Loops, conditions, functions, and error handling in Bash

Here's a script that combines loops, conditions, functions, and error handling:

```
#!/bin/bash

# Function to calculate the average of numbers in a file
calculate_average() {
    local file=$1
    local total=0
    local count=0

    while read -r line
    do
        if [[ $line =~ ^[0-9]+$ ]]; then
            total=$((total + line))
            count=$((count + 1))
        else
            echo "Warning: Non-numeric value '$line' ignored."
        fi
    done < "$file"

    if [ $count -eq 0 ]; then
        echo "Error: No valid numbers in the file."
        exit 1
    fi

    echo "Average: $((total / count))"
}

# Main script
read -p "Enter the file name: " filename

# Error handling for file existence
if [ ! -f $filename ]; then
    echo "Error: File '$filename' not found!"
    exit 1
fi

# Call the function to calculate average
calculate_average $filename
```

Use Case: Log Management and Rotation on Ubuntu

Scenario

You have an Ubuntu server running a web application, and you want to manage the logs generated by the application to prevent disk space from being filled up. Additionally, you need to troubleshoot issues using system and application logs.

1. Log Rotation Using `logrotate`

Log rotation is the process of periodically rotating and archiving log files. This helps in managing disk space and keeps logs organized.

Example Configuration:

Let's assume your web application writes logs to `/var/log/myapp/myapp.log`.

Configuration File for `logrotate`

1. Create a configuration file for your application logs in `/etc/logrotate.d/`:

```
sudo nano /etc/logrotate.d/myapp
```

2. Add the following configuration:

```
/var/log/myapp/myapp.log {
    daily
    rotate 7
    compress
    missingok
    notifempty
    create 0640 www-data www-data
    postrotate
        systemctl reload myapp.service
    endscript
}
```

- **daily**: Rotate the log files daily.
- **rotate 7**: Keep 7 days' worth of log files.
- **compress**: Compress the rotated log files to save space.
- **missingok**: Ignore errors if the log file is missing.
- **notifempty**: Do not rotate the log if it is empty.
- **create 0640 www-data www-data**: Create new log files with specific permissions and ownership.

- **postrotate**: Command to run after rotation (e.g., reload the application service).

2. Steps to Troubleshoot Using Logs

System Logs

System logs contain information about the system's operation. They can be found in the `/var/log` directory.

Example Logs:

- `/var/log/syslog`: General system log.
- `/var/log/auth.log`: Authentication log.
- `/var/log/dmesg`: Kernel ring buffer log.

Application Logs

Application logs contain information specific to your application. They help in diagnosing issues related to the application's performance and behavior.

Example Log:

- `/var/log/myapp/myapp.log`: Custom application log.

Troubleshooting Steps

1. **Check System Logs for Errors:**

```
sudo tail -n 100 /var/log/syslog
sudo tail -n 100 /var/log/auth.log
```

- Use `tail -n 100` to display the last 100 lines of the log file.

2. **Check Application Logs:**

bash
Copy code

```
sudo tail -n 100 /var/log/myapp/myapp.log
```

3. **Search for Specific Errors or Patterns:**

Use `grep` to search for specific errors or patterns in the logs.

```
sudo grep "ERROR" /var/log/myapp/myapp.log
sudo grep "authentication failure" /var/log/auth.log
```

4. **Monitor Logs in Real-Time:**

Use `tail -f` to monitor logs in real-time.

```
sudo tail -f /var/log/myapp/myapp.log
```

5. **Review Logrotate Status:**

Check the status and configuration of `logrotate` to ensure logs are being rotated correctly.

```
sudo logrotate -d /etc/logrotate.conf
sudo logrotate -f /etc/logrotate.conf
```

Example Script for Troubleshooting

Below is a Bash script that automates the process of checking logs and providing a summary.

```
#!/bin/bash

LOG_FILE="/var/log/myapp/myapp.log"
SYSLOG_FILE="/var/log/syslog"
AUTH_LOG_FILE="/var/log/auth.log"

check_logs() {
    local file=$1
    local keyword=$2
    echo "Checking $file for '$keyword'..."
    grep "$keyword" "$file" | tail -n 10
}

echo "Starting log check..."

echo "=== Application Log Errors ==="
check_logs $LOG_FILE "ERROR"

echo "=== System Log Errors ==="
check_logs $SYSLOG_FILE "error"

echo "=== Authentication Failures ==="
check_logs $AUTH_LOG_FILE "authentication failure"

echo "Log check completed."
```

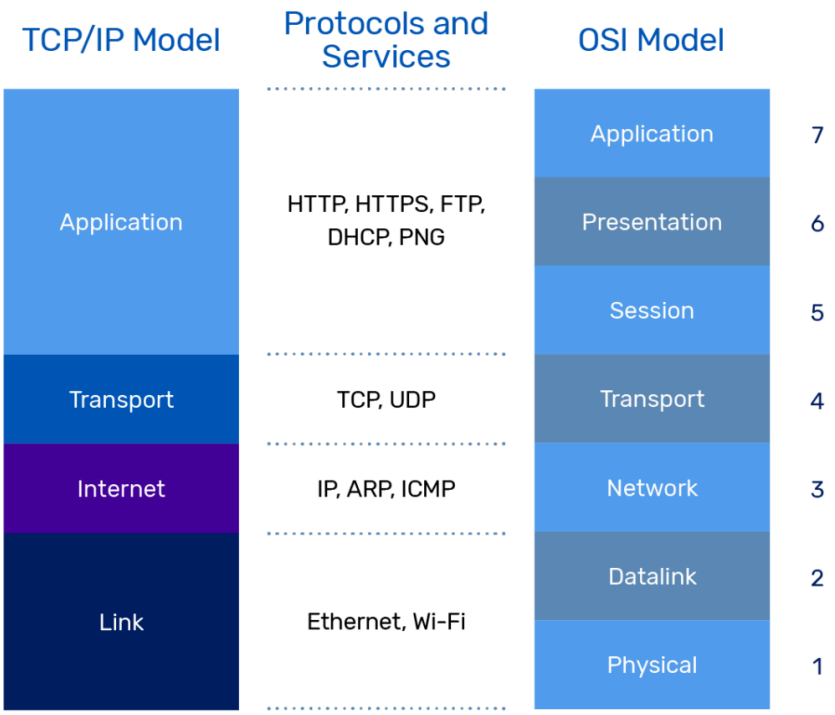
Explanation

1. **Logrotate Configuration:**
- The configuration ensures logs are rotated daily, old logs are compressed, and only the last 7 logs are kept. This prevents the log directory from filling up with old logs and saves disk space.
2. **Troubleshooting Script:**
- The script `check_logs` takes a log file and a keyword as arguments, searches for the keyword in the log file, and displays the last 10 occurrences.
- The main part of the script calls `check_logs` for application errors, system errors, and authentication failures, providing a quick summary of recent issues.

OSI Layers

The OSI (Open Systems Interconnection) model has seven layers that describe the functions of a networking system.

1. **Physical Layer**: Deals with the hardware connection and physical transmission of data.
2. **Data Link Layer**: Manages data frames between physical links.
3. **Network Layer**: Handles logical addressing and routing (e.g., IP addresses).
4. **Transport Layer**: Provides reliable data transfer (e.g., TCP, UDP).
5. **Session Layer**: Manages sessions and controls dialogues between computers.
6. **Presentation Layer**: Translates data formats and handles encryption/decryption.
7. **Application Layer**: Interfaces with the application software (e.g., HTTP, FTP).



IP Addressing and Subnet Masking

IP Address: A unique identifier for a device on a network (e.g., `192.168.1.10`).

Subnet Mask: Divides the IP address into network and host portions (e.g., `255.255.255.0`).

Use Case: Assign IP addresses to devices in the office network.

Assigning IP Addresses with Best Practices

1. **Static IP Addresses**: For critical devices like servers and printers.
2. **Dynamic IP Addresses (DHCP)**: For client devices like laptops and smartphones.
3. **Avoid Conflicts**: Ensure no two devices have the same IP address.

Gateway and NAT (Network Address Translation)

Gateway: A device that routes traffic from a local network to external networks (e.g., the internet).

NAT: Translates private IP addresses to a public IP address for internet access.

SNAT (Source NAT): Translates the source address of outgoing packets.

Broadcast Address: An IP address that allows information to be sent to all devices in a network (e.g., `192.168.1.255`).

Example Configuration

1. Static IP Address Assignment

Assume you are configuring a Linux server with a static IP address.

```
sudo nano /etc/network/interfaces
```

Add the following configuration:

```
auto eth0
iface eth0 inet static
    address 192.168.1.10
    netmask 255.255.255.0
    gateway 192.168.1.1
    dns-nameservers 8.8.8.8 8.8.4.4
```

- **address**: Static IP address (`192.168.1.10`).
- **netmask**: Subnet mask (`255.255.255.0`).

- **gateway**: Default gateway (`192.168.1.1`).
- **dns-nameservers**: DNS servers (`8.8.8.8` and `8.8.4.4`).

Save and restart the network service:

```
sudo systemctl restart networking
```

2. Dynamic IP Address Assignment with DHCP

For client devices, configure DHCP on the router.

Access the router's web interface and configure the DHCP server to assign IP addresses in the range `192.168.1.100` to `192.168.1.200`.

3. NAT Configuration

Configure NAT on a Linux router.

```
sudo nano /etc/rc.local
```

Add the following lines to enable NAT:

```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Enable IP forwarding:

```
sudo sysctl -w net.ipv4.ip_forward=1
```

Persist the setting:

```
sudo nano /etc/sysctl.conf
```

Uncomment or add the following line:

```
net.ipv4.ip_forward=1
```

Troubleshooting Commands

Check IP Address:

```
ip addr show
```

Check Routing Table:
`ip route show`

Check NAT Rules:

```
sudo iptables -t nat -L
```

Ping Gateway:

```
ping 192.168.1.1
```

Explanation of Commands and Scripts

Static IP Configuration

- **`/etc/network/interfaces`**: Configuration file for network interfaces.
- **`auto eth0`**: Automatically brings up the `eth0` interface at boot.
- **`iface eth0 inet static`**: Defines the `eth0` interface with a static IP.
- **`address`**: IP address assigned to the interface.
- **`netmask`**: Subnet mask.
- **`gateway`**: Default gateway.
- **`dns-nameservers`**: DNS servers.
- **`sudo systemctl restart networking`**: Restarts the networking service to apply changes.

DHCP Configuration

- **Router Configuration**: Configure DHCP server to assign IP addresses dynamically.

NAT Configuration

- **`iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE`**: Adds a rule to the NAT table to masquerade (SNAT) outgoing packets on the `eth0` interface.
- **`sysctl -w net.ipv4.ip_forward=1`**: Temporarily enables IP forwarding.
- **`/etc/sysctl.conf`**: Configuration file to persistently enable IP forwarding.

Archiving and Transferring Files

Scenario: You are a system administrator responsible for maintaining backups of critical files on a web server. To efficiently manage storage and simplify the process of transferring backups to a remote server, you need to archive and compress these files. Additionally, you want to ensure the backups can be easily extracted when needed.

Tools and Commands

1. **ZIP and UNZIP**: Used for compressing and extracting `.zip` files.
2. **tar**: Used for archiving files into a single file (tarball).
3. **gzip and gunzip**: Used for compressing and decompressing files, often used with tarballs.

Compression in Linux

1. Archiving and Compressing Files Using tar and gzip

Objective: Archive and compress the `/var/www/html` directory, which contains your website files, into a single compressed file for storage and transfer.

Example

Create a tarball of the `/var/www/html` directory:

```
tar -cvf website_backup.tar /var/www/html
```

- **`tar`**: The archiving utility.
- **`-c`**: Create a new archive.
- **`-v`**: Verbose mode, showing progress in the terminal.
- **`-f`**: Specifies the filename of the archive (`website_backup.tar`).
- **`/var/www/html`**: The directory to be archived.

Compress the tarball using gzip:

```
gzip website_backup.tar
```

- **`gzip`**: The compression utility.
- **`website_backup.tar`**: The file to be compressed. This will create `website_backup.tar.gz`.

Verify the compressed file:

```
ls -lh website_backup.tar.gz
```

2.

- **`ls -lh`**: Lists the file with human-readable file sizes to verify the size and presence of `website_backup.tar.gz`.

2. Extracting Files from a tar.gz Archive

Objective: Extract the archived files to restore the website on a new server.

Step-by-Step Guide:

Copy the compressed file to the new server (using `scp` for secure transfer):

```
scp website_backup.tar.gz user@newserver:/path/to/destination
```

1.
    - **`scp`**: Secure copy command for transferring files over SSH.
    - **`website_backup.tar.gz`**: The file to transfer.
    - **`user@newserver`**: The username and server address.
    - **`/path/to/destination`**: The destination directory on the new server.

Log in to the new server:

```
ssh user@newserver
```

- **`ssh`**: Secure Shell command for logging into the remote server.
- **`user@newserver`**: The username and server address.

Navigate to the destination directory:

```
cd /path/to/destination
```

Decompress the file using gunzip:

```
gunzip website_backup.tar.gz
```

- **`gunzip`**: The decompression utility.
- **`website_backup.tar.gz`**: The compressed file to decompress. This will produce `website_backup.tar`.

Extract the tarball:

```
tar -xvf website_backup.tar
```

- **`tar`**: The archiving utility.
- **`-x`**: Extract the archive.
- **`-v`**: Verbose mode.
- **`-f`**: Specifies the filename of the archive (`website_backup.tar`).

3. Using ZIP and UNZIP

Objective: Archive and compress multiple log files from `/var/log` into a `.zip` file for easy transfer.

Step-by-Step Guide:

Navigate to the `/var/log` directory:

```
cd /var/log
```

Create a ZIP archive of the log files:

```
zip logs_backup.zip *.log
```

- **`zip`**: The archiving and compression utility.
- **`logs_backup.zip`**: The name of the output ZIP file.
- **`*.log`**: Wildcard to include all `.log` files in the current directory.

Verify the ZIP file:

```
ls -lh logs_backup.zip
```

Transfer the ZIP file to a remote server:

```
scp logs_backup.zip user@newserver:/path/to/destination
```

Log in to the remote server and navigate to the destination directory:

```
ssh user@newserver
cd /path/to/destination
```

Extract the ZIP file:

```
unzip logs_backup.zip
```

Virtual Private Cloud

IP

- **IPv4:** default, 32 bit, CIDR - /16 to /28, 0.0.0.0 - 255.255.255.255
- **IPv6:** 128 bit, CIDR - /56, example: 2001:0000:130F:0000:0000:09C0:876A:130B.

CIDR, Classless Inter Domain Routing

-

Subnet
0
Security Group/Firewall
Network ACL

195.78.3.21
175.25.3.9

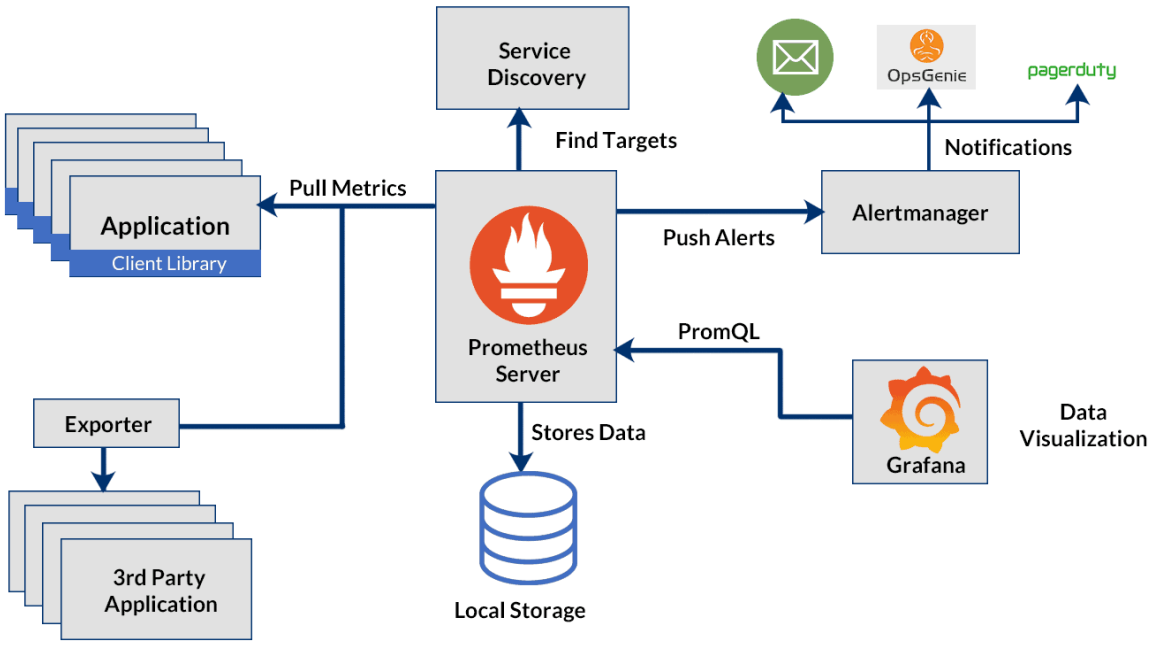
10.0.1.5 - 10.0.1.195

10.0.0.0/16
32 - 16 = 16 → 2 ^ 16 = 65,536

10.0.1.0/24
32 - 24 = 8 → 2 ^ 8 = 256 - 5 = 251

10.0.1.0
10.0.1.1
10.0.1.2
10.0.1.3
10.0.1.255

Prometheus



Installation of Prometheus

[Download | Prometheus]

Prometheus is an open-source monitoring and alerting toolkit designed for reliability and scalability. It is particularly well-suited for monitoring dynamic cloud environments and microservices. Prometheus is based on a pull model, where it scrapes metrics from configured endpoints at regular intervals.

Prometheus Architecture Overview

Prometheus consists of several components, each playing a vital role in its monitoring and alerting functions:

1. **Prometheus Server**: The core component responsible for scraping and storing metrics, querying data, and triggering alerts.
2. **Client Libraries**: Libraries used to instrument applications to expose metrics in a format that Prometheus can scrape.
3. **Pushgateway**: A component used to push metrics from short-lived jobs.
4. **Alertmanager**: Handles alerts generated by the Prometheus server, including routing, silencing, and notifications.
5. **Exporters**: Components that expose metrics from third-party systems as Prometheus-compatible metrics.
6. **Service Discovery**: Mechanisms for dynamically discovering targets to scrape metrics from.

7. **PromQL**: The query language used to retrieve and analyze metrics data stored in Prometheus.

Detailed Explanation of Components

1. Prometheus Server

- **Scrape**: Prometheus server pulls metrics data (scraping) from targets at regular intervals, which are defined in the configuration file. It uses HTTP-based pulls to retrieve metrics.
- **Time Series Database (TSDB)**: The scraped metrics are stored as time series data in a highly efficient, custom database. Each metric is identified by its name and key-value pairs called labels.
- **PromQL (Prometheus Query Language)**: This is the query language used to retrieve and manipulate time series data stored in Prometheus. PromQL enables users to aggregate, filter, and process metrics, making it a powerful tool for building dashboards and generating alerts.

2. Client Libraries

Prometheus client libraries are available in multiple programming languages (e.g., Go, Java, Python, Ruby). They allow developers to instrument their applications by exposing internal metrics such as memory usage, request durations, and more.

- **Instrumentation**: Applications are instrumented using these libraries to expose metrics at an endpoint (usually `/metrics`), where Prometheus can scrape them.

3. Pushgateway

The Pushgateway is designed for short-lived jobs that do not have a persistent runtime, making it difficult for Prometheus to scrape them directly. Instead, these jobs push their metrics to the Pushgateway, which then holds the metrics for Prometheus to scrape.

- **Use Case**: Ideal for batch jobs, CI/CD pipelines, and other ephemeral processes that need to report metrics.

4. Alertmanager

The Alertmanager handles alerts sent by Prometheus. It manages alert notifications, deduplication, grouping, routing, and silencing.

- **Alert Routing**: Alerts can be sent to different receivers based on predefined rules (e.g., email, Slack, PagerDuty).
- **Grouping**: Alerts can be grouped by common labels to avoid flooding receivers with multiple alerts.
- **Silencing**: Alerts can be temporarily silenced based on specific conditions, such as during maintenance windows.

5. Exporters

Exporters are components that allow Prometheus to collect metrics from third-party systems that don't natively expose Prometheus metrics. Common exporters include:

- **Node Exporter**: Exposes hardware and OS metrics from *nix systems.
- **MySQL Exporter**: Exposes MySQL server metrics.
- **Blackbox Exporter**: Allows probing of endpoints for availability and response times.

Exporters translate the native metrics of these systems into Prometheus-compatible metrics that can be scraped.

6. Service Discovery

Prometheus uses service discovery to automatically find targets to scrape, especially in dynamic environments like cloud platforms or Kubernetes.

- **Static Configuration**: Targets can be statically configured in the `prometheus.yml` file.
- **Dynamic Discovery**: Prometheus can integrate with service discovery mechanisms like Kubernetes, Consul, or EC2, allowing it to automatically adapt to changes in the environment (e.g., scaling up/down of instances).

7. PromQL

PromQL is the powerful query language used by Prometheus to retrieve, aggregate, and process time series data. It is designed for working with time series and offers a variety of functions for calculating rates, averages, sums, and more.

- **Examples**:
  - `up`: Simple query to check if targets are up.
  - `rate(http_requests_total[5m])`: Calculates the rate of HTTP requests over the last 5 minutes.

Prometheus Data Flow

1. **Scraping**: Prometheus server scrapes metrics from targets (applications, services, exporters) at defined intervals.
2. **Storage**: The scraped metrics are stored as time series data in Prometheus' TSDB.
3. **Querying**: Users can query this data using PromQL to generate dashboards, reports, or to perform ad-hoc analysis.
4. **Alerting**: Prometheus evaluates alerting rules, and if conditions are met, it sends alerts to the Alertmanager.
5. **Alert Handling**: The Alertmanager processes and routes the alerts to various notification channels.

PromQL (Prometheus Query Language) is a powerful language used to query and analyze the time series data collected by Prometheus. It allows users to select and aggregate data in various ways, making it an essential tool for creating dashboards, generating reports, and setting up alerts.

PromQL Overview

PromQL operates on time series data, which is identified by a metric name and a set of key-value pairs (labels). The queries can filter, aggregate, and transform this data to provide insights.

Basic Concepts

1. **Metric Name**: The name of the metric being queried, e.g., `http_requests_total`.
2. **Labels**: Key-value pairs associated with metrics, e.g., `{method="GET", status="200"}`.
3. **Time Series**: A stream of timestamped values for a particular set of labels and metric name.
4. **Range Vector**: A sequence of time series over a specified time range.
5. **Instant Vector**: A set of time series containing a single sample point per time series.

PromQL Query Types

1. **Instant Queries**: Return the current value of a time series.
2. **Range Queries**: Return values over a specified time range.

Basic PromQL Examples

Let's explore some basic PromQL queries using the data exposed by the Node Exporter on your Ubuntu EC2 instance. Node Exporter exposes various metrics about the system's hardware and OS, such as CPU usage, memory, disk I/O, etc.

1. Querying Current CPU Usage

```
rate(node_cpu_seconds_total[5m])
```

- **Explanation**:
  - `node_cpu_seconds_total`: The total number of seconds the CPU has spent in various modes (user, system, idle, etc.).
  - `rate(node_cpu_seconds_total[5m])`: Calculates the per-second average rate of CPU time spent over the last 5 minutes.
- **Use Case**: Monitoring CPU usage trends to detect abnormal spikes or sustained high usage, which could indicate performance issues.

2. Filtering by Labels

```
node_cpu_seconds_total{mode="idle"}
```

- **Explanation**:
  - `mode="idle"`: Filters the `node_cpu_seconds_total` metric to show only the idle CPU time.
  - This query will return the total idle CPU time.
- **Use Case**: Understanding how much time the CPU spends idle, which can help in capacity planning.

3. Querying Free Memory

```
node_memory_MemFree_bytes
```

- **Explanation**:
  - `node_memory_MemFree_bytes`: Shows the current amount of free memory in bytes.
- **Use Case**: Ensuring that the system has enough free memory to handle current and future workloads.

4. Calculating Memory Usage Percentage

```
100 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes * 100)
```

- **Explanation**:
  - `node_memory_MemAvailable_bytes`: The amount of memory available for new processes.
  - `node_memory_MemTotal_bytes`: The total memory available in the system.
  - This query calculates the percentage of memory in use by subtracting the available memory percentage from 100%.
- **Use Case**: Monitoring memory utilization to detect when the system is running low on memory.

Advanced PromQL Queries

1. Aggregating CPU Usage Across All Cores

```
sum(rate(node_cpu_seconds_total[5m])) by (instance)
```

- **Explanation**:
  - `sum(rate(node_cpu_seconds_total[5m]))`: Sums up the CPU usage across all cores.
  - `by (instance)`: Groups the result by instance, so you can see the aggregated CPU usage per server.
- **Use Case**: Useful for understanding the overall CPU usage on multi-core systems.

2. Monitoring Disk Space Usage

```
100 - (node_filesystem_free_bytes{mountpoint="/"} /
node_filesystem_size_bytes{mountpoint="/"} * 100)
```

- **Explanation**:
  - `node_filesystem_free_bytes{mountpoint="/"}`: The free disk space in bytes for the root filesystem.
  - `node_filesystem_size_bytes{mountpoint="/"}`: The total size of the root filesystem in bytes.
  - This query calculates the percentage of disk space used on the root filesystem.
- **Use Case**: Essential for monitoring disk space to prevent the system from running out of storage.

3. Alerting on High CPU Usage

You can set up an alert for high CPU usage using a PromQL expression:

```
groups:
- name: CPUAlerts
  rules:
  - alert: HighCPUUsage
    expr: sum(rate(node_cpu_seconds_total{mode!="idle"}[5m])) by
(instance) > 0.9
    for: 5m
    labels:
      severity: critical
    annotations:
      summary: "High CPU usage on {{ $labels.instance }}"
      description: "CPU usage is above 90% for more than 5 minutes."
```

- **Explanation**:
  - `expr`: The PromQL expression that triggers the alert. In this case, it checks if the CPU usage is greater than 90% for more than 5 minutes.
  - `for`: The duration the condition must hold true before the alert is fired.
  - `labels` and `annotations`: Additional information attached to the alert for routing and documentation.
- **Use Case**: Automatically notify when a system is under high CPU load, potentially preventing system outages.

Setting Up Prometheus to Scrape Node Exporter

Since you have Node Exporter and Prometheus installed on your Ubuntu EC2 instance, let's ensure that Prometheus is scraping metrics from the Node Exporter.

Prometheus Configuration File (`prometheus.yml`)
Ensure that your Prometheus configuration file includes a job for scraping Node Exporter:

```
scrape_configs:
- job_name: 'node_exporter'
  static_configs:
    - targets: ['localhost:9100']
```

- **Explanation**:
  - `job_name`: A label that will appear in your queries to identify this job.
  - `targets`: The endpoint where Node Exporter is running. By default, Node Exporter runs on port `9100`.

Reload Prometheus Configuration
After updating the configuration, reload Prometheus to apply the changes:

```
kill -HUP $(pgrep prometheus)
```

- **Explanation**: This command sends a SIGHUP signal to the Prometheus process, causing it to reload its configuration without restarting.

2. **Verify Metrics in Prometheus**
   Navigate to Prometheus' web UI (typically at `http://<your-ec2-instance-ip>:9090`) and use the "Graph" or "Explore" tab to execute the PromQL queries mentioned above.

Lab Credentials

Ansible-worker.pem
Username: ubuntu

Vraj: 18.116.235.13
Mayusha: 3.145.125.21
Suryarajsingh: 3.129.73.56
Neel: 3.133.139.68
Utsav: 3.148.108.46
Yash: 18.118.126.26
Santosh: 3.133.121.136
Harshwardhan: 3.144.123.83
Farajnazish: 3.149.27.16
Yaksh: 3.133.149.158
Shital: 3.144.4.28

Manan: 3.135.210.131
Maaz: 3.138.175.243
Jash: 18.218.16.42
Eklavya: 18.219.239.171
Bhavin: 3.140.185.86
Bhavik: 3.15.138.230
Arsh: 18.226.180.134
Aman: 52.15.227.182
Abhinav: 18.218.212.14

Ansible-new.pem
Username: ubuntu

Shreya: 52.53.212.188
Chirag: 54.193.139.181
Shiv: 54.176.172.164
Jasminbanu: 54.193.193.217
Palash: 18.144.176.8
YashMahindrakar: 54.67.58.189
Nensi: 18.144.89.241
Poonam: 50.18.33.247
Siddh: 13.52.211.131
Sudharshan: 54.183.206.226

Project 01:

Project Overview:

The goal of this capstone project is to combine shell scripting with system monitoring and log management practices. You will create a set of automated tools using shell scripts to manage logs, monitor system performance using Prometheus and Node Exporter, and generate insights using PromQL queries. The project will require a systematic approach, covering scripting fundamentals, log management, and monitoring setup.

Project Deliverables:

1. **Shell Scripts for Basic Operations:**
    - **Task:** Write shell scripts to perform basic system operations, such as checking disk usage, memory usage, and CPU load.
    - **Deliverable:**
        - A collection of scripts that output system performance metrics.
        - Scripts should include error handling and logging.
2. **Log Management Script:**
    - **Task:** Develop a script to automate log management tasks such as log rotation and archiving. This script should include the ability to compress old logs and delete logs older than a specified number of days.
    - **Deliverable:**
        - A shell script that performs log rotation based on predefined conditions (e.g., log size, log age).
        - A report generated by the script detailing which logs were rotated, compressed, or deleted.
3. **Advanced Shell Scripting - Loops, Conditions, Functions, and Error Handling:**
    - **Task:** Refactor the previous scripts to include loops, conditionals, and functions for modularity. Implement error handling to manage potential issues during script execution.
    - **Deliverable:**
        - Modular shell scripts that use functions for repeatable tasks.
        - Error-handling mechanisms in place for scenarios like missing files, insufficient permissions, etc.
        - Logs that track script execution and any errors encountered.
4. **Log Checking and Troubleshooting:**
    - **Task:** Write a script that reads through system and application logs, identifies common issues (e.g., out of memory, failed service starts), and provides troubleshooting steps based on log analysis.
    - **Deliverable:**
        - A script that parses logs for errors or warnings and outputs possible root causes.
        - Documentation on the types of logs checked and the issues identified.
        - A troubleshooting guide based on common errors found in the logs.
5. **Installation and Setup of Prometheus and Node Exporter:**
    - **Task:** Install and configure Prometheus and Node Exporter on the system. Ensure that Node Exporter is properly configured to collect system metrics.

- **Deliverable:**
  - A documented installation and configuration process for Prometheus and Node Exporter.
  - A running instance of Prometheus scraping metrics from Node Exporter.

6. **Prometheus Query Language (PromQL) Basic Queries:**
   - **Task:** Create a series of PromQL queries to monitor system performance, such as CPU usage, memory usage, and disk I/O.
   - **Deliverable:**
     - A set of PromQL queries that can be used to monitor key system metrics.
     - A dashboard setup guide or configuration that visualizes these metrics in Prometheus or Grafana.
7. **Final Report and Presentation:**
   - **Task:** Prepare a final report documenting all scripts, the installation and configuration of monitoring tools, and the output of your PromQL queries.
   - **Deliverable:**
     - A comprehensive project report covering all steps, scripts, and results.