QGIS MCP

Références

- Plugin officiel : plugins.qgis.org/plugins/qgis_mcp_plugin — version 0.5.0, compatible QGIS 3.28–4.x
- Dépôt GitHub : nkarasiak/qgis-mcp — 102 outils MCP (couches, traitements, symbologie, export, mise en page/atlas, SQL cross-couches)

Architecture

```
Claude Code ↔ Serveur MCP Python (uvx) ↔ socket TCP local ↔ Plugin QGIS (dock "QGIS MCP")
```

Le plugin QGIS crée un serveur TCP local. Le serveur MCP Python (lancé par Claude Code via `uvx`) s'y connecte. QGIS doit être ouvert et le serveur démarré avant de lancer Claude Code.

Prérequis

Installer `uv` (gestionnaire de paquets Python) si absent :

```
winget install astral-sh.uv
```

Installation

1. Plugin dans QGIS

`Extensions` > `Installer/Gérer les extensions` > chercher **QGIS MCP** > Installer.

Redémarrer QGIS, puis cliquer **Start Server** dans le dock QGIS MCP.

2. Enregistrer le MCP dans Claude Code

```
claude mcp add -s user qgis -- uvx --from https://github.com/nkarasiak/qgis-mcp/archive/refs/heads/main.zip qgis-mcp-server
```

`-s user` = enregistrement global (tous les projets Claude Code).

3. Vérifier

Dans une session Claude Code avec QGIS ouvert et le serveur démarré, demander `diagnose` — confirme la synchronisation plugin/serveur MCP.

Ordre de démarrage (à chaque session)

1. Ouvrir QGIS
2. Cliquer **Start Server** dans le dock QGIS MCP
3. Lancer Claude Code

Authentification (optionnel, machines partagées)

Définir `QGIS_MCP_TOKEN=votre-secret` dans l'environnement QGIS, redémarrer le serveur, puis ajouter la même variable à la config MCP :

```
"env": { "QGIS_MCP_TOKEN": "votre-secret" }
```

Capacités principales

Catégorie	Exemples
Couches	Charger Shapefile, GeoJSON, WMS, PostGIS ; lister, supprimer
Traitements	Buffer, intersection, statistiques zonales (1000+ algorithmes)
Symbologie	Modifier couleurs, classification, étiquettes
Export	PNG, PDF, carte mise en page
SQL cross-couches	Requêtes spatiales directement depuis Claude
Mise en page/Atlas	Créer et exporter des atlas cartographiques

---

Lien avec PostGIS Alteris

La base `alteris_geo` (79.137.14.202:5432, user `alteris_admin`) est directement utilisable depuis QGIS MCP : Claude peut interroger PostGIS en SQL spatial, charger le résultat comme couche QGIS, styliser et exporter en PDF — sans intervention manuelle.

Le port 5432 n'est pas exposé publiquement (pare-feu VPS). Connexion via tunnel SSH obligatoire (voir section Tunnel SSH).

Tunnel SSH vers PostGIS Alteris

Option 1 — Tunnel manuel (terminal externe)

Lancer avant QGIS/Claude Code :

```
ssh -L 5433:localhost:5432 debian@79.137.14.202 -N
```

Puis se connecter sur `localhost:5433` au lieu de `79.137.14.202:5432`.

Option 2 — Tunnel automatique via paramiko (PyQGIS) ✓ validé 2026-06-20

Ouvre le tunnel directement depuis la console Python QGIS, sans terminal externe.

Installation de paramiko (une seule fois)

`sys.executable` dans QGIS pointe sur `qgis-bin.exe` — `subprocess` est inutilisable. Utiliser pip en interne :

```
from pip._internal.cli.main import main as pip_main
pip_main(['install', 'paramiko'])
```

Paramiko s'installe dans `C:\Users\eliob\AppData\Roaming\Python\Python312\site-packages` (dossier utilisateur). QGIS ne l'inclut pas automatiquement dans `sys.path` — ajouter à chaque session :

```
import sys
sys.path.insert(0, r'C:\Users\eliob\AppData\Roaming\Python\Python312\site-packages')
import paramiko
print(paramiko.__version__)  # 5.0.0
```

Script du tunnel (à lancer après l'import paramiko)

```
import socket
import threading
import select

class SSHTunnel(threading.Thread):
    def __init__(self, ssh_host, ssh_user, ssh_password,
                 remote_port, local_port=5433, remote_host='127.0.0.1'):
        super().__init__(daemon=True)
        self.ssh_host = ssh_host
        self.ssh_user = ssh_user
        self.ssh_password = ssh_password
        self.remote_host = remote_host
        self.remote_port = remote_port
        self.local_port = local_port
        self._stop = threading.Event()
        self.transport = None
```

```
        self.server_sock = None

    def run(self):
        client = paramiko.SSHClient()
        client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        client.connect(self.ssh_host, username=self.ssh_user, password=self.ssh_password)
        self.transport = client.get_transport()
        self.server_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.server_sock.bind(('127.0.0.1', self.local_port))
        self.server_sock.listen(5)
        self.server_sock.settimeout(1.0)
        print(f"[Tunnel] localhost:{self.local_port} → {self.ssh_host} →
{self.remote_host}:{self.remote_port}")
        while not self._stop.is_set():
            try:
                conn, _ = self.server_sock.accept()
                threading.Thread(target=self._forward, args=(conn,), daemon=True).start()
            except socket.timeout:
                continue
        self.server_sock.close()
        client.close()
        print("[Tunnel] Arrêté.")

    def _forward(self, local_conn):
        try:
            chan = self.transport.open_channel(
                'direct-tcpip',
                (self.remote_host, self.remote_port),
                local_conn.getpeername()
            )
        except Exception as e:
            print(f"[Tunnel] Erreur canal : {e}")
            local_conn.close()
            return
        while True:
            r, _, _ = select.select([local_conn, chan], [], [], 5)
            if local_conn in r:
                data = local_conn.recv(4096)
                if not data:
```

```
                    break
                chan.send(data)
            if chan in r:
                data = chan.recv(4096)
                if not data:
                    break
                local_conn.send(data)
        chan.close()
        local_conn.close()

    def stop(self):
        self._stop.set()



tunnel = SSHTunnel('79.137.14.202', 'debian', 'RAW+NEXTE!', remote_port=5432, local_port=5433)
tunnel.start()

import time; time.sleep(1)
s = socket.socket(); s.settimeout(3)
try:
    s.connect(('127.0.0.1', 5433)); print("Tunnel OK")
except Exception as e:
    print(f"Tunnel KO : {e}")
finally:
    s.close()
```

Pour arrêter : `tunnel.stop()`

Les avertissements `DEPRECATION: Unexpected import of 'paramiko.*'` sont inoffensifs (pip qui se plaint d'imports tardifs dans la même session).

Connexion PostGIS depuis PyQGIS

Charger une couche PostGIS

```
from qgis.core import QgsProject, QgsVectorLayer
```

```
uri = (
    "host=localhost port=5433 dbname=alteris_geo "
    "user=alteris_admin password=Alteris2026 sslmode=disable "
    'table="altfoncier_28051"."28051_plu_zonage" (geom) sql='
)
layer = QgsVectorLayer(uri, "PLU Zonage 28051", "postgres")
QgsProject.instance().addMapLayer(layer)
```

Point clé — champ JSONB `props`

Les tables altfoncier ont leurs attributs métier dans un champ `props` de type JSONB. Dans PyQGIS, ce champ est un **dict Python** (pas une chaîne). Pour y accéder en expression QGIS :

```
map_get("props", 'typezone')   ✓ correct
"props" LIKE '%typezone%'      ✗ ne fonctionne pas (props n'est pas une chaîne)
```

Stylisation CNIG PLU (zonage)

Renderer catégorisé sur `map_get("props", 'typezone')` :

typezone	Couleur remplissage	Couleur bordure	Label
U	`#FFFF73`	`#A3A300`	Zones urbaines
AU	`#FFAA00`	`#CD6600`	Zones à urbaniser
A	`#D3FFBE`	`#267300`	Zones agricoles
N	`#98E600`	`#267300`	Zones naturelles et forestières

```
from qgis.core import QgsCategorizedSymbolRenderer, QgsRendererCategory, QgsFillSymbol

cnig = {
    'U':  ('#FFFF73', '#A3A300', 'Zones urbaines (U)'),
    'AU': ('#FFAA00', '#CD6600', 'Zones à urbaniser (AU)'),
    'A':  ('#D3FFBE', '#267300', 'Zones agricoles (A)'),
    'N':  ('#98E600', '#267300', 'Zones naturelles et forestières (N)'),
}
categories = []
for tz, (fill, border, label) in cnig.items():
    symbol = QgsFillSymbol.createSimple({
```

```
        'color': fill, 'outline_color': border, 'outline_width': '0.26'
    })
    categories.append(QgsRendererCategory(tz, symbol, label))

renderer = QgsCategorizedSymbolRenderer("map_get(\"props\", 'typezone')", categories)
layer.setRenderer(renderer)
layer.triggerRepaint()
```

Journal de session

2026-06-20 — Premier diagnostic validé

Première connexion réussie entre Claude Code et QGIS via MCP. Résultat du `diagnose` :

Vérification	Résultat
QGIS	4.0.3-Norrköping
Python	3.12.13
Qt	6.11.0
Plugin MCP	0.5.0 (serveur et plugin synchronisés)
Clients connectés	1
Providers de traitement	3d, gdal, grass, model, native, pdal, project, qgis, quickosm, script
Projet ouvert	Aucun (layer_count = 0)

Statut global : **healthy**. Stack complète et opérationnelle.

2026-06-20 — Connexion PostGIS Alteris + stylisation CNIG

- Connexion PostGIS directe (`79.137.14.202:5432`) → **timeout** (port filtré par pare-feu VPS)
- Tunnel SSH manuel (`ssh -L 5433:localhost:5432 debian@79.137.14.202 -N`) → **connexion OK**
- Couche `altfoncier_28051.28051_plu_zonage` chargée (20 entités, commune 28051)
- Stylisation CNIG appliquée et validée visuellement

- **Point clé découvert** : le champ `props` (JSONB PostGIS) arrive comme **dict Python** dans PyQGIS — utiliser `map_get("props", 'typezone')` en expression, pas LIKE sur chaîne

2026-06-20 — Tunnel paramiko validé

- `sys.executable` = `qgis-bin.exe` → subprocess inutilisable pour pip
- Installation via `pip._internal.cli.main` → paramiko 5.0.0 installé dans le dossier utilisateur
- `sys.path.insert(0, ...)` nécessaire à chaque session (QGIS n'inclut pas le dossier utilisateur)
- Tunnel paramiko démarré → **Tunnel OK** confirmé
- Couche PostGIS chargée via `localhost:5433` → **succès**

Prochaine étape :

- Exporter la carte en PDF via mise en page QGIS

Revision #5
Created 2026-06-20 14:30:47 UTC by Julien
Updated 2026-06-20 16:42:29 UTC by Julien