

Go 언어 기반 네트워크 자동화와 AI Observability

코드 기반 데이터 수집과 AI 분석이 만나는 지점

유 동 주



Table of Contents

✓ 네트워크 자동화를 위한 Go 생태계

- ✓ 다양한 자동화 도구 및 라이브러리
- ✓ About Go

✓ Go vs Python

✓ Go 기반 네트워크 자동화

✓ AI-powered Observability

- ✓ MCP 소개
- ✓ MCP Server 활용

Intro

Q1. 자동화에 대한 본인의 생각과 가장 가까운 것을 선택해주세요.

- ☐ 수작업이 편해서 굳이 자동화할 필요는 없다
24표, 19.5%
- ☐ 자동화가 필요하긴 한데, 여유가 없다
47표, 38.2%
- ☐ 지금 자동화를 적극 검토 중이다
13표, 10.6%
- ☐ 이미 자동화를 일부 적용하고 있다
32표, 26%
- ☐ 거의 모든 작업을 자동화하고 있다
7표, 5.7%

Q4. 문제 발생시 자동으로 조치되는 구조가 필요하다고 느낀 적이 있으신가요?

- ☐ 자주 느낀다
34표, 34.3%
- ☐ 가끔 느낀다
32표, 32.3%
- ☐ 아직까지는 수동 대응도 괜찮다
22표, 22.2%
- ☐ 우리 환경에서는 필요 없다
11표, 11.1%

Q2. 실제로 어떤 자동화 도구를 사용하고 있나요? (복수 선택 가능)

복수 선택 가능

- ☐ 없음 (수동 CLI/스크립트만 사용)
43표, 26.5%
- ☐ Python
51표, 31.5%
- ☐ Ansible
27표, 16.7%
- ☐ 네트워크 벤더 솔루션
15표, 9.3%
- ☐ 별도의 NMS
13표, 8%

기타 (직접 구현, 외부 플랫폼 등)

13표, 8%

Q5. 적용중인 자동화 범주에 가장 가까운 것은?

- ☐ 장비 조회/상태 수집 정도만 자동화되어 있다
42표, 43.3%
- ☐ 설정 변경은 일부 수동, 일부는 자동화되어 있다
24표, 24.7%
- ☐ 트러블 발생 시 자동으로 조치되는 구조가 있다
9표, 9.3%
- ☐ 자동화된 구조는 전혀 없다
22표, 22.7%

Q3. 본인이 느끼는 가장 큰 자동화 도입 장애물은 무엇인가요?

- ☐ 시간이 없다
24표, 21.6%
- ☐ 개발 지식이 부족하다
48표, 43.2%
- ☐ 기존 시스템과 충돌이 우려된다
16표, 14.4%
- ☐ 장비 지원이 애매하다
9표, 8.1%
- ☐ 아직 자동화의 효과가 체감되지 않는다
14표, 12.6%

Q6. 자동화를 이미 일부/적극적으로 적용중이지만 겪은 문제가 있다면? (사례를 댓글로 공유해주시면 감사하겠습니다.)

- ☐ 장비가 많아 속도에 병목을 느낀 적 있음
10표, 10.8%
- ☐ 실시간 판단이 안 되어 수동 대응함
27표, 29%
- ☐ 문제를 겪었지만 대안을 못 찾음
18표, 19.4%
- ☐ 해당 경험 없음
38표, 40.9%

네트워크 자동화를 위한 Go 생태계

자동화 도구 소개

다양한 자동화 도구 및 라이브러리



About Go

What is Go?

Google에서 개발한 오픈소스 프로그래밍 언어로, 간결한 문법, 뛰어난 병렬성, 그리고 높은 성능을 갖춘 현대적인 시스템 프로그래밍 언어



- ✓ **정적 타입, 컴파일 언어**
→ 빠른 실행 속도
- ✓ **가비지 컬렉션(Garbage Collection) 내장**
→ 높은 성능과 안정성을 동시에 제공
- ✓ **동시성 프로그래밍 지원 (Goroutines, Channels)**
→ 경량, 강력한 병렬처리
- ✓ **풍부한 표준 라이브러리**
→ HTTP, JSON, SSH, 파일 처리 등 기본 제공
- ✓ **의존성 없는 단일 바이너리 생성**
→ 운영 환경 배포 간편
- ✓ **간결한 문법**
→ 유지보수 용이

About Go

Why Go for Network Automation?

네트워크 자동화에 있어 **Go** 언어는 아래와 같은 특화된 장점을 바탕으로 복잡한 인프라 환경에서도 높은 안정성과 성능을 발휘합니다.



높은 성능

- 컴파일 언어 기반으로 인터프리터 언어 대비 더 빠른 실행 속도
- 대규모 장비를 다루는 네트워크 환경에 적합



동시성 처리

- Goroutine과 Channel을 활용한 경량 동시성 처리 모델
- 수천 개 디바이스와의 통신을 효율적, 안정적으로 병렬 처리



정적 바이너리

- 외부 라이브러리 없이 단일 실행 파일로 배포 가능
- 운영 환경에 관계없이 안정적으로 실행할 수 있어 관리가 간편



타입 안정성

- 정적 타입 시스템을 통해 런타임 오류를 사전에 방지
- 네트워크 자동화 코드의 안정성과 유지보수성 우수

특성	Go	Python
실행 성능	매우 빠름 (컴파일 언어)	상대적으로 느림 (인터프리터)
메모리 사용	효율적인 관리	상대적으로 높음
병렬 처리	Goroutines (경량 스레드)	Threading, Multiprocessing
학습 곡선	중간 수준	쉬운 진입점
네트워크 자동화 도구	성장 중	풍부한 에코시스템

Go vs Python

문법 비교

Go vs Python

변수 선언



특징

- ✓ 정적 타입 언어 → 컴파일 타임에 타입 결정
- ✓ 타입 추론 가능(:=), 타입 변경 불가

코드 예시

```
var x int = 10  
y := "hello"  
// x = "world" // 오류
```



특징

- ✓ 동적 타입 언어 → 런타임에 타입 결정
- ✓ 변수 선언

코드 예시

```
x = 10  
x = "hello" # 가능  
print(x + 5) # TypeError (문자열 + 숫자)
```

Go vs Python

자료형



특징

- ✓ 모든 변수는 선언 시 자료형이 결정
- ✓ Zero value 개념 있음 (자동 초기화)

코드 예시

```
var age int = 30 // int
name := "Alice" // string
active := true // bool
scores := []int{90, 85} // list
settings := map[string]string{"lang": "go"} // map
```



특징

- ✓ 모든 것이 객체이며, mutable/immutable에 따라 동작 차이 있음
- ✓ 명시적 초기화 필요

코드 예시

```
age = 30
name = "Alice"
active = True
scores = [90, 85]
settings = {"lang": "python"}
```

Go vs Python

함수 선언 / 반환



특징

- ✓ 함수 인자와 반환값의 타입 명시 → 컴파일 타임 오류 방지, 유지보수성 향상
- ✓ 복수 반환 지원

코드 예시

```
func add(a int, b int) int {  
    return a + b  
}  
  
func info() (string, int) {  
    return "ok", 200  
}
```



특징

- ✓ 타입 명시 없이 함수 선언 가능
- ✓ 복수 반환도 튜플로 가능

코드 예시

```
def add(a, b):  
    return a + b  
  
def info():  
    return "ok", 200  
  
add("1", 2) # TypeError: can't add str and int
```

Go vs Python

구조체 vs 클래스



특징

- ✓ 구조체(struct)로 데이터 정의
- ✓ 상속 대신 조합(embedding) 사용

코드 예시

```
type Person struct {  
    Name string  
    Age  int  
}  
  
func (p Person) Greet() { // 메서드는 struct에 바인딩  
    fmt.Println("Hello", p.Name)  
}
```



특징

- ✓ Class로 데이터와 메서드 정의
- ✓ 상속 및 생성자(__init__) 등 지원

코드 예시

```
class Person:  
    def __init__(self, name):  
        self.name = name  
  
    def greet(self):  
        print("Hello", self.name)
```

Go vs Python

인터페이스 vs Duck Typing



특징

- ✓ Go의 인터페이스는 구현을 명시하지 않아도 자동으로 적용
- ✓ 강력한 추상화와 컴파일 타임 타입 안전성을 동시에 확보

코드 예시

```
type Speaker interface { // 인터페이스 정의
    Speak()
}

type Dog struct{} // 구현체 정의
func (d Dog) Speak() {
    fmt.Println("Woof!")
}

func talk(s Speaker) { // 인터페이스를 인자로 받는 함수
    s.Speak()
}

func main() {
    var d Dog
    talk(d) // OK: Dog은 Speaker를 암묵적으로 만족
}
```



특징

- ✓ Duck Typing (메서드만 맞으면 OK). 명시적 인터페이스 정의 불필요
- ✓ 정적 타입 체크가 어려워 실수 시 런타임 오류로 이어질 수 있음

코드 예시

```
class Dog:
    def speak(self):
        print("Woof!")

def talk(obj):
    obj.speak()

d = Dog()
talk(d) # 정상 동작
talk("not a speaker") # AttributeError (런타임 오류)
```

Go vs Python

예외 처리 방식



특징

- ✓ try / except 구조 없음 → 에러는 값으로 반환 (error)
- ✓ Go는 예외 흐름이 숨겨지지 않도록 설계됨 → 읽기 쉬운 코드, 명확한 의도

코드 예시

```
result, err := doSomething()
if err != nil {
    log.Println("처리 실패:", err)
    return
}
fmt.Println("성공:", result)
```



특징

- ✓ try / except 문법 제공
- ✓ 예외는 객체로 처리하며, 다양한 예외 타입 지원

코드 예시

```
try:
    result = do_something()
except FileNotFoundError as e:
    print("파일이 없음:", e)
# 예외 흐름이 명시적으로 보이지 않아 예상 밖의 예외 발생 시 디버깅이 어려움
except Exception as e:
    print("알 수 없는 오류:", e)
```

Go vs Python

포인터 유무



특징

- ✓ Go는 포인터를 명시적으로 지원하지만, C처럼 위험한 포인터 연산은 막아둠
- ✓ 함수에 인자를 포인터로 넘기면 실제 값을 직접 수정할 수 있음

코드 예시

```
func update(val *int) {  
    *val = 20  
}  
  
func main() {  
    x := 10  
    update(&x)  
    fmt.Println(x) // 20  
}
```



특징

- ✓ 포인터 없음
- ✓ mutable 객체만 함수나 대입을 통해 간접 수정이 가능

코드 예시

```
def update(val):  
    val += 1  
  
x = 10  
update(x)  
print(x) # 10 (값은 바뀌지 않음)
```

Go vs Python

동시성 (Goroutine)



특징

- ✓ go 키워드로 병렬 실행 가능한 경량 스레드
- ✓ channel로 goroutine 간 안전하게 데이터 공유

코드 예시

```
func sayHello(ch chan string) {
    ch <- "Hello from Goroutine"
}

func main() {
    ch := make(chan string)
    go sayHello(ch) // 한 줄로 병행성 실행
    msg := <-ch
    fmt.Println(msg)
}
```



특징

- ✓ Python의 동시성은 thread, asyncio, multiprocessing 등에 의존
- ✓ 안전하게 데이터를 교환하려면 외부 도구 필요, 스레드가 많아지면 복잡도 증가

코드 예시

```
import threading
from queue import Queue

def say_hello(q):
    q.put("Hello from Thread")

q = Queue() # 채널 개념이 없어 Queue로 간접 동기화
t = threading.Thread(target=say_hello, args=(q,))
t.start()
msg = q.get()
print(msg)
```

Go vs Python

Go만의 문법 특징



특징

- ✓ defer: 함수 종료 시 실행 예약
- ✓ select: 채널 병행 제어 전용 분기문

코드 예시

```
func main() {  
    defer fmt.Println("Done")  
    fmt.Println("Start")  
}  
  
select {  
    case msg := <-ch1:  
        fmt.Println(msg)  
    case <-time.After(1 * time.Second):  
        fmt.Println("timeout")  
}
```



특징

- ✓ defer 없음 → try/finally로 유사 처리

코드 예시

```
try:  
    print("Start")  
finally:  
    print("Done")
```

Go 기반 네트워크 자동화

Go 기반 네트워크 자동화

Gornir + goeapi + gorm + sqlite3



gornir

Go

Go 언어 기반의 병렬 네트워크 자동화 프레임워크로, 인벤토리 기반 작업 실행을 구조화

주요 특징

- ✓ Go의 goroutine을 활용, 다수의 장비에 대한 작업을 빠르게 병렬 실행
- ✓ 자동화 로직을 코드 레벨에서 깔끔하게 분리



GoEAPI

Go

GoEAPI는 Arista EOS 장비와 JSON-RPC API로 통신하기 위한 Go 언어 전용 클라이언트

주요 특징

- ✓ Arista EOS CLI 명령을 JSON 형태로 실행 및 결과 반환 가능
- ✓ 장비와의 연결/인증/명령 실행을 간결한 Go 인터페이스로 제공



GORM

Go

Go 언어에서 SQL 데이터베이스를 쉽게 다룰 수 있도록 해주는 ORM(Object Relational Mapping) 라이브러리

주요 특징

- ✓ Go struct와 DB 테이블 간 자동 매핑 지원
- ✓ 다양한 RDBMS 지원: SQLite, MySQL, PostgreSQL 등



SQLite3

DB

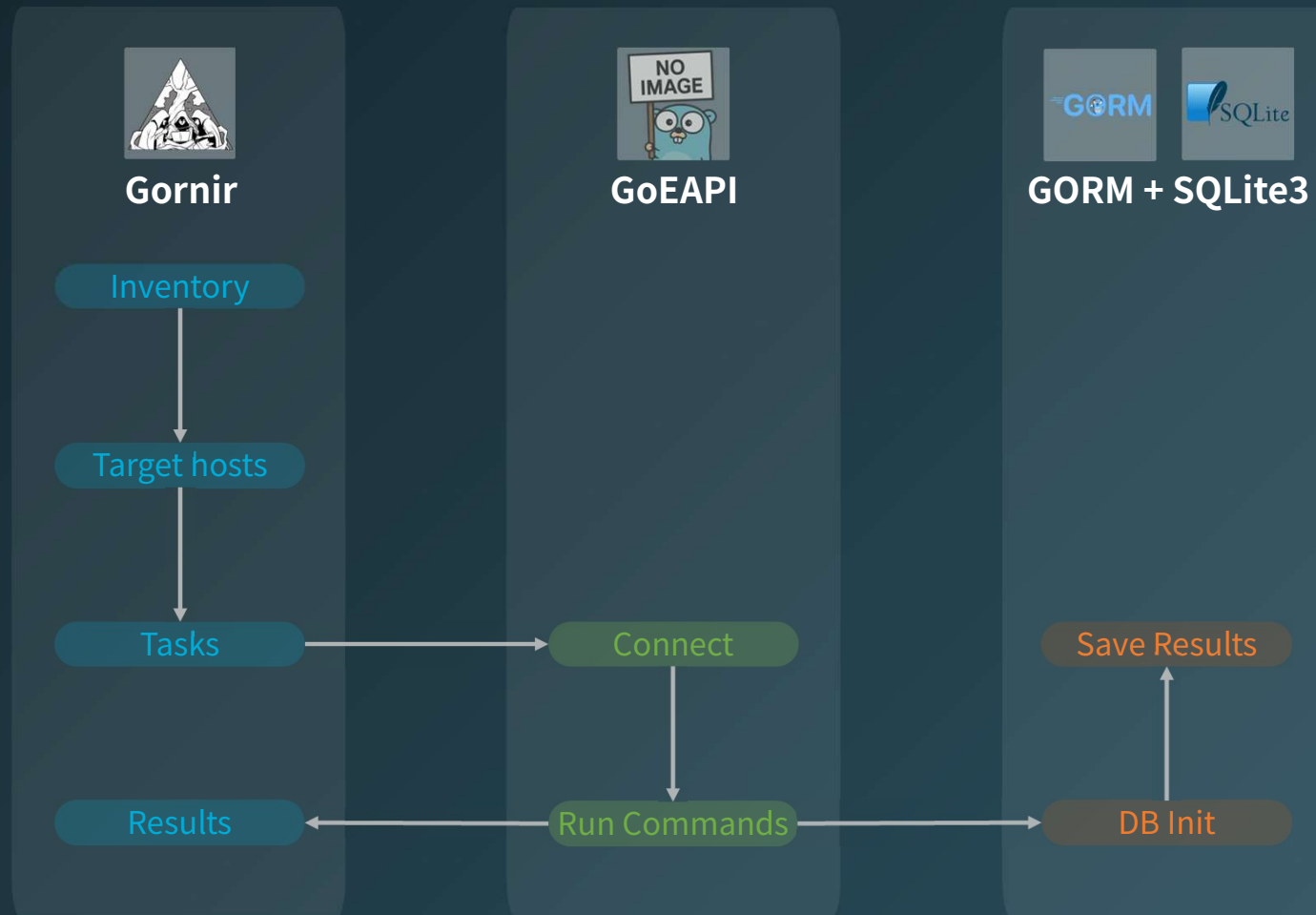
서버 없이 동작하는 경량화된 파일 기반 관계형 데이터베이스

주요 특징

- ✓ 단일 실행 파일 배포 가능 — 내장 DB로 가볍고 설치 불필요
- ✓ 소규모 자동화, 임시 저장, 테스트 환경에서 빠르고 안정적으로 작동

Go 기반 네트워크 자동화

Gornir + goeapi + gorm + sqlite3



Go 기반 네트워크 자동화

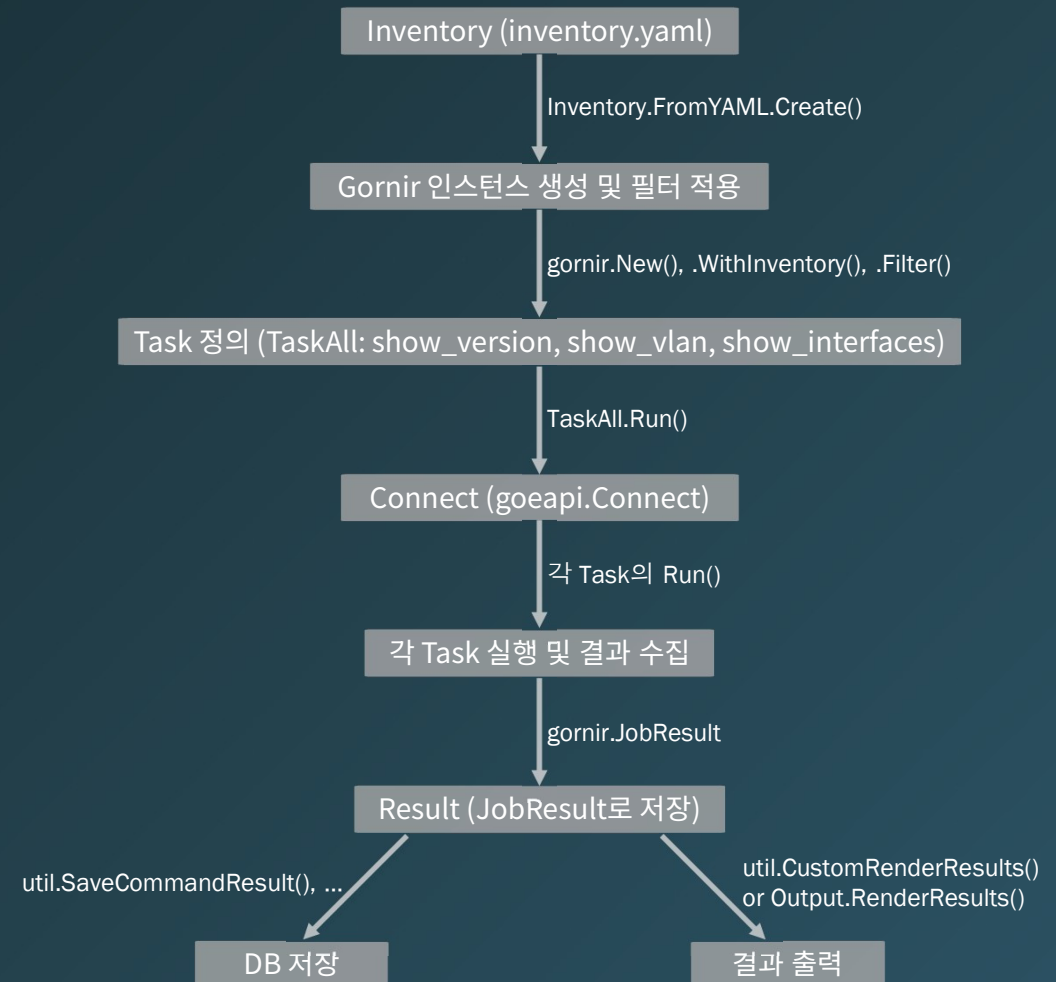
프로젝트 구조

특징

- ✓ Tasks:
- ✓ Util:

디렉토리 구조

```
├── config
│   └── inventory.yaml
├── go.mod
├── go.sum
├── main.go
├── results.db
├── tasks
│   ├── task_show_all.go
│   ├── task_show_version.go
│   ├── task_show_vlan.go
│   └── ...
├── util
│   ├── custom_render_results.go
│   └── db_util.go
```



Go 기반 네트워크 자동화

Inventory

설명

- ✓ Yaml 형태의 호스트 선언: 가독성이 뛰어난 계층적 구조
- ✓ select: 채널 병행 제어 전용 분기문

inventory.yaml

```
---
switch1:
  hostname: 192.168.1.51
  username: admin
  password: admin
  platform: eos
  data:
    groups:
      - arista
      - leaf

switch2:
  hostname: 192.168.1.52
  username: admin
  password: admin
  platform: eos
  data:
    groups:
  ...
```



Go 기반 네트워크 자동화

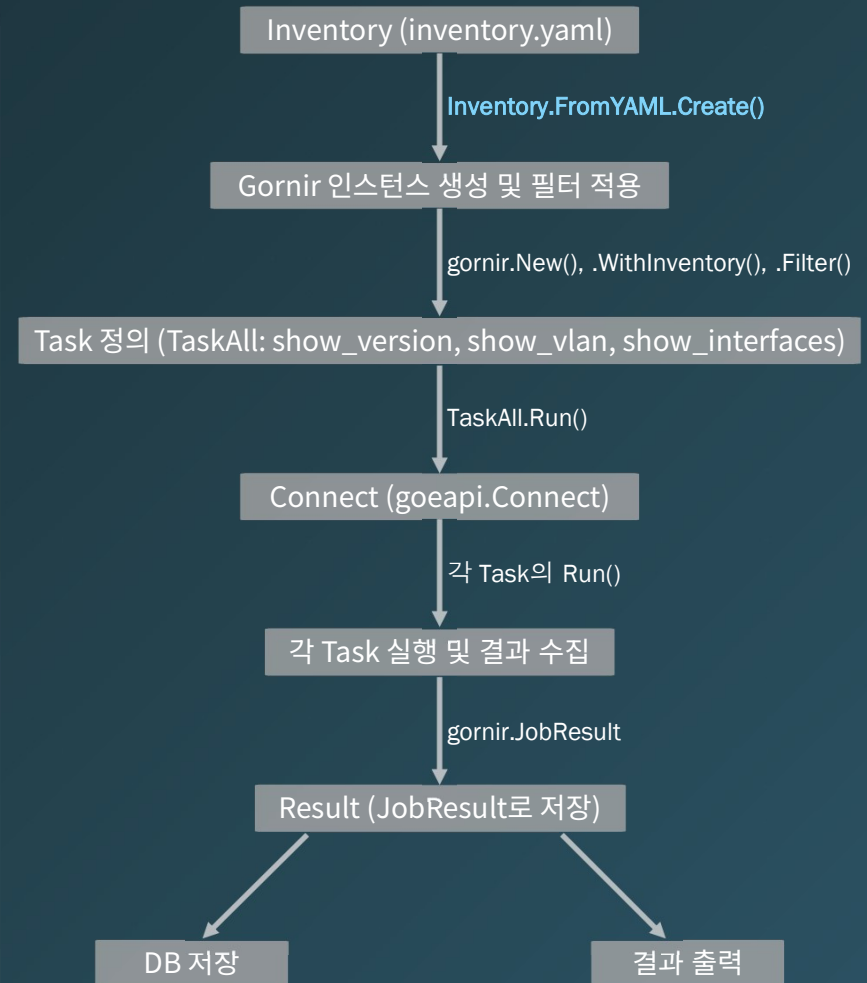
Inventory 파싱

설명

- 네트워크 장비 목록과 접속 정보를 inventory.yaml 파일에서 로드
- inventory.FromYAML.Create() 함수가 YAML 파일을 파싱하여 Gornir에서 사용할 인벤토리 객체 생성

main.go

```
func main() {  
    log := logger.NewLogrus(false)  
  
    // DB 테이블 생성  
    if _, err := util.InitDB("results.db", "show_vlan", "show_version", "show_interfaces"); err != nil {  
        log.Fatal(err)  
    }  
  
    file := "./config/inventory.yaml"  
    filter := "arista"  
    plugin := inventory.FromYAML{HostsFile: file}  
    inv, err := plugin.Create()  
    // ... (생략) ...  
}
```



Go 기반 네트워크 자동화

Gornir 인스턴스 생성 및 필터

설명

- ✓ Gornir 프레임워크의 인스턴스를 생성하고, 인벤토리와 로거, 병렬 실행 러너를 설정
- ✓ 필요에 따라 특정 그룹(예: arista)만 필터링하여 작업 대상을 제한

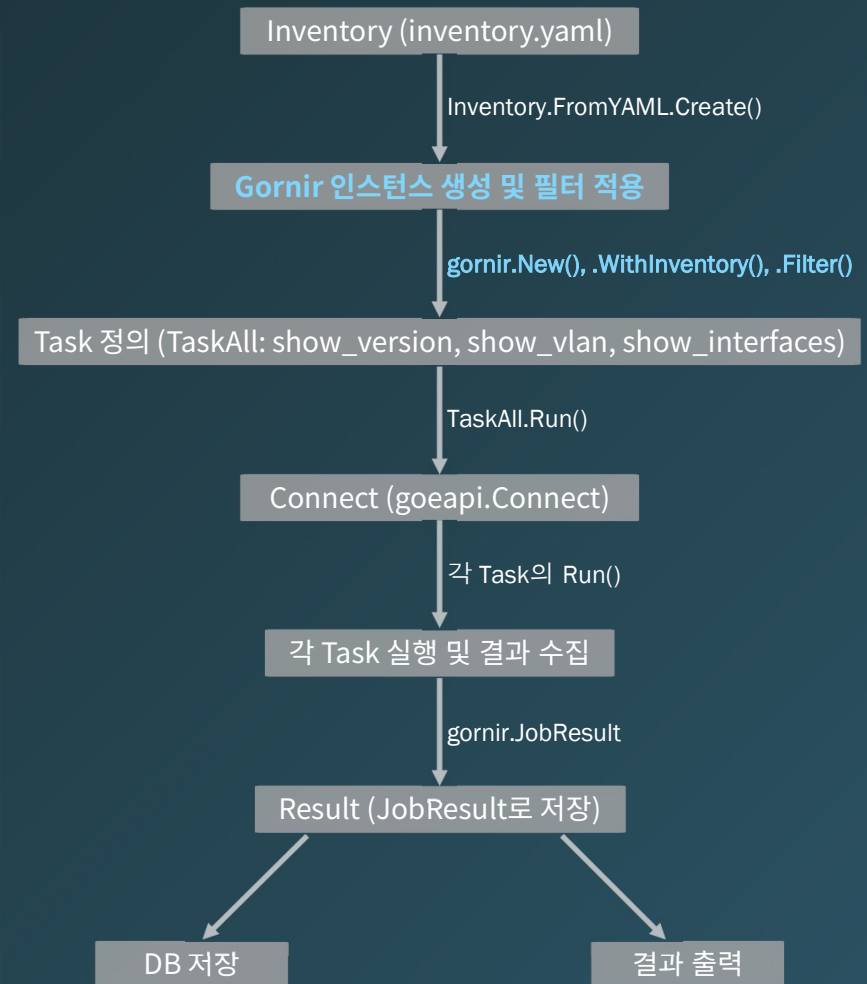
main.go

```
func main() {
    log := logger.NewLogrus(false)

    // ... (생략) ...
    file := "./config/inventory.yaml"
    filter := "arista"
    plugin := inventory.FromYAML{HostsFile: file}
    inv, err := plugin.Create()

    gr := gornir.New().WithInventory(inv).WithLogger(log).WithRunner(runner.Parallel())
    gr = gr.Filter(func(h *gornir.Host) bool {
        // ... (생략) ...
    })

    // 병렬 실행
    results, err := gr.RunSync(
        context.Background(),
        &eapi.TaskShowInterfaces{SaveToDB: true},
    )
    // ... (결과 수집 및 출력) ...
}
```



Go 기반 네트워크 자동화

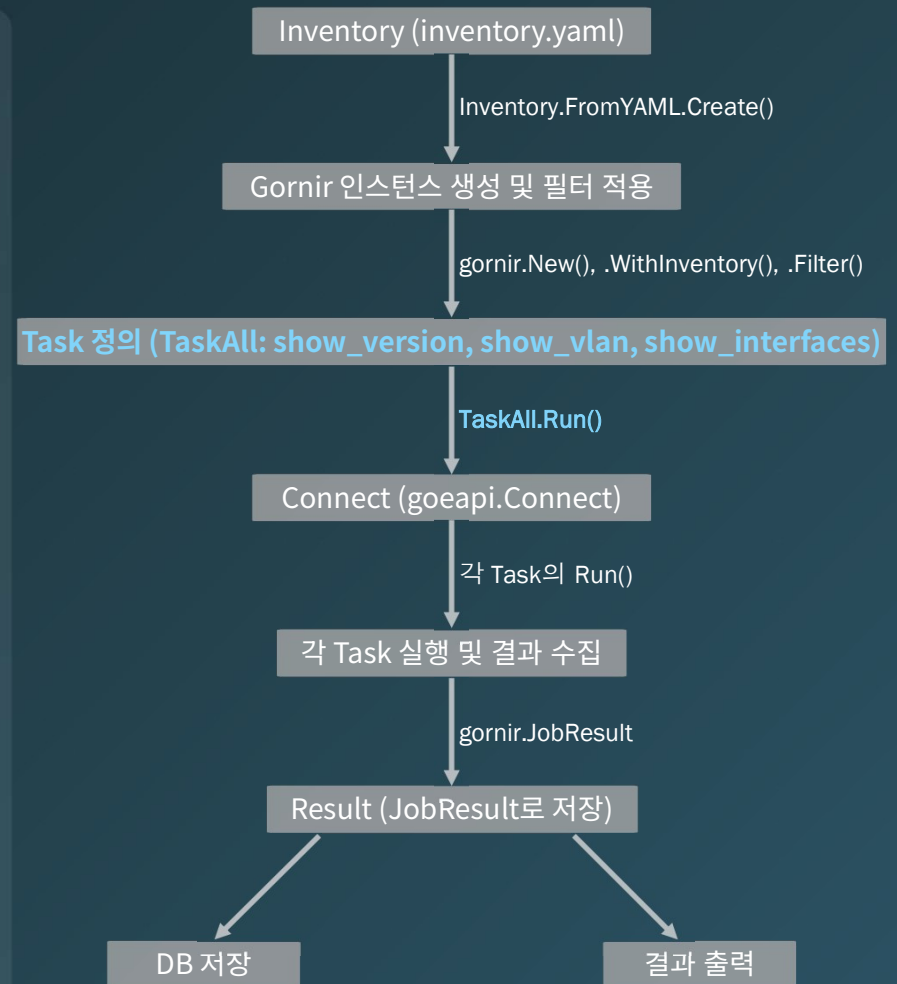
Task

설명

- ✓ 여러 개의 Task(명령어 실행 단위)를 묶어서 한 번에 실행할 수 있도록 TaskAll 구조체에 정의
- ✓ 각 Task는 장비에 접속해 명령어를 실행하고, 결과를 반환

main.go

```
func main() {  
    log := logger.NewLogrus(false)  
  
    // ... (생략) ...  
  
    // 병렬 실행  
    results, err := gr.RunSync(  
        context.Background(),  
        &eapi.TaskAll{  
            Tasks: []gornir.Task{  
                &eapi.TaskShowVersion{SaveToDB: true},  
                &eapi.TaskShowVlan{SaveToDB: true},  
                &eapi.TaskShowInterfaces{SaveToDB: true},  
            },  
        },  
    )  
  
    // ... (생략) ...  
}
```



Go 기반 네트워크 자동화

Connect (goeapi.Connect)

설명

- ✓ 각 Task의 Run() 함수 내부에서 goeapi.Connect()를 통해 실제 장비에 접속
- ✓ 접속 정보는 인벤토리에서 가져오며, HTTP API로 장비에 연결

e.g.) task_show_interfaces.go

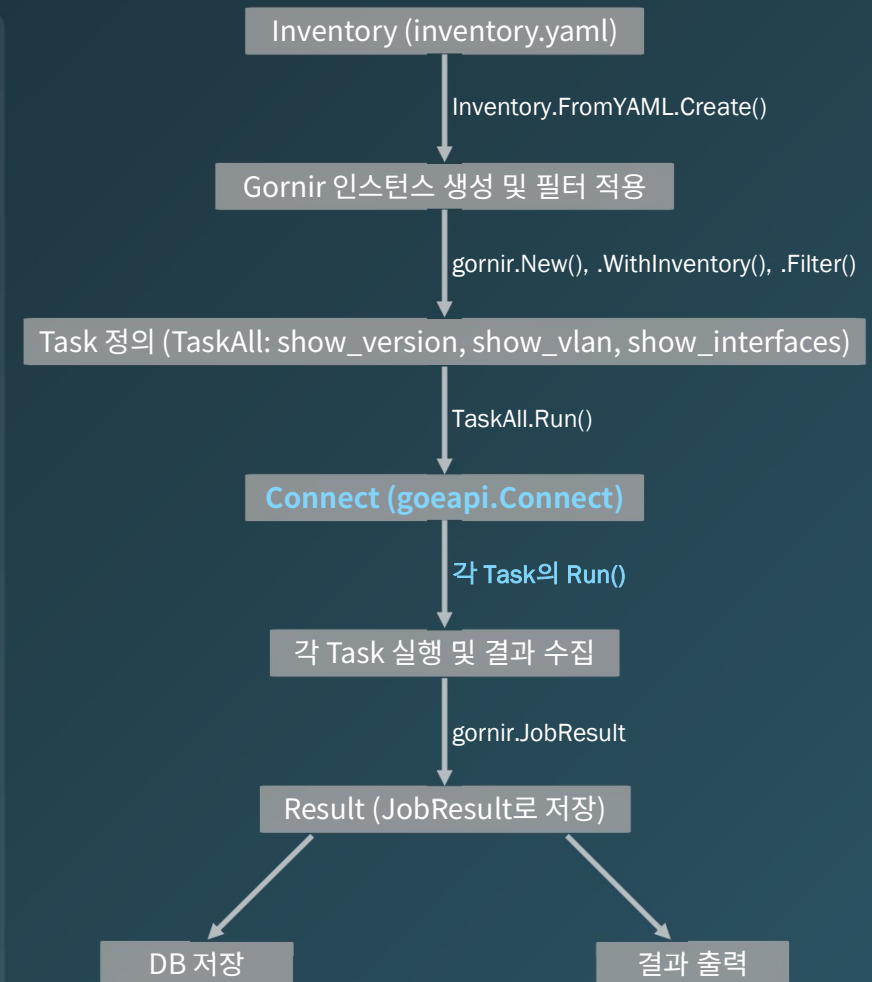
```
type InterfacesInfo struct {
    Interfaces map[string]interface{} `json:"interfaces"`
}

type TaskShowInterfaces struct {
    SaveToDB bool
}

func (t *TaskShowInterfaces) Run(ctx context.Context, logger gornir.Logger, host *gornir.Host) (gornir.TaskInstanceResult, error) {
    // ... (생략) ...
    info := InterfacesInfo{}

    node, err := goeapi.Connect("http", hostname, username, password, 80)
    // ... (결과 파싱) ...
    return info, nil
}

func (t *TaskShowInterfaces) Metadata() *gornir.TaskMetadata {
    return &gornir.TaskMetadata{Identifier: "show_interfaces"}
}
```



Go 기반 네트워크 자동화

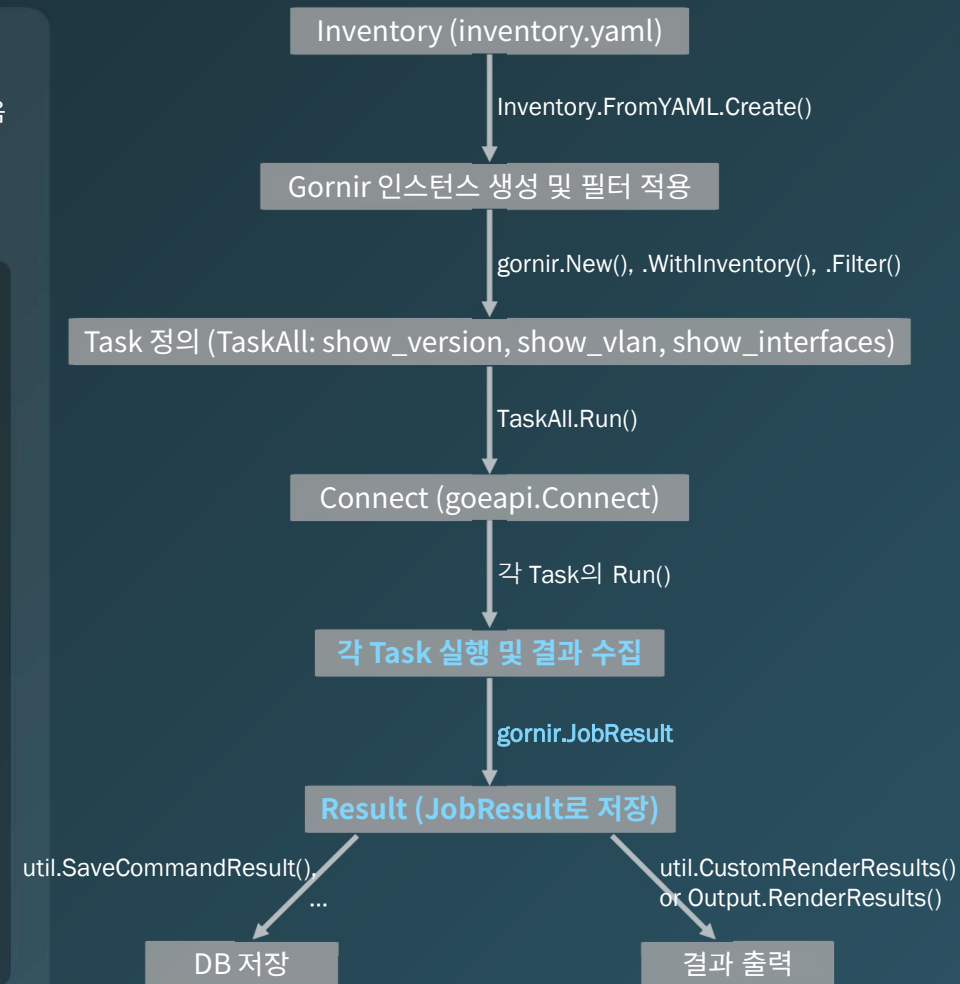
각 Task 실행 및 결과 수집

설명

- 장비에 명령어(예: show version, show vlan, show interfaces)를 실행하고, 결과를 JSON 형태로 받음
- 결과는 각 Task의 결과 구조체(예: InterfacesInfo, VersionInfo, VlanInfo 등)에 파싱

e.g.) task_show_interfaces.go

```
type InterfacesInfo struct {  
    Interfaces map[string]interface{} `json:"interfaces"`  
}  
  
func (t*TaskShowInterfaces) Run(ctx context.Context, logger gornir.Logger, host*gornir.Host)  
(gornir.TaskInstanceResult, error) {  
    // ... (생략) ...  
    commands := []string{"show interfaces"}  
    result, err := node.RunCommands(commands, "json")  
  
    interfacesData := result.Result[0]  
  
    rawJSON, err := json.Marshal(interfacesData)  
    if err != nil {  
        return info, fmt.Errorf("marshal failed: %v", err)  
    }  
  
    if err := json.Unmarshal(rawJSON, &info); err != nil {  
        return info, fmt.Errorf("unmarshal failed: %v", err)  
    }  
    // ... (생략) ...  
}
```



Go 기반 네트워크 자동화

DB 저장

설명

- ✓ 각 Task에서 SaveToDB 옵션이 true일 경우, 결과를 SQLite DB에 저장
- ✓ util.SaveCommandResult() 함수가 테이블명에 맞게 결과를 기록

db_util.go

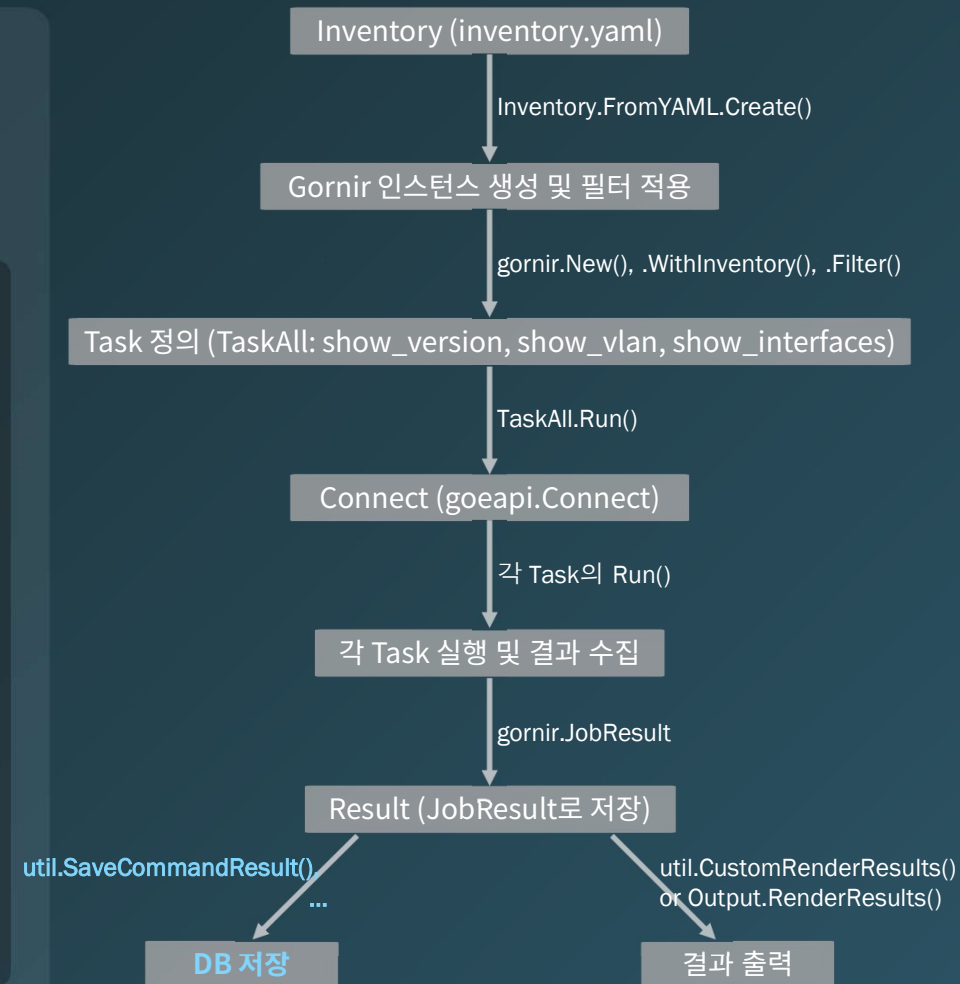
```
type InterfacesInfo struct {
    Interfaces map[string]interface{} `json:"interfaces"`
}

func (t*TaskShowInterfaces) Run(ctx context.Context, logger gornir.Logger, host*gornir.Host)
(gornir.TaskInstanceResult, error) {
    // ... (생략) ...
    commands := []string{"show interfaces"}
    result, err := node.RunCommands(commands, "json")

    interfacesData := result.Result[0]

    rawJSON, err := json.Marshal(interfacesData)
    if err != nil {
        return info, fmt.Errorf("marshal failed: %v", err)
    }

    if err := json.Unmarshal(rawJSON, &info); err != nil {
        return info, fmt.Errorf("unmarshal failed: %v", err)
    }
    // ... (생략) ...
}
```



Go 기반 네트워크 자동화

결과 출력 (CustomRenderResults, output.RenderResults)

설명

- ✓ 커스텀 포맷(CustomRenderResults)과 표준 포맷(output.RenderResults)으로 결과를 출력
- ✓ 성공/실패, 각 장비별 주요 정보를 보기 쉽게 정리

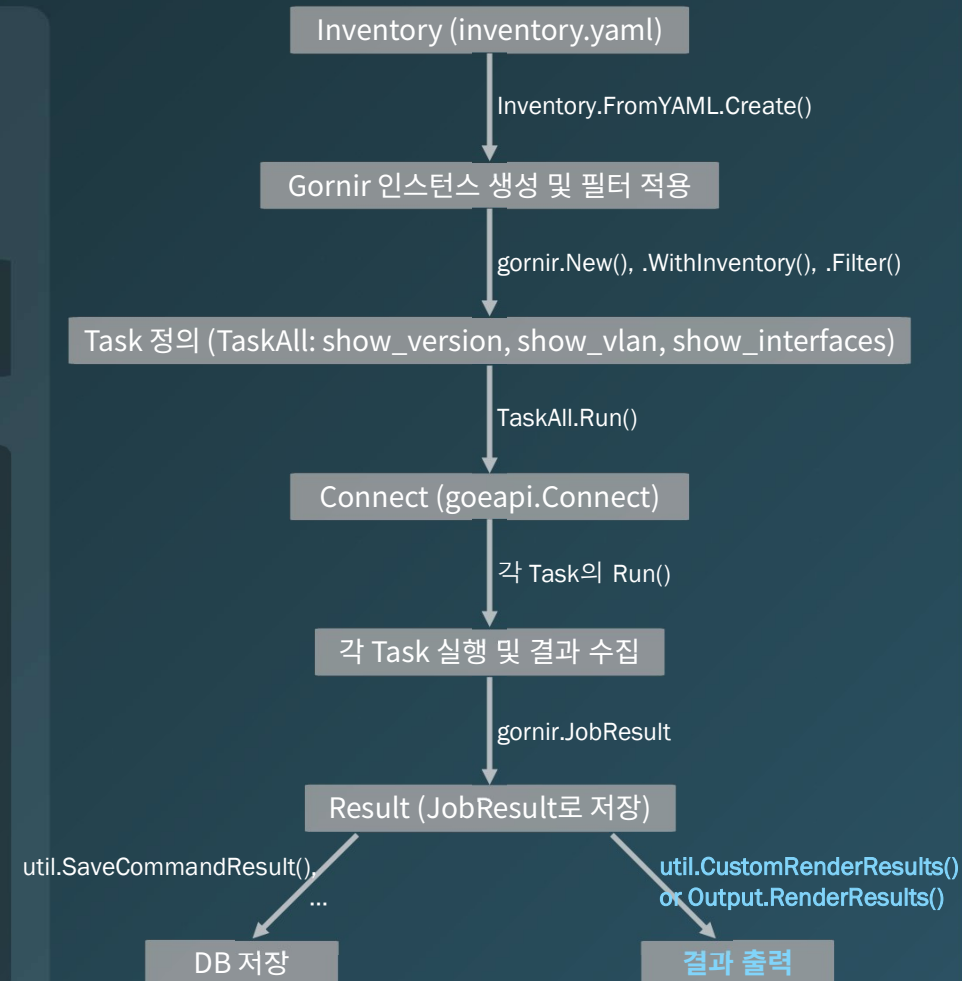
main.go

```
util.CustomRenderResults(os.Stdout, storedResults, "장비 정보") // custom_render_results.go
output.RenderResults(os.Stdout, resultsCopy, "장비 정보", true)
```

task_show_version.go

```
func (i VersionInfo) Format() string { //CustomRenderResults 의 매서드로 포맷 변경시 사용
    return fmt.Sprintf("show version:\n - 버전: %s\n - 모델: %s\n - 시리얼 번호: %s\n - 시스템\nMAC: %s\n - 아키텍처: %s\n",
        i.Version, i.Model, i.SerialNum, i.SystemMac, i.Architecture)
}

func (v VersionInfo) String() string { // RenderResults 의 매서드로 포맷 변경시 사용
    return fmt.Sprintf(
        `{Version: %s, Model: %s, Serial: %s, System MAC: %s, Architecture: %s, Bootup: %.0f, Config\nMAC: %s, HW Revision: %s, HW MAC: %s, Image Format: %s, Optimization: %s, Build ID: %s,\n// ... (생략) ...`
        , v.Version, v.Model, v.SerialNum, v.SystemMac, v.Architecture, v.BootupTimestamp,\n        v.ConfigMacAddress, v.HardwareRevision, v.HwMacAddress, v.ImageFormatVersion,\n        // ... (생략) ...`
    )
}
```



Go 기반 네트워크 자동화

코드 실행 및 결과

각 호스트별 전체 Task 진행 및 결과 DB 저장 확인

```
~/go/s/n.ex.t-38th/gornir_w_goeapi main !6 ?1 go run main.go
Running task: show_version on host: 192.168.1.52
Running task: show_version on host: 192.168.1.54
Running task: show_version on host: 192.168.1.53
Running task: show_version on host: 192.168.1.51
Running task: show_vlan on host: 192.168.1.54
Running task: show_vlan on host: 192.168.1.51
Running task: show_vlan on host: 192.168.1.52
Running task: show_interfaces on host: 192.168.1.54
Running task: show_interfaces on host: 192.168.1.51
Running task: show_interfaces on host: 192.168.1.53
Running task: show_interfaces on host: 192.168.1.52
Running task: show_interfaces on host: 192.168.1.53

===== 장비 정보 =====
총 장비 : 4, 성공 : 4, 실패 : 0

[✓] 192.168.1.54:
show version:
- 버전 : 4.29.2F
- 모델 : vEOS-lab
- 시리얼 번호 : 5C15DB49AD775FA325746437A0940646
- 시스템 MAC : 50:02:46:09:be:0e
- 아키텍처 : x86_64

show vlan:
- VLAN 1 (default): interfaces=1 (Ethernet3)

show interfaces: 5개 인터페이스
[✓] 192.168.1.51:
show version:
- 버전 : 4.29.2F
- 모델 : vEOS-lab
- 시리얼 번호 : E7FFE784AB4D108D5174137D2F8DC763
- 시스템 MAC : 50:cf:78:11:e7:5c
- 아키텍처 : x86_64
```

TABLES					id	host	command	result
show_interfaces					id	host	command	result
ROWID					1	192.168.1.54	show interfaces	{"interfaces":{"Ethernet1":{"autoNegotiate":"unknown","b...
id					2	192.168.1.51	show interfaces	{"interfaces":{"Ethernet1":{"autoNegotiate":"unknown","b...
host					3	192.168.1.52	show interfaces	{"interfaces":{"Ethernet1":{"autoNegotiate":"unknown","b...
command					4	192.168.1.53	show interfaces	{"interfaces":{"Ethernet1":{"autoNegotiate":"unknown","b...
result					5			
created_at								
show_version								
ROWID								
id								
host								
command								
result								
created_at								
show_vlan								
ROWID								
id								
host								
command								
result								
created_at								
sqlite_sequence								

Go 기반 네트워크 자동화

데모 프로젝트

<https://github.com/Aiden-Yoo/n.ex.t-38th>

The screenshot shows the GitHub repository page for 'n.ex.t-38th' by user 'Aiden-Yoo'. The repository is public and has 0 stars, 0 forks, and 0 watchers. The main branch is 'main' with 1 branch and 0 tags. The repository contains 2 commits, the latest being '67c5eaf' from yesterday. The file list shows 'gornir_w_goeapi' (yesterday), 'nornir_w_pyeapi' (3 days ago), and '.gitignore' (3 days ago). The README section is empty with a prompt to 'Add a README'. The right sidebar shows 'About' (no description), 'Activity', 'Releases' (no releases), 'Packages' (no packages), and 'Languages' (Go 58.9%, Python 41.1%).

Aiden-Yoo / n.ex.t-38th

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

n.ex.t-38th Public

Unpin Watch 0 Fork 0 Star 0

main 1 Branch 0 Tags

Go to file

Code

Aiden-Yoo ADD: task_show_interfaces 67c5eaf · yesterday 2 Commits

gornir_w_goeapi	ADD: task_show_interfaces	yesterday
nornir_w_pyeapi	first commit	3 days ago
.gitignore	first commit	3 days ago

README

Add a README

Help people interested in this repository understand your project by adding a README.

Add a README

About

No description, website, or topics provided.

Activity

0 stars

0 watching

0 forks

Releases

No releases published

[Create a new release](#)

Packages

No packages published

[Publish your first package](#)

Languages

Go 58.9% Python 41.1%

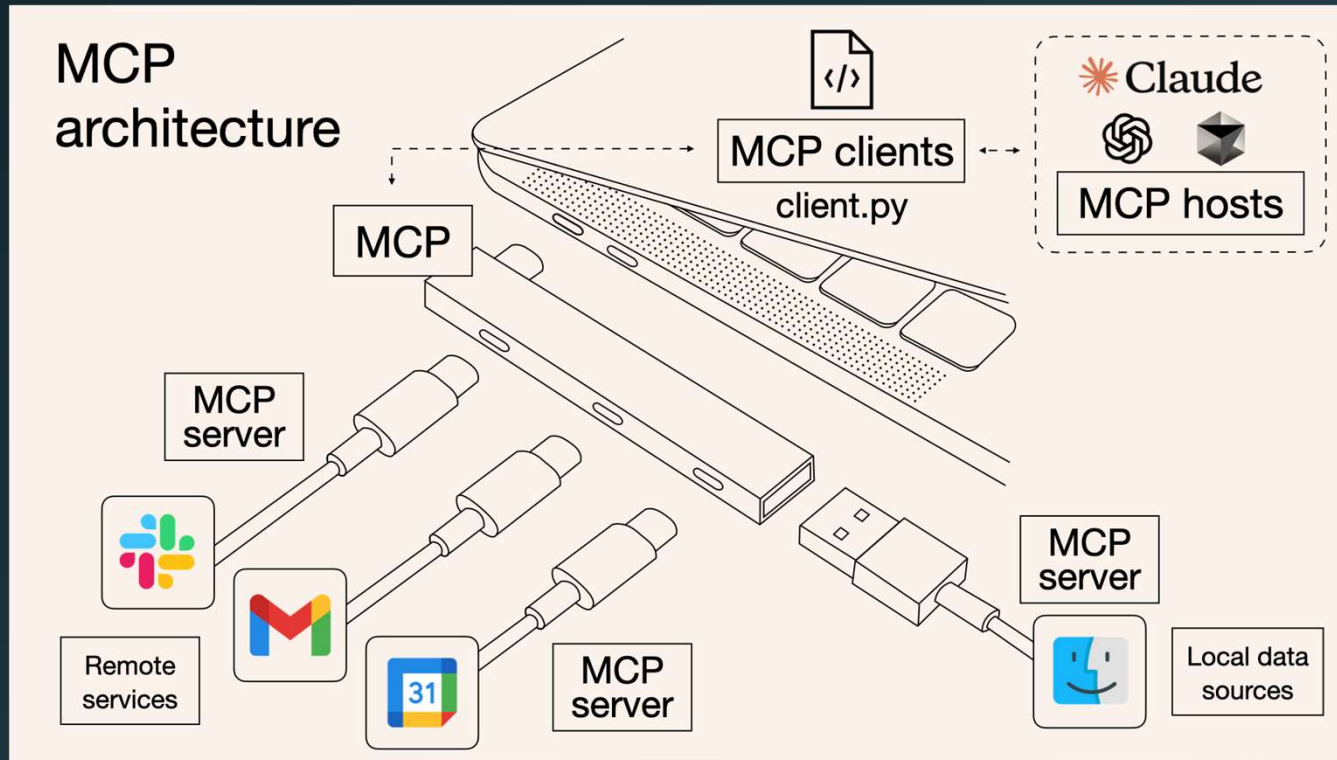
AI-powered Observability

MCP 를 활용한 AI 연동

MCP 소개

Model Context Protocol (MCP)

MCP는 AI 언어 모델(LLM)이 외부 도구, API, 또는 에이전트와 직접 상호 작용할 수 있도록 설계된 통신 프로토콜



MCP 소개

MCP Host & Client & Server

MCP Client

서버와 통신하여 모델의 상태나 명령을 요청하고, 서버로부터 결과를 받아오는 역할을 수행하는 구현체

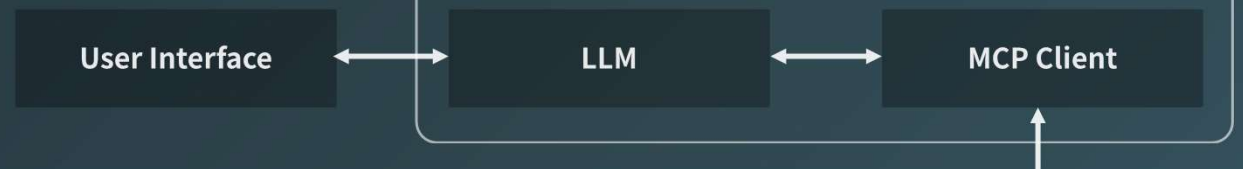
- ✓ 호스트와 서버간의 통신 및 데이터 관리(적절한 MCP 서버로 요청 전달)
- ✓ 메시지 변환 및 처리 등을 담당(MCP 서버로부터 결과를 반환)

MCP Server

클라이언트로부터 요청을 받아 모델의 상태를 관리하고, 명령을 실행하거나 데이터를 제공

- ✓ 언어 모델과 도구 간의 중재자 역할
- ✓ 각종 도구 및 기능, 외부 API 같은 외부 리소스를 제공하는 역할

MCP Host



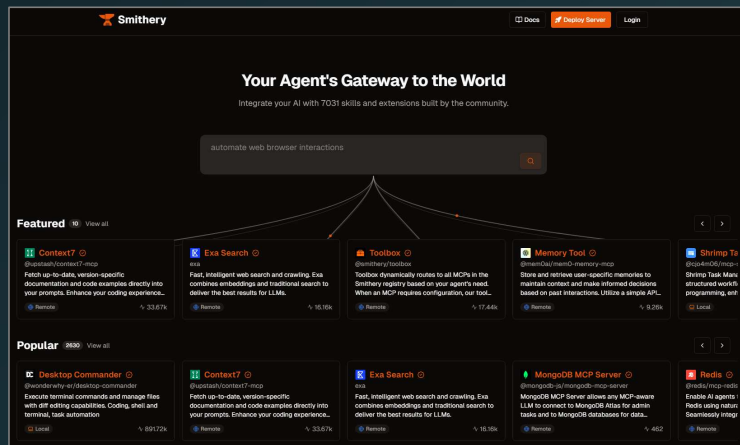
MCP Server



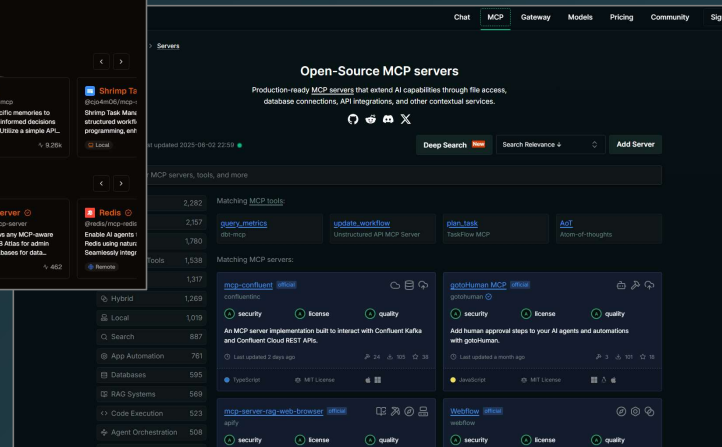
MCP 소개

MCP Marketplace

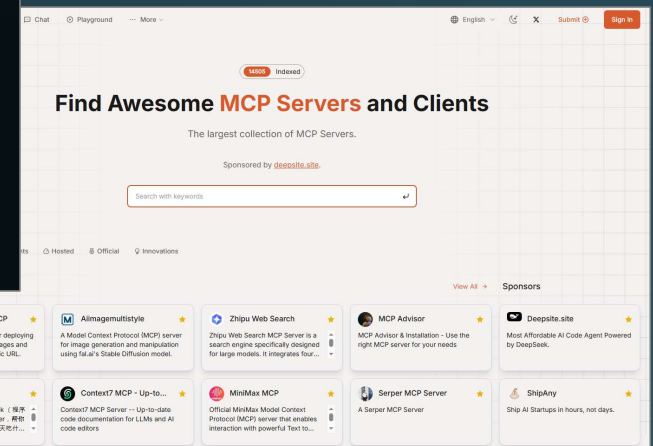
LLM이 사용할 수 있는 도구(tool), 기능(function), 외부 API들을 공식적으로 공유하고 검색 가능한 공간



Smithery : <https://smithery.ai/>



Glama : <https://glama.ai/chat>



MCP.so : <https://mcp.so/>

MCP 소개

MCP Servers



Desktop Commander

@wonderwhy-er/desktop-commander

파일 시스템과 터미널에 접속할 수 있도록 해주는 MCP 도구로 자연어를 이용해 터미널 명령 사용 가능

주요 특징

- ✓ 파일 시스템 작업(파일 읽기/쓰기), 디렉토리 생성/나열, 파일 이동 등
- ✓ 프로세스 관리(프로세스 확인 및 종료), 터미널 명령 실행 등



GitHub

@smithery-ai/github

GitHub Repository와 직접 상호작용할 수 있도록 도와주는 MCP 도구

주요 특징

- ✓ 이슈 생성 및 Pull Request 작성
- ✓ 코드 변경사항 분석



Context7

@upstash/context7-mcp

코드 생성을 위해 최신 문서를 직접 호출하여 정확도를 확보하고 공식문서 기반의 최신 코드 예제를 직접 제공

주요 특징

- ✓ 대화 및 작업의 흐름 유지 (세션 단위 맥락 유지)
- ✓ 최신 버전의 문서를 동적으로 제공 - 할루시네이션 감소 효과



Task Master

@eyaltoledano/claude-task-master

AI 기반 작업 관리 시스템으로 프로젝트를 구조적이고 효율적으로 진행할 수 있도록 돕는 강력한 도구

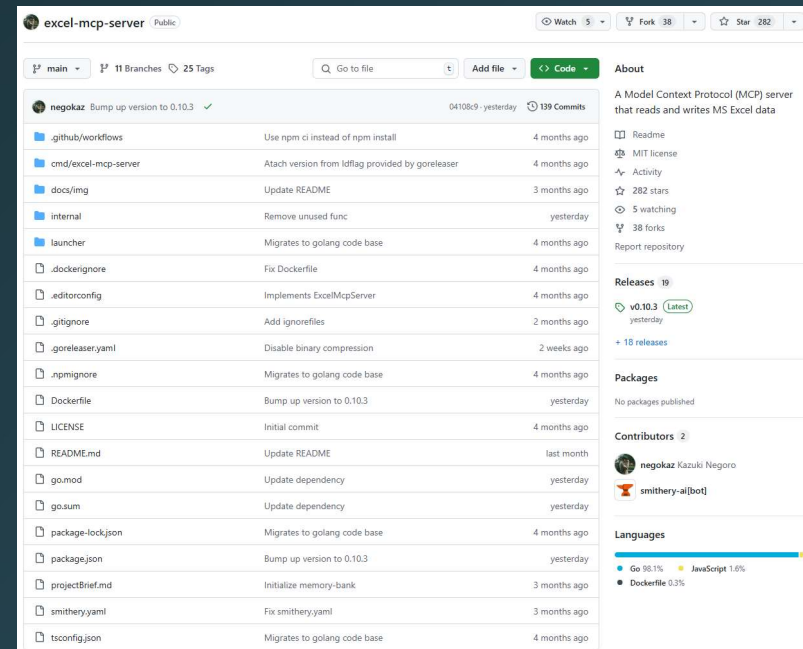
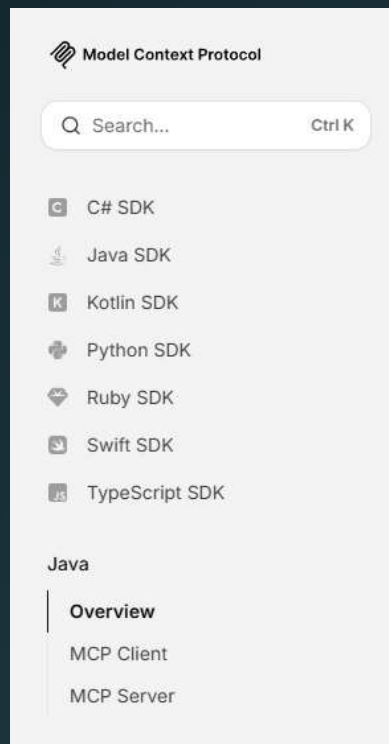
주요 특징

- ✓ 작업을 동적으로 생성, 우선순위를 지정하여 효율적인 워크플로우 보장
- ✓ 작업 진행 상황 및 프로젝트 상태를 모니터링

MCP 소개

MCP SDKs

Model Context Protocol(MCP)을 위한 다양한 언어의 SDK를 제공하며
애플리케이션이 표준화된 인터페이스를 통해 AI 모델 및 도구와 상호 작용할 수 있도록 지원

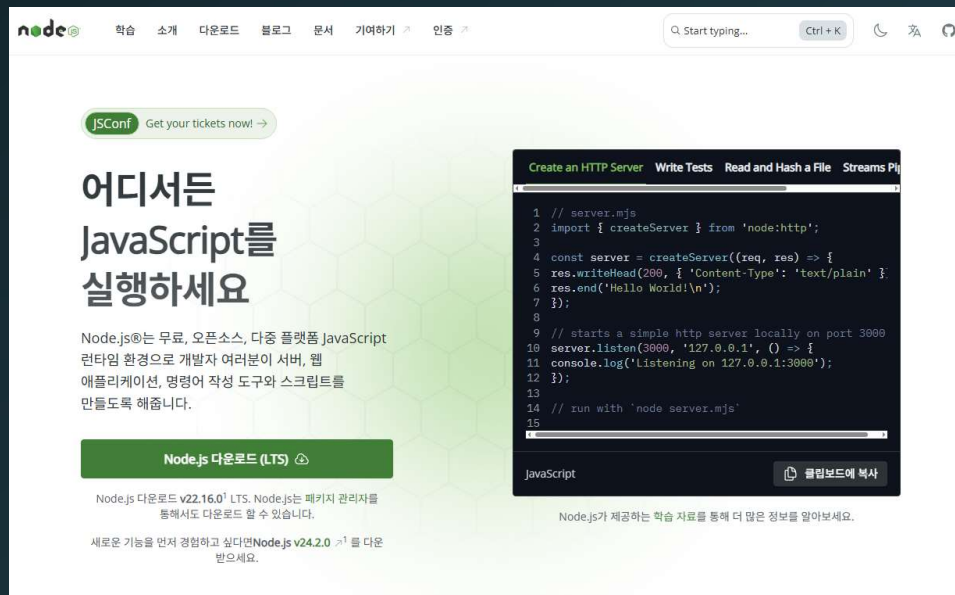


<https://github.com/negokaz/excel-mcp-server>
<https://github.com/mark3labs/mcp-go>

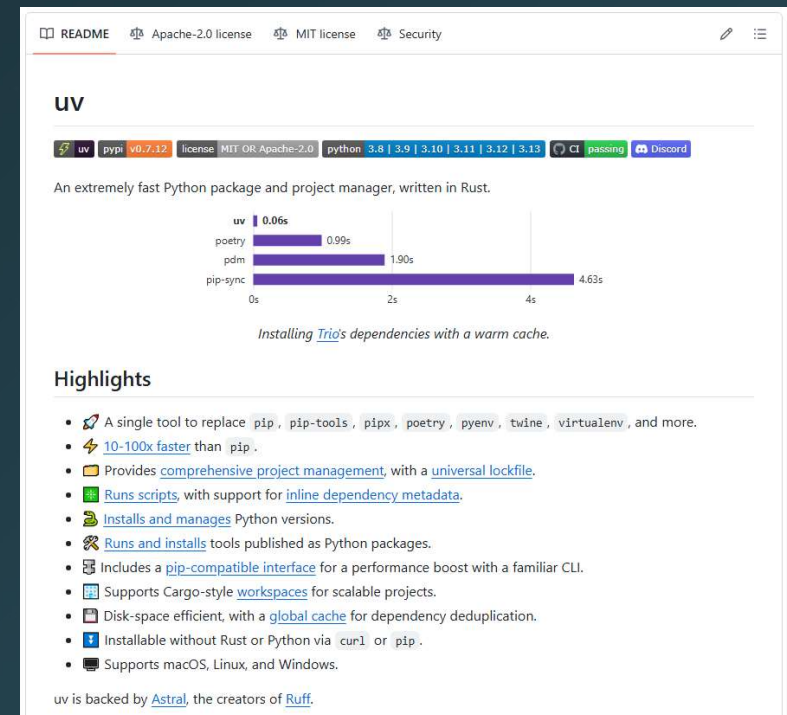
MCP 서버 활용

Installation – npy/uvx

MCP 서버에 따라 실행을 위한 도구가 다를 수 있어 필요에 따라 설치



Nodejs (npm/npv) - <https://nodejs.org>



uv (uvx) - <https://github.com/astral-sh/uv>

MCP 서버 활용

Installation – MCP

Client 별 명령어 기반 설치 or JSON 설정 기반 설치

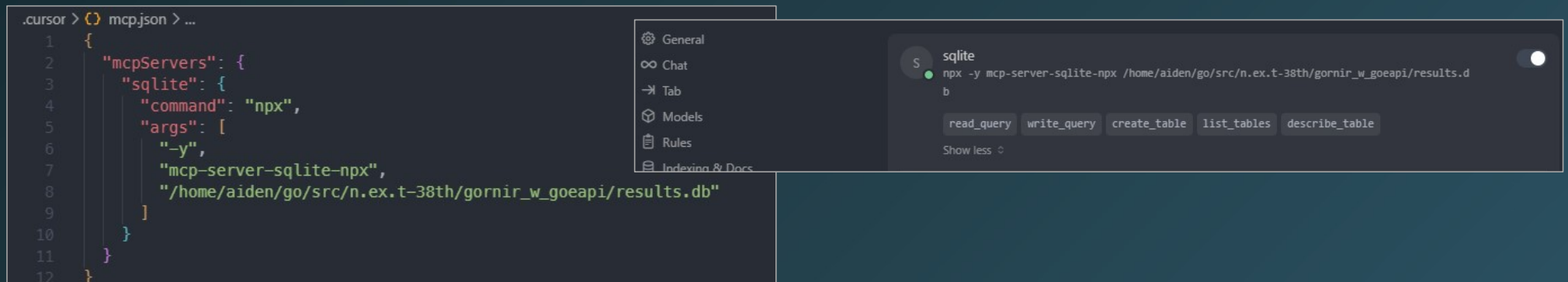
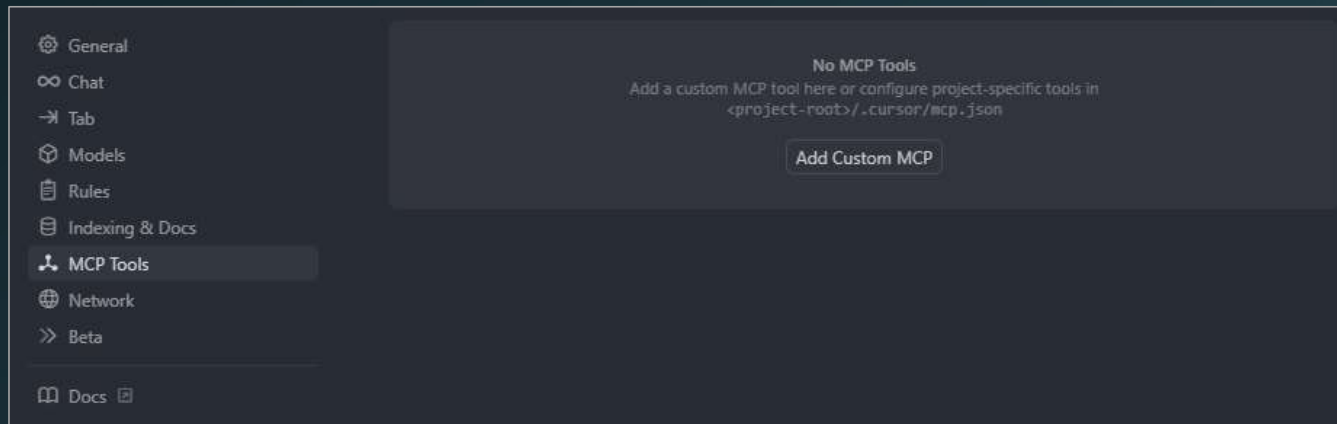
The screenshot shows the Smithery website interface. At the top, there's a navigation bar with links for Docs, Explore, Playground, and Deploy Server. Below this is a search bar with the placeholder text "Search or prompt for servers...". The main content area is titled "SQLite Server" and includes a "Claim Server" button. Below the title, there's a description: "A Node.js implementation of the Model Context Protocol SQLite server, providing an npx-based alternative for environments lacking Python's UVX runner." There are tabs for "Overview", "Tools", and "API". The "Tools" tab is selected, showing a list of tools. A message states "Configuration required to view tools" with an "Explore Tools" button. On the right side of the page, there's an "Install" section with tabs for "Auto", "JSON", and "URL". The "JSON" tab is selected, showing a configuration in JSON format for Mac/Linux, Windows, and WSL. The JSON configuration includes the command "npx" and the arguments "y", "@smithery/cli@latest", "install", "mcp-server-sqlite-npx", and "--client cur". A warning message at the bottom states: "Your smithery key is sensitive. Please don't share it with anyone."

The screenshot shows the "Install" section of the Smithery website. It has tabs for "Auto", "JSON", and "URL". The "Auto" tab is selected, showing a "Cursor" dropdown menu. Below this, there's a text prompt: "Run the following command to install for Cursor." A terminal icon is shown next to the command: `npx -y @smithery/cli@latest install mcp-server-sqlite-npx --client cur`. A "Copy" button is next to the command. At the bottom, there's a warning message: "Your smithery key is sensitive. Please don't share it with anyone." and a "Report Issue" button.

MCP 서버 활용

Installation – MCP (sqlite)

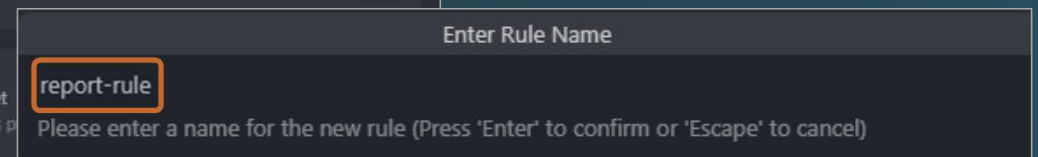
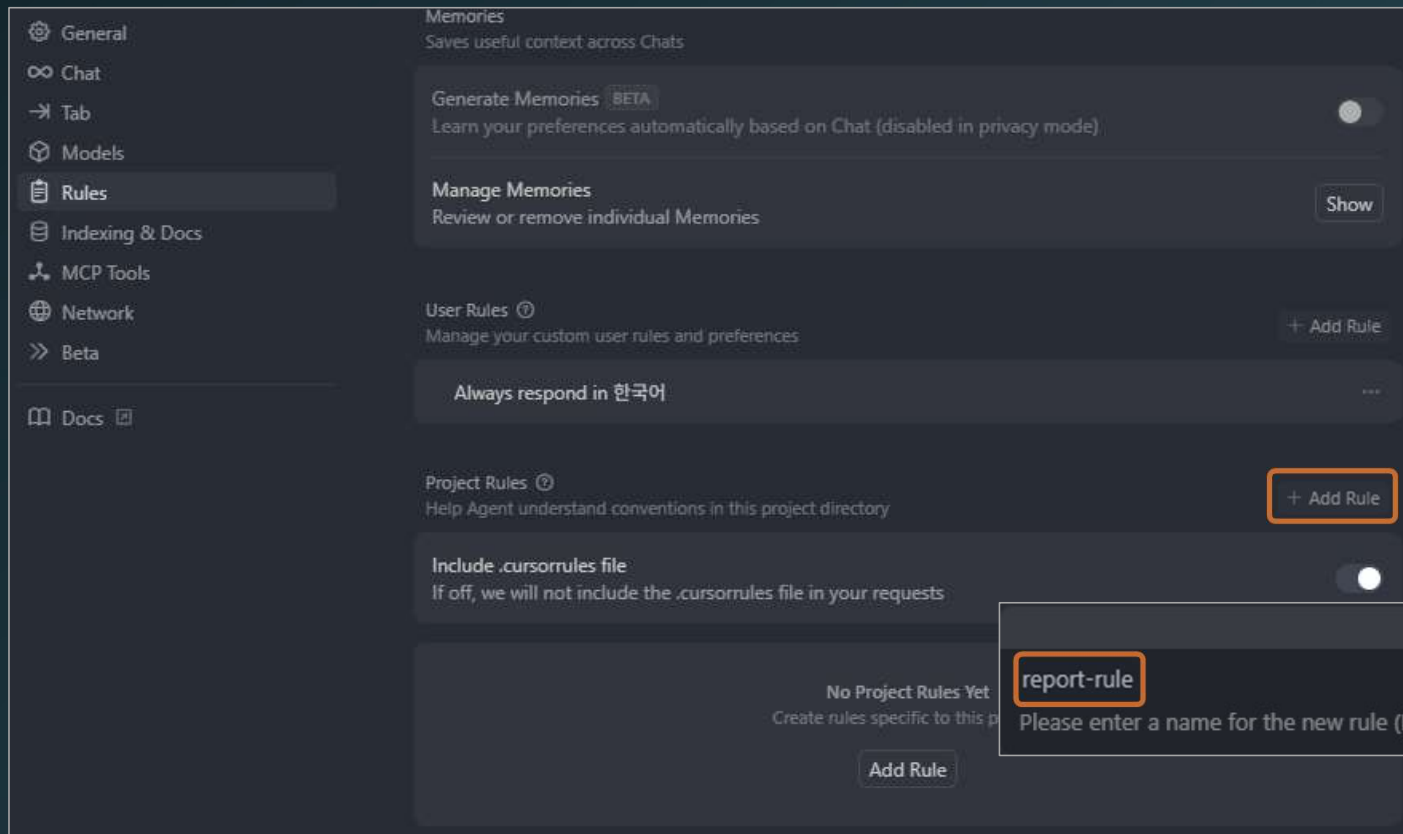
Ctrl + Shift + J → MCP Tools → Add Custom MCP



MCP 서버 활용

Cursor Rules

Ctrl + Shift + J → Rules → Add Rule



MCP 서버 활용

Cursor Rules

Rule Type: Always

Rule Type

Always

This rule attached to every chat and command+k request

MCP 기반 네트워크 점검 지침 (LLM Instruction)

1. 목적

수집된 네트워크 장비 정보를 바탕으로, 잠재적인 문제를 식별하고, 원인과 조치 방법, 그리고 점검 절차까지 포함한 네트워크 점검 리포트를 생성.

2. 데이터 구조 요약

* 수집된 정보는 디렉토리 내 results.db 에 저장되어있고 sqlite라는 이름의 MCP를 통해 접근 가능.

* 각 명령어별 테이블 구조는 다음과 같음:

* 테이블명: `show\_version`, `show\_interfaces`, `show\_vlan`, ...

* 공통 컬럼:

* `hostname`: 장비 이름

* `timestamp`: 수집 시각

* `result`: JSON 문자열 형식의 결과

3. 점검 시작 조건

사용자가 다음과 같이 입력할 경우:

네트워크 점검

LLM은 다음 단계를 수행합니다.

- Step 1: 데이터 파싱

* 각 테이블의 `result` 컬럼을 JSON 파싱합니다.

* 장비별로 데이터를 연관지어 하나의 상태 정보를 구성합니다.

예: 동일 hostname 기준으로 show version, interfaces, vlan 등을 통합.

- Step 2: 분석 기준

MCP 서버 활용

명령 실행

Rule Type: Always

네트워크 점검

> Called MCP tool list_tables ✓

> Called MCP tool read_query ✓

> Called MCP tool read_query ✓

> Called MCP tool read_query ✓

> Called MCP tool read_query ✓

> Called MCP tool read_query ✓

> Called MCP tool read_query ✓

> Called MCP tool read_query ✓

네트워크 분석 리포트

장비: 192.168.1.51

1. [문제] VLAN 구성 다수 및 기타 장비와 불일치

- > 원인 추정: 192.168.1.51 장비에는 VLAN 1, 10, 20, 30이 존재하나, 나머지 장비 (192.168.1.52, 53, 54)는 VLAN 1만 존재
- > 조치 방법: 네트워크 설계 기준에 따라 불필요한 VLAN 정리 또는 타 장비와의 VLAN 동기화 필요성 검토
- > 점검 절차:
 - show vlan
 - 네트워크 설계 문서 확인
 - 필요시 VLAN 추가/삭제 작업

2. [문제] 인터페이스 상태 변화(Flap) 감지

- > 원인 추정: Ethernet1~3, Management1에서 linkStatusChanges=2, 4 등 상태 변화 발생
- > 조치 방법: 포트 flap 원인(물리적 연결, peer 장비 상태 등) 확인
- > 점검 절차:
 - show interfaces status

장비: 192.168.1.52

1. [문제] VLAN 1만 존재

- > 원인 추정: 단일 VLAN만 구성되어 있어, 멀티 VLAN 환경 필요시 추가 필요
- > 조치 방법: 네트워크 정책에 따라 VLAN 추가 여부 검토

2. [문제] 인터페이스 상태 변화(Flap) 감지

- > 원인 추정: Ethernet1~3, Management1에서 linkStatusChanges=2, 4 등 상태 변화 발생
- > 조치 방법: 포트 flap 원인(물리적 연결, peer 장비 상태 등) 확인

3. [문제] OS/버전, 메모리, CPU 등 특이사항 없음

장비: 192.168.1.53

1. [문제] VLAN 1만 존재

- > 원인 추정: 단일 VLAN만 구성되어 있어, 멀티 VLAN 환경 필요시 추가 필요

2. [문제] Ethernet1 인터페이스 에러 카운터 증가

- > 원인 추정: alignmentErrors(10), fcsErrors(90), inDiscards(110) 등 에러 발생 → 케이블 불량, 포트 불량, peer 문제 가능성
- > 조치 방법: 물리적 포트 및 케이블 점검, peer 장비 상태 확인
- > 점검 절차:
 - show interfaces Ethernet1
 - show log | include Ethernet1
 - 케이블 교체 및 peer 포트 상태 확인

3. [문제] 인터페이스 상태 변화(Flap) 감지

4. [문제] OS/버전, 메모리, CPU 등 특이사항 없음

장비: 192.168.1.54

1. [문제] VLAN 1만 존재

- > 원인 추정: 단일 VLAN만 구성되어 있어, 멀티 VLAN 환경 필요시 추가 필요

2. [문제] 인터페이스 상태 변화(Flap) 감지

공통 이슈 요약

- VLAN 구성 불일치: 192.168.1.51만 다수 VLAN 보유, 나머지는 VLAN 1만 존재 → 표준화 필요
- 인터페이스 상태 변화(Flap): 모든 장비에서 linkStatusChanges 값이 2~4로 flap 발생 → 물리적 연결, peer 상태 점검 필요
- 특정 포트 에러: 192.168.1.53의 Ethernet1에서 에러 카운터 증가 → 케이블/포트/peer 점검 필요

추가 조치 권장

- MCP 서버 연동 시, "show interfaces" 및 "show vlan" 명령어를 각 장비에서 재실행하여 최신 상태를 확인하는 것을 권장합니다.

> 추가 점검이 필요한 항목이나, 상세 로그/통계가 필요할 경우 말씀해 주세요.

Appendix

context7 + shrimp-task-manager + desktop-commander

Thought for 7 seconds

현재 프로젝트 구조를 먼저 파악하고 리팩토링 계획을 세워보겠습니다.

> Called MCP tool list_directory ✓

현재 프로젝트 구조를 더 자세히 파악해보겠습니다.

> Called MCP tool read_file ✓

> Called MCP tool read_file ✓

> Called MCP tool list_directory ✓

> Called MCP tool list_directory ✓

> Called MCP tool list_directory ✓

> Called MCP tool list_directory ✓

프로젝트의 주요 구성 요소들을 더 살펴보겠습니다.

> Called MCP tool read_file ✓

> Called MCP tool read_file ✓

> Called MCP tool read_file ✓

> Called MCP tool list_directory ✓

> Called MCP tool list_directory ✓

> Called MCP tool list_directory ✓

이제 핵심 파일들의 내용을 확인해보겠습니다.

> Called MCP tool read_file ✓

> Called MCP tool read_file ✓

> Called MCP tool read_file ✓

> Called MCP tool read_file ✓

> Called MCP tool read_file ✓

> Called MCP tool read_file ✓

이제 프로젝트 분석이 완료되었으니 작업을 구체적인 하위 작업들로 분할

> Called MCP tool split_tasks ✓

연백합니다! Python Nornir+pyeapi 프로젝트를 Golang으로 리팩토링
개의 구체적인 하위 작업으로 성공적으로 분할했습니다.

작업 개요

총 10개의 작업이 생성되었으며, 각 작업은 다음과 같은 특징을 가집니다:

작업 의존성 구조

text

1. Go 프로젝트 초기 구조 설정

- 2. 설정 파일 로더 구현
- 3. GORM 데이터베이스 연결 및 동적 모델 구현
- 4. goeapi 연결 관리자 구현
- 5. 병렬 실행 엔진 구현
- 9. 결과 출력 포맷터 구현
 - 6. Version 정보 수집 (의존: 3,4)
 - 7. Interface 정보 수집 (의존: 3,4)
 - 8. Interface 설정 변경 (의존: 4)
 - 10. 메인 애플리케이션 구현 (의존: 3,4,5,6,7,8,9)

주요 기술 스택 변환

- Python Nornir → Go goroutine + Worker Pool
- Python pyeapi → Go goeapi
- Python SQLAlchemy → Go GORM + SQLite
- Python Rich → Go fatih/color + 테이블 포맷터
- Python YAML → Go gopkg.in/yaml.v3

예상 프로젝트 구조

text

nornir_w_eapi_go/

- main.go
- go.mod
- go.sum
- config/ (기존 YAML 파일 재사용)
- internal/
 - config/
 - inventory/
 - tasks/
 - connection/
 - database/
 - display/
 - executor/
- pkg/
- utils/
- cmd/

첫 번째 작업 완료 요약

작업 목표 및 주요 성과

"Go 프로젝트 초기 구조 및 의존성 설정" 작업이 성공적으로 완료되었습니다.

구현된 솔루션의 핵심 사항

1. 새 Go 프로젝트 생성

- 디렉터리: /Users/dongju/go/src/nornir_w_eapi_go
- Go 모듈 초기화: nornir_w_eapi_go

2. 핵심 의존성 패키지 추가

- github.com/aristanetworks/goeapi v1.0.0 - Arista eAPI 클라이언트
- gorm.io/gorm v1.30.0 - ORM 프레임워크
- gorm.io/driver/sqlite v1.6.0 - SQLite 드라이버
- gopkg.in/yaml.v3 v3.0.1 - YAML 파서
- github.com/fatih/color v1.18.0 - 터미널 색상 출력

3. 프로젝트 디렉터리 구조 생성

text

Apply to hosts.yaml

nornir_w_eapi_go/

- cmd/
- config/
- internal/
 - config/
 - inventory/
 - tasks/
 - connection/
 - database/
 - display/
 - executor/
- pkg/
- utils/

4. 기존 설정 파일 재사용

- Python 프로젝트의 YAML 설정 파일들을 새 프로젝트로 복사
- config.yaml, hosts.yaml, groups.yaml 호환성 유지

주요 도전과제 및 해결방안

1. 의존성 관리 이슈

- 문제: 초기 go get 명령들이 개별적으로 실행되어 일부 의존성이 제대로 추가되지 않음
- 해결: 모든 의존성을 한 번에 추가하는 go get -u 명령 사용으로 해결

2. Go 모듈 경고

- 문제: go mod tidy 실행 시 "all matched no packages" 경고 발생
- 해결: 아직 Go 소스 파일이 없어서 발생하는 정상적인 경고로 확인

NORNIR_W_EAPI_GO

cmd

- interfaces.go
- root.go
- utils.go
- version.go

config

- config.yaml
- groups.yaml
- hosts.yaml

examples

- config_examples
- database

internal

- config

connection

database

display

executor

inventory

tasks

pkg

- debug_db.go
- go.mod
- go.sum
- main.go
- nornir_result
- nornir_w_eapi
- README.md

[connection:spine2]
host=192.168.1.54
username=admin
password=admin
transport=http
port=80

[INFO] 2025/06/13 21:15:57 Config file created, waiting for goeapi to read it...
[INFO] 2025/06/13 21:15:57 Connecting to leaf2 (connection name: leaf2)
[INFO] 2025/06/13 21:15:58 Successfully connected to leaf2
[INFO] 2025/06/13 21:15:58 Successfully connected to leaf2
[INFO] 2025/06/13 21:15:58 Creating connection to leaf1
[INFO] 2025/06/13 21:15:58 Connecting to leaf1 (connection name: leaf1)
[INFO] 2025/06/13 21:15:58 Successfully connected to leaf1
[INFO] 2025/06/13 21:15:58 Successfully connected to leaf1
[INFO] 2025/06/13 21:15:58 Successfully executed show interfaces on leaf2
[INFO] 2025/06/13 21:15:58 Worker 2 job interfaces-leaf2 completed successfully in 448.464791ms
[INFO] 2025/06/13 21:15:58 Result collected for host leaf2 (success)
[INFO] 2025/06/13 21:15:58 Successfully executed show interfaces on leaf1
[INFO] 2025/06/13 21:15:58 Worker 9 job interfaces-leaf1 completed successfully in 499.520833ms
[INFO] 2025/06/13 21:15:58 Result collected for host leaf1 (success)
[INFO] 2025/06/13 21:15:58 Collected 2 results in 499.824ms

=== Interface Information Results ===

HOST STATUS RESULTS

leaf2 SUCCESS {
 "interfaces": {
 "Ethernet1": {
 "name": "Ethernet1",
 "description": "",
 "interfaceStatus": "connected",
 "lineProtocolStatus": ...

leaf1 SUCCESS {
 "interfaces": {
 "Ethernet1": {
 "name": "Ethernet1",
 "description": "",
 "interfaceStatus": "connected",
 "lineProtocolStatus": ...

=== Summary ===
Total: 2
✓ Success: 2
Success Rate: 100.0%

Completed at: 2025-06-13 21:15:58
Duration: 500.599ms
=== Interface Information Collection Completed ===

Q&A

Thank You!