Day 2: PV VG Go

- :
    - PV (pvcreate, pvs, pvdisplay)
    - VG (vgcreate, vgs, vgdisplay, vgextend)
    - PE LVM
- :
    - LVM
    -
    - **Go JSON PV VG**

PV

PV LVM pvcreate

/dev/sdb **LVM LVM Label** :

1. **LVM** : “ LVM ”
2. **UUID**: PV
3. : PV
4. **Metadata Area**: VG

(Physical Extent - PE): pvcreate LVM PV PE**PE LVM** 4MBLV PE

VG

PV “” VG “”vgcreate

vgcreate vg_data_01 /dev/sdb /dev/sdc :

1. **VG UUID**:
2. : vg_data_01 “” /dev/sdb /dev/sdc PV PV PE
3. : LVM “” /dev/sdb /dev/sdc **PV VG**   LVM VG PV

□ □□□□

/dev/sdb, /dev/sdc, /dev/sdd, /dev/sde 10GB

1. Physical Volume (PV)

1: PV /dev/sdb /dev/sdc

```
#  sudo  root
sudo pvcreate /dev/sdb /dev/sdc
```

:

```
  Physical volume "/dev/sdb" successfully created.
  Physical volume "/dev/sdc" successfully created.
```

2: PV pvs PV

```
sudo pvs
```

:

```
  PV         VG Fmt  Attr PSize   PFree
  /dev/sdb      lvm2 ---  <10.00g <10.00g
  /dev/sdc      lvm2 ---  <10.00g <10.00g
```

- PV:
- VG: VG
- Fmt: lvm2
- Attr: ---
- PSize: PV
- PFree: PV

3: PV pvdisplay PV

```
sudo pvdisplay /dev/sdb
```

:

```
  "/dev/sdb" is a new physical volume of "<10.00 GiB"
  --- NEW Physical volume ---
  PV Name               /dev/sdb
  VG Name
  PV Size               <10.00 GiB
  Allocatable           NO
  PE Size               0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

- PV Name:
- VG Name:
- PV UUID: **ID**LVM PV
- PE Size, Total PE: 0 PV VG PE

2. Volume Group (VG)

1: VG PV vg_data_01

```
sudo vgcreate vg_data_01 /dev/sdb /dev/sdc
```

:

```
  Volume group "vg_data_01" successfully created
```

2: VG vgs

```
sudo vgs
```

:

```
  VG         #PV #LV #SN Attr   VSize   VFree
  vg_data_01   2   0   0 wz--n- <19.99g <19.99g
```

- VG:
- #PV: PV 2
- #LV, #SN: 0
- Attr: wz--n-
- VSize: VG PV
- VFree: VG

3: VG vgdisplay vg_data_01

```
sudo vgdisplay vg_data_01
```

:

```
  --- Volume group ---
  VG Name               vg_data_01
  System ID
  Format                lvm2
  Metadata Areas        2
  Metadata Sequence No  1
  VG Access             read/write
  VG Status             resizable
  ...
  VG Size               <19.99 GiB
  PE Size               4.00 MiB
  Total PE              5118
  Alloc PE / Size       0 / 0
  Free  PE / Size       5118 / <19.99 GiB
  VG UUID               yyyyyyyy-yyyy-yyyy-yyyy-yyyyyyyyyyyy
```

- Metadata Areas: 2 2 /dev/sdb /dev/sdc
- PE Size: **4.00 MiB** VG PE
- Total PE: VG PE
- Free PE / Size: PE

4: PV pvs

```
sudo pvs
```

:

```
  PV         VG         Fmt  Attr PSize   PFree
  /dev/sdb   vg_data_01 lvm2 a--  <10.00g <10.00g
  /dev/sdc   vg_data_01 lvm2 a--  <10.00g <10.00g
```

VG vg_data_01

Go

: Go pvs vgs JSON Go

:

```
#  lvm-manager
mkdir -p cmd/day02
cd cmd/day02
```

(main.go):

```
package main

import (
        "bytes"
        "encoding/json"
        "fmt"
        "log"
        "os/exec"
        "strings"
)

// LVMReport is the top-level structure for LVM JSON reports.
type LVMReport struct {
        Report []map[string][]map[string]string `json:"report"`
}

// PhysicalVolume defines the structure for a PV's attributes.
type PhysicalVolume struct {
        Name  string `json:"pv_name"`
        VG    string `json:"vg_name"`
        Size  string `json:"pv_size"`
```

```go
	Free  string `json:"pv_free"`
	UUID  string `json:"pv_uuid"`
}

// VolumeGroup defines the structure for a VG's attributes.
type VolumeGroup struct {
	Name    string `json:"vg_name"`
	PVCount string `json:"pv_count"`
	LVCount string `json:"lv_count"`
	Size    string `json:"vg_size"`
	Free    string `json:"vg_free"`
	UUID    string `json:"vg_uuid"`
}

// runLVMCommand executes an LVM command with JSON reporting options.
func runLVMCommand(command string, args ...string) ([]byte, error) {
	// Prepend sudo to run with root privileges
	fullArgs := append([]string{command}, args...)
	fullArgs = append(fullArgs, "--reportformat", "json")

	cmd := exec.Command("sudo", fullArgs...)

	var stdout, stderr bytes.Buffer
	cmd.Stdout = &stdout
	cmd.Stderr = &stderr

	err := cmd.Run()
	if err != nil {
		return nil, fmt.Errorf("command `sudo %s %s` failed: %v\nStderr: %s", command, strings.Join(args, " "), err, stderr.String())
	}
	return stdout.Bytes(), nil
}

// GetPhysicalVolumes fetches and parses PV information.
func GetPhysicalVolumes() ([]PhysicalVolume, error) {
	output, err := runLVMCommand("pvs", "-o", "pv_name,vg_name,pv_size,pv_free,pv_uuid")
	if err != nil {
		return nil, err
	}

	var report LVMReport
	if err := json.Unmarshal(output, &report); err != nil {
		return nil, fmt.Errorf("failed to parse pvs JSON: %v", err)
	}

	var pvs []PhysicalVolume
	if len(report.Report) > 0 && report.Report[0]["pv"] != nil {
		for _, pvMap := range report.Report[0]["pv"] {
			pvs = append(pvs, PhysicalVolume{
				Name: pvMap["pv_name"],
				VG:   pvMap["vg_name"],
				Size: pvMap["pv_size"],
				Free: pvMap["pv_free"],
				UUID: pvMap["pv_uuid"],
			})
		}
	}
	return pvs, nil
}

// GetVolumeGroups fetches and parses VG information.
func GetVolumeGroups() ([]VolumeGroup, error) {
	output, err := runLVMCommand("vgs", "-o", "vg_name,pv_count,lv_count,vg_size,vg_free,vg_uuid")
	if err != nil {
		return nil, err
	}

	var report LVMReport
	if err := json.Unmarshal(output, &report); err != nil {
		return nil, fmt.Errorf("failed to parse vgs JSON: %v", err)
	}

	var vgs []VolumeGroup
	if len(report.Report) > 0 && report.Report[0]["vg"] != nil {
		for _, vgMap := range report.Report[0]["vg"] {
			vgs = append(vgs, VolumeGroup{
				Name:    vgMap["vg_name"],
				PVCount: vgMap["pv_count"],
				LVCount: vgMap["lv_count"],
				Size:    vgMap["vg_size"],
				Free:    vgMap["vg_free"],
				UUID:    vgMap["vg_uuid"],
			})
		}
	}
	return vgs, nil
}

func main() {
	log.Println("Fetching LVM information...")

	pvs, err := GetPhysicalVolumes()
	if err != nil {
		log.Fatalf("Error getting physical volumes: %v", err)
	}

	vgs, err := GetVolumeGroups()
	if err != nil {
		log.Fatalf("Error getting volume groups: %v", err)
	}

	fmt.Println("\n--- Physical Volumes (PVs) ---")
	fmt.Printf("%-15s %-15s %-12s %-12s %-s\n", "PV Name", "VG Name", "Size", "Free", "UUID")
```

```
        fmt.Println(strings.Repeat("-", 80))
        for _, pv := range pvs {
                fmt.Printf("%-15s %-15s %-12s %-12s %-s\n", pv.Name, pv.VG, pv.Size, pv.Free, pv.UUID)
        }

        fmt.Println("\n--- Volume Groups (VGs) ---")
        fmt.Printf("%-15s %-5s %-5s %-12s %-12s %-s\n", "VG Name", "#PV", "#LV", "Size", "Free", "UUID")
        fmt.Println(strings.Repeat("-", 80))
        for _, vg := range vgs {
                fmt.Printf("%-15s %-5s %-5s %-12s %-12s %-s\n", vg.Name, vg.PVCount, vg.LVCount, vg.Size, vg.Free, vg.UUID)
        }

        log.Println("LVM information fetched successfully.")
}
```

:

1. `cmd/day02` `go run main.go`
2. :
    - LVM `--reportformat json`
    - `runLVMCommand` `stderr`
    - `GetPhysicalVolumes` `GetVolumeGroups`
    - `main`

- : `pvcreate` "Device /dev/sdb is already in use"
    - :
    - : `lsblk -f` `mount`
    - :
- : VG

  ```
  # VG /tmp
  sudo vgcfgbackup -f /tmp/vg_data_01.backup vg_data_01
  ```

  VG

: VG Go VG PV

1. **VG**: `/dev/sdd` `vg_archive_01`

   ```
   sudo pvcreate /dev/sdd
   sudo vgcreate vg_archive_01 /dev/sdd
   ```

2. **Go** : `main.go` VG VG VG PV
    - : `vgs` `pvs` `pv.VG == vg.Name`

1. **VG** :
    - `sudo pvcreate /dev/sde` `/dev/sde` PV
    - `sudo vgextend vg_archive_01 /dev/sde` PV `vg_archive_01`
    - `vgs` `vgdisplay vg_archive_01`
    - `vgreduce vg_archive_01 /dev/sde`
2. **Go** :
    - : Go `Size` `Free` `<19.99g` Go `parseSize(sizeStr string) (float64, error)` LVM `<`, `>` `g`, `m`, `t` GB `float64`
    - : Go VG