Warpgate: Modern Bastion Host and Access Management with Authentik SSO

Warpgate: Modern Bastion Host and Access Management with Authentik SSO

Managing administrative SSH access, database connections, and management consoles across multi-server environments is a core challenge in perimeter defense. Without centralized access control, static key sprawl and untracked credentials undermine security policies.

To establish structured, scalable, and fully open-source access governance, **Warpgate** provides a compelling architectural framework.

Warpgate is a modern bastion host implemented in Rust that serves as a transparent proxy for SSH, HTTP, MySQL, PostgreSQL, and Kubernetes clusters. Its notable capabilities include native Single Sign-On (SSO) support via OpenID Connect (OIDC). This guide explores containerized deployment with Docker, reverse proxy configuration through Nginx, and role-based access control (RBAC) integration with Authentik.

Architectural Comparison and Operational Trade-offs

Before diving into configuration details, it is helpful to contrast Warpgate with traditional access mechanisms:

- **SSH Jump Host (ProxyJump)**: The native OpenSSH approach. Lightweight and zero-overhead, but restricted to SSH, lacking native federated SSO authentication and built-in audit recording.
- **Teleport**: A comprehensive platform in Go, but frequently complex for lean teams. The community edition restricts certain enterprise workflows and introduces proprietary client tooling (`tsh`).

- **CyberArk (PAM)**: The legacy enterprise proprietary standard. Heavy footprint, substantial licensing overhead, and high administrative friction for engineering teams.
- **Warpgate**: It stands in the middle. It is 100% open-source, does not require dedicated clients (you use your usual software), integrates OIDC SSO, and is very lightweight to run in a simple container.

Let's skip the small talk and get started.

Step 1: Installation (good old Docker)

Let's create the project directory on the server:

```
# mkdir -p /opt/warpgate && cd /opt/warpgate
```

Prepare our `docker-compose.yml`:

```
services:
  warpgate:
    image: ghcr.io/warp-tech/warpgate:latest
    container_name: warpgate
    restart: unless-stopped
    ports:
      - "2222:2222"      # SSH inbound
      - "8888:8888"      # Admin Web UI / HTTPS proxy
      - "33306:33306"    # MySQL proxy
      - "55432:55432"    # PostgreSQL proxy
    volumes:
      - ./data:/data     # Configuration, sqlite database, and session
recordings
    stdin_open: true
    tty: true
```

Before starting the container, we must run the initial configuration wizard to generate the bastion's SSH keys and the internal SQLite database:

```
# docker compose run --rm warpgate setup
```

Follow the on-screen steps, setting a temporary password for the local administrator. This command will create the `./data` folder containing the `warpgate.yaml` file.

Step 2: Placing Warpgate behind Nginx (and fighting TLS)

If you want to expose the Warpgate web interface on `warpgate.example.org`, you need to configure a reverse proxy. But here is the first quirk: on startup, Warpgate generates self-signed certificates and enables TLS on web port `8888`.

If you try to configure Nginx with a classic `proxy_pass http://127.0.0.1:8888;`, you will get a nice `upstream sent no valid HTTP/1.0 header` error. Nginx is talking in plain HTTP to a port that expects encryption.

The cure is to force Nginx to speak HTTPS to the backend, disabling the internal self-signed certificate verification:

```
server {
    listen 80;
    server_name warpgate.example.org;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl http2;
    server_name warpgate.example.org;

    ssl_certificate /etc/letsencrypt/live/warpgate.example.org/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/warpgate.example.org/privkey.pem;

    client_max_body_size 4G;

    location / {
        # HTTPS is mandatory, on port 8888
        proxy_pass https://127.0.0.1:8888;
        proxy_ssl_verify off; # Ignore the internal self-signed certificate

        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;

        # WebSocket support for interactive consoles
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        proxy_connect_timeout 600s;
        proxy_read_timeout 600s;
        proxy_send_timeout 600s;
    }
}
```

Run `nginx -t` and reload the service.

Step 3: Configuration on Authentik (OIDC)

Let's move to our Authentik instance (`auth.example.org`) to configure the application.

1. **Create the OIDC Provider**:
    - Go to **Applications** > **Providers** > **Create**.
    - Select **OAuth2/OpenID Provider**.
    - Name: `warpgate`.
    - Client Type: `Confidential`.
    - Redirect URI: `https://warpgate.example.org/@warpgate/api/sso/return` (do not specify port `8888` if Nginx responds on port 443).
    - Take note of the **Client ID** and **Client Secret**.
2. **Create the Application**:
    - Go to **Applications** > **Applications** > **Create**.
    - Name: `Warpgate`, Slug: `warpgate`.
    - Select the Provider created in the previous step.

The OIDC Tricks (Mandatory Property Mappings)

If you try to log in now, you'll run into two of Warpgate's security blocks. Let's fix them right away by creating custom **Property Mappings** of type **Scope** in Authentik (**Customization** > **Property Mappings**):

1. Forcing verified email (`warpgate-email-verified`)

In recent versions, Authentik sends the `email_verified` claim as `false` by default. Warpgate will reject the login. Create a mapping with:

- **Scope name**: `email`
- **Expression**:

```
return {
    "email": request.user.email,
    "email_verified": True,
}
```

2. Mapping administrative roles (`warpgate-admin-roles-mapping`)

Warpgate separates target access from administrative privileges. It looks for the `warpgate_admin_roles` claim in the token. Let's create a mapping to tell Authentik that users in the `WARP-Admins` group are Warpgate administrators:

- **Scope name**: `openid`
- **Expression**:

```
admin_roles = []
for group in user.groups.all():
    if group.name == "WARP-Admins":
        admin_roles.append("warpgate:admin")
return {
    "warpgate_admin_roles": admin_roles
}
```

3. Mapping normal access roles (`warpgate-roles-mapping`)

To specify which machines users can connect to, we use the `warpgate_roles` claim:

- **Scope name**: `openid`
- **Expression**:

```
roles = []
for group in user.groups.all():
    if group.name != "WARP-Admins":
        roles.append(group.name) # Pass all other groups (e.g., WARP-
Netadmins)
```

```
return {
    "warpgate_roles": roles
}
```

Make sure to go to the Provider settings in Authentik and select these three mappings in the **Property Mappings** field.

Step 4: Configuring `warpgate.yaml`

Now we modify the `data/warpgate.yaml` file on the server to enable SSO and tell Warpgate to trust Nginx:

```
external_host: warpgate.example.org

sso_providers:
  - name: authentik
    label: "Accedi con Authentik"
    provider:
      type: custom
      client_id: <YOUR_AUTHENTIK_CLIENT_ID>
      client_secret: <YOUR_AUTHENTIK_CLIENT_SECRET>
      issuer_url: https://auth.example.org/application/o/warpgate/
      scopes:
        - openid
        - email
        - profile
    auto_create_users: true

http:
  listen: '[::]:8888'
  external_port: 443
  external_host: warpgate.example.org
  certificate: /data/tls.certificate.pem
  key: /data/tls.key.pem
  trust_x_forwarded_headers: true # Crucial for reading Nginx headers
  session_max_age: 30m
  cookie_max_age: 1day
  sni_certificates: []
```

Save the file and restart the container:

```
# docker compose restart warpgate
```

Connecting to Target Machines

Now that SSO is working and you are logged in as an administrator, you can configure your targets (e.g., an SSH server on port 22) directly from the Warpgate Web UI.

When configuring a target, enter the name of the Authentik group you want to authorize in the **Allowed Roles** field (e.g., `WARP-Netadmins`).

Finally, remember that Warpgate does not forward your private SSH key to the destination. You must copy Warpgate's public key (found in the **SSH** section of the Web UI or by running `docker compose exec warpgate warpgate client-keys`) and append it to the `~/.ssh/authorized_keys` file of the user on the remote machine.

Zero personal keys scattered around, zero passwords written in plain text. Everything managed centrally.

[May the Talpone be with you]

[EOF]