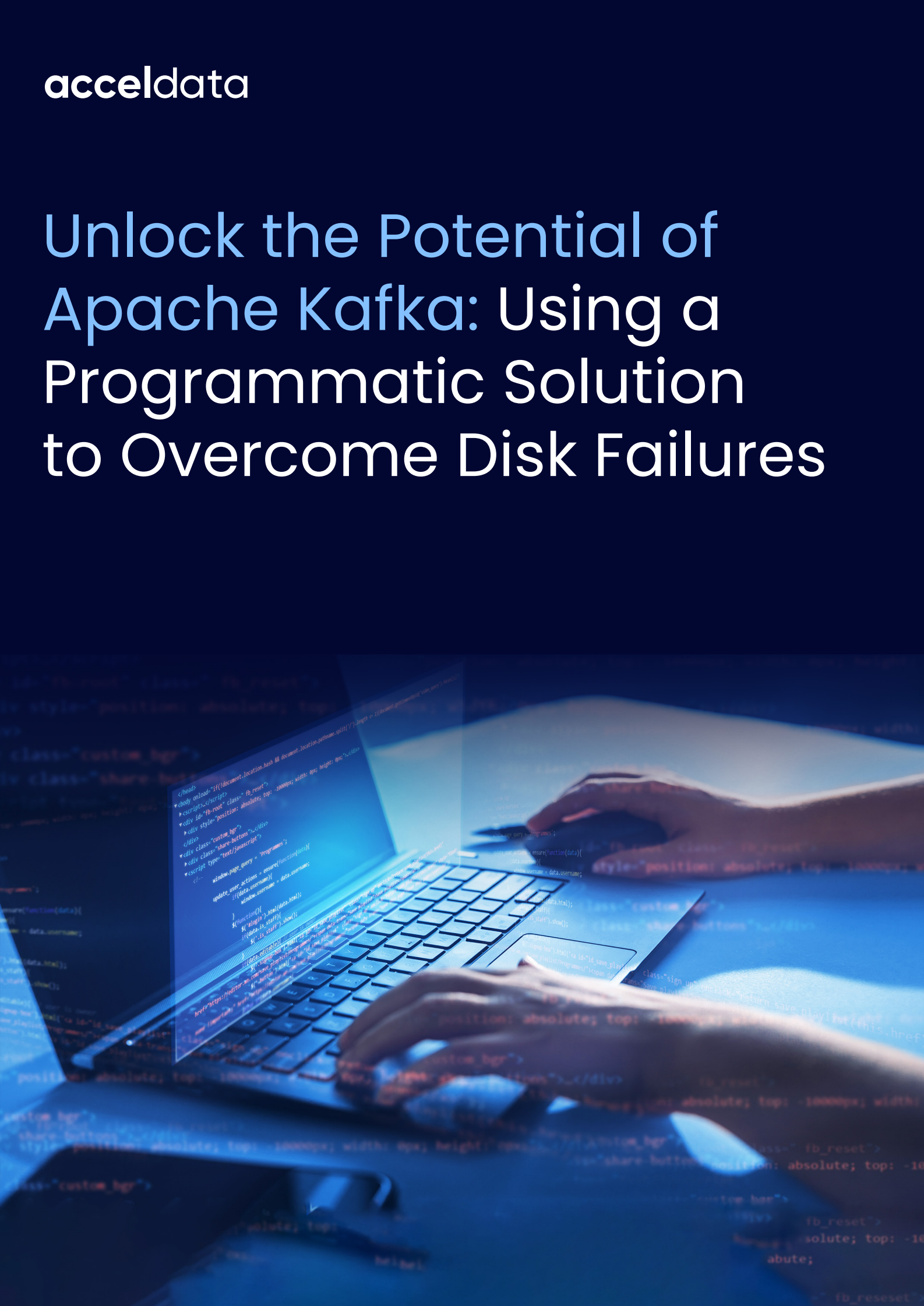


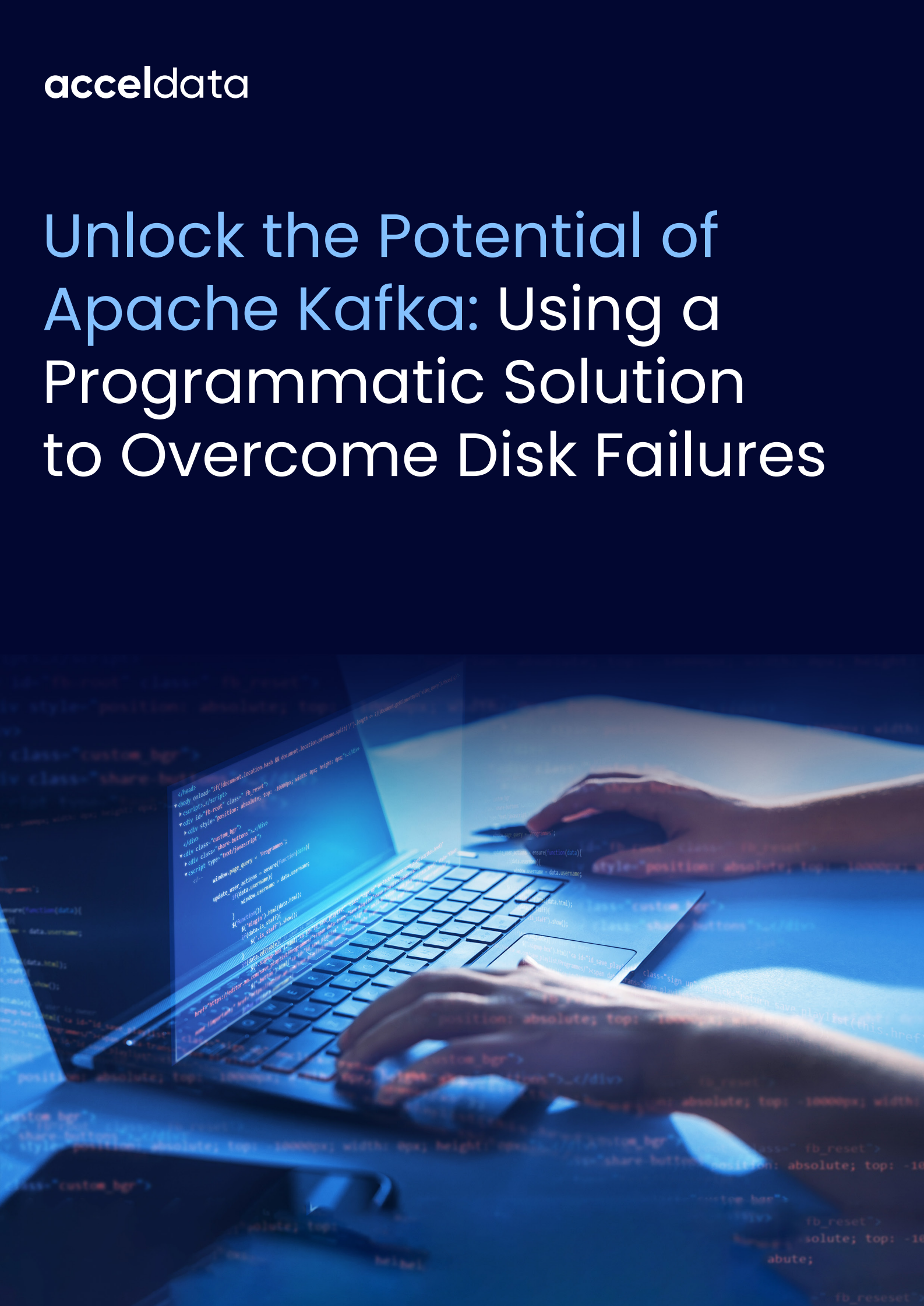
# acceldata

## Unlock the Potential of Apache Kafka: Using a Programmatic Solution to Overcome Disk Failures

A person's hands are shown typing on a laptop keyboard, which is illuminated with a strong blue light. Overlaid on the scene is a semi-transparent digital interface displaying lines of JavaScript code. The code includes DOM manipulation and jQuery-related functions, such as `$(document).ready()`, `$('.share-button')`, and `$('.fb_reset')`. The overall aesthetic is high-tech and digital, with a focus on programming and data management.

# acceldata

## Unlock the Potential of Apache Kafka: Using a Programmatic Solution to Overcome Disk Failures



The background image shows a person's hands typing on a laptop keyboard. The scene is dimly lit with a strong blue light source, creating a high-tech, digital atmosphere. A semi-transparent code editor window is overlaid on the laptop screen, displaying JavaScript code. The code includes DOM manipulation and jQuery-like functions, such as `$(document).ready()`, `$('.share-button').click()`, and `$.ajax()`. The overall aesthetic is modern and tech-oriented, emphasizing data and programming.

# Introduction

Kafka is a widely adopted distributed streaming platform known for its ability to handle large data streams. But, because of its distributed architecture and demanding throughput needs, it faces the risk of disk failures and other issues including network outages, hardware failures, software bugs, configuration errors, overload conditions, and security breaches. All these risks can impact the performance, availability, and data integrity of the Kafka streaming platform.

This technical whitepaper provides a programmatic solution to overcome disk failures in Kafka using ZFS (Zettabyte File System). First, we will discuss the five advantages of using ZFS to counter the disk failure problems in Kafka. Then, we will walk through a 11-step process to implement ZFS as the file system and logical volume manager for Kafka to ensure data reliability by protecting against disk failures.

## The Kafka Fire-and-Forget Model vs. the Three Acknowledgement Models in Kafka

In the Fire and Forget model in Kafka, producers publish messages to a Kafka topic without waiting for any acknowledgment or confirmation from the broker or consumers. Once the message is sent, the producer does not block or wait for a response and can continue with its own processing. The Fire and Forget model prioritizes high throughput and non-blocking communication, but it comes with the tradeoff of potential data loss if messages are not successfully processed.

Kafka provides different **acknowledgment modes** that producers can configure based on their desired tradeoff between data reliability and performance. These acknowledgment modes are defined by the acknowledgment ("acks") configuration parameter. The following are the three common acknowledgment modes:

### 1. Acknowledgment All (acks = all)

When using acknowledgment all mode, producers require acknowledgments from all in-sync replicas (ISRs) of the topic partition after publishing a message. The producer waits for acknowledgments from all replicas before considering the message as successfully published. This mode provides the strongest guarantee of data reliability since it ensures that all replicas have received and acknowledged the message. It offers the highest durability but may impact throughput and latency due to the additional synchronization required.

### 2. Acknowledgment One (acks = 1)

In acknowledgment one mode, producers require acknowledgment from the leader replica of the topic partition after publishing a message. Once the leader replica successfully receives the message, the producer considers it as successfully published. This mode offers a balance between data reliability and performance. It ensures that at least one replica has received the message, providing some level of reliability while maintaining a reasonable level of throughput.

### 3. Acknowledgment Minus One (acks = -1 or acks = all / -1)

Acknowledgment minus one mode is a combination of acknowledgment all and acknowledgment one. With this mode, the producer does not require any acknowledgments before considering the message as successfully published. It immediately continues processing without any acknowledgment delay. However, it still waits for the message to be written to the leader replica for durability. This mode offers the highest throughput but provides the least guarantee of data reliability since it doesn't require any acknowledgment...

### 3. Acknowledgment Minus One (acks = -1 or acks = all / -1)

...Kafka provides different **acknowledgment modes** that producers can configure based on their desired tradeoff between data reliability and performance. These acknowledgment modes are defined by the acknowledgement ("acks") configuration parameter. The following are the three common acknowledgment modes:

*To summarize:*

*If acks=0 -> Just fire and forget. Producer won't wait for an acknowledgement.*

*If acks=1 -> Acknowledgement is sent by the broker when message is successfully written on the leader.*

*If acks=all -> Acknowledgement is sent by the broker when message is successfully written on all replicas*

The choice of acknowledgment mode depends on the desired tradeoff between data reliability and performance. Acknowledgment all provides the strongest durability guarantees but may impact latency and throughput. Acknowledgment one offers a balance between reliability and performance, while acknowledgment minus one prioritizes maximum throughput with potential tradeoffs in data reliability.



## The Tradeoff Between Data Throughput and Data Reliability

As seen above, the Fire-and-Forget model prioritizes speed and does not wait for acknowledgments, which increases throughput but carries a higher risk of data loss. On the other hand, the acknowledgment modes provide varying levels of reliability by introducing acknowledgment processes that can impact latency and potentially reduce throughput.

Ultimately, the choice between the Fire-and-Forget model and the acknowledgment modes depends on the specific requirements of the application. If data loss is unacceptable and higher reliability is crucial, acknowledgment modes like acknowledgement all or acknowledgement one are typically used...



...However, if maximizing throughput and minimizing latency are the primary concerns, the Fire-and-Forget model is used, accepting the associated tradeoff in data reliability.

The tradeoff lies in the fact that ensuring higher data reliability typically comes at the cost of reduced data throughput. Mechanisms like replication, acknowledgment modes, and synchronization among replicas introduce additional overhead and latency, which can impact the overall throughput of the system. On the other hand, prioritizing higher data throughput may involve sacrificing some level of reliability, as faster processing may leave less time for replication or acknowledgment.

Finding the right balance depends on the specific requirements of the use case. Some applications may prioritize maximum throughput, accepting a higher risk of potential data loss or temporary inconsistencies. Other use cases, such as financial systems or critical data pipelines, may prioritize data reliability and durability, even if it comes at the expense of slightly lower throughput. Kafka offers configurable parameters like acknowledgment modes, replication factor, and durability settings to strike the desired balance between data throughput and reliability based on the application's needs.

## Problems Resulting From Disk Failures in Kafka

Apache Kafka is arguably one of the most popular open-source distributed streaming platforms today because of its exceptional capability to handle vast volumes of data streams. Used by over 80% of the Fortune 100, it has countless advantages for any organization that benefits from real-time data, stream processing, integration, or analytics.

However, due to its inherent design as a distributed streaming platform, Kafka operates across multiple nodes and brokers, making it vulnerable to various types of failures, including disk failures. Disk failures in Kafka can result in data loss or corruption and cluster...

...outages, which can prove to be very expensive for organizations.

Unpredictable disk crashes on brokers can cause cluster-wide downtime and affect the availability of partitions. Restoring a Kafka cluster after a disk failure may result in some data loss. Without proper mechanisms in place, there is no built-in parity solution to recover corrupted data blocks. Disk failures may require restarting multiple services at the cluster level. Hence, protecting against disk failures is a major consideration when implementing Kafka in production environments in order to ensure reliability of data.

## What is ZFS?

ZFS (Zettabyte File System) is a combined file system and logical volume manager originally developed by Sun Microsystems (and is now owned by Oracle Corporation). It is an advanced and scalable file system designed to address the limitations and challenges of traditional file systems.

ZFS offers a wide range of features and capabilities that make it popular in various environments, including enterprise storage systems, data centers, and high-performance computing. The benefits of utilizing ZFS as a file system and logical volume manager include its scalability, robust protection against data corruption, support for large storage capacities, efficient data compression, seamless integration of filesystem and volume management functionalities, capabilities for snapshots and copy-on-write clones, continuous integrity checks, and automatic repairs. Additionally, ZFS provides software-defined RAID pools to ensure disk redundancy for enhanced data reliability.

Overall, ZFS provides a robust and scalable file system solution with features that enhance data integrity, reliability, and storage management. Its advanced capabilities make it suitable for a wide range of applications and storage environments.

# ZFS as a Programmatic Solution to Disk Failures in Kafka

The main purpose of this whitepaper is to explore the use of ZFS as a programmatic solution for disk failures in Kafka. Since ZFS offers numerous benefits such as efficient I/O utilization, automatic checksumming, and pooled storage, it is an ideal choice for ensuring data integrity and availability in Kafka deployments.

Implementing ZFS as the file system and logical volume manager for Kafka can greatly enhance data reliability and protection against disk failures. With ZFS, Kafka benefits from efficient I/O utilization, automatic checksumming, and pooled storage. By creating software-defined RAID pools, ZFS ensures data redundancy and availability, even in the face of disk failures. Moreover, the flexibility of adding or removing buffer disks allows for dynamic adaptation to changing storage requirements.

Let's explore some of these advantages of using Kafka on ZFS in detail first, and then we will go through the 11 steps that will help you confidently set up ZFS for Kafka and leverage its robust features to build a reliable and resilient streaming platform.

**Note:** The implementation steps provided in this whitepaper are a summary and should be adapted to suit specific system configurations.

## Functionality Improvements From Using ZFS with Kafka

Using ZFS with Kafka provides the following functionality improvements:

### Improved Broker Read Performance

ZFS offers an advantageous feature in the form of a file system cache that significantly improves the read performance of brokers in Kafka. By utilizing this...

...cache, ZFS can store frequently accessed data in memory, reducing the need to fetch it from disk repeatedly. This caching mechanism effectively minimizes latency and accelerates the retrieval of data, resulting in improved overall broker read performance.

By leveraging ZFS's file system cache, Kafka can handle large-scale data streams more efficiently, enabling faster and more responsive data processing and consumption. This enhancement in read performance helps ensure smoother and more reliable data flow within Kafka, ultimately enhancing the overall performance and efficiency of the distributed streaming platform.

### Efficient I/O Utilization

ZFS is designed to utilize I/O efficiently, ensuring optimal performance and resource utilization. Efficient I/O utilization is the key advantage of using ZFS with Kafka.

ZFS is specifically engineered to make efficient use of I/O operations, resulting in enhanced performance and optimized resource utilization. This efficiency is achieved through various mechanisms within ZFS, such as adaptive read and write algorithms, intelligent caching mechanisms, and the ability to parallelize I/O across multiple disks. By maximizing I/O efficiency, ZFS minimizes latency and improves throughput, enabling Kafka to handle large-scale data streams more effectively. This translates to faster data processing, reduced bottlenecks, and improved overall performance of the Kafka platform.

Whether it's reading data from disk or writing data to disk, ZFS's efficient I/O utilization ensures that the system operates at its full potential, delivering optimal performance for data-intensive workloads in Kafka deployments.

## Pooled Storage

Using ZFS with Kafka provides the pooled storage feature, which enhances data availability and resilience. With ZFS, storage is pooled, meaning that the failure of one disk does not impact the entire Zpool.

Hence, the failure of one disk within the pool does not have a significant impact on the overall storage system. The data stored in the pool is distributed across multiple disks, allowing for redundancy and fault tolerance. In the event of a disk failure, ZFS can automatically reconstruct the data from the remaining disks in the pool, ensuring that data availability is maintained and minimizing the risk of data loss. This pooled storage architecture enhances the resilience of the storage system, making it more reliable and suitable for high-performance and critical data streaming applications like Kafka.

## Automatic Checksumming

Using ZFS with Kafka provides a very useful feature called automatic checksumming. This feature ensures data integrity by automatically performing checksum calculations on the stored data. With checksumming, ZFS can detect if any data blocks become corrupted on the disks. In the event of a corrupted block, ZFS can use the parity information to rebuild the block and restore it to its original state, all without requiring any downtime. This automatic process helps to maintain data integrity and minimize the risk of data loss or inconsistencies.

By seamlessly handling and recovering from corrupted blocks, ZFS provides a reliable and robust solution for protecting data in Kafka deployments, enhancing the overall reliability and availability of the system.

## Virtual Volumes

One of the key features of ZFS is its support for dynamic volume management, specifically through the use of virtual volumes. With virtual volumes, organizations using Kafka can effortlessly add and remove buffer disks as needed, enabling flexible and scalable storage management. This feature empowers users to easily...

...adapt their storage infrastructure to accommodate changing data demands, optimizing resource utilization and minimizing costs.

By providing seamless volume allocation and deallocation, ZFS simplifies the process of managing storage volumes in Kafka environments, allowing organizations to efficiently scale their data processing capabilities while ensuring optimal performance and operational efficiency.

## Snapshots and Clones

ZFS's snapshot and copy-on-write clone capabilities are powerful features of ZFS that offer efficient methods for creating point-in-time copies of Kafka data. These are valuable for various use cases, such as data backups, testing, and data versioning.

With snapshots, organizations can capture the state of their data at a specific moment without the need to copy or duplicate the entire dataset. This enables easy and efficient data backups, as snapshots only store the changes made since the snapshot was taken, saving both time and storage space. Additionally, snapshots can be used for testing purposes, allowing developers to experiment with different configurations or perform regression testing without impacting the production environment.

Clones, on the other hand, are writable snapshots that provide a complete, independent copy of a dataset. This is useful for creating multiple instances of a Kafka cluster for different purposes, such as creating development or testing environments. By offering these snapshot and clone capabilities, ZFS simplifies data management and enhances the flexibility and versatility of Kafka deployments.

# Advantages of Using ZFS with Kafka

The above functionality improvements offer the following advantages:

## Data Integrity

ZFS provides extensive protection against data corruption through its built-in checksumming mechanism. This ensures that data written to Kafka topics remains intact and is not compromised during storage or retrieval.

Data integrity is a critical aspect of any data storage system, and ZFS excels in this regard. With its built-in checksumming mechanism, ZFS offers robust protection against data corruption in Kafka. The checksums are calculated for each block of data written to Kafka topics, allowing ZFS to detect and mitigate any potential corruption issues. This ensures that the integrity of the data is maintained throughout its lifecycle. By automatically verifying the integrity of stored data, ZFS provides peace of mind to users, knowing that their Kafka data is protected against silent data corruption and other potential risks.

This advantage of ZFS contributes to the overall reliability and trustworthiness of Kafka deployments, minimizing the chances of data loss or compromise. The data integrity feature of ZFS helps organizations maintain the consistency and accuracy of their Kafka data, making it a valuable asset for applications that require high levels of data reliability.

## High Storage Capacities

One of the advantages of ZFS is its support for high storage capacities, which is particularly beneficial for Kafka deployments dealing with massive data volumes. With ZFS, organizations can scale their storage infrastructure to accommodate the increasing demands of high-throughput Kafka systems...

This capability ensures that Kafka can efficiently handle and process large-scale data streams without being constrained by storage limitations. This is crucial for high-throughput Kafka deployments that deal with large-scale data streams.

By leveraging ZFS's ability to handle large storage capacities, organizations can seamlessly expand their Kafka clusters as data volumes grow, ensuring continuous and uninterrupted data processing. This scalability feature of ZFS enables Kafka to operate effectively in demanding environments where substantial amounts of data need to be ingested, processed, and distributed in real-time. Overall, ZFS's support for high storage capacities empowers Kafka deployments to handle the challenges posed by massive data volumes with ease and efficiency.

## Efficient Data Compression

Efficient data compression is a notable advantage of ZFS for Kafka deployments. With ZFS's efficient data compression algorithms, organizations can achieve significant reductions in storage space requirements for Kafka topics. This means that less physical storage is needed to store the same amount of data, resulting in improved storage utilization and potentially reducing costs associated with storage resources. ZFS employs advanced compression techniques that efficiently compress data while maintaining its integrity and minimizing performance impact.

By compressing data, organizations can store more data within the same storage infrastructure, allowing for increased scalability and accommodating larger data volumes. This advantage not only saves on storage costs but also enhances overall system performance by reducing the amount of data that needs to be read from or written to disk, resulting in faster data access and improved throughput for Kafka applications.

## Data Consistency and Reduced Risk of Data Corruption

Another significant advantage of ZFS is its ability to provide data consistency and reduce the risk of data corruption. ZFS continuously performs integrity checks on stored data, verifying the integrity of each block using checksums. If any errors are detected, ZFS automatically repairs the corrupted blocks using redundant copies or parity information. This proactive approach to data integrity ensures that the data remains consistent and reliable, reducing the chances of data corruption.

By mitigating the risk of data corruption, ZFS enhances the overall reliability and durability of the data stored in Kafka deployments. This advantage is particularly crucial for mission-critical applications where data integrity is paramount, as it helps maintain the integrity and accuracy of the data throughout its lifecycle.

Additionally, ZFS's automatic repair mechanism minimizes the need for manual intervention, simplifying data management tasks and reducing administrative overhead.

## Data Availability and Durability

Another notable advantage of ZFS is its ability to provide data availability and durability through software-defined RAID pools. With these pools, administrators can easily configure disk redundancy, which safeguards Kafka data against disk failures. By distributing data across multiple disks with redundancy, ZFS ensures that even if one disk fails, the data remains accessible and intact, minimizing the risk of data loss or service disruption. This high availability and durability are crucial for Kafka deployments, as it enables continuous data processing and minimizes the impact of hardware failures.

ZFS's software-defined approach also offers flexibility in adapting to changing storage needs, allowing administrators to add or remove disks from the RAID pool without disrupting data availability. With ZFS, organizations can confidently rely on a robust and resilient storage solution for their Kafka data, ensuring uninterrupted operations and data integrity.

## Implementing Kafka on ZFS

The next few sections will guide you through an 11-step process for implementing ZFS using the following steps:

1. Check the current version of CentOS installed on the server.
2. Add the ZFSOnLinux repository using the provided command.
3. Update the ZFS repository to use kABI as the module loader.
4. Install the ZFS file system on CentOS 7 using the given command.
5. Load the ZFS kernel module and verify its status.
6. Create two ZFS pools (data1 and data2) based on the server's disk setup.
7. Fine-tune ZFS configurations for better performance and reliability.
8. Set L2ARC (Level 2 Adaptive Replacement Cache) configurations for improved caching.
9. Enable auto-import on reboot for ZFS.
10. Reboot the server to finalize the ZFS setup.
11. Set up alerts from Pulse to monitor and receive notifications for kernel-level errors and disk issues.

**Please note that step 11 mentions setting up alerts specifically for kernel-level errors and bad sectors on disks to ensure timely detection and mitigation of potential issues.**



# Step-by-Step Implementation for Using Kafka on ZFS

The following will guide you through these 11 steps using screenshots:

## STEP 1: Check the current version of CentOS installed on the server

To check the current version of CentOS installed on the server, you can use the command "cat /etc/redhat-release" in the terminal. This will display the CentOS Linux release information, including the version number and any additional details. It helps in identifying the specific CentOS version running on the server, which is useful for troubleshooting and ensuring compatibility with software and updates.

```
[root@server1 ~]# cat /etc/redhat-release
CentOS Linux release 7.9.2009 (Core)
```

## STEP 2: Add the ZFSOnLinux repository using the provided command

To add the ZFSOnLinux repository, you can use the provided command: "yum install http://download.zfsnlinux.org/epel/zfs-release.el7\_9.noarch.rpm". This command downloads and installs the repository package, enabling you to access and install the ZFS file system on your CentOS 7 server.

```
[root@server1 ~]# yum install
http://download.zfsnlinux.org/epel/zfs-release.el7_9.noarch.rp
m
Failed to set locale, defaulting to C
Loaded plugins: fastestmirror
zfs-release.el7_9.noarch.rpm
| 5.3 kB 00:00:00
Examining
/var/tmp/yum-root-YaPpb0/zfs-release.el7_9.noarch.rpm:
zfs-release-1-7.9.noarch
Marking /var/tmp/yum-root-YaPpb0/zfs-release.el7_9.noarch.rpm
to be installed
Resolving Dependencies
--> Running transaction check
--> Package zfs-release.noarch 0:1-7.9 will be installed
--> Finished Dependency Resolution
[zfs]

Dependencies Resolved

=====
Package                               Arch
Version                               Repository
Size
=====
Installing:
zfs-release                           noarch
1-7.9
/zfs-release.el7_9.noarch             2.9 k

Transaction Summary
=====
```

## STEP 3: Update the ZFS repository to use kABI as the module loader

Updating the ZFS repository to use kABI as the module loader involves disabling the DKMS (Dynamic Kernel Module System) based repository and enabling the kABI (Kernel Application Binary Interface) based repository. This step is important because using DKMS requires recompiling the ZFS module every time the CentOS kernel is updated, while kABI eliminates the need for recompilation. By enabling kABI, the ZFS module can be loaded into the kernel without the need for frequent recompilation, providing a more efficient and convenient solution.

When we install the ZFS repository on CentOS 7, the DKMS (Dynamic Kernel Module System) based repository is enabled by default. So we have to disable the DKMS based repository and enable the kABI (Kernel Application Binary Interface) based repository.

Why kABI? DKMS and kABI are two ways a ZFS module can be loaded into the kernel. If DKMS is used then if CentOS kernel is ever updated, the ZFS module will need to be recompiled again. But with kABI, no recompilation is necessary.

This is the final configuration:

In zfs section set, enabled=0

In zfs-kmod section set, enabled=1

```
[root@server1 yum.repos.d]# cat /etc/yum.repos.d/zfs.repo
[zfs]
name=ZFS on Linux for EL7 - dkms
baseurl=http://download.zfsnlinux.org/epel/7.9/$basearch/
enabled=0
metadata_expire=7d
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-zfsnlinux

[zfs-kmod]
name=ZFS on Linux for EL7 - kmod
baseurl=http://download.zfsnlinux.org/epel/7.9/kmod/$basearch/
enabled=1
metadata_expire=7d
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-zfsnlinux
```

## STEP 4: Install the ZFS file system on CentOS 7 using the given command

To install the ZFS file system on CentOS 7, you can use the command "yum install zfs". This command will download and install the necessary packages and dependencies for ZFS. Once the installation is complete, you can verify the installation by checking the ZFS kernel module using the "lsmod | grep zfs" command.

```
[root@server1 yum.repos.d]# yum install zfs
Failed to set locale, defaulting to C
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: mirrors.nxtgen.com
 * extras: mirrors.nxtgen.com
 * updates: mirrors.nxtgen.com
zfs-kmod
| 2.9 kB 00:00:00
zfs-kmod/x86_64/primary_db
| 175 kB 00:00:01
Resolving Dependencies
--> Running transaction check
.
.
.
=====
Package Arch
Version Size
Repository
=====
Installing:
zfs x86_64
2.0.7-1.el7 zfs-kmod
595 k
Installing for dependencies:
kmod-zfs x86_64
2.0.7-1.el7 zfs-kmod
1.4 M
libnvpair3 x86_64
2.0.7-1.el7 zfs-kmod
32 k
libuutil3 x86_64
```

```
Transaction Summary
=====
Install 1 Package (+6 Dependent packages)
.
.
.
/sbin/dracut: line 682: warning: setlocale: LC_CTYPE:
cannot change locale (): No such file or directory
Installing : zfs-2.0.7-1.el7.x86_64
7/7
Verifying : kmod-zfs-2.0.7-1.el7.x86_64
1/7
Verifying : sysstat-10.1.5-19.el7.x86_64
2/7
Verifying : zfs-2.0.7-1.el7.x86_64
3/7
Verifying : libuutil3-2.0.7-1.el7.x86_64
4/7
Verifying : libzpool4-2.0.7-1.el7.x86_64
5/7
Verifying : libzfs4-2.0.7-1.el7.x86_64
6/7
Verifying : libnvpair3-2.0.7-1.el7.x86_64
7/7

Installed:
zfs.x86_64 0:2.0.7-1.el7

Dependency Installed:
kmod-zfs.x86_64 0:2.0.7-1.el7 libnvpair3.x86_64
0:2.0.7-1.el7 libuutil3.x86_64 0:2.0.7-1.el7 libzfs4.x86_64
0:2.0.7-1.el7 libzpool4.x86_64 0:2.0.7-1.el7 sysstat.x86_64
0:10.1.5-19.el7

Complete!
```

## STEP 5: Load the ZFS kernel module and verify its status

To load the ZFS kernel module, run the command "modprobe zfs" in the terminal. After loading the module, you can verify its status by running "lsmod | grep zfs" to check if it appears in the list of loaded modules. The output should display information about the ZFS module, confirming that it has been successfully loaded.

```
[root@server1 yum.repos.d]# lsmod | grep zfs
[root@server1 yum.repos.d]# modprobe zfs
[root@server1 yum.repos.d]# lsmod | grep zfs
zfs 4224878 0
zunicode 331170 1 zfs
zzstd 460780 1 zfs
zlua 151526 1 zfs
zcommon 94285 1 zfs
znvpair 94388 2 zfs,zcommon
zavl 15698 1 zfs
icp 301775 1 zfs
spl 96750 6 icp,zfs,zavl,zzstd,zcommon,znvpair
```

There are three types of pools that can be created in ZFS:

- Stripped Pool
- Mirrored Pool
- Raid Pool

Reference Link for more information related to types of zfs pool as per the required use case :

<https://www.thegeekdiary.com/zfs-tutorials-creating-zfs-pool-and-file-systems/>

We will choose Raid Pool as all Raid-ZX in ZFS works similarly with the difference in disk tolerance. The main difference between Raid-Z1, Raid-Z2 and Raid-Z3 are they can tolerate a maximum of 1, 2 and 3 disk failure respectively without any data loss.

**ashift:** This ZFS property allows you to manually set the sector size on SSD, ashift=12, means sector size would be 2<sup>12</sup>, 4096 bytes, no penalty to setting ashift too high but if ashift is set too low, an astronomical read/write amplification penalty is incurred—writing a 512 byte "sectors" to a 4KiB real sector means having to write the first "sector", then read the 4KiB sector, modify it with the second 512 byte "sector", write it back out to a \*new\* 4KiB sector, and so forth, for every single write.

## STEP 6: Create two ZFS pools (data1 and data2) based on the server's disk setup

Create 2 zpools (data1 and data2) based on the disk OS is setup:

```
sudo zpool create -o ashift=12 -f data1
raidz1 sda sdb sdc sdd
sudo zpool create -o ashift=12 -f data2
raidz1 sdf sdg sdh sdi
```

- OR -

```
sudo zpool create -o ashift=12 -f data1
raidz1 sdb sdc sdd sde
sudo zpool create -o ashift=12 -f data2
raidz1 sdg sdh sdi sdj
```

```
[root@server1 ~]# zpool create -o ashift=12 -f data1 raidz1
sdb sdc
[root@server1 ~]# zpool create -o ashift=12 -f data2 raidz1
sdd sde
[root@server1 ~]# zpool status
pool: data1
state: ONLINE
config:

NAME        STATE      READ WRITE CKSUM
data1       ONLINE    0     0     0
raidz1-0    ONLINE    0     0     0
  sdb       ONLINE    0     0     0
  sdc       ONLINE    0     0     0

errors: No known data errors

pool: data2
state: ONLINE
config:

NAME        STATE      READ WRITE CKSUM
data2       ONLINE    0     0     0
raidz1-0    ONLINE    0     0     0
  sdd       ONLINE    0     0     0
  sde       ONLINE    0     0     0

errors: No known data errors
[root@mstr3-hdp1fs ~]# zfs list
NAME      USED  AVAIL  REFER  MOUNTPOINT
data1     384K  19.1G  96K    /data1
data2     372K  19.1G  96K    /data2
[root@mstr3-hdp1fs ~]# df -hP
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        16G   0    16G   0% /dev
tmpfs           16G   0    16G   0% /dev/shm
tmpfs           16G  8.9M   16G   1% /run
tmpfs           16G   0    16G   0%
/sys/fs/cgroup
/dev/mapper/centos-root 98G  1.4G  96G   2% /
/dev/sda1       1014M 150M  865M  15% /boot
tmpfs           3.2G   0    3.2G   0% /run/user/0
data1           20G  128K   20G   1% /data1
data2           20G  128K   20G   1% /data2
```

## STEP 7: Fine-tune ZFS configurations for better performance and reliability

In this step, several ZFS configurations are adjusted to enhance performance and reliability. First, the "atime" property is disabled to improve I/O performance for frequently accessed small files. Second, the "redundant\_metadata" property, which stores an additional copy of most block metadata, is set to "most" to enhance random write performance. Third, the extended attributes and process execution within the "data1" file system are controlled by disabling "xattr" and "exec" respectively. Lastly, the "overlay" property is set to "off" on every pool to expedite mount failure recovery. These fine-tuning measures aim to achieve better overall performance and reliability in the ZFS file system.

### sudo zfs set atime=off data1

Disabling atime is a great way to improve I/O performance on filesystems with lots of small files that are accessed frequently.

### sudo zfs set redundant\_metadata=most data1

When set to most, ZFS stores an extra copy of most types of block metadata. This can improve performance of random writes, because less metadata must be written.

### sudo zfs set xattr=off data1

Controls whether extended attributes are enabled for the ZFS file system.

### sudo zfs set exec=off data1

Controls whether processes can be executed from within this file system.

### sudo zfs set overlay=off data1

Set overlay=off on every single pool, for a faster mount failure recovery.

```
sudo zfs set atime=off data2
```

```
sudo zfs set redundant_metadata=most
data2
```

```
sudo zfs set xattr=off data2
```

```
sudo zfs set exec=off data2
```

```
sudo zfs set overlay=off data2
```

## STEP 8: Set L2ARC (Level 2 Adaptive Replacement Cache) configurations for improved caching

The L2ARC is the 2nd Level Adaptive Replacement Cache, and is an SSD based cache that is accessed before reading from the much slower pool disks. The L2ARC is currently intended for random read workloads.

```
sudo vi /etc/modprobe.d/zfs.conf
```

```
options zfs zfs_arc_max=322122547200 (As per the max
memory from the Kafka Broker server)
options zfs l2arc_write_max=536870912 (Max write bytes per
interval)
options zfs zfs_vdev_async_read_max_active=30 (Maximum
asynchronous read I/O operations active to each device.)
options zfs zfs_vdev_async_write_max_active=100 (Maximum
asynchronous write I/O operations active to each device)
options zfs zfs_vdev_sync_read_max_active=30 (Maximum
synchronous read I/O operations active to each device)
options zfs zfs_vdev_sync_write_max_active=100 (Maximum
synchronous write I/O operations active to each device)
options zfs zfs_vdev_max_active=4000 (The maximum number of
I/O operations active to each device. Ideally, this will be
at least the sum of each queue's max_active.)
```

## STEP 9: Enable the auto import on reboot

```
systemctl enable zfs-import-cache -l
systemctl start zfs-import-cache -l
```

## STEP 10: Alert Setup from Pulse

ZFS setup for Kafka broker is completed now and it can be used for log.dir /data1 and /data2 with raidz1 to sustain 1 disk failure.

**Important note:** Please keep 2 disks in the buffer for the hot swapping with the bad disk in the zpool.

## STEP 11: Setup Alert for any Kernel Level Error which includes exceptions for bad sector within the disk before crashing

## Additional Context About the 11-step Implementation

This whitepaper discusses the challenges faced with normal file systems (such as XFS and ext4 on SSDs) in a Kafka cluster. When the "acks=0" (Acknowledgment One) configuration is set at the cluster level, a disk crash on the broker can lead to a cluster outage and result in leader -1 for affected partitions. Restoring the cluster incurs data loss when "unclean.leader.election.enable=true" is set, and there is no built-in parity mechanism to recover corrupted data blocks. Multiple cluster-level service restarts are required in such scenarios.

The proposed resolution suggests an architectural change from XFS and ext4 to ZFS as the file system. This change helps in sustaining disk failures and uncertain outages. Data loss is mitigated through the use of parity checksums to recover or rebuild corrupted data blocks. Additionally, the proposed solution eliminates the need for cluster-level service restarts, streamlining the recovery process. However, it's important to note that the drawback of implementing ZFS systems is the increased storage requirement on each node. This includes the need to allocate space for spare disks and parity, impacting overall storage capacity.



## Summary

Organizations have to make constant tradeoffs between data throughput/performance and data reliability, and as a result of these tradeoffs, data teams encounter problems in their data landscapes - such as disk failures in their Kafka environments. This technical whitepaper offers a programmatic solution to address disk failures in Kafka by leveraging the ZFS (Zettabyte File System).

In this whitepaper, we explored how Kafka, a popular distributed streaming platform, often faces challenges related to disk failures and other failures due to its distributed architecture and high throughput requirements. This whitepaper highlights the notable functionality improvements and key advantages of adopting ZFS with Kafka to tackle these issues and provides an 11-step guide to implementing ZFS as the file system and logical volume manager for Kafka. By incorporating ZFS, organizations can enhance data reliability and safeguard against disk failures, ensuring the uninterrupted operation of Kafka's large-scale data streams.

In recent years, data has become increasingly critical and invaluable across various industries and sectors. As organizations make constant tradeoffs between data throughput and data reliability, a continuous data observability platform like Acceldata can be invaluable in helping data engineers and data reliability engineers [monitor and optimize data failure metrics](#).

Acceldata's [comprehensive, multi-layered data observability platform](#) continually monitors and validates data pipelines from end to end for both performance bottlenecks and [data reliability](#) issues. If a hard disk fails, Acceldata will instantly recognize the issue, alert the appropriate data engineer or admin so that they can turn care of the situation or can trigger actions that can automatically remediate the issue. In this day and age, when the reliability and availability of data is measured in nanoseconds, having the assurance of Acceldata as your continuous and comprehensive data observability is key.

## About Acceldata:

Acceldata is the market leader in **enterprise data observability**. With Acceldata's multi-layered data observability solution, enterprises gain comprehensive insights into their data stack to improve data quality, pipeline reliability, compute performance, and spend efficiency. Acceldata is the only multidimensional and industry agnostic Data Observability solution that presents a clear solution to diverse predicaments.



## Our Customers



ORACLE

true

HERSHEY'S

dun & bradstreet

& many more

Interested in seeing Acceldata in action? Please schedule a personalized demonstration or sign up for a 30-day free trial.

Get started