**PLUGIN DEVELOPMENT GUIDE**

**Создание плагина**

**Python Plugin (Local)**

# my\_plugins/custom\_filter.py

from app.plugins.base import LogPlugin

from app.models.log\_entry import LogEntry

from typing import Dict, Any, Optional

class MyCustomPlugin(LogPlugin):

def \_\_init\_\_(self, my\_param: str = "default"):

self.my\_param = my\_param

def process\_json(self, json\_obj: Dict[str, Any]) -> Optional[Dict[str, Any]]:

# Обработка JSON перед парсингом

# return None - фильтровать

return json\_obj

def process\_entry(self, entry: LogEntry) -> Optional[LogEntry]:

# Обработка готовой записи

# return None - фильтровать

return entry

**gRPC Plugin (External)**

// plugin.proto

syntax = "proto3";

service LogPluginService {

rpc ProcessJSON(JSONRequest) returns (JSONResponse);

rpc ProcessEntry(EntryRequest) returns (EntryResponse);

}

message JSONRequest {

string json\_data = 1;

}

message JSONResponse {

string json\_data = 1;

bool filter = 2; // true = отфильтровать

}

message EntryRequest {

string timestamp = 1;

string level = 2;

string message = 3;

string raw\_json = 4;

}

message EntryResponse {

string timestamp = 1;

string level = 2;

string message = 3;

string raw\_json = 4;

bool filter = 5;

}

# my\_grpc\_plugin/server.py

import grpc

from concurrent import futures

import plugin\_pb2

import plugin\_pb2\_grpc

class LogPluginServicer(plugin\_pb2\_grpc.LogPluginServiceServicer):

def ProcessJSON(self, request, context):

# Твоя логика

return plugin\_pb2.JSONResponse(

json\_data=request.json\_data,

filter=False

)

def ProcessEntry(self, request, context):

# Твоя логика

return plugin\_pb2.EntryResponse(

timestamp=request.timestamp,

level=request.level,

message=request.message,

raw\_json=request.raw\_json,

filter=False

)

def serve():

server = grpc.server(futures.ThreadPoolExecutor(max\_workers=10))

plugin\_pb2\_grpc.add\_LogPluginServiceServicer\_to\_server(

LogPluginServicer(), server

)

server.add\_insecure\_port('[::]:50051')

server.start()

server.wait\_for\_termination()

**gRPC Adapter (в основном проекте)**

# app/plugins/grpc\_adapter.py

import grpc

import json

from typing import Dict, Any, Optional

from app.plugins.base import LogPlugin

from app.models.log\_entry import LogEntry

import plugin\_pb2

import plugin\_pb2\_grpc

class GRPCPluginAdapter(LogPlugin):

def \_\_init\_\_(self, host: str, port: int):

self.channel = grpc.insecure\_channel(f'{host}:{port}')

self.stub = plugin\_pb2\_grpc.LogPluginServiceStub(self.channel)

def process\_json(self, json\_obj: Dict[str, Any]) -> Optional[Dict[str, Any]]:

request = plugin\_pb2.JSONRequest(json\_data=json.dumps(json\_obj))

response = self.stub.ProcessJSON(request)

if response.filter:

return None

return json.loads(response.json\_data)

def process\_entry(self, entry: LogEntry) -> Optional[LogEntry]:

request = plugin\_pb2.EntryRequest(

timestamp=entry.timestamp.isoformat(),

level=entry.level.name,

message=entry.message,

raw\_json=json.dumps(entry.raw\_json) if entry.raw\_json else ""

)

response = self.stub.ProcessEntry(request)

if response.filter:

return None

# Обновить entry из response

entry.message = response.message

return entry

**Подключение плагинов**

**plugin\_config.json**

{

"default": {

"MyCustomPlugin": {

"enabled": true,

"type": "local",

"module": "my\_plugins.custom\_filter",

"class": "MyCustomPlugin",

"params": {

"my\_param": "value"

}

},

"ExternalGRPCPlugin": {

"enabled": true,

"type": "grpc",

"host": "localhost",

"port": 50051,

"timeout": 5

}

}

}

**app/config.py (расширенный)**

def create\_plugins\_from\_config(self, profile: str = "default") -> List:

config = self.get\_config(profile)

plugins = []

for plugin\_name, plugin\_config in config.items():

if not plugin\_config.get("enabled", False):

continue

try:

plugin\_type = plugin\_config.get("type", "local")

if plugin\_type == "local":

# Динамический импорт

module\_path = plugin\_config.get("module")

class\_name = plugin\_config.get("class")

params = plugin\_config.get("params", {})

module = \_\_import\_\_(module\_path, fromlist=[class\_name])

PluginClass = getattr(module, class\_name)

plugins.append(PluginClass(\*\*params))

elif plugin\_type == "grpc":

host = plugin\_config.get("host", "localhost")

port = plugin\_config.get("port", 50051)

plugins.append(GRPCPluginAdapter(host, port))

logger.info(f"Loaded plugin: {plugin\_name}")

except Exception as e:

logger.error(f"Error loading plugin {plugin\_name}: {e}")

return plugins

**UI для управления плагинами**

**app/main.py (новые эндпоинты)**

@app.get("/api/v1/plugins", tags=["Plugins"])

async def get\_plugins():

"""Список всех доступных плагинов"""

return {

"available\_plugins": [

{

"name": "SensitiveDataPlugin",

"type": "local",

"description": "Скрывает чувствительные данные",

"params": ["redact\_value", "keep\_prefix"]

},

{

"name": "ExternalGRPCPlugin",

"type": "grpc",

"description": "Внешний gRPC плагин",

"params": ["host", "port", "timeout"]

}

],

"active\_plugins": config\_loader.get\_config("default")

}

@app.post("/api/v1/plugins/toggle", tags=["Plugins"])

async def toggle\_plugin(plugin\_name: str, enabled: bool):

"""Включить/выключить плагин"""

config = config\_loader.get\_config("default")

if plugin\_name in config:

config[plugin\_name]["enabled"] = enabled

# Сохранить обратно в файл

with open("plugin\_config.json", "w") as f:

json.dump({"default": config}, f, indent=2)

return {"status": "ok"}

@app.put("/api/v1/plugins/{plugin\_name}", tags=["Plugins"])

async def update\_plugin\_params(plugin\_name: str, params: Dict[str, Any]):

"""Обновить параметры плагина"""

config = config\_loader.get\_config("default")

if plugin\_name in config:

config[plugin\_name].update(params)

with open("plugin\_config.json", "w") as f:

json.dump({"default": config}, f, indent=2)

return {"status": "ok"}

**static/js/viewer.js (добавить)**

async function loadPlugins() {

const response = await fetch(`${API\_BASE}/api/v1/plugins`);

const data = await response.json();

const container = document.getElementById('pluginsContainer');

container.innerHTML = data.available\_plugins.map(plugin => {

const active = data.active\_plugins[plugin.name];

const enabled = active?.enabled || false;

return `

<div class="plugin-item">

<div class="plugin-header">

<h4>${plugin.name}</h4>

<label class="toggle">

<input type="checkbox"

${enabled ? 'checked' : ''}

onchange="togglePlugin('${plugin.name}', this.checked)">

<span class="slider"></span>

</label>

</div>

<p>${plugin.description}</p>

${enabled ? `

<div class="plugin-params">

${Object.entries(active.params || {}).map(([key, value]) => `

<input type="text"

value="${value}"

onchange="updatePluginParam('${plugin.name}', '${key}', this.value)">

`).join('')}

</div>

` : ''}

</div>

`;

}).join('');

}

async function togglePlugin(name, enabled) {

await fetch(`${API\_BASE}/api/v1/plugins/toggle?plugin\_name=${name}&enabled=${enabled}`, {

method: 'POST'

});

loadPlugins();

}

**terraform\_viewer.html (добавить вкладку)**

<button class="tab" onclick="switchTab('plugins')" data-tab="plugins">🔌 Plugins</button>

<div class="tab-content" id="plugins-tab">

<div class="plugins-header">

<h3>Plugin Management</h3>

<button onclick="loadPlugins()">Refresh</button>

</div>

<div id="pluginsContainer"></div>

</div>