DevOps Shack

Top 50 Most Asked Prometheus Questions and Answers

1. What is Prometheus?

Prometheus is an open-source monitoring and alerting toolkit originally developed at SoundCloud and later donated to the Cloud Native Computing Foundation (CNCF). It is designed for high-dimensional time-series data collection, storage, querying, and alerting.

Key Features:

- **Multi-dimensional data model with key-value pairs (labels)**
- **Powerful query language (PromQL)**
- **Pull-based architecture**
- **Service discovery & static configurations**
- **Time-series storage with efficient data compression**
- **Built-in alerting via Alertmanager**
- **Visualization support with Grafana**

2. How does Prometheus work?

Prometheus operates on a pull model where it periodically scrapes targets (exporters) over HTTP endpoints.

Prometheus Workflow:

1. **Scrapes Metrics: It queries the configured endpoints (e.g., `/metrics`).**

2. **Stores Time-Series Data: The collected data is stored in a time-series format with timestamps.**
3. **Runs Queries (PromQL): Users can query data using PromQL for insights.**
4. **Alerts Based on Rules: It sends alerts when defined conditions are met.**
5. **Visualizes Data: Dashboards in Grafana provide insights.**

3. What are exporters in Prometheus?

Exporters are services that expose metrics for Prometheus to scrape. They translate metrics from third-party applications into Prometheus-compatible format.

Common Exporters:

- **Node Exporter: Provides OS-level metrics (CPU, memory, disk, etc.).**
- **cAdvisor: Monitors container performance.**
- **Blackbox Exporter: Probes network endpoints (HTTP, DNS, TCP).**
- **MySQL Exporter: Exposes MySQL database metrics.**
- **JMX Exporter: Collects Java-based application metrics.**

4. What is PromQL and why is it important?

PromQL (Prometheus Query Language) is used to retrieve and manipulate time-series data.

Example Queries:

- **CPU Usage:** `rate(node_cpu_seconds_total[5m])`
- **Memory Usage:** `node_memory_Active_bytes`
- **HTTP Request Count:** `sum by (method) (rate(http_requests_total[1m]))`

PromQL is powerful because:

- **It supports filtering by labels.**
- **It enables aggregation functions.**
- **It allows math operations on metrics.**

5. What is the difference between Pull vs Push in monitoring?

Pull Model (Prometheus)

- **Prometheus scrapes metrics from services at intervals.**
- **Easier to scale and control.**
- **No need for services to push metrics.**

Push Model (StatsD, Graphite)

- **Services push metrics to a central database.**
- **Requires additional configurations.**
- **Works better for short-lived jobs.**

6. What is a Prometheus TSDB (Time-Series Database)?

Prometheus stores data in its own TSDB, optimized for:

- **High-speed writes.**
- **Efficient storage compression.**
- **Label-based indexing.**

Storage Internals:

- **Head Block (In-Memory): Stores recent data.**

- **WAL (Write-Ahead Log): Ensures durability.**
- **Compacted Blocks (on Disk): Older data gets stored in blocks.**

7. What is the role of labels in Prometheus?

Labels are key-value pairs that uniquely identify time-series data.

Example metric:

```
http_requests_total{method="POST", status="200",
instance="server1"}
```

Advantages of labels:

- **Enable advanced filtering.**
- **Improve data organization.**
- **Allow precise alerting.**

8. How does Prometheus handle high cardinality?

High cardinality occurs when too many unique label combinations exist.

Ways to handle it:

- **Avoid unnecessary labels (e.g., `user_id`, `request_id`).**
- **Use aggregations (e.g., `sum()`, `avg()`).**
- **Increase retention period wisely to prevent excessive storage.**

9. How does Prometheus handle alerting?

Prometheus triggers alerts based on defined rules and sends them to Alertmanager.

Example Alert Rule:

```
groups:
  - name: High CPU Usage
    rules:
      - alert: HighCPU
        expr: avg by (instance)
(rate(node_cpu_seconds_total[5m])) > 0.8
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "High CPU usage detected on {{
$labels.instance }}"
```

10. What is the role of Alertmanager?

Alertmanager handles and routes alerts from Prometheus.

Key Features:

- **Silencing: Suppresses alerts for a period.**
- **Grouping: Combines similar alerts.**
- **Routing: Sends alerts to Email, Slack, PagerDuty, etc.**

11. How to monitor RabbitMQ with Prometheus?

RabbitMQ provides an official Prometheus exporter for monitoring.

Step 1: Enable RabbitMQ Prometheus Plugin

Run:

```
rabbitmq-plugins enable rabbitmq_prometheus
```

This exposes metrics at:

http://localhost:15692/metrics

Step 2: Configure Prometheus to Scrape RabbitMQ

Add this to `prometheus.yml`:

```
scrape_configs:
  - job_name: 'rabbitmq'
    static_configs:
      - targets: ['localhost:15692']
```

Step 3: Key RabbitMQ Metrics to Monitor

Total Queued Messages:

```
rabbitmq_queue_messages
```

Consumer Count:

```
Rabbitmq_queue_consumers
```

Queue Depth Over Time:

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

Step 4: Use Grafana for RabbitMQ Monitoring

- **Create dashboards with real-time queue statistics.**
- **Import RabbitMQ dashboards from Grafana Labs.**

12. How to integrate Prometheus with Kubernetes?

Kubernetes metrics are collected via:

1. **kube-state-metrics (Pod, Deployment, Node data)**
2. **cAdvisor (Container-level metrics)**
3. **Node Exporter (Node OS metrics)**

Example `prometheus.yml`:

```
scrape_configs:
  - job_name: 'kubernetes-nodes'
    kubernetes_sd_configs:
      - role: node
```

13. What are recording rules in Prometheus?

Recording rules precompute expensive queries for efficiency.

Example:

```
groups:
  - name: cpu_usage
    Rules:
```

```
- record: instance:cpu_usage:rate5m
  expr: rate(node_cpu_seconds_total[5m])
```

Instead of running `rate(node_cpu_seconds_total[5m])`, we can directly query `instance:cpu_usage:rate5m`.

14. How to secure Prometheus?

- Enable authentication (Basic Auth, OAuth, JWT).
- Use HTTPS (TLS Encryption).
- Restrict API access.
- Monitor Prometheus itself.
- Use RBAC in Kubernetes.

15. How to scale Prometheus?

- Sharding: Split targets across multiple Prometheus instances.
- Federation: Aggregate data from multiple instances.
- Remote Storage: Store data in Thanos, Cortex, or VictoriaMetrics.
- Long-term Storage Solutions: Use AWS S3, GCP, or Postgres for extended retention.

16. How does Prometheus handle downsampling?

Downsampling reduces storage by summarizing data over time.

Methods:

- Recording rules: Store aggregated versions.
- External solutions: Thanos Compact, Mimir.

17. Can Prometheus monitor Windows servers?

Yes, using the Windows Exporter, which provides metrics like:

- **CPU usage**
- **Memory utilization**
- **Disk I/O**
- **Network statistics**

18. What are histograms and summaries in Prometheus?

- **Histogram: Groups values into buckets (e.g., request durations).**
- **Summary: Provides quantiles (e.g., 95th percentile latency).**

Example:

```
http_request_duration_seconds_bucket{le="0.1"} 50
```

This means 50 requests took ≤ 100ms.

19. What is the difference between a Counter and a Gauge in Prometheus?

In Prometheus, metric types define how data is collected and interpreted.

A Counter is a cumulative metric that only increases over time. It is useful for tracking total occurrences of events such as the number of HTTP requests or errors. Counters cannot decrease, except when they are reset (e.g., service restart).

Example of a Counter:

```
http_requests_total{method="GET"} 2000
```

If 5 new requests occur, the new value will be 2005.

A Gauge, on the other hand, represents a value that can increase or decrease over time. It is useful for tracking metrics like CPU usage, memory utilization, and active connections.

Example of a Gauge:

```
node_memory_free_bytes 1234567890
```

The value of this metric can go up or down based on system usage.

Key Difference

- **Use a Counter when tracking an event that only increases (e.g., request count).**
- **Use a Gauge when tracking a value that fluctuates (e.g., temperature, memory).**

20. What is a Histogram in Prometheus? How does it work?

A Histogram in Prometheus is used to measure the distribution of observed values over a set of predefined buckets. It is useful for tracking request duration, response sizes, or latencies.

When a metric is recorded in a histogram, it is stored in multiple bucket ranges. Each bucket keeps track of the number of values that fall within that range.

Example metric in Prometheus:

```
http_request_duration_seconds_bucket{le="0.1"} 500
http_request_duration_seconds_bucket{le="0.2"} 750
http_request_duration_seconds_bucket{le="0.5"} 1000
http_request_duration_seconds_bucket{le="1"} 1200
http_request_duration_seconds_bucket{le="+Inf"} 1500
```

This means:

- **500 requests completed in ≤ 0.1 seconds**
- **750 requests completed in ≤ 0.2 seconds**
- **1000 requests completed in ≤ 0.5 seconds**
- **1200 requests completed in ≤ 1 second**
- **1500 requests in total.**

To analyze this data, you can use:

```
histogram_quantile(0.95,
rate(http_request_duration_seconds_bucket[5m]))
```

This will return the 95th percentile request duration.

Use Cases

- **Measuring request latencies in an application.**
- **Analyzing distribution of file sizes in a system.**

21. What is a Summary in Prometheus? How is it different from a Histogram?

A Summary in Prometheus is another way to track distributions, but instead of buckets, it calculates quantiles directly.

Example Summary metric:

```
http_request_duration_seconds{quantile="0.5"} 0.15
http_request_duration_seconds{quantile="0.9"} 0.3
http_request_duration_seconds{quantile="0.99"} 0.5
```

This tells us:

- **50% of requests were ≤ 150ms.**
- **90% of requests were ≤ 300ms.**
- **99% of requests were ≤ 500ms.**

Key Differences Between Histogram and Summary

- **A Histogram stores bucketed data and calculates percentiles later.**
- **A Summary computes quantiles on ingestion, which makes it harder to aggregate.**

Use a Histogram if you need flexible, post-processing aggregation.
Use a Summary if you need precomputed quantiles and don't need aggregation.

22. How does Prometheus handle service discovery?

Prometheus supports dynamic service discovery using various mechanisms. This eliminates the need for manual configuration when adding new targets.

Supported Service Discovery Mechanisms

1. **Kubernetes SD: Automatically discovers pods, nodes, and services.**
2. **Consul SD: Used in dynamic environments like microservices.**

3. **EC2 SD: Finds AWS instances.**
4. **File-based SD: Uses JSON or files to define targets.**

Example Kubernetes service discovery configuration:

```
scrape_configs:
  - job_name: 'kubernetes-nodes'
    kubernetes_sd_configs:
      - role: node
```

This dynamically collects metrics from all Kubernetes nodes.

Advantages of Service Discovery

- **No need to manually update target lists.**
- **Automatically adapts to scaling services.**
- **Reduces operational overhead in dynamic environments.**

23. What is Prometheus Federation? How does it work?

Federation is a scalability feature in Prometheus that allows multiple Prometheus instances to aggregate metrics.

Why Use Federation?

- **When a single Prometheus server cannot handle all the metrics.**
- **When monitoring multiple data centers or clusters.**
- **When you need global and local views of metrics.**

How Federation Works

- **A federating Prometheus instance pulls specific metrics from multiple Prometheus instances.**

Example configuration:

```
scrape_configs:
  - job_name: 'federate'
    honor_labels: true
    scrape_interval: 15s
    metrics_path: '/federate'
    params:
      'match[]':
        - '{job="node"}'
        - '{job="application"}'
    static_configs:
      - targets:
          - 'prometheus-1:9090'
          - 'prometheus-2:9090'
```

This setup collects only the `{job="node"}` and `{job="application"}` metrics from two Prometheus instances.

Key Benefits

- **Improves horizontal scalability.**
- **Helps in multi-cluster environments.**
- **Enables centralized monitoring.**

24. What are the limitations of Prometheus?

While Prometheus is powerful, it has some limitations:

1. **No built-in long-term storage**
    - **Prometheus stores data locally, and older data is deleted after retention.**
    - **To store metrics long-term, external storage like Thanos, Cortex, or VictoriaMetrics is needed.**
2. **Single-node architecture**
    - **Prometheus does not scale horizontally on its own.**
    - **High-availability setups require multiple Prometheus instances with federation.**
3. **Limited multi-tenancy support**
    - **Prometheus does not have strong multi-tenant isolation.**
    - **Solutions like Mimir, Thanos, and Cortex solve this.**
4. **High cardinality issues**
    - **Prometheus struggles with high-cardinality metrics (e.g., metrics with unique user IDs).**
    - **Too many unique labels increase memory usage.**
5. **No built-in anomaly detection**
    - **Prometheus does not provide ML-based anomaly detection.**
    - **External tools like Prometheus AI models, Grafana Loki, or Unomaly are required.**

How to Overcome These Limitations?

- **Use Thanos or Cortex for long-term storage.**
- **Use federation for horizontal scalability.**
- **Optimize metrics by reducing label cardinality.**

25. What is Thanos, and how does it extend Prometheus?

Thanos is an open-source project that extends Prometheus with long-term storage and high availability.

Key Features of Thanos

1. **Long-term Storage: Stores data in cloud storage (AWS S3, GCS, etc.).**
2. **Global View: Aggregates multiple Prometheus instances into one queryable system.**
3. **Downsampling: Reduces storage by compressing older data.**
4. **Highly Available: Allows HA Prometheus setups.**

How Thanos Works

Thanos runs as a sidecar with Prometheus and uploads data to object storage. It then allows querying from this long-term storage.

Example Thanos deployment:

```
[Prometheus] → [Thanos Sidecar] → [Object Storage]
                                        ↓
                                [Thanos Query Layer]
```

This enables global queries across Prometheus instances.

26. What is Cortex, and how does it compare to Thanos?

Cortex is an open-source system that enables multi-tenant, horizontally scalable Prometheus storage.

Key Features of Cortex

- **Multi-tenancy: Multiple teams can use the same Cortex cluster with isolation.**
- **Horizontal scalability: Unlike Prometheus, Cortex scales across multiple nodes.**
- **Long-term storage: Supports storage in object stores like AWS S3, GCS, and DynamoDB.**
- **Query across clusters: Allows querying multiple Prometheus instances in one place.**

Cortex vs. Thanos

- **Cortex is designed for multi-tenancy, while Thanos is better for single-tenant, HA use cases.**
- **Thanos uses object storage for long-term storage, while Cortex uses distributed databases.**
- **Cortex supports a microservices-based architecture, whereas Thanos extends Prometheus with a sidecar approach.**

27. How does Prometheus handle data retention and compaction?

Prometheus automatically deletes older data based on retention settings.

Data Retention Settings

Prometheus keeps data for a configured period using the `--storage.tsdb.retention.time` flag.

Example:

```
prometheus --storage.tsdb.retention.time=15d
```

This keeps data for 15 days before deletion.

Data Compaction Process

Prometheus stores data in block files, and compaction merges smaller blocks into larger ones to optimize storage.

- **Newly written data stays in the Head block (in-memory).**
- **After 2 hours, it is flushed to disk as a block.**
- **Blocks older than the retention period are deleted automatically.**

To control retention and storage location:

```
--storage.tsdb.retention.time=30d
--storage.tsdb.path=/data/prometheus
```

This stores data for 30 days in `/data/prometheus`.

28. What happens when Prometheus crashes? How does it recover?

Prometheus uses a Write-Ahead Log (WAL) to ensure data durability.

Crash Recovery Mechanism

1. **When Prometheus starts, it reads the WAL from disk.**
2. **It reconstructs time-series data from the WAL.**
3. **Any unsaved in-memory data is restored.**

The WAL is stored in:

`/data/prometheus/wal`

This ensures minimal data loss even after a crash.

29. How to monitor Prometheus itself?

Prometheus exposes its own metrics under the `/metrics` endpoint.

Important Metrics to Monitor

Prometheus scrape health:

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

1. **This checks how often Prometheus scrapes targets.**

Storage usage:

```
prometheus_tsdb_storage_blocks_bytes
```

2. **This tracks disk space usage.**

Query execution time:

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

3. **Ensures queries aren't too slow.**

Target failures:

```
count(up == 0)
```

4. **Counts the number of down targets.**

30. What is the WAL (Write-Ahead Log) in Prometheus?

The Write-Ahead Log (WAL) ensures data durability by recording metrics before writing them to disk.

How WAL Works

- **All new metrics are first written to WAL (before committing to TSDB).**
- **If Prometheus crashes, the WAL is replayed on restart.**
- **Old WAL files are deleted automatically when not needed.**

Location of WAL Files

Stored in:

`/data/prometheus/wal`

Each WAL segment contains compressed time-series records.

31. How to reduce Prometheus memory usage?

Prometheus can be memory-intensive, but you can optimize it:

1. **Reduce label cardinality:**
   - **Avoid high-cardinality labels like `user_id`, `session_id`.**
   - **Instead, aggregate data at scrape time.**
2. **Reduce scrape intervals:**
   - **Instead of `scrape_interval=5s`, use `scrape_interval=15s`.**
3. **Enable memory-mapped TSDB blocks:**
   - **Set `GOMEMLIMIT` in environment variables to control memory.**

4. Lower retention time:
    - Instead of `--storage.tsdb.retention.time=90d`, use `--storage.tsdb.retention.time=30d`.
5. Use Remote Write for long-term storage:
    - Offload old data to Thanos, Cortex, or VictoriaMetrics.

32. How does Prometheus remote storage work?

Prometheus natively supports remote storage for long-term data retention.

How Remote Storage Works

- Prometheus writes data to an external system instead of its own TSDB.
- It uses the `remote_write` and `remote_read` configurations.

Example configuration:

```
remote_write:
  - url: "http://thanos-receiver:10901/api/v1/receive"
```

This sends metrics to Thanos for long-term storage.

Popular Remote Storage Solutions

- Thanos – Cloud-native storage.
- Cortex – Multi-tenant storage.
- VictoriaMetrics – High-performance, efficient storage.

33. How to debug slow PromQL queries?

PromQL queries can be slow due to large datasets or inefficient queries.

Steps to Optimize PromQL Queries

1. **Use aggregation functions**

Instead of:

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

Use:

```
sum(rate(http_requests_total[5m]))
```

2. **Avoid high-cardinality labels**
    - **Queries with `group by user_id` are slow.**
3. **Limit time range**
    - **Instead of querying 30 days, limit queries to 1 day.**

Check query execution time

```
prometheus_engine_query_duration_seconds
```

4. **Enable query logging**

Run Prometheus with:

```
--log.level=debug
```

34. How to visualize Prometheus metrics in Grafana?

Grafana is used for visualizing Prometheus metrics.

Steps to Connect Prometheus to Grafana

Install Grafana

```
sudo apt-get install grafana
```

1. **Add Prometheus as a data source**
    - **Open Grafana UI → Configuration → Data Sources → Add Prometheus.**

Set URL as:

```
http://localhost:9090
```

2. **Create a new dashboard**
    - **Go to Dashboards → New Panel.**

Use a PromQL query:

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

- **Save and visualize.**

35. How to scale Prometheus for large environments?

In large infrastructures, scaling Prometheus is necessary to handle more metrics.

Methods to Scale Prometheus

1. **Federation:**
    - **Deploy multiple Prometheus instances per environment.**
    - **Use a central Prometheus to query aggregated data.**

2. Sharding:
    - Distribute scraping across multiple Prometheus instances.
    - Example:
        - Prometheus-1 scrapes Node A, B.
        - Prometheus-2 scrapes Node C, D.
3. Use Thanos or Cortex for long-term storage:
    - Offload historical data to Thanos, Cortex, or VictoriaMetrics.
4. Reduce scrape interval for non-critical metrics:
    - Critical metrics: `scrape_interval=5s`
    - Non-critical metrics: `scrape_interval=30s`

36. What are Prometheus relabeling rules, and how do they work?

Relabeling in Prometheus is a powerful mechanism used to modify, filter, or transform labels before storing metrics or scraping targets.

Use Cases of Relabeling

1. **Drop unwanted targets**
2. **Rename labels**
3. **Modify label values**
4. **Reduce high-cardinality metrics**
5. **Filter data before ingestion**

Example Relabeling Rule

Let's assume Prometheus scrapes from Kubernetes, but you want to drop all `pod` labels.

```
Scrape_configs:
```

```
- job_name: 'kubernetes-pods'
  relabel_configs:
    - source_labels: [__meta_kubernetes_pod_name]
      regex: ".*"
      action: drop
```

This drops the `pod_name` label from all targets.

Relabeling Actions

- drop: Excludes the target from scraping.
- keep: Keeps only matching targets.
- replace: Renames or modifies labels.
- labelmap: Renames multiple labels using a regex.

Example: Renaming a Label

```
relabel_configs:
  - source_labels: [instance]
    regex: "(.*):9090"
    target_label: instance
    replacement: "$1"
```

This removes the `:9090` port from the `instance` label.

37. How does Prometheus handle multi-tenancy?

Prometheus does not natively support multi-tenancy, meaning each team typically runs its own Prometheus instance. However, you can achieve multi-tenancy using external tools.

Approaches for Multi-Tenancy

1. **Cortex or Thanos**
    - **Both support tenant-based isolation.**
    - **Each tenant has its own metrics storage and queries.**
2. **Separate Prometheus Instances**
    - **Run multiple Prometheus servers, one per tenant.**
    - **Each instance scrapes only its own team's services.**
3. **Label-based Isolation**
    - **Use labels like `{tenant="teamA"}` in queries.**

Example PromQL for filtering tenant data:

```
http_requests_total{tenant="teamA"}
```

4. **Federation**
    - **Each team has a Prometheus instance that feeds into a global Prometheus.**

38. What is the default scrape interval in Prometheus?

By default, Prometheus scrapes targets every 15 seconds.

Defined in `prometheus.yml`:

```
global:
  scrape_interval: 15s
```

Customizing Scrape Intervals

Increase it for low-priority services

```
scrape_configs:
  - job_name: 'slow_metrics'
    scrape_interval: 1m
```

Decrease it for high-precision monitoring

```
scrape_configs:
  - job_name: 'critical_services'
    scrape_interval: 5s
```

Setting `scrape_interval` too low (e.g., 1s) can overload Prometheus.

39. How to create custom metrics for Prometheus?

You can expose custom application metrics using Prometheus client libraries.

Step 1: Install a Prometheus Client

For Python:

```
pip install prometheus-client
```

Step 2: Create a Metric

Example Python code:

```
from prometheus_client import start_http_server,
Counter
import time
```

```
REQUEST_COUNT = Counter('http_requests_total', 'Total
HTTP requests')

def process_request():
    REQUEST_COUNT.inc()
    time.sleep(1)

if __name__ == "__main__":
    start_http_server(8000)
    while True:
        process_request()
```

This exposes `http_requests_total` on `http://localhost:8000/metrics`.

40. What are Pushgateway and its use cases?

Unlike Prometheus' pull-based model, the Pushgateway allows short-lived jobs to push metrics.

When to Use Pushgateway?

- **Batch jobs that run and exit before Prometheus scrapes.**
- **CI/CD pipelines that execute quick tasks.**
- **Cron jobs that generate logs.**

Example Usage

```
echo "job_duration_seconds 3.5" | curl --data-binary @-
```

```
http://localhost:9091/metrics/job/myjob
```

This stores `job_duration_seconds` in Pushgateway.

⚠️ Do not use Pushgateway for real-time services (use exporters instead).

41. How to filter metrics in Prometheus?

Prometheus allows filtering using label matchers.

Common Filtering Operators

- `{key="value"}` → Exact match
- `{key!="value"}` → Not equal
- `{key=~"regex"}` → Regex match
- `{key!~"regex"}` → Regex not match

Example Queries

Filter by status code

```
http_requests_total{status="500"}
```

Exclude 404 errors

```
http_requests_total{status!="404"}
```

Regex match for API endpoints

```
http_requests_total{endpoint=~"/api/.*"}
```

42. How does Prometheus handle high availability (HA)?

Prometheus itself does not support HA, but HA can be achieved with redundant Prometheus instances.

Approaches for HA

1. **Run multiple Prometheus instances**
    - **Two Prometheus servers scrape the same targets.**
    - **Use Grafana or Alertmanager deduplication to avoid duplicate alerts.**
2. **Use Thanos**
    - **Thanos de-duplicates data across multiple Prometheus instances.**
3. **Use Cortex**
    - **Cortex runs multiple Prometheus servers in parallel, ensuring HA.**

43. How to set up Alertmanager with Prometheus?

Alertmanager routes alerts from Prometheus to notification channels like Slack, PagerDuty, or email.

Step 1: Configure Prometheus to Send Alerts

```
alerting:
  alertmanagers:
    - static_configs:
        - targets:
          - "alertmanager:9093"
```

Step 2: Define an Alerting Rule

```
groups:
  - name: CPUAlerts
    rules:
      - alert: HighCPU
        expr: rate(node_cpu_seconds_total[5m]) > 0.9
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "High CPU usage detected"
```

Step 3: Configure Alertmanager to Route Alerts

Example Slack notification:

```
receivers:
  - name: "slack-notifier"
    slack_configs:
      - channel: "#alerts"
        send_resolved: true
        api_url:
"https://hooks.slack.com/services/xxxxx"
```

This sends critical alerts to Slack.

44. How to monitor microservices with Prometheus?

To monitor microservices, follow these steps:

1. **Expose Metrics in Each Microservice**
    - **Use Prometheus client libraries.**
    - **Add a `/metrics` endpoint.**

Deploy Prometheus Scraping Rules

```
scrape_configs:
 - job_name: 'microservices'
   static_configs:
     - targets: ['service1:8000', 'service2:8000']
```

2. **Set Up Dashboards in Grafana**

Use queries like:

```
rate(http_requests_total{service="payment"}[5m])
```

- **Create separate dashboards for each microservice.**

Set Up Alerts for Microservice Failures

```
groups:
  - name: ServiceDownAlerts
    rules:
      - alert: ServiceDown
        expr: up{job="service1"} == 0
        for: 2m
        labels:
          severity: critical
```

45. How to integrate Prometheus with Kubernetes for monitoring?

Prometheus has native support for Kubernetes, allowing it to automatically discover and monitor containers, nodes, and services.

Steps to Set Up Prometheus with Kubernetes

Deploy Prometheus in Kubernetes using Helm (Recommended)

```
helm repo add prometheus-community
https://prometheus-community.github.io/helm-charts
helm install prometheus
prometheus-community/kube-prometheus-stack
```

1. **This installs:**
    - **Prometheus Server (collects metrics)**
    - **Alertmanager (handles alerts)**
    - **Grafana (visualizes data)**
    - **Kubernetes Exporters (collects cluster metrics)**

Configure Kubernetes Service Discovery Prometheus automatically discovers services, nodes, and pods.

Example `prometheus.yml`:

```
scrape_configs:
  - job_name: 'kubernetes-nodes'
    kubernetes_sd_configs:
      - role: node

  - job_name: 'kubernetes-pods'
    Kubernetes_sd_configs:
```

```
      - role: pod
```

2. This dynamically scrapes all Kubernetes nodes and pods.
3. Expose Metrics in Kubernetes Pods
    - Each pod should expose metrics at `/metrics`.

Example Python Flask app:

```
from prometheus_client import start_http_server,
Counter
import time

REQUEST_COUNT = Counter('http_requests_total', 'Total
HTTP requests')

if __name__ == "__main__":
    start_http_server(8000)
    while True:
        REQUEST_COUNT.inc()
        time.sleep(1)
```

4. Deploy Prometheus Operator (for Advanced Users)
    - Prometheus Operator simplifies deployment.
    - Manages Prometheus, Alertmanager, and rules as Kubernetes CRDs.

```
helm install prometheus-operator
prometheus-community/kube-prometheus-stack
```

Why Use Prometheus for Kubernetes?

- **Auto-discovery of services and pods.**
- **Deep integration with Kubernetes events and state.**
- **Powerful alerting and querying on cluster health.**

46. What are Kubernetes Exporters in Prometheus?

Prometheus exporters are agents that expose metrics from Kubernetes components.

Common Kubernetes Exporters

1. **kube-state-metrics**
    - **Provides cluster state metrics (pods, deployments, nodes).**

Example metric:

```
kube_pod_status_ready{pod="nginx", namespace="default"}
```

2. **cAdvisor (Container Advisor)**
    - **Collects container-level resource metrics (CPU, memory, disk I/O).**

Example metric:

```
container_memory_usage_bytes{name="nginx"} 1048576
```

3. **Node Exporter**
    - **Monitors host-level system metrics.**

Example metric:

```
node_cpu_seconds_total{mode="user"} 12345
```

4. **Blackbox Exporter**
    - **Probes network endpoints (HTTP, TCP, DNS).**

Example usage:

```
curl
"http://localhost:9115/probe?target=https://example.com
"
```

How to Use These Exporters?

- **Deploy exporters as Kubernetes DaemonSets.**
- **Add them to Prometheus scrape targets.**

47. How to secure Prometheus?

By default, Prometheus does not have authentication, making it vulnerable to unauthorized access.

Ways to Secure Prometheus

1. **Enable Basic Authentication with Nginx**

Install Nginx and create a password file:

```
sudo apt install apache2-utils
htpasswd -c /etc/nginx/.htpasswd admin
```

Configure Nginx as a reverse proxy:

```
server {
  listen 9090;
  location / {
    auth_basic "Restricted";
    auth_basic_user_file /etc/nginx/.htpasswd;
    proxy_pass http://localhost:9090;
  }
}
```

2. **Use TLS for Encryption**
    - **Enable HTTPS using `nginx` or `Traefik`.**

Example TLS settings in Nginx:

```
ssl_certificate /etc/ssl/certs/prometheus.crt;
ssl_certificate_key /etc/ssl/private/prometheus.key;
```

3. **Use OAuth Authentication**
    - **Integrate Prometheus with OAuth providers (Google, GitHub, Okta).**

Example using OAuth2 Proxy:

```
args:
  - --provider=google
  - --upstream=http://localhost:9090
```

4. **Restrict API Access**

Disable UI and API with:

```
--web.enable-admin-api=false
```

5. Use RBAC in Kubernetes
    - Limit Prometheus' access with role-based access control.

48. How to troubleshoot missing metrics in Prometheus?

If a metric is not appearing in Prometheus, follow these steps:

Step 1: Check If the Exporter is Running

Run:

```
curl http://localhost:9100/metrics
```

- If it responds, the exporter is working.
- If not, check the service logs.

Step 2: Verify Prometheus Scraping

Use:

```
up
```

- If `up{job="node"}` is `0`, Prometheus cannot reach the target.

Step 3: Check Scrape Configuration

Open `prometheus.yml` and verify:

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

- Ensure the target exists and is accessible.

Step 4: Check Prometheus Logs

Run:

```
docker logs prometheus
```

- Look for scrape errors or connection timeouts.

Step 5: Check Label Filters

Query:

```
{job="node"}
```

- If empty, the labels may not match in queries.

49. How does Prometheus handle metric expiration?

Prometheus automatically deletes metrics if a target stops sending data.

How Metric Expiry Works

1. If a metric is no longer scraped, it remains in the database until retention expires.
2. After retention time, Prometheus deletes the old data.

Example retention setting:

```
--storage.tsdb.retention.time=15d
```

This keeps data for 15 days.

How to Prevent Expiry?

- **Ensure services expose metrics consistently.**
- **Use recording rules to keep derived metrics.**

50. How does Prometheus handle timestamp alignment?

Prometheus aligns timestamps to the nearest scrape interval.

Example:

If `scrape_interval=10s`, metrics appear at:
makefile

```
12:00:00
12:00:10
12:00:20
```

- **If a metric arrives late, Prometheus adjusts the timestamp.**

Avoiding Timestamp Issues

1. **Ensure time synchronization using `ntpd` or `chrony`.**
2. **Use `rate()` instead of `increase()` to handle missing data.**