0.1:2181	Broker1	zookeeperServers=127.0.0.1:2186
configurationStoreServers=127.0.0.1:2184			

Prepare deployment

1. Download Pulsar and untar the tarball. In this example, Pulsar 2.7.0 is installed.

2. Create empty directories using the following structure and then change the names accordingly. You can create the directories anywhere in your local environment.

   Input
   ```
   |-separate-clusters
       |-configuration-store
           |-zk1
       |-cluster1
           |-zk1
           |-bk1
           |-broker1
       |-cluster2
           |-zk1
           |-bk1
           |-broker1
   ```

3. Copy the files to each directory that you created in step 2.

4. Start configuration store. Configuration store operates at the instance level and provides configuration management and task coordination across clusters. In this example, cluster1 and cluster2 share one configuration store.

   Input
   ```
   cd configuration-store/zk1
   ```

   ```
   bin/pulsar-daemon start configuration-store
   ```

Deploy Pulsar cluster1

1. Start a local ZooKeeper.
   For each Pulsar cluster, you need to deploy 1 local ZooKeeper to manage configurations and coordinate tasks.

   Input
   ```
   cd cluster1/zk1

   bin/pulsar-daemon start zookeeper
   ```

2. Initialize metadata. Write metadata to ZooKeeper.

   Input
   ```
   cd cluster1/zk1

   bin/pulsar initialize-cluster-metadata \
     --cluster cluster1 \
     --zookeeper localhost:2181 \
     --configuration-store localhost:2184 \
     --web-service-url http://localhost:8080/ \
     --web-service-url-tls https://localhost:8443/ \
     --broker-service-url pulsar://localhost:6650/ \
     --broker-service-url-tls pulsar+ssl://localhost:6651/
   ```

3. Deploy BookKeeper.
   BookKeeper provides persistent storage for messages on Pulsar. Each Pulsar broker owns its bookie. BookKeeper clusters and Pulsar clusters share the local ZooKeeper.

   (1) Configure bookies.
   Change the value of the following configurations in the `cluster1/bk1/conf/bookkeeper.conf` file.
   ```
   allowLoopback=true
   prometheusStatsHttpPort=8002
   httpServerPort=8002
   ```

   (2) Start bookies.

   Input
   ```
   cd cluster1/bk1

   bin/pulsar-daemon start bookie
   ```

Check whether the bookie is started successfully.

Input
```
bin/bookkeeper shell bookiesanity
```

Output
```
Bookie sanity test succeeded
```

4. Deploy brokers.

   (1) Configure brokers. Change the value of the following configurations in the `cluster1/broker1/conf/broker.conf` file.
   ```
   zookeeperServers=127.0.0.1:2181
   configurationStoreServers=127.0.0.1:2184
   clusterName=cluster1
   managedLedgerDefaultEnsembleSize=1
   managedLedgerDefaultWriteQuorum=1
   managedLedgerDefaultAckQuorum=1
   ```

   (2) Start brokers.

   Input
   ```
   cd cluster1/broker1
   ```

   ```
   bin/pulsar-daemon start broker
   ```

Deploy Pulsar cluster2

1. Deploy a local ZooKeeper.

   (1) Configure a local ZooKeeper.

   Change the value of the following configurations in the `cluster2/zk1/conf/zookeeper.conf` file.
   ```
   clientPort=2186
   admin.serverPort=9992
   ```

   Add the following configurations to the `cluster2/zk1/conf/pulsar_env.sh` file.
   ```
   OPTS="-Dstats_server_port=8011"
   ```

(2) Start a local ZooKeeper.

Input

```
cd cluster2/zk1

bin/pulsar-daemon start zookeeper
```

2. Initialize metadata.

Input

```
bin/pulsar initialize-cluster-metadata \
  --cluster cluster2 \
  --zookeeper localhost:2186 \
  --configuration-store localhost:2184 \
  --web-service-url http://localhost:8081/ \
  --web-service-url-tls https://localhost:8444/ \
  --broker-service-url pulsar://localhost:6660/ \
  --broker-service-url-tls pulsar+ssl://localhost:6661/
```

3. Deploy BookKeeper.

(1) Configure bookies.
Change the value of the following configurations in the `cluster2/bk1/conf/bookkeeper.conf` file.

```
bookiePort=3182
zkServers=localhost:2186
allowLoopback=true
prometheusStatsHttpPort=8003
httpServerPort=8003
```

(2) Start bookies.

```
      **Input**
cd cluster2/bk1
bin/pulsar-daemon start bookie

      Check whether the bookie is started successfully.

      **Input**

bin/bookkeeper shell bookiesanity
```

Output

```
Bookie sanity test succeeded
```

4. Deploy brokers.

   (1) Configure brokers.

   Change the value of the following configurations in the `cluster2/broker1/conf/broker.conf` file.

```
clusterName=cluster2
zookeeperServers=127.0.0.1:2186
configurationStoreServers=127.0.0.1:2184
brokerServicePort=6660
webServicePort=8081
managedLedgerDefaultEnsembleSize=1
managedLedgerDefaultWriteQuorum=1
managedLedgerDefaultAckQuorum=1
```

   Change the value of the following configurations in the `cluster2/broker1/conf/client.conf` file.

```
webServiceUrl=http://localhost:8081/
brokerServiceUrl=pulsar://localhost:6660/
```

   (2) Start brokers.

   Input

```
cd cluster2/broker1

bin/pulsar-daemon start broker
```

2. Verify data isolation of clusters

This section verifies whether the data in the two Pulsar clusters are isolated.

1. Create namespace1 and assign it to cluster1.

   Tip: The format of a namespace name is `<tenant-name>/<namespace-name>`. For more information, see Namespaces.

   Input

```
cd cluster1/broker1
```

```
bin/pulsar-admin namespaces create -c cluster1
public/namespace1
```

Check the result.

Input
```
bin/pulsar-admin namespaces list public
```

Output
```
"public/default"
"public/namespace1"
```

2. Set the retention policy for namespace1.

   Note: If the retention policy is not set and the topic is not subscribed, the data stored on the topic is deleted automatically after a while.

   Input
   ```
   bin/pulsar-admin namespaces set-retention -s 100M -t 3d
   public/namespace1
   ```

3. Create topic1 in namespace1 and write 1,000 messages to this topic.

   Tip: The pulsar-client is a command line tool to send and consume data. For more information, see Pulsar command line tools.

   Input
   ```
   bin/pulsar-client produce -m 'hello c1 to c2' -n 1000
   public/namespace1/topic1
   ```

   ```
   09:56:34.504 [main] INFO
   org.apache.pulsar.client.cli.PulsarClientTool - 1000 messages
   successfully produced
   ```

   Check the result.

   Input
   ```
   bin/pulsar-admin --admin-url http://localhost:8080 topics
   stats-internal public/namespace1/topic1
   ```

   Output
   The `entriesAddedCounter` parameter shows that 1000 messages are added.
   ```
   {
   ```

```
  "entriesAddedCounter" : 1000,
  "numberOfEntries" : 1000,
  "totalSize" : 65616,
  "currentLedgerEntries" : 1000,
  "currentLedgerSize" : 65616,
  "lastLedgerCreatedTimestamp" :
"2021-04-22T10:24:00.582+08:00",
  "waitingCursorsCount" : 0,
  "pendingAddEntriesCount" : 0,
  "lastConfirmedEntry" : "4:999",
  "state" : "LedgerOpened",
  "ledgers" : [ {
    "ledgerId" : 4,
    "entries" : 0,
    "size" : 0,
    "offloaded" : false
  } ],
  "cursors" : { },
  "compactedLedger" : {
    "ledgerId" : -1,
    "entries" : -1,
    "size" : -1,
    "offloaded" : false
  }
}
```

4. Check the data stored on public/namespace1/topic1 by cluster2 (localhost:8081).

   Input

   ```
   bin/pulsar-admin --admin-url http://localhost:8081 topics
   stats-internal public/namespace1/topic1
   ```

   Output

   The attempt failed. The error message shows that the data stored on public/namespace1 is assigned only to cluster1. This proves that the data is isolated.

   ```
   Namespace missing local cluster name in clusters list:
   local_cluster=cluster2 ns=public/namespace1 clusters=[cluster1]

   Reason: Namespace missing local cluster name in clusters list:
   local_cluster=cluster2 ns=public/namespace1 clusters=[cluster1]
   ```

5. Write data to public/namespace1/topic1 in cluster2.

Input
```
cd cluster2/broker1

bin/pulsar-client produce -m 'hello c1 to c2' -n 1000
public/namespace1/topic1
```

Output
The error message shows that 0 message is written. The attempt failed because namespace1 is assigned only to cluster1. This proves that the data is isolated.
```
12:09:50.005 [main] INFO
org.apache.pulsar.client.cli.PulsarClientTool - 0 messages
successfully produced
```

3. Synchronize and migrate data between clusters

After verifying that the data are isolated, you can synchronize (using geo-replication) and migrate data between clusters.

1. Assign namespace1 to cluster2, that is, adding cluster2 to the cluster list of namespace1.

   This enables geo-replication to synchronize the data between cluster1 and cluster2.

   Input
   ```
   bin/pulsar-admin namespaces set-clusters --clusters
   cluster1,cluster2 public/namespace1
   ```

   Check the result.

   Input
   ```
   bin/pulsar-admin namespaces get-clusters public/namespace1
   ```

   Output
   ```
   "Cluster1"
   "Cluster2"
   ```

2. Check whether topic1 is in cluster2.

   Input
   ```
   bin/pulsar-admin --admin-url http://localhost:8081 topics
   ```

```
stats-internal public/namespace1/topic1
```

Output

The output shows that there are 1000 messages on cluster2/topic1. This proves that the data stored on cluster1/topic1 is replicated to cluster2 successfully.


```
{
  "entriesAddedCounter" : 1000,
  "numberOfEntries" : 1000,
  "totalSize" : 75616,
  "currentLedgerEntries" : 1000,
  "currentLedgerSize" : 75616,
  "lastLedgerCreatedTimestamp" :
"2021-04-23T12:02:52.929+08:00",
  "waitingCursorsCount" : 1,
  "pendingAddEntriesCount" : 0,
  "lastConfirmedEntry" : "1:999",
  "state" : "LedgerOpened",
  "ledgers" : [ {
    "ledgerId" : 1,
    "entries" : 0,
    "size" : 0,
    "offloaded" : false
  } ],
  "cursors" : {
    "pulsar.repl.cluster1" : {
      "markDeletePosition" : "1:999",
      "readPosition" : "1:1000",
      "waitingReadOp" : true,
      "pendingReadOps" : 0,
      "messagesConsumedCounter" : 1000,
      "cursorLedger" : 2,
      "cursorLedgerLastEntry" : 2,
      "individuallyDeletedMessages" : "[]",
      "lastLedgerSwitchTimestamp" :
"2021-04-23T12:02:53.248+08:00",
      "state" : "Open",
      "numberOfEntriesSinceFirstNotAckedMessage" : 1,
      "totalNonContiguousDeletedMessagesRange" : 0,
      "properties" : { }
    }
  },
  "compactedLedger" : {
    "ledgerId" : -1,
    "entries" : -1,
    "size" : -1,
```

```
    "offloaded" : false
  }
}
```

3. Migrate the producer and consumer from cluster1 to cluster2.

```
PulsarClient pulsarClient1 =
PulsarClient.builder().serviceUrl("pulsar://localhost:6650").bu
ild();
// migrate the client to cluster2 pulsar://localhost:6660
PulsarClient pulsarClient2 =
PulsarClient.builder().serviceUrl("pulsar://localhost:6660").bu
ild();
```

4. Remove cluster1 from the cluster list of namespace1.

   Input

```
bin/pulsar-admin namespaces set-clusters --clusters cluster2
public/namespace1
```

   Check if the data is stored on cluster1/topic1.

   Input

```
cd cluster1/broker1

bin/pulsar-admin --admin-url http://localhost:8080 topics
stats-internal public/namespace1/topic1
```

   Output

   The data is removed from cluster1/topic1 successfully since the value of `numberOfEntries` parameter is 0.

```
{
  "entriesAddedCounter" : 0,
  "numberOfEntries" : 0,
  "totalSize" : 0,
  "currentLedgerEntries" : 0,
  "currentLedgerSize" : 0,
  "lastLedgerCreatedTimestamp" : "2021-04-23T15:20:08.1+08:00",
  "waitingCursorsCount" : 1,
  "pendingAddEntriesCount" : 0,
  "lastConfirmedEntry" : "3:-1",
  "state" : "LedgerOpened",
  "ledgers" : [ {
    "ledgerId" : 3,
```

```
    "entries" : 0,
    "size" : 0,
    "offloaded" : false
  } ],
  "cursors" : {
    "pulsar.repl.cluster2" : {
      "markDeletePosition" : "3:-1",
      "readPosition" : "3:0",
      "waitingReadOp" : true,
      "pendingReadOps" : 0,
      "messagesConsumedCounter" : 0,
      "cursorLedger" : 4,
      "cursorLedgerLastEntry" : 0,
      "individuallyDeletedMessages" : "[]",
      "lastLedgerSwitchTimestamp" :
"2021-04-23T15:20:08.122+08:00",
      "state" : "Open",
      "numberOfEntriesSinceFirstNotAckedMessage" : 1,
      "totalNonContiguousDeletedMessagesRange" : 0,
      "properties" : { }
    }
  },
  "compactedLedger" : {
    "ledgerId" : -1,
    "entries" : -1,
    "size" : -1,
    "offloaded" : false
  }
}
```


At this point, you replicated data from cluster1/topic1 to cluster2 and then removed the data from cluster1/topic1.

4. Scale up and down nodes

This section outlines how to scale up and scale down nodes (brokers and bookies) as workloads increase or decrease.

Broker

Scale up brokers

In this procedure, you'll create 2 partitioned topics on cluster1/broker1 and add 2 brokers. Then, you'll offload the data stored on partitioned topics and check the data distribution among 3 brokers.

1. Check the information about brokers in cluster1.
   Input
   ```
   cd/cluster1/broker1
   ```

   ```
   bin/pulsar-admin brokers list cluster1
   ```

   Output
   The output shows that broker1 is the only broker in cluster1.
   ```
   "192.168.0.105:8080"
   ```

2. Create 2 partitioned topics on cluster1/broker1.

   Create 6 partitions for partitioned-topic1 and 7 partitions for partitioned-topic2.

   Input
   ```
   bin/pulsar-admin topics create-partitioned-topic -p 6
   public/namespace1/partitioned-topic1
   ```

   ```
   bin/pulsar-admin topics create-partitioned-topic -p 7
   public/namespace1/partitioned-topic2
   ```

   Check the result.

   Input
   ```
   bin/pulsar-admin topics partitioned-lookup
   public/namespace1/partitioned-topic1
   ```

   Output

   All data of partitioned-topic1 is from broker1.
   ```
   "persistent://public/namespace1/partitioned-topic1-partition-0
   pulsar://192.168.0.105:6650"
   "persistent://public/namespace1/partitioned-topic1-partition-1
   pulsar://192.168.0.105:6650"
   "persistent://public/namespace1/partitioned-topic1-partition-2
   pulsar://192.168.0.105:6650"
   "persistent://public/namespace1/partitioned-topic1-partition-3
   pulsar://192.168.0.105:6650"
   ```

```
"persistent://public/namespace1/partitioned-topic1-partition-4
pulsar://192.168.0.105:6650"
"persistent://public/namespace1/partitioned-topic1-partition-5
pulsar://192.168.0.105:6650"
```

Input

```
bin/pulsar-admin topics partitioned-lookup
public/namespace1/partitioned-topic2
```

Output

All data of partitioned-topic2 is from broker1.

```
"persistent://public/namespace1/partitioned-topic2-partition-0
pulsar://192.168.0.105:6650"
"persistent://public/namespace1/partitioned-topic2-partition-1
pulsar://192.168.0.105:6650"
"persistent://public/namespace1/partitioned-topic2-partition-2
pulsar://192.168.0.105:6650"
"persistent://public/namespace1/partitioned-topic2-partition-3
pulsar://192.168.0.105:6650"
"persistent://public/namespace1/partitioned-topic2-partition-4
pulsar://192.168.0.105:6650"
"persistent://public/namespace1/partitioned-topic2-partition-5
pulsar://192.168.0.105:6650"
"persistent://public/namespace1/partitioned-topic2-partition-6
pulsar://192.168.0.105:6650"
```

3. Add broker2 and broker3.

   (1) Prepare for deployment.

   Create two empty repositories (broker2 and broker3) under cluster1 repository. Copy the untarred files in the Pulsar repository to these two repositories.

```
|-separate-clusters
    |-configuration-store
        |-zk1
    |-cluster1
        |-zk1
        |-bk1
        |-broker1
        |-broker2
        |-broker3
    |-cluster2
        |-zk1
```

```
        |-bk1
        |-broker1
```

(2) Deploy brokers.

(2.a) Configure brokers.

Configure broker2	Configure broker3
Change the values of the following configurations in the `cluster1/broker2/conf/broker.conf` file.	Change the values of the following configurations in the `cluster1/broker3/conf/broker.conf` file.
brokerServicePort=6652 webServicePort=8082 zookeeperServers=127.0.0.1:2181 configurationStoreServers=127.0.0.1:2184 clusterName=cluster1 managedLedgerDefaultEnsembleSize=1 managedLedgerDefaultWriteQuorum=1 managedLedgerDefaultAckQuorum=1	brokerServicePort=6653 webServicePort=8083 zookeeperServers=127.0.0.1:2181 configurationStoreServers=127.0.0.1:2184 clusterName=cluster1 managedLedgerDefaultEnsembleSize=1 managedLedgerDefaultWriteQuorum=1 managedLedgerDefaultAckQuorum=1

(2.b) Start brokers.

Start broker2	Start broker3

Input
`cd cluster1/broker2`
`bin/pulsar-daemon start broker`
	Input
	`cd cluster1/broker3`
	`bin/pulsar-daemon start broker`

(2.c) Check the information about the running brokers in cluster1.

Input

```
bin/pulsar-admin brokers list cluster1
```

Output

```
"192.168.0.105:8080" // broker1
```

```
"192.168.0.105:8082" // broker2
"192.168.0.105:8083" // broker3
```

4. Offload the data stored on namespace1/partitioned-topic1.
   Input
   ```
   bin/pulsar-admin namespaces unload public/namespace1
   ```

   Check the result.

   (1) Check the distribution of data stored on partitioned-topic1.

   Input
   ```
   bin/pulsar-admin topics partitioned-lookup
   public/namespace1/partitioned-topic1
   ```

   Output

   The output shows that the data stored on partitioned-topic1 is distributed evenly on broker1, broker2, and broker3.
   ```
   "persistent://public/namespace1/partitioned-topic1-partition-0
   pulsar://192.168.0.105:6650"
   "persistent://public/namespace1/partitioned-topic1-partition-1
   pulsar://192.168.0.105:6653"
   "persistent://public/namespace1/partitioned-topic1-partition-2
   pulsar://192.168.0.105:6652"
   "persistent://public/namespace1/partitioned-topic1-partition-3
   pulsar://192.168.0.105:6653"
   "persistent://public/namespace1/partitioned-topic1-partition-4
   pulsar://192.168.0.105:6650"
   "persistent://public/namespace1/partitioned-topic1-partition-5
   pulsar://192.168.0.105:6653"
   ```

   (2) Check the distribution of data stored on partitioned-topic2.

   Input
   ```
   bin/pulsar-admin topics partitioned-lookup
   public/namespace1/partitioned-topic2
   ```

   The output shows that the data stored on partitioned-topic2 is distributed evenly on broker1, broker2, and broker3.

   Output
   ```
   "persistent://public/namespace1/partitioned-topic2-partition-0
   ```

```
pulsar://192.168.0.105:6653"
"persistent://public/namespace1/partitioned-topic2-partition-1
pulsar://192.168.0.105:6650
"persistent://public/namespace1/partitioned-topic2-partition-2
pulsar://192.168.0.105:6653"
"persistent://public/namespace1/partitioned-topic2-partition-3
pulsar://192.168.0.105:6652"
"persistent://public/namespace1/partitioned-topic2-partition-4
pulsar://192.168.0.105:6653"
"persistent://public/namespace1/partitioned-topic2-partition-5
pulsar://192.168.0.105:6650"
"persistent://public/namespace1/partitioned-topic2-partition-6
pulsar://192.168.0.105:6653"
```

Scale down brokers

Tip: The following steps continue from the previous section “Scale up brokers”.

In this procedure, you’ll stop 1 broker in cluster1 and check how the data stored on the partitioned topics is distributed among other brokers.

1. Stop broker3.

   Input
   ```
   cd/cluster1/broker3
   ```

   ```
   bin/pulsar-daemon stop broker
   ```

   Check the result.

   Input
   ```
   bin/pulsar-admin brokers list cluster1
   ```

   Output

   The output shows that only broker1 and broker2 are running in cluster1.
   ```
   "192.168.0.105:8080" // broker1
   "192.168.0.105:8082" // broker2
   ```

2. Check the distribution of data stored on partitioned-topic1.

Input

```
bin/pulsar-admin topics partitioned-lookup
public/namespace1/partitioned-topic1
```

Output

The output shows that the data stored on partitioned-topic1 is distributed evenly between broker1 and broker2, which means that the data from broker3 is redistributed.

```
"persistent://public/namespace1/partitioned-topic1-partition-0
pulsar://192.168.0.105:6650"
"persistent://public/namespace1/partitioned-topic1-partition-1
pulsar://192.168.0.105:6650"
"persistent://public/namespace1/partitioned-topic1-partition-2
pulsar://192.168.0.105:6652"
"persistent://public/namespace1/partitioned-topic1-partition-3
pulsar://192.168.0.105:6652"
"persistent://public/namespace1/partitioned-topic1-partition-4
pulsar://192.168.0.105:6650"
"persistent://public/namespace1/partitioned-topic1-partition-5
pulsar://192.168.0.105:6650"
```

Similarly, the data stored on partitioned-topic2 is distributed evenly between broker1 and broker2.

Input

```
bin/pulsar-admin topics partitioned-lookup
public/namespace1/partitioned-topic2
```

Output

```
"persistent://public/namespace1/partitioned-topic2-partition-0
pulsar://192.168.0.105:6650"
"persistent://public/namespace1/partitioned-topic2-partition-1
pulsar://192.168.0.105:6650"
"persistent://public/namespace1/partitioned-topic2-partition-2
pulsar://192.168.0.105:6652"
"persistent://public/namespace1/partitioned-topic2-partition-3
pulsar://192.168.0.105:6652"
"persistent://public/namespace1/partitioned-topic2-partition-4
pulsar://192.168.0.105:6650"
"persistent://public/namespace1/partitioned-topic2-partition-5
pulsar://192.168.0.105:6650"
"persistent://public/namespace1/partitioned-topic2-partition-6
pulsar://192.168.0.105:6652"
```

Bookie

Scale up bookies

In this procedure, you'll add 2 bookies to cluster1/bookkeeper1. Then, you'll write data to topic1 and check whether the replicas are saved.

1. Check the information about bookies in cluster1.
   Input
   ```
   cd cluster1/bk1

   bin/bookkeeper shell listbookies -rw -h
   ```

   Output

   The output shows that broker1 is the only broker in cluster1.
   ```
   12:31:34.933 [main] INFO
   org.apache.bookkeeper.tools.cli.commands.bookies.ListBookiesCom
   mand - ReadWrite Bookies :
   12:31:34.946 [main] INFO
   org.apache.bookkeeper.tools.cli.commands.bookies.ListBookiesCom
   mand - BookieID:192.168.0.105:3181, IP:192.168.0.105,
   Port:3181, Hostname:192.168.0.105
   ```

2. Allow 3 bookies to serve.

   Change the values of the following configurations in the `cluster1/broker1/conf/broker.conf` file.
   ```
   managedLedgerDefaultEnsembleSize=3 // specify the number of
   bookies to use when creating a ledger
   managedLedgerDefaultWriteQuorum=3 // specify the number of
   copies to store for each message
   managedLedgerDefaultAckQuorum=2  // specify the number of
   guaranteed copies (acks to wait before writing is completed)
   ```

3. Restart broker1 to enable the configurations.

   Input
   ```
   cd cluster1/broker1

   bin/pulsar-daemon stop broker
   ```

```
bin/pulsar-daemon start broker
```

4. Set the retention policy for the messages in public/default.

   Note: If the retention policy is not set and the topic is not subscribed, the data of the topic is deleted automatically after a while.

   Input
   ```
   cd cluster1/broker1
   ```

   ```
   bin/pulsar-admin namespaces set-retention -s 100M -t 3d
   public/default
   ```

5. Create topic1 in public/default and write 100 messages to this topic.

   Input
   ```
   bin/pulsar-client produce -m 'hello' -n 100 topic1
   ```

   Output

   The data is not written successfully because of the insufficient number of bookies.
   ```
   …
   ```

   ```
   12:40:38.886 [pulsar-client-io-1-1] WARN
   org.apache.pulsar.client.impl.ClientCnx - [id: 0x56f92aff,
   L:/192.168.0.105:53069 - R:/192.168.0.105:6650] Received error
   from server:
   org.apache.bookkeeper.mledger.ManagedLedgerException: Not
   enough non-faulty bookies available
   ```

   ```
   …
   ```

   ```
   12:40:38.886 [main] ERROR
   org.apache.pulsar.client.cli.PulsarClientTool - Error while
   producing messages
   ```

   ```
   …
   ```

   ```
   12:40:38.890 [main] INFO
   org.apache.pulsar.client.cli.PulsarClientTool - 0 messages
   successfully produced
   ```

6. Add bookie2 and bookie3.

   (1) Prepare for deployment.

   Create two empty repositories (bk2 and bk3) under cluster1 repository. Copy the untarred files in Pulsar repository to these two repositories.

```
|-separate-clusters
    |-configuration-store
        |-zk1
    |-cluster1
        |-zk1
        |-bk1
        |-bk2
        |-bk3
        |-broker1
    |-cluster2
        |-zk1
        |-bk1
        |-broker1
```

   (2) Deploy bookies.

   (2.a) Configure bookies.

Configure bookie2	Configure bookie3
Change the values of the following configurations in the `cluster1/bk2/conf/bookkeeper.conf` file.	Change the values of the following configurations in the `cluster1/bk3/conf/bookkeeper.conf` file.
allowLoopback=true bookiePort=3183 prometheusStatsHttpPort=8004 httpServerPort=8004	allowLoopback=true bookiePort=3184 prometheusStatsHttpPort=8005 httpServerPort=8005

   (2.b) Start bookies.

Start bookie2	Start bookie3
Input `cd cd cluster1/bk2`	Input `cd cluster1/bk3`

`bin/pulsar-daemon start bookie` `bin/bookkeeper shell bookiesanity`	`bin/pulsar-daemon start bookie` `bin/bookkeeper shell bookiesanity`

(2.c) Check the running bookies in cluster1.

Input

```
bin/bookkeeper shell listbookies -rw -h
```

Output

All three bookies are running in cluster1:

bookie1 : 192.168.0.105:3181
bookie2 : 192.168.0.105:3183
bookie3 : 192.168.0.105:3184

```
12:12:47.574 [main] INFO
org.apache.bookkeeper.tools.cli.commands.bookies.ListBookiesCom
mand - BookieID:192.168.0.105:3183, IP:192.168.0.105,
Port:3183, Hostname:192.168.0.105
12:12:47.575 [main] INFO
org.apache.bookkeeper.tools.cli.commands.bookies.ListBookiesCom
mand - BookieID:192.168.0.105:3184, IP:192.168.0.105,
Port:3184, Hostname:192.168.0.105
12:12:47.576 [main] INFO
org.apache.bookkeeper.tools.cli.commands.bookies.ListBookiesCom
mand - BookieID:192.168.0.105:3181, IP:192.168.0.105,
Port:3181, Hostname:192.168.0.105
```

7. Set the retention policy for messages in public/default.

   Note: If the retention policy is not set and the topic is not subscribed, the data stored on the topic is deleted automatically after a while.

   Input

   ```
   cd cluster1/broker1
   bin/pulsar-admin namespaces set-retention -s 100M -t 3d
   public/default
   ```

8. Create topic1 in public/default and write 100 messages to this topic.

   Input

```
bin/pulsar-client produce -m 'hello' -n 100 topic1
```

Output

The messages are written successfully.

```
…
12:17:40.222 [main] INFO
org.apache.pulsar.client.cli.PulsarClientTool - 100 messages
successfully produced
```

9. Check the information about topic1.

   Input

   ```
   bin/pulsar-admin topics stats-internal topic1
   ```

   Output

   The output shows that the data stored on topic1 is saved in the ledger with ledgerId 5.

   ```
   {
     "entriesAddedCounter" : 100,
     "numberOfEntries" : 100,
     "totalSize" : 5500,
     "currentLedgerEntries" : 100,
     "currentLedgerSize" : 5500,
     "lastLedgerCreatedTimestamp" :
   "2021-05-11T12:17:38.881+08:00",
     "waitingCursorsCount" : 0,
     "pendingAddEntriesCount" : 0,
     "lastConfirmedEntry" : "5:99",
     "state" : "LedgerOpened",
     "ledgers" : [ {
       "ledgerId" : 5,
       "entries" : 0,
       "size" : 0,
       "offloaded" : false
     } ],
     "cursors" : { },
     "compactedLedger" : {
       "ledgerId" : -1,
       "entries" : -1,
       "size" : -1,
       "offloaded" : false
     }
   ```

```
}
```

10. Check in which bookies the ledger with ledgerId 5 is saved.

    Input
    ```
    bin/bookkeeper shell ledgermetadata -ledgerid 5
    ```

    Output

    As configured previously, the ledger with ledgerId 5 is saved on bookie1 (3181), bookie2 (3181), and bookie3 (3184).
    ```
    …
    12:23:17.705 [main] INFO
    org.apache.bookkeeper.tools.cli.commands.client.LedgerMetaDataC
    ommand - ledgerID: 5
    12:23:17.714 [main] INFO
    org.apache.bookkeeper.tools.cli.commands.client.LedgerMetaDataC
    ommand - LedgerMetadata{formatVersion=3, ensembleSize=3,
    writeQuorumSize=3, ackQuorumSize=2, state=OPEN,
    digestType=CRC32C, password=base64:,
    ensembles={0=[192.168.0.105:3184, 192.168.0.105:3181,
    192.168.0.105:3183]},
    customMetadata={component=base64:bWFuYWdlZC1sZWRnZXI=,
    pulsar/managed-ledger=base64:cHVibGljL2RlZmF1bHQvcGVyc2lzdGVudC
    90b3BpYzE=, application=base64:cHVsc2Fy}}
    …
    ```

Scale down bookies

Tip: The following steps continue from the previous section “Scale up bookies”.

In this procedure, you’ll remove 2 bookies. Then, you’ll write data to topic2 and check where the data is saved.

1. Allow 1 bookie to serve.
   Change the values of the following configurations in the `cluster1/broker1/conf/broker.conf` file.
   ```
   managedLedgerDefaultEnsembleSize=1 // specify the number of
   bookies to use when creating a ledger
   managedLedgerDefaultWriteQuorum=1 // specify the number of
   copies to store for each message
   ```

```
managedLedgerDefaultAckQuorum=1  // specify the number of
guaranteed copies (acks to wait before writing is completed)
```

2. Restart broker1 to enable the configurations.

   Input
   ```
   cd cluster1/broker1

   bin/pulsar-daemon stop broker

   bin/pulsar-daemon start broker
   ```

3. Check the information about bookies in cluster1.

   Input
   ```
   cd cluster1/bk1

   bin/bookkeeper shell listbookies -rw -h
   ```

   Output

   All three bookies are running in cluster1, including bookie1 (3181), bookie2 (3183), and bookie3 (3184).
   ```
   …
   15:47:41.370 [main] INFO
   org.apache.bookkeeper.tools.cli.commands.bookies.ListBookiesCom
   mand - ReadWrite Bookies :
   15:47:41.382 [main] INFO
   org.apache.bookkeeper.tools.cli.commands.bookies.ListBookiesCom
   mand - BookieID:192.168.0.105:3183, IP:192.168.0.105,
   Port:3183, Hostname:192.168.0.105
   15:47:41.383 [main] INFO
   org.apache.bookkeeper.tools.cli.commands.bookies.ListBookiesCom
   mand - BookieID:192.168.0.105:3184, IP:192.168.0.105,
   Port:3184, Hostname:192.168.0.105
   15:47:41.384 [main] INFO
   org.apache.bookkeeper.tools.cli.commands.bookies.ListBookiesCom
   mand - BookieID:192.168.0.105:3181, IP:192.168.0.105,
   Port:3181, Hostname:192.168.0.105
   …
   ```

4. Stop bookie2 and bookie3.

Tip: For more information about how to stop bookies, see Decommission Bookies.

Input

```
cd cluster1/bk2

bin/bookkeeper shell listunderreplicated

bin/pulsar-daemon stop bookie

bin/bookkeeper shell decommissionbookie
```

Input

```
cd cluster1/bk3

bin/bookkeeper shell listunderreplicated

bin/pulsar-daemon stop bookie

bin/bookkeeper shell decommissionbookie
```

5. Check the information about bookies in cluster1.

   Input

   ```
   cd cluster1/bk1

   bin/bookkeeper shell listbookies -rw -h
   ```

   Output

   The output shows that bookie1 (3181) is the only running bookie in cluster1.

   ```
   …
   16:05:28.690 [main] INFO
   org.apache.bookkeeper.tools.cli.commands.bookies.ListBookiesCom
   mand - ReadWrite Bookies :
   16:05:28.700 [main] INFO
   org.apache.bookkeeper.tools.cli.commands.bookies.ListBookiesCom
   mand - BookieID:192.168.0.105:3181, IP:192.168.0.105,
   Port:3181, Hostname:192.168.0.105
   …
   ```

6. Set the retention policy for the messages in public/default.

Note: If the retention policy is not set and the topic is not subscribed, the data stored on the topic is deleted automatically after a while.

Input

```
cd cluster1/broker1

bin/pulsar-admin namespaces set-retention -s 100M -t 3d
public/default
```

7. Create topic2 in public/default and write 100 messages to this topic.

   Input

   ```
   bin/pulsar-client produce -m 'hello' -n 100 topic2
   ```

   Output

   The data is written successfully.

   ```
   …
   16:06:59.448 [main] INFO
   org.apache.pulsar.client.cli.PulsarClientTool - 100 messages
   successfully produced
   ```

8. Check the information about topic2.

   Input

   ```
   bin/pulsar-admin topics stats-internal topic2
   ```

   Output

   The data stored on topic2 is saved in the ledger with ledgerId 7.

   ```
   {
     "entriesAddedCounter" : 100,
     "numberOfEntries" : 100,
     "totalSize" : 5400,
     "currentLedgerEntries" : 100,
     "currentLedgerSize" : 5400,
     "lastLedgerCreatedTimestamp" :
   "2021-05-11T16:06:59.058+08:00",
     "waitingCursorsCount" : 0,
     "pendingAddEntriesCount" : 0,
     "lastConfirmedEntry" : "7:99",
     "state" : "LedgerOpened",
     "ledgers" : [ {
   ```

```
    "ledgerId" : 7,
    "entries" : 0,
    "size" : 0,
    "offloaded" : false
  } ],
  "cursors" : { },
  "compactedLedger" : {
    "ledgerId" : -1,
    "entries" : -1,
    "size" : -1,
    "offloaded" : false
  }
}
```

9. Check where the ledger with ledgerId 7 is saved.

   Input

   ```
   bin/bookkeeper shell ledgermetadata -ledgerid 7
   ```

   Output

   The ledger with ledgerId 7 is saved on bookie1 (3181).

   ```
   …
   16:11:28.843 [main] INFO
   org.apache.bookkeeper.tools.cli.commands.client.LedgerMetaDataC
   ommand - ledgerID: 7
   16:11:28.846 [main] INFO
   org.apache.bookkeeper.tools.cli.commands.client.LedgerMetaDataC
   ommand - LedgerMetadata{formatVersion=3, ensembleSize=1,
   writeQuorumSize=1, ackQuorumSize=1, state=OPEN,
   digestType=CRC32C, password=base64:,
   ensembles={0=[192.168.0.105:3181]},
   customMetadata={component=base64:bWFuYWdlZC1sZWRnZXI=,
   pulsar/managed-ledger=base64:cHVibGljL2RlZmF1bHQvcGVyc2lzdGVudC
   90b3BpYzM=, application=base64:cHVsc2Fy}}
   …
   ```

Get started with Pulsar

- **Learn Pulsar Fundamentals with StreamNative Academy:** If you're new to Pulsar, we recommend taking our self-paced Pulsar courses developed by the original creators of Pulsar.
- **Spin up a Pulsar cluster in minutes:** Sign up for StreamNative Free Cloud today. StreamNative Cloud is the simple, fast, and cost-effective way to run Pulsar in the public cloud.
- **Connect with others in the Pulsar community:** Join the Pulsar community on Slack to ask questions and engage with other Pulsar enthusiasts. You can also sign up for the Pulsar mailing list.