

How to Use Vultr's Prometheus Marketplace Application

Learn how to deploy, configure, and use Vultr's Prometheus Marketplace Application for effective monitoring of your infrastructure and applications.

Contents

01	Introduction	3
02	Deploy Vultr's Prometheus Marketplace Application	3
03	Initial Setup and Configuration	4
04	Configure Prometheus	9
05	Explore Prometheus Features	12
06	Integrations and Monitoring	14
07	Best Practices and Configuration	18
08	Troubleshooting	21
09	Use Cases	23
010	Conclusion	23

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 key-value labels. With its powerful query language (PromQL), service discovery mechanisms, and pull-based architecture, Prometheus excels at monitoring dynamic cloud environments and microservices. The Vultr Marketplace provides a pre-configured Prometheus instance, enabling quick deployment and setup on a Vultr server.

This guide explains deploying and using [Vultr's Prometheus Marketplace Application](#). You will deploy an instance, configure security and SSL, set up monitoring targets, explore PromQL queries, integrate with Node Exporter and Alertmanager, and implement best practices for production monitoring.

Deploy Vultr's Prometheus Marketplace Application

1. Log in to your [Vultr Customer Portal](#) and click the **Deploy Server** button.
2. Select your preferred server type.
3. Choose a server location.
4. Select a server plan with at least **2GB RAM** and **2 CPU cores** for production workloads.
5. Click the **Configure** button to proceed.
6. Under **Marketplace Apps**, search for `Prometheus` and select it as the Marketplace Application.
7. Select the **Limited Login** option from the **Additional Features** section to create a limited user with sudo access.

- Review your configurations and click the **Deploy Now** button to start deployment.

Note

It may take up to 10 minutes for your server to finish installing Prometheus.

- After the instance shows the status of Running, navigate to the **Server Overview** page and copy the SSH connection details.

Initial Setup and Configuration

After deployment, configure DNS, verify the installation, and secure your Prometheus instance with proper authentication and SSL/TLS before exposing it to production traffic.

- Create a [DNS A record](#) pointing to your server's IP address, such as `prometheus.example.com`.
- Connect to your Vultr server instance over [SSH](#) using the connection details from the Server Overview page.

Verify Prometheus Installation

- Check the Prometheus service status.

CONSOLE

```
$ sudo systemctl status prometheus
```

The service should show as `active (running)`.

- Verify the installed Prometheus version.

CONSOLE

```
$ prometheus --version
```

Output:

```
prometheus, version 3.7.0 (branch: HEAD, revision:
09814effe6c3b4fb4c8e98b578f7205d0228f04d)
  build user:      root@84cfe63a2ab5
  build date:      20251015-10:17:59
  go version:      go1.25.3
  platform:        linux/amd64
  tags:            netgo,builtinassets
```

3. Access Prometheus by visiting `prometheus.example.com` in a web browser.
4. Log in with the default credentials from the **Server Overview** page.

Configure Firewall Security

Secure your server by configuring the firewall to allow only necessary traffic.

1. Allow SSH connections.

CONSOLE

```
$ sudo ufw allow OpenSSH
```

2. Allow HTTP and HTTPS traffic for Nginx and Certbot.

CONSOLE

```
$ sudo ufw allow 80/tcp
$ sudo ufw allow 443/tcp
```

3. Enable the firewall.

CONSOLE

```
$ sudo ufw enable
```

4. Verify firewall status.

```
CONSOLE
```

```
$ sudo ufw status
```

Port 9090 is used by Prometheus's web UI. You will remove this rule after enabling SSL through the Nginx reverse proxy.

Configure Reverse Proxy with Nginx

Set up Nginx as a reverse proxy to serve Prometheus over standard HTTP/HTTPS ports with authentication.

1. Install the Nginx web server package.

```
CONSOLE
```

```
$ sudo apt install nginx -y
```

2. Create a password file for basic authentication.

```
CONSOLE
```

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

Enter a secure password when prompted. This creates the admin user for accessing Prometheus.

3. Create an Nginx virtual host configuration for Prometheus.

```
CONSOLE
```

```
$ sudo nano /etc/nginx/sites-available/prometheus
```

```
INI
```

```
server {  
    listen 80;  
    server_name prometheus.example.com;
```

```
auth_basic "Prometheus Authentication";
auth_basic_user_file /etc/nginx/.htpasswd;

location / {
    proxy_pass http://localhost:9090;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_read_timeout 300;
    proxy_send_timeout 300;
}
}
```

Replace `prometheus.example.com` with your domain name.

Save and close the file.

4. Enable the Prometheus server block.

CONSOLE

```
$ sudo ln -s /etc/nginx/sites-available/prometheus /etc/nginx/sites-enabled/
```

5. Test the Nginx configuration syntax.

CONSOLE

```
$ sudo nginx -t
```

Output:

```
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

6. Reload Nginx to apply the changes.

CONSOLE

```
$ sudo systemctl reload nginx
```

Secure Prometheus with SSL/TLS

Protect your Prometheus instance with HTTPS using Let's Encrypt certificates via Certbot.

1. Install Certbot and the Nginx plugin.

CONSOLE

```
$ sudo apt install certbot python3-certbot-nginx -y
```

2. Request an SSL certificate for your domain.

CONSOLE

```
$ sudo certbot --nginx -d prometheus.example.com
```

Follow the prompts and select the option to redirect HTTP traffic to HTTPS when asked.

3. Verify SSL certificate auto-renewal.

CONSOLE

```
$ sudo certbot renew --dry-run
```

4. Configure Prometheus to use the external URL for correct link generation.

CONSOLE

```
$ sudo nano /etc/default/prometheus
```

5. Add the following configuration.

INI

```
ARGS="--web.external-url=https://prometheus.example.com --web.route-prefix=/"
```

This ensures Prometheus generates correct URLs when behind a reverse proxy.

- Restart Prometheus to apply changes.

CONSOLE

```
$ sudo systemctl restart prometheus
```

- Access Prometheus securely at `https://prometheus.example.com`.
- Close direct port 9090 access now that traffic goes through the Nginx reverse proxy.

CONSOLE

```
$ sudo ufw delete allow 9090/tcp
```

Configure Prometheus

Set up monitoring targets, adjust scrape intervals, and configure Prometheus for your infrastructure.

Update Prometheus Configuration

- Edit the Prometheus configuration file.

CONSOLE

```
$ sudo nano /etc/prometheus/prometheus.yml
```

- Review the default global configuration and update as needed.

YAML

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s
  external_labels:
    monitor: 'vultr-prometheus'
    environment: 'production'

alerting:
  alertmanagers:
    - static_configs:
      - targets: []

rule_files:
  # - "alerts.yml"

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

Save and close the file.

3. Validate the configuration before restarting.

CONSOLE

```
$ promtool check config /etc/prometheus/prometheus.yml
```

4. Restart Prometheus to apply changes.

CONSOLE

```
$ sudo systemctl restart prometheus
```

5. Verify configuration was loaded successfully.

CONSOLE

```
$ sudo systemctl status prometheus
```

Add Monitoring Targets

Configure Prometheus to scrape metrics from additional targets.

1. Edit the Prometheus configuration to add new scrape targets.

CONSOLE

```
$ sudo nano /etc/prometheus/prometheus.yml
```

2. Add additional scrape configurations for your services.

YAML

```
scrape_configs:  
  - job_name: 'prometheus'  
    static_configs:  
      - targets: ['localhost:9090']  
  
  - job_name: 'node_exporter'  
    static_configs:  
      - targets: ['localhost:9100']  
  
  - job_name: 'custom_app'  
    static_configs:  
      - targets: ['app-server-1:8080', 'app-server-2:8080']  
    metrics_path: '/metrics'  
    scrape_interval: 10s
```

Replace the targets with your actual service endpoints.

Save and close the file.

3. Validate the configuration.

CONSOLE

```
$ promtool check config /etc/prometheus/prometheus.yml
```

4. Restart Prometheus.

```
CONSOLE
```

```
$ sudo systemctl restart prometheus
```

Explore Prometheus Features

Learn to query metrics, explore time-series data, and understand Prometheus's capabilities.

Access the Prometheus Web UI

1. Navigate to `https://prometheus.example.com` in a web browser.
2. Enter the basic authentication credentials you created earlier.
3. The Prometheus web interface provides:
 - **Graph**: Query and visualize metrics
 - **Alerts**: View active and pending alerts
 - **Status**: Check targets, configuration, and service discovery
 - **Help**: PromQL documentation and examples

Run PromQL Queries

Prometheus Query Language (PromQL) allows you to select and aggregate time-series data.

1. Navigate to the **Graph** tab in the Prometheus UI.
2. Query CPU utilization percentage.

```
PROMQL
```

```
> 100 - (avg by (instance)  
(irate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)
```

3. Query memory usage percentage.

PROMQL

```
> 1 - (node_memory_MemAvailable_bytes /  
node_memory_MemTotal_bytes) * 100
```

4. Query HTTP request rate.

PROMQL

```
> rate(prometheus_http_requests_total[5m])
```

5. Aggregate metrics across labels.

PROMQL

```
> sum by (job) (up)
```

This shows how many instances are up for each job.

6. Filter by label values.

PROMQL

```
> node_cpu_seconds_total{cpu="0", mode="system"}
```

View Target Status

1. Navigate to **Status > Target health** in the Prometheus UI.
2. Verify all configured targets show as **UP**.
3. If a target is down, check:
 - The target service is running
 - Firewall rules allow connections
 - The metrics endpoint is accessible

Integrations and Monitoring

Extend Prometheus capabilities by integrating with exporters, Alertmanager, and visualization tools.

Install Node Exporter

Node Exporter exposes system metrics like CPU, memory, disk, and network for monitoring.

1. Download and install Node Exporter.

CONSOLE

```
$ cd /tmp
$ curl -LO https://github.com/prometheus/node_exporter/releases/download/v1.7.0/node_exporter-1.7.0.linux-amd64.tar.gz
$ tar -xvf node_exporter-1.7.0.linux-amd64.tar.gz
$ sudo mv node_exporter-1.7.0.linux-amd64/node_exporter /usr/local/bin/
```

2. Create a dedicated system user for Node Exporter.

CONSOLE

```
$ sudo useradd --no-create-home --system --shell /usr/sbin/nologin node_exporter
```

3. Create a systemd service for Node Exporter.

CONSOLE

```
$ sudo nano /etc/systemd/system/node_exporter.service
```

4. Add the following configuration.

```
INI
```

```
[Unit]
Description=Node Exporter
After=network.target

[Service]
User=node_exporter
Group=node_exporter
Type=simple
ExecStart=/usr/local/bin/node_exporter

[Install]
WantedBy=multi-user.target
```

Save and close the file.

5. Start and enable Node Exporter.

```
CONSOLE
```

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable --now node_exporter
```

6. Verify Node Exporter is running.

```
CONSOLE
```

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

7. Prometheus will automatically scrape Node Exporter metrics if you added the `node_exporter` job in the configuration.

Configure Alertmanager

Alertmanager handles alerts sent by Prometheus, managing deduplication, grouping, and routing to notification channels.

1. Install Alertmanager.

CONSOLE

```
$ sudo apt install prometheus-alertmanager -y
```

2. Configure Alertmanager for email notifications.

CONSOLE

```
$ sudo nano /etc/alertmanager/alertmanager.yml
```

YAML

```
global:
  resolve_timeout: 5m

route:
  receiver: 'email-notifications'

receivers:
- name: 'email-notifications'
  email_configs:
  - to: 'admin@example.com'
    from: 'prometheus@example.com'
    smarthost: 'smtp.example.com:587'
    auth_username: 'prometheus@example.com'
    auth_password: 'your_email_password'
```

Replace with your SMTP settings.

Save and close the file.

3. Create alert rules.

CONSOLE

```
$ sudo nano /etc/prometheus/alerts.yml
```

YAML

```
groups:
- name: system_alerts
```

```
interval: 30s
rules:
  - alert: InstanceDown
    expr: up == 0
    for: 5m
    labels:
      severity: critical
    annotations:
      summary: "Instance {{ $labels.instance }} down"
      description: "{{ $labels.instance }} has been down
for more than 5 minutes."
```

Save and close the file.

4. Update Prometheus configuration to load alert rules.

CONSOLE

```
$ sudo nano /etc/prometheus/prometheus.yml
```

Add the following:

YAML

```
rule_files:
  - "alerts.yml"

alerting:
  alertmanagers:
    - static_configs:
      - targets: ['localhost:9093']
```

Save and close the file.

5. Validate and restart services.

CONSOLE

```
$ promtool check config /etc/prometheus/prometheus.yml
$ sudo systemctl restart prometheus
$ sudo systemctl restart alertmanager
```

Integrate with Grafana

Visualize Prometheus metrics using Grafana dashboards.

1. Deploy a Grafana instance following the [Vultr Grafana Marketplace Application guide](#).
2. Add Prometheus as a data source in Grafana:
 - Navigate to **Connections > Data sources**
 - Click **Add data source**
 - Select **Prometheus**
 - **URL:** `https://prometheus.example.com` OR `http://localhost:9090`
 - Enable **Basic auth** if using the authenticated endpoint
 - Click **Save & test**
3. Import pre-built Prometheus dashboards from [Grafana Dashboards](#):
 - Node Exporter Full: `1860`
 - Prometheus Stats: `3662`
 - System Monitoring: `11074`

Best Practices and Configuration

Implement these recommendations to ensure your Prometheus instance runs efficiently and reliably.

Data Retention and Storage

1. Configure data retention period based on your needs.

```
CONSOLE
```

```
$ sudo nano /etc/default/prometheus
```

2. Add storage retention flags.

INI

```
ARGS="--storage.tsdb.retention.time=30d --  
storage.tsdb.retention.size=50GB"
```

This retains data for 30 days or up to 50GB, whichever limit is reached first.

3. Restart Prometheus to apply changes.

CONSOLE

```
$ sudo systemctl restart prometheus
```

Recording Rules for Performance

Use recording rules to precompute frequently used queries.

1. Create a recording rules file.

CONSOLE

```
$ sudo nano /etc/prometheus/recording_rules.yml
```

2. Add recording rules.

YAML

```
groups:  
  - name: performance_rules  
    interval: 30s  
    rules:  
      - record: instance:node_cpu_utilization:rate5m  
        expr: 100 - (avg by (instance)  
(irate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)  
      - record: instance:node_memory_utilization:percentage  
        expr: (1 - (node_memory_MemAvailable_bytes /  
node_memory_MemTotal_bytes)) * 100
```

3. Update Prometheus configuration.

CONSOLE

```
$ sudo nano /etc/prometheus/prometheus.yml
```

4. Add the recording rules file.

YAML

```
rule_files:  
  - "alerts.yml"  
  - "recording_rules.yml"
```

Save and close the file.

5. Validate the configuration.

CONSOLE

```
$ promtool check config /etc/prometheus/prometheus.yml
```

6. Restart Prometheus.

CONSOLE

```
$ sudo systemctl restart prometheus
```

Security Hardening

1. Regularly update Prometheus and dependencies.

CONSOLE

```
$ sudo apt update  
$ sudo apt upgrade prometheus
```

2. Review and restrict access to configuration files.

```
CONSOLE
```

```
$ sudo chmod 640 /etc/prometheus/prometheus.yml
```

Troubleshooting

This section covers common issues and diagnostic commands to help resolve problems with your Prometheus instance.

Check Service Status and Logs

1. Verify service status and view logs.

```
CONSOLE
```

```
$ sudo systemctl status prometheus  
$ sudo journalctl -u prometheus -e  
$ sudo tail -f /var/log/nginx/error.log
```

Common Issues

Targets Showing as Down

1. Verify the target service is running and accessible.

```
CONSOLE
```

```
$ curl http://target-host:port/metrics
```

2. Check firewall rules allow Prometheus to connect.
3. Verify the target configuration in `prometheus.yml`.

High Memory Usage

1. Check storage usage and retention settings.

```
CONSOLE
```

```
$ du -sh /var/lib/prometheus/
```

2. Reduce retention time or size if storage is full.
3. Implement recording rules to reduce query load.

Configuration Errors

1. Validate Prometheus configuration.

```
CONSOLE
```

```
$ promtool check config /etc/prometheus/prometheus.yml
```

2. Validate alert rules.

```
CONSOLE
```

```
$ promtool check rules /etc/prometheus/alerts.yml
```

Cannot Access Web UI

1. Verify Nginx is running and properly configured.

```
CONSOLE
```

```
$ sudo systemctl status nginx  
$ sudo nginx -t
```

2. Check authentication credentials.

```
CONSOLE
```

```
$ sudo cat /etc/nginx/.htpasswd
```

3. Verify SSL certificate is valid.

```
CONSOLE
```

```
$ sudo certbot certificates
```

Use Cases

Prometheus excels in various monitoring scenarios:

- **Infrastructure Monitoring:** Track server health, resource utilization, and system metrics across your infrastructure with Node Exporter and custom exporters.
- **Container and Kubernetes Monitoring:** Monitor containerized applications, pod metrics, and cluster health with native Kubernetes service discovery.
- **Application Performance Monitoring:** Instrument applications to expose custom metrics for tracking business KPIs, request rates, and latency.
- **Microservices Monitoring:** Monitor distributed systems with service discovery, multi-dimensional data collection, and powerful querying capabilities.
- **Alerting and Incident Response:** Set up sophisticated alerting rules with Alertmanager for proactive incident detection and notification routing.
- **Capacity Planning:** Analyze historical metrics trends to predict resource needs and plan infrastructure scaling.

Conclusion

In this guide, you deployed Vultr's Prometheus Marketplace Application and configured it for production use. You secured the instance with SSL/TLS through Nginx reverse proxy, configured firewall rules and authentication, set up monitoring targets with Node Exporter, created alert rules with Alertmanager, and integrated with Grafana for visualization. You also implemented best practices including data retention policies, recording rules, and security hardening. With Prometheus's powerful monitoring capabilities and Vultr's

infrastructure, you can build comprehensive observability solutions for modern applications and infrastructure.



VULTR

