GeoServer Deployment – Architecture & Support Document

Netcup Ubuntu 24.04 · GeoServer 2.x · OpenLayers · Plesk Integration (planned)

1. System Overview

This document describes the current architecture and configuration of the GeoServer installation running on a Netcup Ubuntu 24.04 root server. It is intended as a support and handover document so administrators and developers can maintain and extend the system.

2. Server Environment

Provider: Netcup (root server / virtual server)
Operating System: Ubuntu 24.04.3 LTS (64-bit)
Access: SSH with user 'shinedown99'
Primary runtime: OpenJDK 17 (Java 17)

Key Commands Executed:

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y openjdk-17-jdk unzip wget git python3 python3-venv python3-pip
```

3. GeoServer Installation

GeoServer has been installed from the official binary distribution (ZIP) and unpacked into `/usr/share/geoserver`. The installation includes an embedded Jetty application server. GeoServer listens on port 8080.

Installation path: `/usr/share/geoserver`
Data directory: `/usr/share/geoserver/data_dir`
Startup script: `/usr/share/geoserver/bin/startup.sh`
Shutdown script: `/usr/share/geoserver/bin/shutdown.sh`

Test command (local):

```
curl -I http://localhost:8080/geoserver/
```

Expected response: HTTP 302 redirect to `/geoserver/index.html`.

4. Service Management (systemd)

A dedicated system user can be used to run GeoServer and a systemd service file can be created for automatic start on boot. The recommended user is 'geoserver' without a login shell and home directory.

Create service user:

```
sudo adduser --system --no-create-home --group geoserver
sudo chown -R geoserver:geoserver /usr/share/geoserver
```

Example systemd unit:

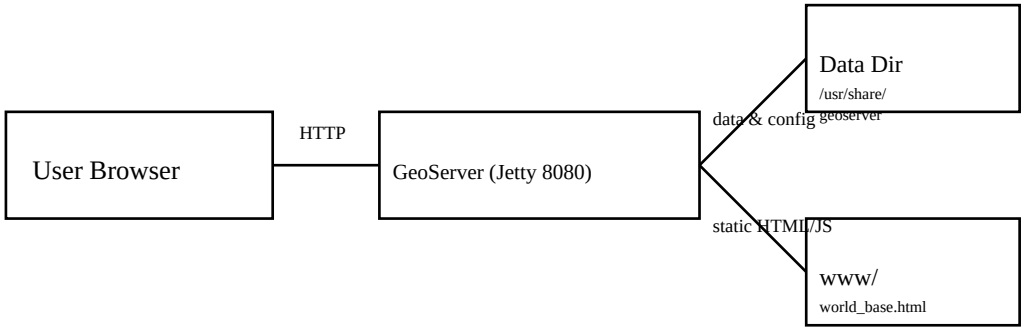
```
[Unit]
Description=GeoServer Service
After=network.target

[Service]
Type=simple
User=geoserver
Environment="GEOSERVER_HOME=/usr/share/geoserver"
Environment="JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64"
ExecStart=/usr/share/geoserver/bin/startup.sh
ExecStop=/usr/share/geoserver/bin/shutdown.sh
Restart=on-failure

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

Enable and control service:

```
sudo systemctl daemon-reload
sudo systemctl enable geoserver
sudo systemctl start geoserver
sudo systemctl status geoserver
```

5. Current Architecture (Direct IP Access)

At present, GeoServer is accessed directly via the server IP address and port 8080. The browser communicates with Jetty (embedded in GeoServer), which reads data and configuration from the GeoServer data directory. A custom public map viewer is also served from the data directory under the 'www' folder.



Public URLs:
`http://SERVER_IP:8080/geoserver/` – GeoServer web interface
`http://SERVER_IP:8080/geoserver/www/world_base.html` – Custom world map viewer

6. GeoServer Configuration & Security

The GeoServer web interface is available at `/geoserver/web/`. The default admin credentials ('admin' / 'geoserver') must be changed immediately after installation. This has been done as part of the initial configuration.

Important: The GeoServer UI should not be exposed publicly without HTTPS and possibly IP restrictions or a reverse proxy with additional security controls.

Content Security Policy (CSP):
GeoServer sends a strict Content-Security-Policy header by default, similar to the following:

```
default-src 'none';
base-uri 'self';
child-src 'self';
connect-src 'self';
font-src 'self';
img-src 'self' data:;
script-src 'self';
style-src 'self' 'unsafe-inline';
form-action 'self';
frame-ancestors 'self';
```

Because of this CSP, the browser is not allowed to load JavaScript, CSS, or images from external domains (CDNs, satellite tile providers, etc.). To work around this, OpenLayers and other assets are served locally from the GeoServer 'www' directory, and map images are served via GeoServer WMS (also same origin).

7. Data Sources for the World Map

The current implementation of the public world map viewer uses a layer from the sample data shipped with GeoServer. By default, GeoServer includes demonstration workspaces such as 'topp', 'sf', 'ne', etc.

Example data source:
Workspace: `ne` (Natural Earth)
Layer name: `ne:countries` (or similar Natural Earth layer)

Natural Earth is a public domain map dataset providing vector and raster data at multiple scales. GeoServer's 'ne' workspace typically contains country boundaries and other global layers derived from Natural Earth. These files are stored under the GeoServer data directory:
`/usr/share/geoserver/data_dir/data/ne/`

The data is accessed via GeoServer's WMS interface, which dynamically renders map tiles on request.

WMS Endpoint:
`http://SERVER_IP:8080/geoserver/wms`

Typical WMS request parameters used by the viewer:

```
SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap
LAYERS=ne:countries
FORMAT=image/png
SRS=EPSG:3857
BBOX, WIDTH, HEIGHT
```

8. Custom Public Map Viewer (world_base.html)

A custom HTML page is used to provide a public, no-login world map view. This page lives under the GeoServer data directory in the 'www' folder, which GeoServer serves as static content.

Location: `/usr/share/geoserver/data_dir/www/world_base.html`

Local OpenLayers files:
`/usr/share/geoserver/data_dir/www/ol/ol.js`
`/usr/share/geoserver/data_dir/www/ol/ol.css`

Key idea:
The viewer uses OpenLayers loaded from the local 'ol' directory (same origin), and it requests map imagery from GeoServer via WMS, which also shares the same origin. This respects the strict CSP policy and avoids any blocked external resources.

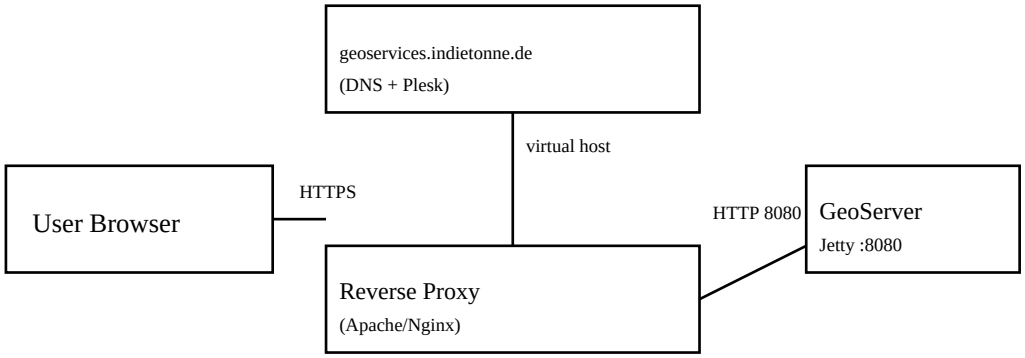
Simplified code structure:

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="ol/ol.css">
</head>
<body>
  <div id="map"></div>
  <script src="ol/ol.js"></script>
  <script>
    const LAYER_NAME = 'ne:countries';  // example layer

    const wmsLayer = new ol.layer.Image({
      source: new ol.source.ImageWMS({
        url: '/geoserver/wms',
        params: { LAYERS: LAYER_NAME, TILED: true },
        serverType: 'geoserver'
      })
    });

    const map = new ol.Map({
      target: 'map',
      layers: [wmsLayer],
      view: new ol.View({
        center: ol.proj.fromLonLat([0, 20]),
        zoom: 2
      })
    });
  </script>
</body>
```

```
</html>
```

9. Target Architecture – Subdomain & HTTPS

The next planned step is to publish GeoServer behind a subdomain managed by Plesk, for example `geoservices.indietonne.de`, and to secure it with HTTPS using Let's Encrypt. Plesk (Apache or Nginx) will act as a reverse proxy, forwarding external HTTPS traffic to the internal GeoServer instance on port 8080.



High-level steps (not yet fully implemented):
1. Create subdomain `geoservices.indietonne.de` in Plesk.
2. Issue Let's Encrypt certificate for the subdomain via Plesk.
3. Configure Apache/Nginx in Plesk to act as reverse proxy:

```
ProxyPreserveHost On
ProxyPass / http://127.0.0.1:8080/geoserver/
ProxyPassReverse / http://127.0.0.1:8080/geoserver/
```

4. Optionally redirect HTTP (port 80) to HTTPS (port 443).

10. Status Summary

Working:
- GeoServer installed and running on Ubuntu 24.04.
- Access via IP: `http://SERVER_IP:8080/geoserver/`.
- Admin login configured and default password changed.
- Sample layers (e.g., `topp:states`) render correctly.
- Custom public world map viewer (`world_base.html`) working via WMS.
- OpenLayers and assets served locally to comply with CSP.

Not yet complete / planned:
- Subdomain `geoservices.indietonne.de` fully wired to GeoServer via reverse proxy.
- HTTPS termination and automatic certificate renewal through Plesk.
- Integration of an external or higher-quality satellite imagery source as a GeoServer store.
- Installation and integration of the dNBR application (GitHub repository to be defined).
- Monitoring, backup strategy, and performance tuning.

This document provides sufficient technical detail for system administrators and developers to maintain the current setup and continue with development (subdomain, HTTPS, satellite imagery, and application integration).