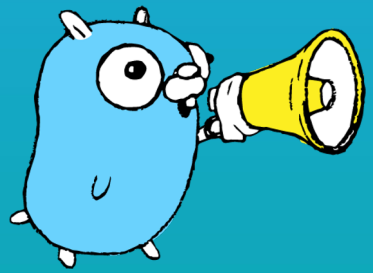
SOUTH KOREA, MAY 28 2020

Go와 Python의 이점을 둘 다 누릴 수 있는 마이크로 서비스 아키텍처

Hyunwoo Lee

Cloudbric

Backend-dev



Today’s (glorious) blather.

SECTION ONE

Requirements

왜 Go와 Python을 동시에 사용했을까?

Requirements

Go

- 가볍고 빨라야한다.
  - Server-side I/O Performance: Node vs PHP vs Java vs Go
  - Go vs Java: Differences and Similarities Compared

Python pandas

- **Database** 서버의 부하를 최소화 해야한다.
- 복잡한 데이터 연산을 빠르고 효율적으로 해야한다.
- 개발 생산성이 좋아야 한다.
  - SQL vs pandas: How to balance tasks between server and client side
  - 쿼리 튜닝 **+** 디비 설정 튜닝 **vs** 기본 설정 **+ pandas**로 유연한 데이터 처리

	slowest (seconds)	fastest (seconds)	median (seconds)
sql1	7.5	6.9	7.0
pandas	6.7	5.8	6.0
sql2	6.6	6.1	6.2
sql3	7.4	6.4	6.6

Fetch using SQL and manipulate data with pandas

PK	DetectionRuleID	vhostName	blackIPv4	countryCode	DetectedCount	WebApplicationFirewallID	calculatedDate
1	1	abc.com	3422294014	KR	3	4d7eb389-4c2b-44ab-9584-c00676552c5f	2020-03-05
2	50	def.com	3422294014	KR	15	3d7eb389-4c2b-44ab-9584-c00676552c5f	2020-03-06
3	1001	ghi.com	660983137	KR	27	2d7eb389-4c2b-44ab-9584-c00676552c5f	2020-03-07


```
"blackIpv4Reputation": [
  {
    "detectedAt": "2020-05-20",
    "country": "China",
    "countryCode": "CN",
    "ipv4": "223.112.190.70",
    "threatScore": 10,
    "attackVector": {
      "interruptingServer": 1,
      "identityTheft": 12
    },
    "detectedWafs": 12,
    "attackedHosts": 12,
    "description": "223.112.190.70 attacked 12 domains with the following intensions: Interrupting Server,Identity Theft. These attackes were detected in a total of 12 Cloudbric WAF worldwide"
  }, ...
```


```
query = f'SELECT {fiedls} FROM {table} WHERE blackIPv4={ipv4}'
df = pd.read_sql(query, alchemy_engine)

ipv4_waf_detected =
df.groupby(['calculatedDate','wafId'].size().reset_index().groupby('calculatedDate').size()

ipv4_vhost_detected =
df.groupby(['calculatedDate','vhost'].size().reset_index().groupby('calculatedDate').size()
# ...

for created in uniq_dates:
    biv4_reputation = reputation.black_ipv4_reputation.add()

    br = biv4_pb2.BlackIPv4Reputation()
    br.detected_at = str(created)
    attack_purpose_set = set()

    if int(ap_map[i][0]) > 0:
        br.attack_vector.vulnerability_scan = int(ap_map[i][0])
        attack_purpose_set.add(_translate_attack_purpose(request.language, attack_purposes[0]))
    # ...
    if int(ap_map[i][5]) > 0:
        br.attack_vector.monetary_loss = int(ap_map[i][5])
        attack_purpose_set.add(_translate_attack_purpose(request.language, attack_purposes[5]))
```

어떤 로우데이터에서 어떤 데이터를 추출해야하는가?

If pandas is slow, maybe that because of you.

```python
# 각 아이피가 어떤 공격 목적으로 몇번 공격을 했는지를 구하고 DataFrame으로 완성하는 로직
stat_pd = pd.DataFrame(columns=['ipv4', 'vulnerability_scan', 'falsifying_websites', 'interrupting_server',
'spreading_malware', 'identity_theft', 'monetary_loss'])

# pandas라이브러리의 최장점 중 하나는 1차원 벡터를 DataFrame의 컬럼으로 그대로 부하없이 삽입할 수 있다는 점입니다.
vulnerability_scan_vector = list()
falsifying_websites_vector = list()
# ...

# len(ipv4_list) == 5,000,000
for idx in range(len(ipv4_list)):
    ipv4 = ipv4_list[idx]
    for attack_purpose, detected_cnt in grp_attack_purpose[ipv4].items():
        if attack_purpose == 'Vulnerability Scan':
            vulnerability_scan_vector.append(int(detected_cnt))
        if len(vulnerability_scan_vector) <= idx:
            vulnerability_scan_vector.append(0)

        if attack_purpose == 'Falsifying Websites':
            falsifying_websites_vector.append(int(detected_cnt))
        if len(falsifying_websites_vector) <= idx:
            falsifying_websites_vector.append(0)
# ...

# 완성된 리스트들을 각각 데이터프레임에 삽입해줍니다.
stat_pd['vulnerability_scan'] = vulnerability_scan_vector
stat_pd['falsifying_websites'] = falsifying_websites_vector
# ...
```

```
> grp_attack_purpose = ildf.groupby('ipv4')['attack_purpose'].value_counts()
> print(grp_attack_purpose)

ipv4          attack_purpose
16794174      Identity Theft         2
16802351      Interrupting Server    1
16810176      Identity Theft         1
              Spreading Malware      1
16810599      Identity Theft         1
                                     ..
3758089786    Falsifying Websites    1
3758090117    Falsifying Websites    1
3758090120    Falsifying Websites    1
3758090131    Falsifying Websites    1
3758090133    Falsifying Websites    1
```

pandas is powerful with numpy

```
# 각 아이피가 어떤 공격 목적으로 몇번 공격을 했는지를 구하고 DataFrame으로 완성하는 로직

attack_purposes = ['Vulnerability Scan', 'Falsifying Websites', 'Interrupting
Server', 'Spreading Malware', 'Identity Theft', 'Monetary Loss']
r_len = len(ipv4_list)
c_len = len(attack_purposes)
attack_purposes_map = np.zeros((r_len, c_len))

before = None
start = True
row_idx = 0
for tup, detected_count in grp_attack_purpose.items():
    ipv4 = tup[0]
    attack_purpose = tup[1]
    if ipv4 != before:
        row_idx += 1
    if start:
        start = False
        row_idx = 0
        col_idx = attack_purposes.index(attack_purpose)
        ap_map[row_idx, col_idx] = detected_count

        before = ipv4

# 다차원 배열의 각 컬럼(공격 목적을 의미)을 그대로 데이터프레임에 삽입해줍니다.
stat_pd['vulnerability_scan'] = attack_purposes_map[:,0]
stat_pd['falsifying_websites'] = attack_purposes_map[:,1]
# ...
# ... stat_pd.to_sql()
```

```
grp_attack_purpose =
ildf.groupby('ipv4')['attack_purpose'].value_counts()
print(grp_attack_purpose)
```

```
ipv4          attack_purpose
16794174      Identity Theft         2
16802351      Interrupting Server    1
16810176      Identity Theft         1
              Spreading Malware      1
16810599      Identity Theft         1
                                     ..
3758089786    Falsifying Websites    1
3758090117    Falsifying Websites    1
3758090120    Falsifying Websites    1
3758090131    Falsifying Websites    1
3758090133    Falsifying Websites    1
```

2d_numpy_array_with_attack_purposes.csv

Raw

Search this file...

	No	Vulnerability Scan	Falsifying Websites	Interrupting Server	Spreading Malware	Identity Theft	Monetary Loss
2	0	0	0	0	0	1	0
3	1	0	0	1	0	0	0
4	2	0	0	0	1	1	0
5	3	0	1	1	0	1	0
6	4	0	0	8	0	9	0
7	5	0	20	0	1	30	0
8	6	0	190	0	7	18	0
9	7	0	0	1	5	50	0
10	8	0	0	0	1	1	100

numpy와 함께 사용해서 문제 해결

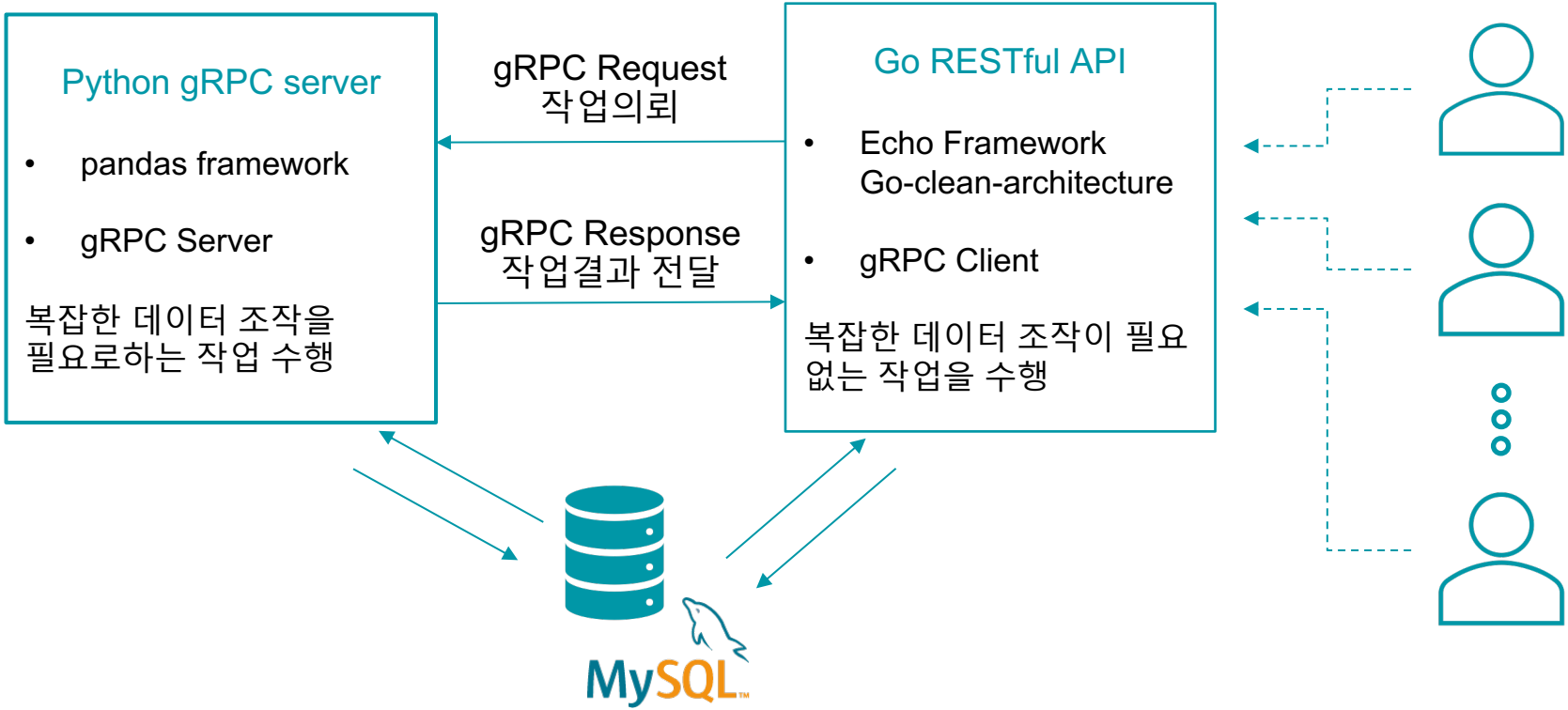
SECTION TWO

Architecture

Go:: RESTful API / gRPC Client

Python:: gRPC server

Architecture



Architecture Summary

Go RESTful API Server

- 쿼리만으로 충분히 빠르게 처리 가능한 작업을 수행
- 복잡한 데이터 연산을 요구하는 액션이면, Python gRPC서버에게 작업을 요청 (gRPC Request)
  응답을 받으면 JSON 포맷으로 클라이언트에게 리턴

Python gRPC Server

- 쿼리만으로 처리하기 어려운 복잡한 데이터 연산이 필요한 작업을 Go 서버로부터
  의뢰 받으면 pandas 프레임워크로 처리 후 결과 전달 (gRPC Response)
  E.g., extracting statistics based on tons of data.

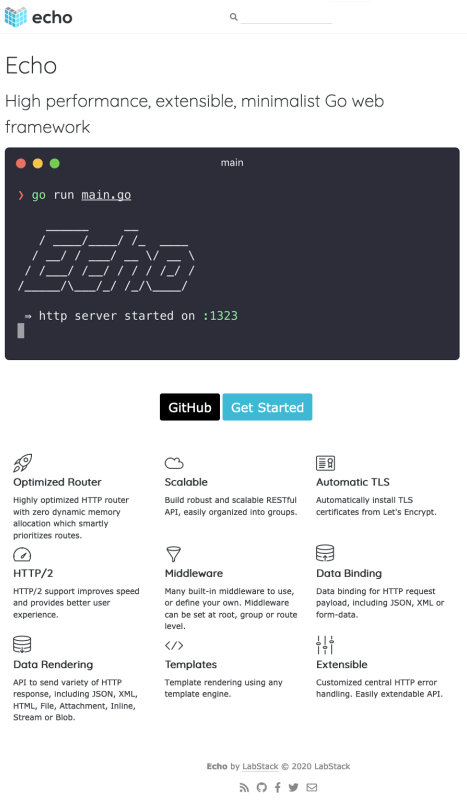
SECTION THREE

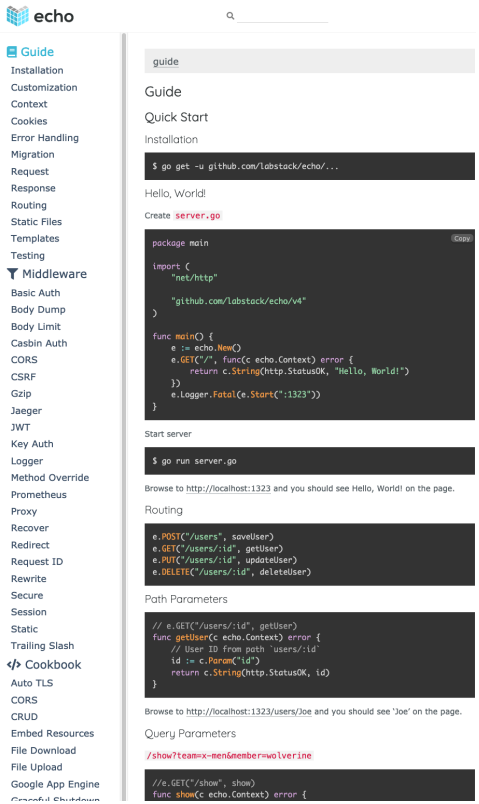
Implementation

Go:: RESTful API / gRPC Client

Python:: gRPC server

Go Implementation: Echo Framework





문서화가 잘 되어 있어 쉽고 빠르게 사용 가능한 웹 프레임워크

What about Directory Structure...?

app

The `app` directory is where all of your application code lives. This comes with a default directory structure that works well for many applications. The following folders make up the basic contents:

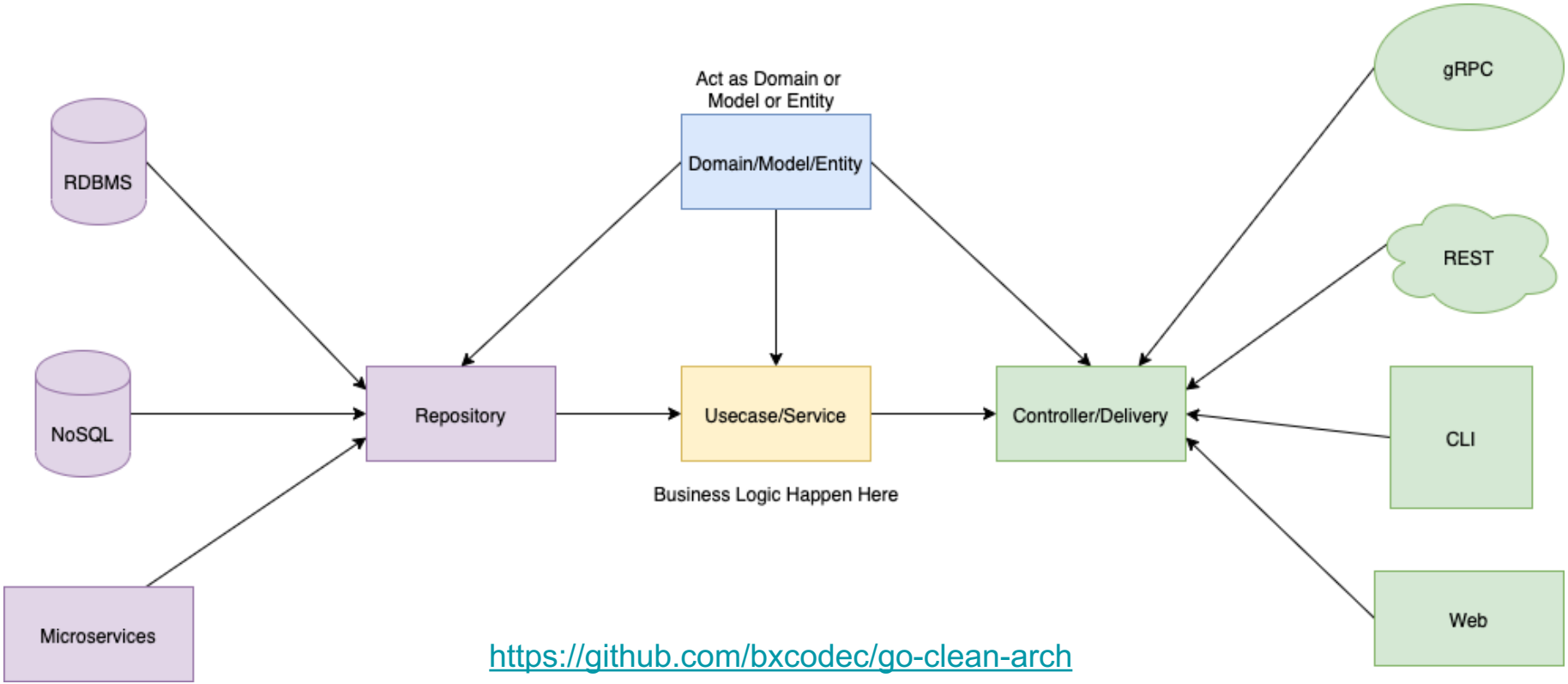
```
/app
        /Config         Stores the configuration files
        /Controllers    Controllers determine the program flow
        /Database       Stores the database migrations and seeds files
        /Filters        Stores filter classes that can run before and after controller
        /Helpers        Helpers store collections of standalone functions
        /Language       Multiple language support reads the language strings from here
        /Libraries      Useful classes that don't fit in another category
        /Models         Models work with the database to represent the business entities.
        /ThirdParty     ThirdParty libraries that can be used in application
        /Views          Views make up the HTML that is displayed to the client.
```

Because the `app` directory is already namespaced, you should feel free to modify the structure of this directory to suit your application's needs. For example, you might decide to start using the Repository pattern and Entity Models to work with your data. In this case, you could rename the `Models` directory to `Repositories`, and add a new `Entities` directory.

Note

If you rename the `Controllers` directory, though, you will not be able to use the automatic method of routing to controllers, and will need to define all of your routes in the routes file.

Go Implementation: Clean Architecture



https://github.com/bxcodec/go-clean-arch

Clean Architecture: Overview

Models(Domain) Layer: 데이터 모델을 정의하는 레이어

어떤 데이터들을 다룰 건가요?

서비스에서 다루는 데이터, 모델을 정의

Repository Layer: 데이터 베이스 또는 마이크로 서비스와 직접 통신하는 레이어

데이터는 어디서 받아 오실 건가요?

SQL, NoSQL, Microservice 등과 직접 상호작용하며 데이터 관련 연산을 진행

Usecase Layer: 비즈니스 로직을 담당하는 레이어

클라이언트에게 전달되는 데이터는 어떤 형태인가요?

Repository에 정의한 연산들을 사용하여 클라이언트에게 전달하고 싶은 데이터를 추출

Delivery Layer: 데이터를 클라이언트에게 전달하는 레이어

클라이언트는 어떤 매개체로 데이터를 전달 받나요?

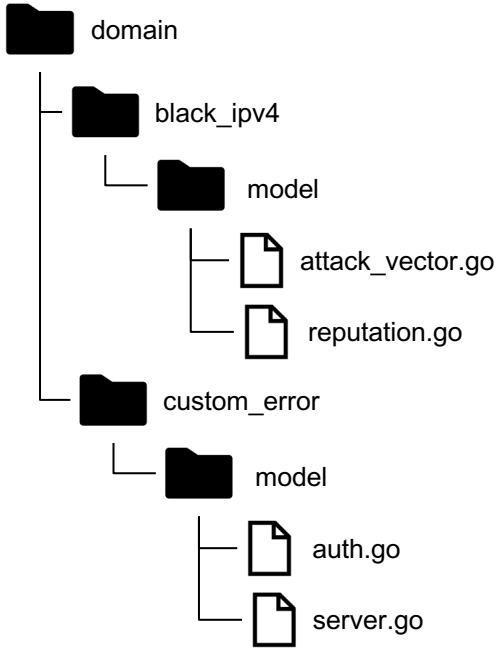
REST, CLI, WEB 등 다양한 매개체로 데이터를 클라이언트에게 전달

Clean Architecture: Models / Domain

```go
// Reputation contains meaningful statistics of black ipv4 during specific
period
type BlackIPv4Reputation struct {
    DetectedAt    string       `json:"detectedAt"`
    Country       string       `json:"country"`
    CountryCode   string       `json:"countryCode"`
    IPv4          string       `json:"ipv4"`
    ThreatScore   uint32       `json:"threatScore"`
    AttackVector  AttackVector `json:"attackVector"`
    DetectedWafs  uint32       `json:"detectedWafs"`
    AttackedHosts uint32       `json:"attackedHosts"`
    Description   string       `json:"description"`
}


type AttackVector struct {
    VulnerabilityScan  uint32 `json:"vulnerabilityScan,omitempty"`
    FalsifyingWebsites uint32 `json:"falsifyingWebsites,omitempty"`
    // ...
}

func (av *AttackVector) TranslateField(lang string, field string) (str string) {
    avMap := map[string]map[string]string{
        "VulnerabilityScan":  map[string]string{},
        "FalsifyingWebsites": map[string]string{},
        // ...
    }
    avMap["VulnerabilityScan"]["kr"] = "Vulnerability Scan"
    // ...
    avMap["VulnerabilityScan"]["en"] = "Vulnerability Scan"
    // ...
    avMap["VulnerabilityScan"]["jp"] = "脆弱性スキャン"
    // ...
    return avMap[field][lang]
}
```



서비스에서 다루고 싶은 데이터 모델을 정의

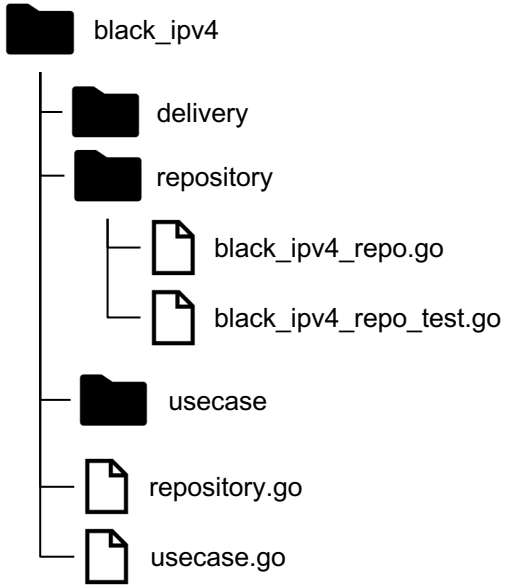
Clean Architecture: Repository

```
type Repository interface {
   GetReputationByIPv4(c context.Context, rr *model.ReputationRequest) (*pb.Reputation, error)
   GetReportByPeriod(c context.Context, rr *model.ReportRequest) (*model.Report, error)
   // ...
}
```

```
type BlackIPv4Repository struct {
   ClientConn   *grpc.ClientConn
   DatabaseConn *sql.DB
   Logger       *logrus.Logger
}

func NewBlackIPv4Repostiory(c *grpc.ClientConn, d *sql.DB, l *logrus.Logger) black_ipv4.Repository {
   return &BlackIPv4Repository{ClientConn: c, DatabaseConn: d, Logger: l}
}

func (r *BlackIPv4Repository) GetReputationByIPv4(c context.Context, rr
*model.ReputationRequest) (reputation *pb.Reputation, err error) {
   cli := pb.NewBlackIPv4ServiceClient(r.ClientConn)
   l := r.Logger
   reputation, err = cli.GetReputationByIPv4(
      c,
      &pb.ReputationRequest{
         StartDate: rr.StartDate, Ipv4: rr.IPv4, Descending: rr.Descending, Limit: rr.Limit,
Language: rr.Language,
      },
   )
   if err != nil {
      l.Error(err)
   }
   return reputation, nil
}
```



데이터의 출처(DB, Microservice, rpcServer, etc...)와 소통하는 연산들을 정의

Clean Architecture: Usecase

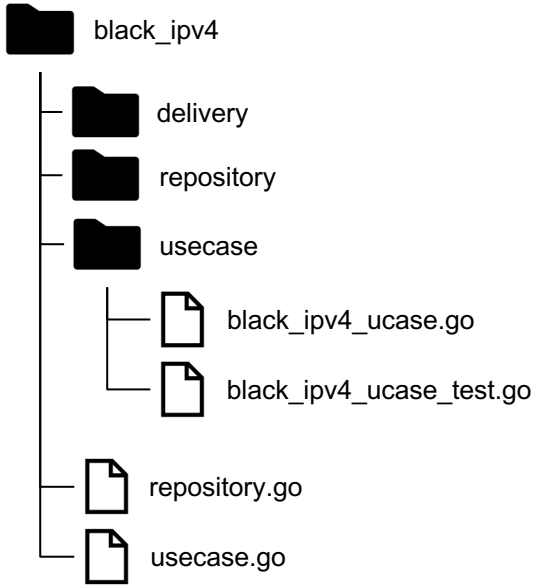
```go
type Usecase interface {
   GetReputationByIPv4(c context.Context, rr *model.ReputationRequest) (*pb.Reputation, error)
   GetReportByPeriod(c context.Context, rr *model.ReportRequest) (*model.Report, error)
   // ...
}
```

```go
// NewBlackIPv4Usecase will create a new BlackIPv4Usecase object representation
of blackIPv4.Uscase interface
func NewBlackIPv4Usecase(r black_ipv4.Repository, t time.Duration, l
*logrus.Logger) black_ipv4.Usecase {
   return &BlackIPv4Usecase{
      Repo:           r,
      ContextTimeout: t,
      Logger:         l,
   }
}


func (u *BlackIPv4Usecase) GetReputationByIPv4(c context.Context, rr
*model.ReputationRequest) (*pb.Reputation, error) {
   c, cancel := context.WithTimeout(c, u.ContextTimeout)
   defer cancel()

   reputation, err := u.Repo.GetReputationByIPv4(c, rr)
   if err != nil {
      return nil, err
   }
   // Business Logic ...
   // Add some fields for our client ...

   return reputation, nil
}
```



Repository에 정의된 연산들을 사용해서 Business Logic을 정의

Clean Architecture: Delivery / http

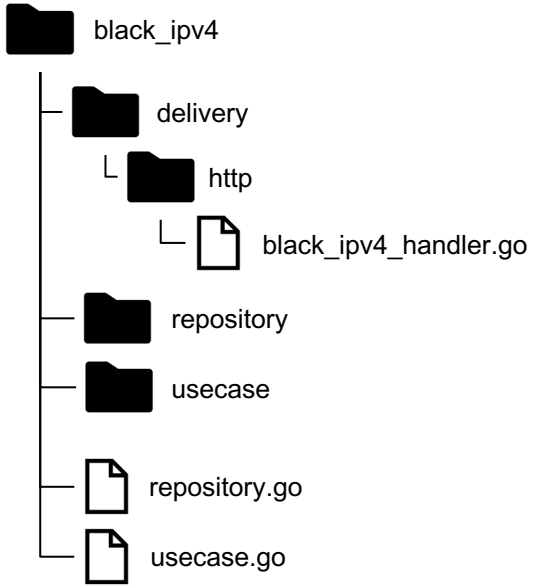
```go
func (h *BlackIPv4Handler) GetReputationByIPv4(c echo.Context) error {
    u, p, ok := c.Request().BasicAuth()
    rr := new(model.ReputationRequest) // 클라이언트의 요청정보
    if err := c.Bind(rr); // For simplicity, error handling was excluded.

    lang := c.Request().Header.Get("Accept-Language")
    supported := getSupportedLang()
    /* change language info of request */

    err := h.Validate.Struct(rr) // 클라이언트의 요청정보 검증
    k, err := h.AuthRepo.VerifyApiKey(ctx, u, p) // API Key 검증
    /* check quota of Api Key and etc... */

    reputation, err := h.Usecase.GetReputationByIPv4(ctx, rr)
    err = h.AuthRepo.DeductFromQuota(ctx, k, reputation.Returned)

    jm := jsonpb.Marshaler{} // turn field name with snake_case into camelCase
    j, _ := jm.MarshalToString(reputation)
    c.Response().Header().Set(echo.HeaderContentType, echo.MIMEApplicationJSON)
    c.Response().Header().Set("Content-Language", rr.Language)
    return c.String(http.StatusOK, reputation)
}
```



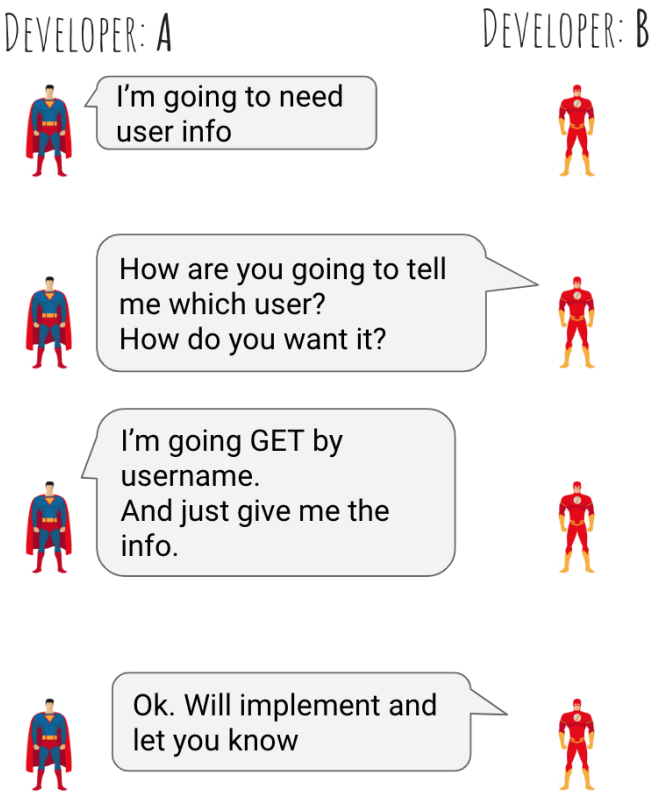
클라이언트에게 어떻게 전달할지를 정의 (http, cli, etc...)

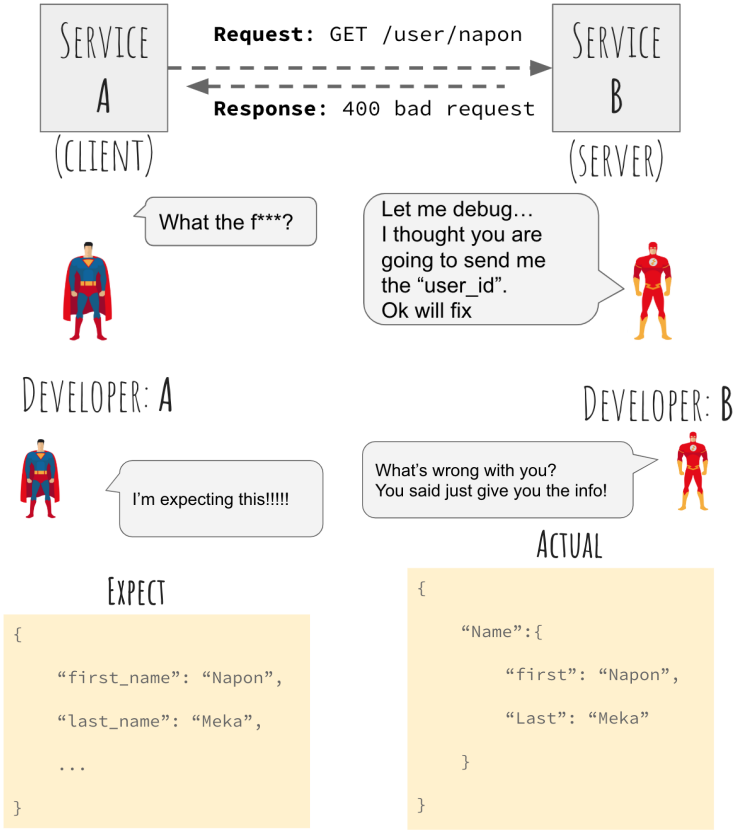
Using an IDL such as Protocol Buffers ("Protobuf") provides numerous benefits over JSON:

- Generated stubs for each language you use.
- Forwards and backwards compatibility for your data types.
- Payload sizes are up to 10 times smaller.
- Serialization speed is up to 100 times faster.
- Structured RPCs for your APIs instead of documented HTTP endpoints.

reference: https://github.com/bufbuild/buf

gRPC를 사용하지 않는다면?





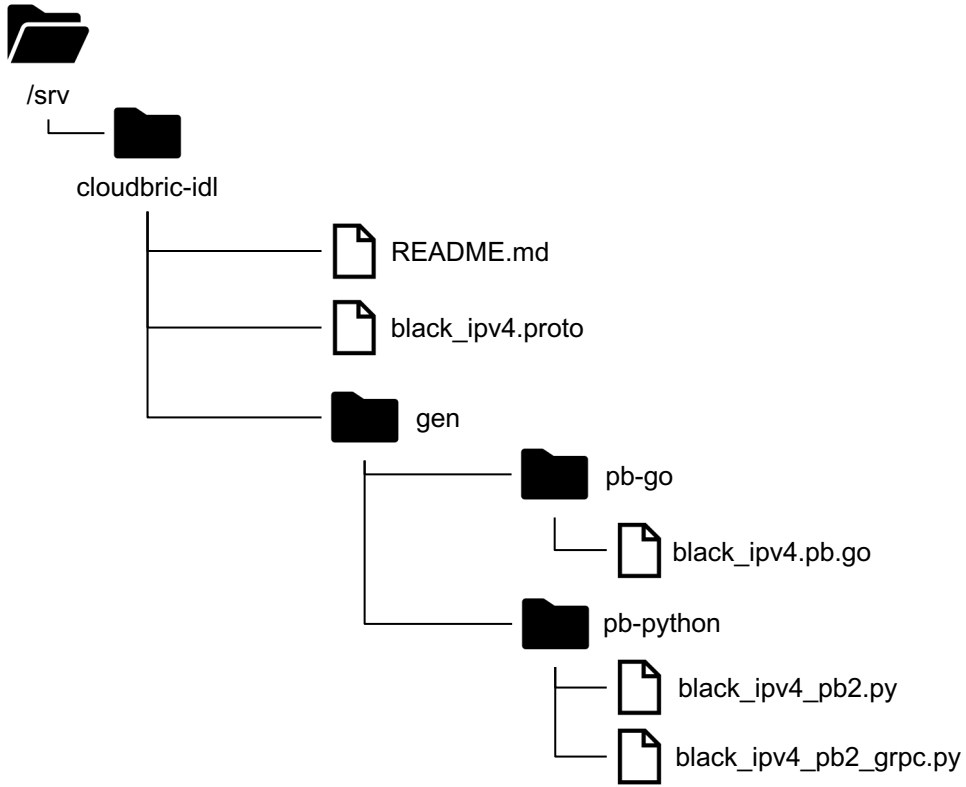
출처: gRPC and why it can save you development time

gRPC / .proto

```
syntax = "proto3";
package black_ipv4;
service BlackIPv4Service {
    rpc GetReputationByIPv4(ReputationRequest) returns (Reputation);
}
message AttackVector {
    uint32 vulnerability_scan = 1;
    // ...
    uint32 monetary_loss = 6;
}
message BlackIPv4Reputation {
    string detected_at = 1;
    // ...
    string description = 6;
}
message ReputationRequest {
    string start_date = 1;
    // ...
    string language = 5;
}
message Reputation {
    ReputationRequest request = 1;
    // ...
    repeated BlackIPv4Reputation black_ipv4_reputation = 5;
}
```

커뮤니케이션을 주고받을 데이터 타입, 그리고 방법론을 정의

gRPC / IDL 관리 방법



Reference

gRPC/Protocol Buffer Compiler Containers

프로덕션 환경에서 사용하는 golang과 gRPC

Python Implementation: gRPC server

```
import black_ipv4_pb2 as biv4_pb2
import black_ipv4_pb2_grpc as biv4_pb2_grpc

class BlackIPv4Servicer(biv4_pb2_grpc.BlackIPv4ServiceServicer):
    def __init__(self):
        self.cnxpool = repository.get_connection_pool()

    def GetReputationByIPv4(self, request, context):
        try:
            cnx = self.cnxpool.get_connection()
            reputation = repository.get_reputation_by_ipv4(cnx, request)
            if reputation != None:
                logging.debug('GetBlackIPv4Reputation is done')
                return reputation
        except Exception as e:
            logging.error(e)

def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    biv4_pb2_grpc.add_BlackIPv4ServiceServicer_to_server(
        BlackIPv4Servicer(), server)
    server.add_insecure_port('[::]:40404')
    server.start()
    server.wait_for_termination()

if __name__ == '__main__':
    logging.basicConfig(filename='rpc.log', format='[%(levelname)s] %(asctime)s | %(message)s', level=logging.DEBUG, datefmt='%m/%d/%Y
%I:%M:%S')
    serve()
```

Python Implementation: gRPC server

```
import black_ipv4_pb2 as biv4_pb2
import black_ipv4_pb2_grpc as biv4_pb2_grpc

def get_reputation_by_ipv4(cnx, request):
    try:
        ildf = pd.read_sql(query, cnx)
        cnx.close()
    except Exception as e:
        raise e

    reputation = biv4_pb2.Reputation()
    # business logic ...
    wapples_detected = ildf.groupby(['created', 'waf_id']).size().reset_index().groupby('created').size()
    attacked_vhosts = ildf.groupby(['created', 'vhost']).size().reset_index().groupby('created').size()
    grp_attack_purpose = ildf.groupby('created')['attack_vector'].value_counts()
    # pandas work...
    for created in uniq_dates:
        biv4_reputation = reputation.black_ipv4_reputation.add()
        br = biv4_pb2.BlackIPv4Reputation()
        br.detected_at = str(created)
        # Some Logic...
        br.detected_wafs = num_detected_wafs
        br.attacked_hosts = num_attacked_vhosts
        br.threat_score = _calculate_threat_score(num_detected_wafs, num_attacked_vhosts)
        br.description = _create_desc(request.language, ipv4_str, num_attacked_vhosts, attack_purpose_str, num_detected_wafs)
        biv4_reputation.CopyFrom(br)
return reputation
```

pandas 프레임워크로 원하는 데이터를 추출하여 데이터 생성 및 전달

SECTION FOUR

Summary

각자 잘하는 걸 살려서 아키텍처를 설계

SECTION FIVE

Q & A

END

감사합니다.