

How to Deploy Prometheus – Metrics Collection Server

Deploy Prometheus with Docker Compose and Traefik for secure, scalable metrics monitoring.

Contents

01	Introduction	3
02	Prerequisites	3
03	Set Up the Directory Structure and Environment Variables	3
04	Deploy with Docker Compose	5
05	Access Prometheus	9
06	Conclusion	10

Introduction

Prometheus is an open-source monitoring and alerting toolkit designed for reliability and scalability. It collects and stores metrics as time-series data with timestamps and optional labels, enabling powerful querying and analysis of your infrastructure and applications.

In this article, you will deploy Prometheus using Docker Compose, configure persistent storage for metrics data, and set up Traefik as a reverse proxy to securely access your Prometheus instance.

Prerequisites

Before you begin, you need to:

- Have access to an [Ubuntu 24.04-based server](#) as a [non-root user with `sudo` privileges](#).
- Install [Docker and Docker Compose](#).
- Configure a [domain A record](#) pointing to your server's IP address (for example, `prometheus.example.com`).

Set Up the Directory Structure and Environment Variables

In this section, you prepare the required directory structure for Prometheus and define environment variables in a `.env` file.

1. Create the directory structure for Prometheus.

CONSOLE

```
$ mkdir -p ~/prometheus-monitoring/{prometheus-  
data,prometheus-config}
```

These directories serve different purposes:

- **prometheus-data**: Contains the time-series database where metrics are stored.
- **prometheus-config**: Stores YAML configuration files for Prometheus.

2. Navigate into the `prometheus-monitoring` directory.

```
CONSOLE
```

```
$ cd ~/prometheus-monitoring
```

3. Set proper ownership for the Prometheus data directory. Prometheus runs as the `nobody` user (UID 65534) inside the container.

```
CONSOLE
```

```
$ sudo chown -R 65534:65534 prometheus-data
```

4. Create a `.env` file.

```
CONSOLE
```

```
$ nano .env
```

Add the following variables:

```
INI
```

```
DOMAIN=prometheus.example.com  
LETSENCRYPT_EMAIL=admin@example.com
```

Replace:

- `prometheus.example.com` with your domain.
- `admin@example.com` with your email.

Save and close the file.

5. Create a Prometheus configuration file.

CONSOLE

```
$ nano prometheus-config/prometheus.yml
```

6. Add the following content.

YAML

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

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
```

Save and close the file.

The configuration includes:

- **global**: Sets the default intervals that control how often Prometheus scrapes metrics from targets and evaluates recording and alerting rules (every 15 seconds).
- **scrape_configs**: Defines which targets to scrape metrics from. Here, Prometheus monitors its own `/metrics` endpoint.

Deploy with Docker Compose

In this section, you create and deploy the Docker Compose stack that runs Prometheus behind Traefik. Docker Compose manages both containers, applies the environment variables from your `.env` file, and automatically configures HTTPS routing through Traefik.

1. Create a new Docker Compose manifest.

CONSOLE

```
$ nano docker-compose.yaml
```

2. Add the following content.

YAML

```
services:
  traefik:
    image: traefik:v3.6
    container_name: traefik
    command:
      - "--providers.docker=true"
      - "--providers.docker.exposedbydefault=false"
      - "--entrypoints.web.address=:80"
      - "--entrypoints.websecure.address=:443"
      - "--
entrypoints.web.http.redirections.entrypoint.to=websecure"
      - "--
entrypoints.web.http.redirections.entrypoint.scheme=https"
      - "--
certificatesresolvers.letsencrypt.acme.httpchallenge=true"
      - "--
certificatesresolvers.letsencrypt.acme.httpchallenge.entrypoint=web"
      - "--certificatesresolvers.letsencrypt.acme.email=${LETSENCRYPT_EMAIL}"
      - "--certificatesresolvers.letsencrypt.acme.storage=/letsencrypt/acme.json"
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - "letsencrypt:/letsencrypt"
      - "/var/run/docker.sock:/var/run/docker.sock:ro"
    restart: unless-stopped

  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    hostname: prometheus
    expose:
      - "9090"
    volumes:
```

```
- "./prometheus-config:/etc/prometheus"
- "./prometheus-data:/prometheus"
command:
- '--config.file=/etc/prometheus/prometheus.yml'
- '--storage.tsdb.path=/prometheus'
labels:
- "traefik.enable=true"
- "traefik.http.routers.prometheus.rule=Host(`${DOMAIN}`)"
-
"traefik.http.routers.prometheus.entrypoints=websecure"
-
"traefik.http.routers.prometheus.tls.certresolver=letsencrypt"
-
restart: unless-stopped

volumes:
letsencrypt:
```

Save and close the file.

This configuration defines:

- **services:** Two containerized components deliver the monitoring solution:
 - **traefik:** Accepts external connections, provisions TLS certificates, and proxies traffic to Prometheus.
 - **prometheus:** Executes the metrics collection engine and maintains time-series databases.
- **image:** Container images are retrieved from their respective Docker Hub repositories.
- **container_name:** Static container names simplify administrative tasks such as log analysis and service inspection.
- **command (Traefik):** Establishes Traefik's operational parameters including Docker container discovery, port binding (80/443), HTTP-to-HTTPS enforcement, and automated Let's Encrypt certificate acquisition via HTTP challenge validation.
- **ports (Traefik):** Binds ports 80 and 443 to the host network interface for accepting inbound HTTP/HTTPS connections.

- **expose (Prometheus)**: Makes port 9090 available within the Docker network for Traefik communication while preventing direct external access.
- **volumes**:
 - Bind-mounted directories (`./prometheus-config`, `./prometheus-data`) ensure configuration files and metric databases survive container recreation.
 - The `letsencrypt` named volume retains TLS certificate data between container lifecycles.
 - Mounting `/var/run/docker.sock` grants Traefik read-only access to the Docker daemon for service discovery.
- **command (Prometheus)**: Defines configuration file location and specifies the directory path for storing time-series metric data.
- **labels (Prometheus)**: Traefik routing directives that activate proxy functionality, establish domain-based routing rules, and select the appropriate certificate resolver.
- **restart: unless-stopped**: Implements automatic container recovery following host reboots or service failures, excluding manual stop events.

3. Create and start the services.

CONSOLE

```
$ docker compose up -d
```

4. Verify that the services are running.

CONSOLE

```
$ docker compose ps
```

Output:

NAME	IMAGE	COMMAND	SERVICE
prometheus	prom/prometheus:latest	"/bin/prometheus --c..."	prometheus
seconds ago	Up 44 seconds	9090/tcp	45
traefik	traefik:v3.6	"/entrypoint.sh --pr..."	traefik
			45

```
seconds ago   Up 44 seconds   0.0.0.0:80->80/tcp, [::]:80->80/tcp, 0.0.0.0:443->443/tcp, [::]:443->443/tcp
```

Both containers are running as expected. Prometheus collects metrics while Traefik accepts connections on ports `80` and `443`.

5. View the logs of the services.

CONSOLE

```
$ docker compose logs
```

For more information on managing a Docker Compose stack, see the [How To Use Docker Compose](#) article.

Access Prometheus

In this section, you access the Prometheus web interface and verify your monitoring setup by running basic queries to explore collected metrics.

1. Open the Prometheus web interface in your browser.

```
https://prometheus.example.com
```

2. Click **Status > Target health** to view monitored endpoints. You should see the `prometheus` job with a state of `UP`.

3. Navigate to the **Query** tab to explore metrics using PromQL.

4. Try these example queries:

- Check target scrape status:

PROMQL

```
> _ up
```

- View CPU usage:

```
PROMQL
```

```
> _ process_cpu_seconds_total
```

- View memory usage:

```
PROMQL
```

```
> _ process_resident_memory_bytes
```

5. Click **Execute** after entering each query to see the results. Switch to the **Graph** tab to visualize metrics over time.

Conclusion

You have successfully deployed Prometheus for system monitoring with encrypted HTTPS connectivity. The Docker Compose deployment includes both the Prometheus metrics engine and Traefik reverse proxy, with durable storage ensuring data retention across restarts. Let's Encrypt certificate automation provides secure connections while Traefik handles intelligent request routing. Your monitoring infrastructure is ready for expansion through additional exporters, custom application instrumentation, and alerting rules.



VULTR

