

Command line Service creation

✦ Here is a comprehensive step-by-step guide explaining how to programmatically configure a proxy-protected application from scratch in Authentik using the command-line Django shell.

CLI Guide: Configuring an Authentik Application & Proxy Provider from Scratch

When web administration is unavailable or API tokens are restricted, you can fully configure, update, and manage your Authentik instances using its built-in Python/Django administration console.

1. Accessing the Authentik CLI Shell

Authentik runs as a Python/Django application inside Docker. You can launch an interactive Django administrative shell by executing the following command on the host machine running your authentik-server container:

```
1 docker exec -it authentik-server /lifecycle/ak shell
```

Note: For non-interactive single-command scripting (like the ones used in this session), append `-c` followed by your Python code wrapped in quotes:

```
1 docker exec -i authentik-server /lifecycle/ak shell -c "print('Connected to Authentik DB')"
```

2. Full Configuration Script: From Zero to Protected App

Below is the complete, working Python example matching the production-grade www configuration. This script performs the following actions:

1. Looks up the default authorization flow.
2. Creates the Proxy Provider with appropriate grant types, redirect URIs, and cookie domains.
3. Attaches standard HTTP headers property mappings.
4. Creates the Application and links it to the provider.
5. Assigns the provider to the Embedded Outpost so it instantly begins protecting the domain.

Run this script inside the Authentik shell:

```

1 from authentik.core.models import Application, PropertyMapping
2 from authentik.flows.models import Flow
3 from authentik.providers.proxy.models import ProxyProvider
4 from authentik.outposts.models import Outpost
5
6 # 1. RETRIEVE THE AUTHORIZATION FLOW
7 # By default, Authentik uses the implicit consent authorization flow.
8 # You can look it up by its default slug or specific UUID:
9 flow = Flow.objects.get(slug="default-provider-authorization-implicit-consent")
10
11
12 # 2. CREATE THE PROXY PROVIDER
13 # A Proxy Provider defines how Authentik's reverse-proxy behaves for your domain.
14 prov, created_prov = ProxyProvider.objects.get_or_create(
15     name="www",
16     defaults={
17         "mode": "forward_single",          # Options: 'forward_single' (Nginx auth_request) or
'proxy'
18         "external_host": "https://www.drheck.dev", # The public-facing domain
19         "internal_host": "",                # Left blank for Nginx auth_request configurations
20         "authorization_flow": flow,        # Link to the authorization flow
21         "cookie_domain": "drheck.dev",    # Base domain for session cookies (shared
among subdomains)
22         "grant_types": [                  # Necessary grant types for authorization redirects
23             "authorization_code",
24             "client_credentials",
25             "password"
26         ]
27     }
28 )
29
30 if created_prov:
31     # Set the Redirect URIs that Authentik's OAuth core will trust
32     prov._redirect_uris = [
33         {
34             "matching_mode": "strict",
35             "redirect_uri_type": "authorization",
36             "url": "https://www.drheck.dev/outpost.goauthentik.io/callback?X-authentik-auth-
callback=true"
37         },
38         {
39             "matching_mode": "strict",
40             "redirect_uri_type": "authorization",
41             "url": "https://www.drheck.dev?X-authentik-auth-callback=true"
42         }
43     ]

```

```

44
45 # Associate default HTTP headers property mappings (Usernames, emails, etc.)
46 # These map Authentik attributes into HTTP request headers forwarded to your app.
47 mapping_uuids = [
48     "d735215e-ee27-43a9-8218-5c8518f7c41a", # authentik_providers_proxy_header_uid
49     "c9546a27-7d86-4414-88f8-b735a413612f", #
authentik_providers_proxy_header_username
50     "39477eed-8af5-40e5-ab32-773904aaadd6", # authentik_providers_proxy_header_email
51     "1769fa05-2d96-407e-a2f6-4831ab37d653", # authentik_providers_proxy_header_name
52     "8a217b98-9d44-46da-9bf5-9a3b38b5ac41" #
authentik_providers_proxy_header_groups
53 ]
54 mappings = PropertyMapping.objects.filter(pk__in=mapping_uuids)
55 prov.property_mappings.set(mappings)
56
57 prov.save()
58 print("ProxyProvider created and configured successfully.")
59 else:
60     print("ProxyProvider already exists.")
61
62
63 # 3. CREATE THE APPLICATION
64 # The Application links the frontend card/dashboard item to your provider configuration.
65 app, created_app = Application.objects.get_or_create(
66     name="www",
67     defaults={
68         "slug": "www", # URL-safe slug for authorization endpoints
69         "provider": prov, # Link the provider we created above
70         "meta_launch_url": "https://www.drheck.dev/" # The URL opened when clicked in the
dashboard
71     }
72 )
73
74 if created_app:
75     print("Application created and linked successfully.")
76 else:
77     print("Application already exists.")
78
79
80 # 4. ASSIGN THE PROVIDER TO THE EMBEDDED OUTPOST
81 # In Authentik, Outposts must be explicitly told which providers they serve.
82 # We fetch the embedded proxy outpost and append our provider to its targets.
83 try:
84     outpost = Outpost.objects.get(name="authentik Embedded Outpost")
85     if prov not in outpost.providers.all():
86         outpost.providers.add(prov)

```

```
87     outpost.save()
88     print("Successfully assigned provider to the Embedded Outpost.")
89 else:
90     print("Provider is already assigned to the Embedded Outpost.")
91 except Outpost.DoesNotExist:
92     print("ERROR: Could not locate the 'authentik Embedded Outpost'.")
```

3. Useful Inspection Cheat Sheet

Here are useful queries you can run in /lifecycle/ak shell to check or debug your configurations:

* List all defined Applications, slugs, and associated Launch URLs:

```
1 from authentik.core.models import Application
2 for app in Application.objects.all():
3     print(f"Name: {app.name} | Slug: {app.slug} | URL: {app.meta_launch_url}")
```

* Verify active Outposts and check which applications/providers are bound to them:

```
1 from authentik.outposts.models import Outpost
2 for op in Outpost.objects.all():
3     providers = [p.name for p in op.providers.all()]
4     print(f"Outpost: {op.name} ({op.type}) | Serving: {providers}")
```

* Inspect a specific Provider's raw DB columns and values:

```
1 from authentik.providers.proxy.models import ProxyProvider
2 import pprint
3 p = ProxyProvider.objects.get(name="www")
4 pprint.pprint(p.__dict__)
```

Revision #1

Created 2026-07-12 22:24:07 UTC by admin

Updated 2026-08-16 20:03:03 UTC by admin