Day 4: Thin Provisioning

- LVM
- Thin Provisioning
-
- Go
-
-

1. LVM

1.1

LVM Copy-on-Write (CoW)

- :
- :
- :

```
├──  (Exception Table)
│   ├──  →
│   └──
├──
│   ├── UUID
│   └──
└──
    ├──
    └──
```

1.2

vs Thin

Thin

1.3

- : + +
- : 7 + 4 + 12
- :

2. Thin Provisioning

2.1 Thin Provisioning

```
Thin Pool
├──  (Metadata Device)
│   ├──
│   ├──
│   └──
├──  (Data Device)
│   ├──
```

```
│   └──
└── Thin Volume
    ├──
    └──
```

- :
- : chunk
- :
- :

2.2

```
# Thin Pool
chunk_size=64K          #
low_water_mark=20%      #
error_if_no_space=yes   #
```

□ □□□□

1.

1.1

```
# 1.
mkdir -p /mnt/data
mount /dev/storage_vg/data_lv /mnt/data
echo "Original data content" > /mnt/data/test.txt
dd if=/dev/zero of=/mnt/data/large_file bs=1M count=100

# 2.  -  20%
lvcreate -L 400M -s -n data_lv_backup /dev/storage_vg/data_lv

# 3.
lvdisplay /dev/storage_vg/data_lv_backup
lvs -o +snap_percent storage_vg

# 4.
echo "Modified content" > /mnt/data/test.txt
mkdir /mnt/snapshot
mount /dev/storage_vg/data_lv_backup /mnt/snapshot
cat /mnt/snapshot/test.txt  #
```

1.2

```
#
lvextend -L +200M /dev/storage_vg/data_lv_backup

#
watch 'lvs -o +snap_percent storage_vg'

#
dmsetup table storage_vg-data_lv_backup
dmsetup status storage_vg-data_lv_backup
```

2. Thin Provisioning

2.1 Thin Pool Thin Volume

```
# 1.  Thin Pool ()
# :  0.1%  1%
lvcreate -L 100M -n thin_meta storage_vg
lvcreate -L 8G -n thin_data storage_vg

# 2.  Thin Pool
lvconvert --type thin-pool --poolmetadata storage_vg/thin_meta storage_vg/thin_data
```

```
lvrename storage_vg/thin_data storage_vg/thin_pool

# 3.  Thin Pool
lvchange --monitor y storage_vg/thin_pool
lvs -o +seg_monitor storage_vg

# 4.  Thin Volume
lvcreate -V 10G -T storage_vg/thin_pool -n thin_lv1
lvcreate -V 15G -T storage_vg/thin_pool -n thin_lv2

# 5.
mkfs.ext4 /dev/storage_vg/thin_lv1
mkdir -p /mnt/thin1 /mnt/thin2
mount /dev/storage_vg/thin_lv1 /mnt/thin1
```

2.2 Thin

```
# 1.  Thin  ()
lvcreate -s -n thin_lv1_snap1 storage_vg/thin_lv1

# 2.
dd if=/dev/urandom of=/mnt/thin1/test_data bs=1M count=500

# 3.
lvcreate -s -n thin_lv1_snap2 storage_vg/thin_lv1

# 4.
lvs -o +data_percent,metadata_percent storage_vg
```

3.

3.1

```
#  LVM
vim /etc/lvm/lvm.conf

#
activation {
    thin_pool_autoextend_threshold = 80
    thin_pool_autoextend_percent = 20
    monitoring = 1
}

#
systemctl enable lvm2-monitor
systemctl start lvm2-monitor
```

3.2 (TRIM/DISCARD)

```
#  DISCARD
tune2fs -o discard /dev/storage_vg/thin_lv1

#  TRIM
fstrim -v /mnt/thin1

#  TRIM
echo '0 2 * * 0 root /usr/sbin/fstrim -a' >> /etc/crontab
```

Go

1. LVM

1.1

```
// filepath: internal/snapshot/snapshot.go
package snapshot

import (
```

```
    "encoding/json"
    "fmt"
    "os/exec"
    "regexp"
    "strconv"
    "strings"
    "time"
)

// SnapshotInfo
type SnapshotInfo struct {
    Name         string    `json:"name"`
    VGName       string    `json:"vg_name"`
    OriginLV     string    `json:"origin_lv"`
    Size         string    `json:"size"`
    UsedPercent  float64   `json:"used_percent"`
    Status       string    `json:"status"`
    CreatedTime  time.Time `json:"created_time"`
    IsActive     bool      `json:"is_active"`
}

// SnapshotManager
type SnapshotManager struct {
    DefaultSize    string
    RetentionDays  int
    AutoExtend     bool
    ExtendPercent  int
}

// NewSnapshotManager
func NewSnapshotManager() *SnapshotManager {
    return &SnapshotManager{
        DefaultSize:   "20%ORIGIN",  // 20%
        RetentionDays: 7,            // 7
        AutoExtend:    true,         //
        ExtendPercent: 20,           // 20%
    }
}

// CreateSnapshot
func (sm *SnapshotManager) CreateSnapshot(vgName, lvName, snapshotName string, size string) error {
    if size == "" {
        size = sm.DefaultSize
    }

    originLV := fmt.Sprintf("/dev/%s/%s", vgName, lvName)

    //  lvcreate
    cmd := exec.Command("lvcreate", "-L", size, "-s", "-n", snapshotName, originLV)

    output, err := cmd.CombinedOutput()
    if err != nil {
        return fmt.Errorf(": %v, : %s", err, string(output))
    }

    fmt.Printf(": %s\n", snapshotName)
    return nil
}

// ListSnapshots
func (sm *SnapshotManager) ListSnapshots(vgName string) ([]SnapshotInfo, error) {
    //  lvs
    cmd := exec.Command("lvs", "--noheadings", "--separator=|",
        "-o", "lv_name,vg_name,origin,lv_size,snap_percent,lv_attr", vgName)

    output, err := cmd.Output()
    if err != nil {
        return nil, fmt.Errorf(": %v", err)
    }

    var snapshots []SnapshotInfo
    lines := strings.Split(strings.TrimSpace(string(output)), "\n")

    for _, line := range lines {
        fields := strings.Split(strings.TrimSpace(line), "|")
```

```
        if len(fields) < 6 {
            continue
        }

        //  ( 's')
        if !strings.Contains(fields[5], "s") {
            continue
        }

        usedPercent, _ := strconv.ParseFloat(strings.TrimSpace(fields[4]), 64)

        snapshot := SnapshotInfo{
            Name:        strings.TrimSpace(fields[0]),
            VGName:      strings.TrimSpace(fields[1]),
            OriginLV:    strings.TrimSpace(fields[2]),
            Size:        strings.TrimSpace(fields[3]),
            UsedPercent: usedPercent,
            Status:      strings.TrimSpace(fields[5]),
            IsActive:    strings.Contains(fields[5], "a"),
        }

        snapshots = append(snapshots, snapshot)
    }

    return snapshots, nil
}

// ExtendSnapshot
func (sm *SnapshotManager) ExtendSnapshot(vgName, snapshotName string, extendSize string) error {
    snapshotPath := fmt.Sprintf("/dev/%s/%s", vgName, snapshotName)

    cmd := exec.Command("lvextend", "-L", "+"+extendSize, snapshotPath)
    output, err := cmd.CombinedOutput()
    if err != nil {
        return fmt.Errorf(": %v, : %s", err, string(output))
    }

    fmt.Printf(": %s  %s\n", snapshotName, extendSize)
    return nil
}

// MonitorSnapshots
func (sm *SnapshotManager) MonitorSnapshots(vgName string, threshold float64) error {
    snapshots, err := sm.ListSnapshots(vgName)
    if err != nil {
        return err
    }

    for _, snapshot := range snapshots {
        if snapshot.UsedPercent > threshold {
            fmt.Printf(":  %s  %.2f%%  %.2f%%\n",
                snapshot.Name, snapshot.UsedPercent, threshold)

            if sm.AutoExtend {
                extendSize := fmt.Sprintf("%d%%ORIGIN", sm.ExtendPercent)
                err := sm.ExtendSnapshot(snapshot.VGName, snapshot.Name, extendSize)
                if err != nil {
                    fmt.Printf(": %v\n", err)
                }
            }
        }
    }

    return nil
}

// RemoveSnapshot
func (sm *SnapshotManager) RemoveSnapshot(vgName, snapshotName string) error {
    snapshotPath := fmt.Sprintf("/dev/%s/%s", vgName, snapshotName)

    cmd := exec.Command("lvremove", "-f", snapshotPath)
    output, err := cmd.CombinedOutput()
    if err != nil {
        return fmt.Errorf(": %v, : %s", err, string(output))
    }
```

```
    fmt.Printf(": %s\n", snapshotName)
    return nil
}
```

1.2

```
// filepath: internal/snapshot/policy.go
package snapshot

import (
    "fmt"
    "regexp"
    "sort"
    "strings"
    "time"
)

// SnapshotPolicy
type SnapshotPolicy struct {
    VGName           string        `json:"vg_name"`
    LVName           string        `json:"lv_name"`
    Schedule         string        `json:"schedule"`        // cron
    RetentionDays    int           `json:"retention_days"`
    SnapshotSize     string        `json:"snapshot_size"`
    NamePrefix       string        `json:"name_prefix"`
    MaxSnapshots     int           `json:"max_snapshots"`
    AutoCleanup      bool          `json:"auto_cleanup"`
}

// PolicyManager
type PolicyManager struct {
    policies []SnapshotPolicy
    manager  *SnapshotManager
}

// NewPolicyManager
func NewPolicyManager(manager *SnapshotManager) *PolicyManager {
    return &PolicyManager{
        policies: make([]SnapshotPolicy, 0),
        manager:  manager,
    }
}

// AddPolicy
func (pm *PolicyManager) AddPolicy(policy SnapshotPolicy) {
    pm.policies = append(pm.policies, policy)
}

// ExecutePolicy
func (pm *PolicyManager) ExecutePolicy(policy SnapshotPolicy) error {
    //  ()
    timestamp := time.Now().Format("20060102-150405")
    snapshotName := fmt.Sprintf("%s-%s", policy.NamePrefix, timestamp)

    //
    err := pm.manager.CreateSnapshot(policy.VGName, policy.LVName,
        snapshotName, policy.SnapshotSize)
    if err != nil {
        return fmt.Errorf(": %v", err)
    }

    //
    if policy.AutoCleanup {
        err = pm.CleanupOldSnapshots(policy)
        if err != nil {
            fmt.Printf(": %v\n", err)
        }
    }

    return nil
}
```

```
// CleanupOldSnapshots
func (pm *PolicyManager) CleanupOldSnapshots(policy SnapshotPolicy) error {
    snapshots, err := pm.manager.ListSnapshots(policy.VGName)
    if err != nil {
        return err
    }

    //
    var policySnapshots []SnapshotInfo
    for _, snapshot := range snapshots {
        if strings.HasPrefix(snapshot.Name, policy.NamePrefix) &&
           snapshot.OriginLV == policy.LVName {
            policySnapshots = append(policySnapshots, snapshot)
        }
    }

    //  ()
    sort.Slice(policySnapshots, func(i, j int) bool {
        return extractTimestamp(policySnapshots[i].Name) >
               extractTimestamp(policySnapshots[j].Name)
    })

    //
    cutoffTime := time.Now().AddDate(0, 0, -policy.RetentionDays)

    for i, snapshot := range policySnapshots {
        snapshotTime := extractTimestamp(snapshot.Name)

        //
        if i >= policy.MaxSnapshots || snapshotTime.Before(cutoffTime) {
            err := pm.manager.RemoveSnapshot(policy.VGName, snapshot.Name)
            if err != nil {
                fmt.Printf(" %s : %v\n", snapshot.Name, err)
            } else {
                fmt.Printf(": %s\n", snapshot.Name)
            }
        }
    }

    return nil
}

// extractTimestamp
func extractTimestamp(snapshotName string) time.Time {
    // : prefix-20060102-150405
    re := regexp.MustCompile(`(\d{8}-\d{6})`)
    matches := re.FindStringSubmatch(snapshotName)

    if len(matches) > 1 {
        t, err := time.Parse("20060102-150405", matches[1])
        if err == nil {
            return t
        }
    }

    return time.Time{} //
}
```

2. Thin Provisioning

2.1 Thin Pool

```
// filepath: internal/thin/monitor.go
package thin

import (
    "encoding/json"
    "fmt"
    "os/exec"
    "strconv"
    "strings"
    "time"
)
```

```
// ThinPoolInfo Thin Pool
type ThinPoolInfo struct {
    Name             string  `json:"name"`
    VGName           string  `json:"vg_name"`
    DataSize         string  `json:"data_size"`
    DataUsedPercent  float64 `json:"data_used_percent"`
    MetaSize         string  `json:"meta_size"`
    MetaUsedPercent  float64 `json:"meta_used_percent"`
    ChunkSize        string  `json:"chunk_size"`
    DiscardPassdown  bool    `json:"discard_passdown"`
    ZeroDetection    bool    `json:"zero_detection"`
}

// ThinVolumeInfo Thin Volume
type ThinVolumeInfo struct {
    Name         string  `json:"name"`
    VGName       string  `json:"vg_name"`
    PoolName     string  `json:"pool_name"`
    VirtualSize  string  `json:"virtual_size"`
    UsedPercent  float64 `json:"used_percent"`
    DeviceID     int     `json:"device_id"`
}

// ThinMonitor Thin
type ThinMonitor struct {
    DataThreshold     float64
    MetadataThreshold float64
    CheckInterval     time.Duration
    AlertCallback     func(alert string)
}

// NewThinMonitor
func NewThinMonitor() *ThinMonitor {
    return &ThinMonitor{
        DataThreshold:     80.0,  //
        MetadataThreshold: 90.0,  //
        CheckInterval:     time.Minute * 5,
    }
}

// GetThinPools  Thin Pool
func (tm *ThinMonitor) GetThinPools() ([]ThinPoolInfo, error) {
    cmd := exec.Command("lvs", "--noheadings", "--separator=|",
        "-o", "lv_name,vg_name,lv_size,data_percent,metadata_percent,chunk_size,discards,zero",
        "-S", "lv_layout=pool")

    output, err := cmd.Output()
    if err != nil {
        return nil, fmt.Errorf(" Thin Pool : %v", err)
    }

    var pools []ThinPoolInfo
    lines := strings.Split(strings.TrimSpace(string(output)), "\n")

    for _, line := range lines {
        if strings.TrimSpace(line) == "" {
            continue
        }

        fields := strings.Split(strings.TrimSpace(line), "|")
        if len(fields) < 8 {
            continue
        }

        dataPercent, _ := strconv.ParseFloat(strings.TrimSpace(fields[3]), 64)
        metaPercent, _ := strconv.ParseFloat(strings.TrimSpace(fields[4]), 64)

        pool := ThinPoolInfo{
            Name:             strings.TrimSpace(fields[0]),
            VGName:           strings.TrimSpace(fields[1]),
            DataSize:         strings.TrimSpace(fields[2]),
            DataUsedPercent:  dataPercent,
            MetaUsedPercent:  metaPercent,
            ChunkSize:        strings.TrimSpace(fields[5]),
```

```
            DiscardPassdown:  strings.TrimSpace(fields[6]) == "passdown",
            ZeroDetection:    strings.TrimSpace(fields[7]) == "detect",
        }

        pools = append(pools, pool)
    }

    return pools, nil
}

// GetThinVolumes  Thin Pool  Thin Volume
func (tm *ThinMonitor) GetThinVolumes(poolName string) ([]ThinVolumeInfo, error) {
    cmd := exec.Command("lvs", "--noheadings", "--separator=|",
        "-o", "lv_name,vg_name,pool_lv,lv_size,data_percent,lv_device_id",
        "-S", fmt.Sprintf("pool_lv=%s", poolName))

    output, err := cmd.Output()
    if err != nil {
        return nil, fmt.Errorf(" Thin Volume : %v", err)
    }

    var volumes []ThinVolumeInfo
    lines := strings.Split(strings.TrimSpace(string(output)), "\n")

    for _, line := range lines {
        if strings.TrimSpace(line) == "" {
            continue
        }

        fields := strings.Split(strings.TrimSpace(line), "|")
        if len(fields) < 6 {
            continue
        }

        usedPercent, _ := strconv.ParseFloat(strings.TrimSpace(fields[4]), 64)
        deviceID, _ := strconv.Atoi(strings.TrimSpace(fields[5]))

        volume := ThinVolumeInfo{
            Name:        strings.TrimSpace(fields[0]),
            VGName:      strings.TrimSpace(fields[1]),
            PoolName:    strings.TrimSpace(fields[2]),
            VirtualSize: strings.TrimSpace(fields[3]),
            UsedPercent: usedPercent,
            DeviceID:    deviceID,
        }

        volumes = append(volumes, volume)
    }

    return volumes, nil
}

// StartMonitoring
func (tm *ThinMonitor) StartMonitoring() {
    go func() {
        ticker := time.NewTicker(tm.CheckInterval)
        defer ticker.Stop()

        for range ticker.C {
            tm.checkAlerts()
        }
    }()
}

// checkAlerts
func (tm *ThinMonitor) checkAlerts() {
    pools, err := tm.GetThinPools()
    if err != nil {
        if tm.AlertCallback != nil {
            tm.AlertCallback(fmt.Sprintf(" Thin Pool : %v", err))
        }
        return
    }

    for _, pool := range pools {
```

```
        //
        if pool.DataUsedPercent > tm.DataThreshold {
            alert := fmt.Sprintf("Thin Pool %s/%s  %.2f%%  %.2f%%",
                pool.VGName, pool.Name, pool.DataUsedPercent, tm.DataThreshold)

            if tm.AlertCallback != nil {
                tm.AlertCallback(alert)
            }
        }

        //
        if pool.MetaUsedPercent > tm.MetadataThreshold {
            alert := fmt.Sprintf("Thin Pool %s/%s  %.2f%%  %.2f%%",
                pool.VGName, pool.Name, pool.MetaUsedPercent, tm.MetadataThreshold)

            if tm.AlertCallback != nil {
                tm.AlertCallback(alert)
            }
        }
    }
}

// ExtendThinPool  Thin Pool
func (tm *ThinMonitor) ExtendThinPool(vgName, poolName, extendSize string) error {
    poolPath := fmt.Sprintf("/dev/%s/%s", vgName, poolName)

    cmd := exec.Command("lvextend", "-L", "+"+extendSize, poolPath)
    output, err := cmd.CombinedOutput()
    if err != nil {
        return fmt.Errorf(" Thin Pool : %v, : %s", err, string(output))
    }

    fmt.Printf("Thin Pool : %s  %s\n", poolName, extendSize)
    return nil
}

// CreateThinVolume  Thin Volume
func (tm *ThinMonitor) CreateThinVolume(vgName, poolName, volumeName, virtualSize string) error {
    poolPath := fmt.Sprintf("%s/%s", vgName, poolName)

    cmd := exec.Command("lvcreate", "-V", virtualSize, "-T", poolPath, "-n", volumeName)
    output, err := cmd.CombinedOutput()
    if err != nil {
        return fmt.Errorf(" Thin Volume : %v, : %s", err, string(output))
    }

    fmt.Printf("Thin Volume : %s (: %s)\n", volumeName, virtualSize)
    return nil
}
```

2.2

```
// filepath: internal/thin/automanage.go
package thin

import (
    "fmt"
    "log"
    "strconv"
    "strings"
    "time"
)

// AutoManager
type AutoManager struct {
    monitor          *ThinMonitor
    AutoExtendThreshold  float64
    AutoExtendPercent    int
    MetadataExtendSize   string
    Enabled             bool
}

// NewAutoManager
```

```
func NewAutoManager(monitor *ThinMonitor) *AutoManager {
    return &AutoManager{
        monitor:             monitor,
        AutoExtendThreshold: 75.0,  // 75%
        AutoExtendPercent:   20,    //  20%
        MetadataExtendSize:  "100M", //  100MB
        Enabled:            true,
    }
}

// StartAutoManagement
func (am *AutoManager) StartAutoManagement() {
    if !am.Enabled {
        return
    }

    go func() {
        ticker := time.NewTicker(time.Minute * 2)
        defer ticker.Stop()

        for range ticker.C {
            am.performAutoActions()
        }
    }()

    log.Println("Thin Pool ")
}

// performAutoActions
func (am *AutoManager) performAutoActions() {
    pools, err := am.monitor.GetThinPools()
    if err != nil {
        log.Printf(" Thin Pool : %v", err)
        return
    }

    for _, pool := range pools {
        //
        if pool.DataUsedPercent > am.AutoExtendThreshold {
            err := am.autoExtendData(pool)
            if err != nil {
                log.Printf(": %v", err)
            }
        }

        //
        if pool.MetaUsedPercent > 80.0 {
            err := am.autoExtendMetadata(pool)
            if err != nil {
                log.Printf(": %v", err)
            }
        }
    }
}

// autoExtendData
func (am *AutoManager) autoExtendData(pool ThinPoolInfo) error {
    //  ( AutoExtendPercent%)
    currentSizeGB := parseSize(pool.DataSize)
    extendSizeGB := currentSizeGB * float64(am.AutoExtendPercent) / 100.0
    extendSize := fmt.Sprintf("%.0fG", extendSizeGB)

    log.Printf(" Thin Pool %s/%s : +%s",
        pool.VGName, pool.Name, extendSize)

    return am.monitor.ExtendThinPool(pool.VGName, pool.Name, extendSize)
}

// autoExtendMetadata
func (am *AutoManager) autoExtendMetadata(pool ThinPoolInfo) error {
    metaPoolName := pool.Name + "_tmeta"

    log.Printf(" Thin Pool %s/%s : +%s",
        pool.VGName, metaPoolName, am.MetadataExtendSize)
```

```
    return am.monitor.ExtendThinPool(pool.VGName, metaPoolName, am.MetadataExtendSize)
}

// parseSize  ( "10.00g" -> 10.0)
func parseSize(sizeStr string) float64 {
    sizeStr = strings.ToLower(strings.TrimSpace(sizeStr))

    var multiplier float64 = 1
    if strings.HasSuffix(sizeStr, "g") {
        multiplier = 1
        sizeStr = strings.TrimSuffix(sizeStr, "g")
    } else if strings.HasSuffix(sizeStr, "m") {
        multiplier = 0.001
        sizeStr = strings.TrimSuffix(sizeStr, "m")
    } else if strings.HasSuffix(sizeStr, "t") {
        multiplier = 1024
        sizeStr = strings.TrimSuffix(sizeStr, "t")
    }

    size, err := strconv.ParseFloat(sizeStr, 64)
    if err != nil {
        return 0
    }

    return size * multiplier
}

// TrimSupport TRIM/DISCARD
func (am *AutoManager) EnableTrimSupport(vgName, poolName string) error {
    poolPath := fmt.Sprintf("/dev/%s/%s", vgName, poolName)

    //  DISCARD
    cmd := fmt.Sprintf("lvchange --discards passdown %s", poolPath)
    _, err := executeCommand(cmd)
    if err != nil {
        return fmt.Errorf(" DISCARD : %v", err)
    }

    log.Printf(" Thin Pool %s/%s  TRIM ", vgName, poolName)
    return nil
}

// executeCommand
func executeCommand(command string) (string, error) {
    //
    return "", nil
}
```

3.

```
// filepath: cmd/lvmtools/main.go
package main

import (
    "flag"
    "fmt"
    "log"
    "os"
    "time"

    "github.com/yourproject/internal/snapshot"
    "github.com/yourproject/internal/thin"
)

func main() {
    var (
        operation = flag.String("op", "", ": snapshot, thin, monitor")
        vgName    = flag.String("vg", "", "")
        lvName    = flag.String("lv", "", "")
        name      = flag.String("name", "", "")
        size      = flag.String("size", "", "")
        monitor   = flag.Bool("monitor", false, "")
    )
```

```
    flag.Parse()

    if *operation == "" {
        printUsage()
        os.Exit(1)
    }

    switch *operation {
    case "snapshot":
        handleSnapshotOperations(*vgName, *lvName, *name, *size, *monitor)
    case "thin":
        handleThinOperations(*vgName, *name, *size, *monitor)
    case "monitor":
        startMonitoring()
    default:
        fmt.Printf(": %s\n", *operation)
        printUsage()
        os.Exit(1)
    }
}

func handleSnapshotOperations(vgName, lvName, name, size string, monitor bool) {
    manager := snapshot.NewSnapshotManager()

    if monitor {
        fmt.Println("...")
        for {
            err := manager.MonitorSnapshots(vgName, 80.0)
            if err != nil {
                log.Printf(": %v", err)
            }
            time.Sleep(time.Minute * 5)
        }
    }

    if name != "" && lvName != "" && vgName != "" {
        err := manager.CreateSnapshot(vgName, lvName, name, size)
        if err != nil {
            log.Fatalf(": %v", err)
        }
    }

    //
    snapshots, err := manager.ListSnapshots(vgName)
    if err != nil {
        log.Fatalf(": %v", err)
    }

    fmt.Println(":")
    for _, snap := range snapshots {
        fmt.Printf("  %s (: %s, : %.2f%%)\n",
            snap.Name, snap.OriginLV, snap.UsedPercent)
    }
}

func handleThinOperations(vgName, name, size string, monitor bool) {
    thinMonitor := thin.NewThinMonitor()
    autoManager := thin.NewAutoManager(thinMonitor)

    if monitor {
        fmt.Println(" Thin ...")

        //
        thinMonitor.AlertCallback = func(alert string) {
            log.Printf(": %s", alert)
        }

        thinMonitor.StartMonitoring()
        autoManager.StartAutoManagement()

        //
        select {}
    }

    //  Thin Pool
```

```
    pools, err := thinMonitor.GetThinPools()
    if err != nil {
        log.Fatalf(" Thin Pool : %v", err)
    }

    fmt.Println("Thin Pool :")
    for _, pool := range pools {
        fmt.Printf("  %s/%s - : %.2f%%, : %.2f%%\n",
            pool.VGName, pool.Name, pool.DataUsedPercent, pool.MetaUsedPercent)
    }
}

func startMonitoring() {
    fmt.Println("...")

    //
    snapManager := snapshot.NewSnapshotManager()
    go func() {
        for {
            //  VG
            time.Sleep(time.Minute * 5)
        }
    }()

    // Thin
    thinMonitor := thin.NewThinMonitor()
    thinMonitor.AlertCallback = func(alert string) {
        log.Printf(": %s", alert)
        //
    }
    thinMonitor.StartMonitoring()

    autoManager := thin.NewAutoManager(thinMonitor)
    autoManager.StartAutoManagement()

    log.Println(" Ctrl+C ")
    select {} //
}

func printUsage() {
    fmt.Println("LVM :")
    fmt.Println("  : -op=snapshot -vg=vg_name -lv=lv_name -name=snap_name [-size=size]")
    fmt.Println("  : -op=snapshot -vg=vg_name -monitor")
    fmt.Println("   Thin: -op=thin -monitor")
    fmt.Println("  : -op=monitor")
}
```

1.

1.1

```
#
lvs -o +snap_percent storage_vg
#  snap_percent  100.00
```

```
# 1.
dmsetup status storage_vg-data_lv_backup

# 2.
journalctl -u lvm2-monitor | grep -i snapshot

# 3.  CoW
dmsetup table storage_vg-data_lv_backup
```

```
# 1:
lvextend -L +500M /dev/storage_vg/data_lv_backup

# 2:
dd if=/dev/storage_vg/other_backup of=/dev/storage_vg/data_lv

# 3:
```

1.2

```
#!/bin/bash
#

echo "..."
fio --name=original --filename=/dev/storage_vg/data_lv --rw=randwrite \
    --bs=4k --numjobs=4 --time_based --runtime=60s --group_reporting

echo "..."
fio --name=snapshot --filename=/dev/storage_vg/data_lv_backup --rw=randwrite \
    --bs=4k --numjobs=4 --time_based --runtime=60s --group_reporting
```

2. Thin Provisioning

2.1 Chunk Size

chunk_size

```
#  chunk_size  thin pool
for chunk in 64K 128K 256K 512K; do
    echo " chunk_size: $chunk"
    lvcreate -L 100M -n thin_meta_$chunk storage_vg
    lvcreate -L 4G -n thin_data_$chunk storage_vg
    lvconvert --type thin-pool --chunksize $chunk \
        --poolmetadata storage_vg/thin_meta_$chunk storage_vg/thin_data_$chunk
    lvrename storage_vg/thin_data_$chunk storage_vg/thin_pool_$chunk

    #
    lvcreate -V 2G -T storage_vg/thin_pool_$chunk -n test_$chunk
    fio --name=test --filename=/dev/storage_vg/test_$chunk \
        --rw=randwrite --bs=4k --numjobs=1 --time_based --runtime=30s
done
```

2.2

SSD

```
#
pvcreate /dev/nvme0n1p1  # SSD
vgextend storage_vg /dev/nvme0n1p1

#  LV  SSD
lvcreate -L 200M -n thin_meta storage_vg /dev/nvme0n1p1
```

3.

3.1

```
// filepath: internal/metrics/collector.go
package metrics

import (
    "time"
)

// StorageMetrics
type StorageMetrics struct {
    Timestamp        time.Time `json:"timestamp"`
```

```
    //
    SnapshotCount    int       `json:"snapshot_count"`
    SnapshotMaxUsage float64   `json:"snapshot_max_usage"`
    SnapshotAvgUsage float64   `json:"snapshot_avg_usage"`

    // Thin Pool
    ThinPoolDataUsage    float64 `json:"thin_pool_data_usage"`
    ThinPoolMetaUsage    float64 `json:"thin_pool_meta_usage"`
    ThinVolumeCount      int     `json:"thin_volume_count"`
    ThinOverallocation   float64 `json:"thin_overallocation"`

    //
    ReadIOPS    int64 `json:"read_iops"`
    WriteIOPS   int64 `json:"write_iops"`
    ReadBW      int64 `json:"read_bandwidth"`  // MB/s
    WriteBW     int64 `json:"write_bandwidth"` // MB/s
}

// MetricsCollector
type MetricsCollector struct {
    interval time.Duration
    history  []StorageMetrics
}

// CollectMetrics
func (mc *MetricsCollector) CollectMetrics() StorageMetrics {
    //
    return StorageMetrics{
        Timestamp: time.Now(),
        // ...
    }
}
```

3.2

```
# filepath: configs/alerts.yaml
alerts:
  snapshot:
    usage_threshold: 80.0
    invalid_snapshot: true
    cleanup_failed: true

  thin_pool:
    data_threshold: 85.0
    metadata_threshold: 90.0
    auto_extend_failed: true

  performance:
    iops_drop_threshold: 50 # IOPS  50%
    latency_threshold: 100 #  100ms
```

LVM

1.
    -
    -
    -
2. **Thin Provisioning**
    -
    -
    - TRIM/DISCARD
3.
    -

    -
    -

4. **Web**
    - RESTful API
    -
    -

- > 80%
- Go modules
-
-

1.
    - cron
    -
    -
2. **Thin Pool**
    - chunk_size
    -
    -

1.
    - LVM cluster
    -
    -
2.
    -
    -
    -

1.
    -
    -
    -
2.
    - Prometheus/Grafana
    -
    -

- CoW
-
- Go
-