Toggly Python

Python, Django, Flask, FastAPI, async, contexts & snapshots

Install

```
pip install toggly             # core
pip install toggly-django      # Django
pip install toggly-flask       # Flask
pip install toggly-fastapi     # FastAPI
pip install toggly-cache[redis]
pip install toggly-cache[memcached]
```

Initialize

```
from toggly import TogglyClient, TogglyConfig

cfg = TogglyConfig(
    app_key="YOUR-APP-KEY",
    environment="Production",
    refresh_interval=180.0,
    connect_timeout=10.0,
    enable_usage_tracking=True,
)
client = TogglyClient(cfg)
client.init()  # client.close() on shutdown
```

Evaluate

```
if client.is_enabled("new-feature"):
    ...
on = client.is_enabled("new-feature", default=False)

# multi-flag gate
from toggly import FeatureRequirement
if client.evaluate_gate(
        ["beta", "pricing"],
        FeatureRequirement.ANY, negate=False):
    ...
```

Evaluation context

```
from toggly import EvaluationContext, RequestContext

ctx = EvaluationContext(
    identity="user-123",
    groups=["beta", "premium"],
    claims={"plan": "enterprise"},
    request=RequestContext(country="US"),
)

client.is_enabled("premium-only", ctx)
```

Variants (catalog-local)

```
cfg = TogglyConfig(
    app_key="YOUR-APP-KEY", identity="user-123")
client = TogglyClient(cfg)
client.init()
# Or: client.set_identity("user-123")

v = client.get_variant("checkout-flow")
print(v.name if v else "default")
# Override: get_variant(key, user_id=…, groups=…)
```

Async client

```
from toggly import AsyncTogglyClient

async with AsyncTogglyClient(cfg) as client:
    await client.init()
    if await client.is_enabled("new-api", ctx):
        ...
```

Django — settings.py

```
INSTALLED_APPS = [..., "toggly_django"]

MIDDLEWARE = [
    ...,
    "toggly_django.middleware.TogglyMiddleware",
]

import os
TOGGLY = {
    "APP_KEY": os.environ.get("TOGGLY_APP_KEY"),
    "ENVIRONMENT": "Production",
    "REFRESH_INTERVAL": 180.0,
}
```

Django — views & templates

```
from django.http import JsonResponse
from toggly_django.decorators import (
    feature_flag_required)

@feature_flag_required("new-dashboard")
def dashboard(request):
    return JsonResponse({"version": "v2"})

def home(request):
    return JsonResponse({
        "home": request.toggly.is_enabled("home")})
# template: {% load toggly_tags %}
# toggly-django 0.4.0+: matching enabled/disabled blo
# {% iffeature "nav" %}New{% endiffeature %}
# {% iffeature "nav" negate=True %}Classic{% endiffea
```

Flask — extension

```
from flask import Flask, render_template
from toggly_flask import (
    Toggly, feature_flag_required)

app = Flask(__name__)
app.config.update(
    TOGGLY_APP_KEY     = "YOUR-APP-KEY",
    TOGGLY_ENVIRONMENT = "Production",
    TOGGLY_REFRESH_INTERVAL = 180.0,
)
toggly = Toggly(app)

@app.route("/dashboard")
@feature_flag_required("new-dashboard")
def dashboard(): return render_template("d.html")
```

JINJA2

```
{% if toggly.check("new-nav") %}
  <nav>new</nav>
{% endif %}
```

Flask — factory pattern

```
import os
toggly = Toggly()

def create_app():
    app = Flask(__name__)
    app.config["TOGGLY_APP_KEY"] = (
        os.environ.get("TOGGLY_APP_KEY"))
    toggly.init_app(app)
    return app
```

FastAPI

```
from contextlib import asynccontextmanager
from fastapi import FastAPI, Depends
from toggly_fastapi import (
    configure_toggly, TogglyMiddleware,
    get_toggly, require_feature)

@asynccontextmanager
async def lifespan(app):
    client = configure_toggly(
        app_key="YOUR-APP-KEY")
    try:
        yield
    finally:
        client.close()

app = FastAPI(lifespan=lifespan)
app.add_middleware(TogglyMiddleware)

@app.get("/dashboard",
   dependencies=[
     Depends(require_feature("new-dashboard"))])
async def dashboard():
    return {"version": "v2"}
```

FastAPI — runtime check

```
@app.get("/checkout")
async def checkout(toggly = Depends(get_toggly)):
    if toggly.is_enabled("new-checkout"):
        return {"v": 2}
    return {"v": 1}
```

Persistent snapshot

```
from toggly.providers import FileSnapshotProvider

cfg = TogglyConfig(
    app_key="YOUR-APP-KEY",
    environment="Production",
    snapshot_provider=FileSnapshotProvider(
        directory="./cache/checkout-production"))
client = TogglyClient(cfg)
client.init()  # close at worker shutdown
```

Each worker owns its refresh loop. Redis/Memcached utilities expose separate save/load APIs and cannot attach directly as core snapshot providers.

Testing

```
client = TogglyClient(TogglyConfig(
    feature_defaults={
        "new-feature": True,
        "beta":        False,
    },
    disable_background_refresh=True,
    enable_live_updates=False,
    enable_usage_tracking=False))
client.init()

assert client.is_enabled("new-feature")
client.close()
```

Entity per check

```
from toggly import TogglyEntityContext

order = TogglyEntityContext(
    kind="Order", key="order-123",
    attributes={"Vip": True, "Total": 120.0})
order_ctx = ctx.with_entity(order)
client.is_enabled("ExpressCheckout",
                  context=order_ctx)
```

Set the entity before the first native request check:

```
# Django
request.toggly_entity = order
# Flask
from flask import g
g.toggly_entity = order
# FastAPI
request.state.toggly_entity = order
```

For another Order, use a new per-call context; helpers cache their first context.

Claims & request filters

```
from toggly import HttpRequestMapper

# base_ctx comes from the adapter's
# get_context_from_request(...).
ctx = HttpRequestMapper.merge_into(
    request.headers, base_ctx)
# verified_claims comes from authentication.
ctx = ctx.with_claims(verified_claims)
client.is_enabled("browser-offer", context=ctx)
```

Pass the enriched context to the core client. Native helpers do not automatically map claims or structured request fields. Trust country headers only from your proxy.

Pitfalls

- `client.init()` spawns a background refresh thread — call `client.close()` in shutdown / fixtures.
- In tests set `disable_background_refresh=True` to keep the loop deterministic.
- Django middleware must come **after** `AuthenticationMiddleware` so identity is populated from `request.user`.
- Flask: register a supplied client once with `set_default_client(client)` before `Toggly(app, client=client)` so native decorators use it.
- `toggly-cache[redis]` and `[memcached]` are optional extras; install only what you use.
- Native helpers use the synchronous client. Claims and request segment fields require an explicit core evaluation context.

Resources

- Docs: `docs.toggly.io/sdks/python`
- PyPI: `toggly`, `toggly-django`, `toggly-flask`, `toggly-fastapi`
- GitHub: `github.com/ops-ai/Toggly.FeatureManagement`
- Samples catalog: `github.com/ops-ai/Toggly.Samples`