



golang으로 multi account/project에 있는 aws/gcp 리소스 메타데이터를 효율적으로 다뤄볼 수 있는 방법

김효민

당근마켓 SRE

Derek



Google Developer Groups

Cloud • Golang korea



1. 만들고 싶은 기능, 사용하려는 클라우드 서비스
2. aws multi account와 gcp multi project 환경에서의
인증과 권한
3. GCP cloud asset api vs GCP other services' api
4. 동시성 프로그래밍을 통해 gcp asset inventory
대체해보기

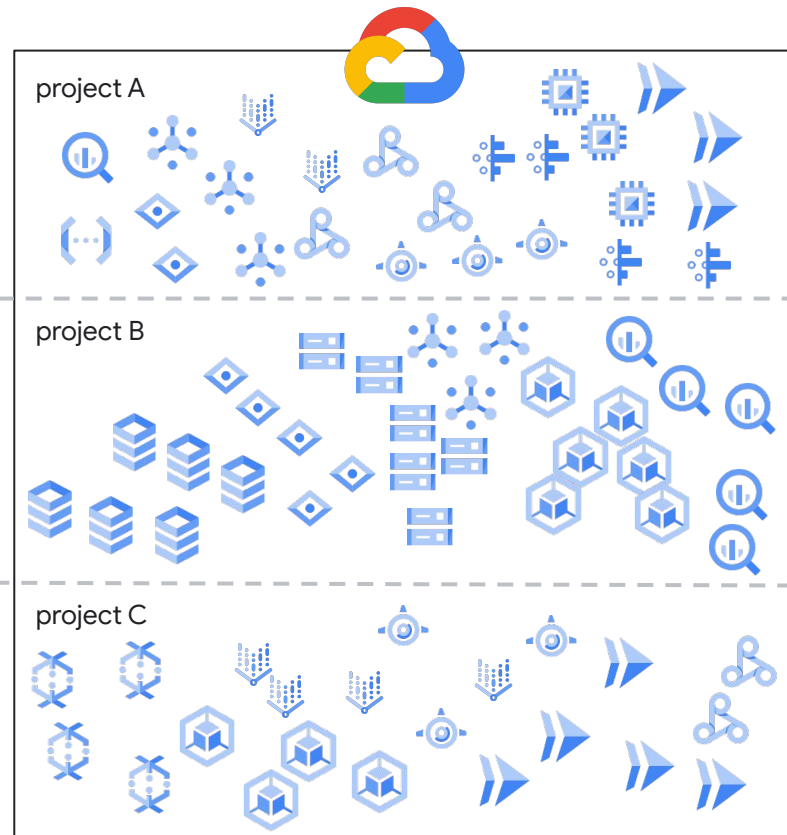


SECTION 1

만들고 싶은 기능,
사용하려는 클라우드 서비스



특정 태그(AWS) 라벨(GCP)을 기준으로 복잡한 리소스들을 정리



각 서비스별로 사용하고 있는 클라우드 리소스 정리(환경)

dev

[AWS]

- redis
- memcached
- dynamodb

[GCP]

- cloud function
- gcs
- bigquery

stage

[AWS]

- redis
- memcached
- dynamodb

[GCP]

- cloud function
- gcs
- bigquery

prod

[AWS]

- redis
- memcached
- dynamodb

[GCP]

- cloud function
- gcs
- bigquery



각 서비스별로 사용하고 있는 클라우드 리소스 정리(오너쉽)

결제 서비스

[AWS]

- redis
- memcached
- dynamodb

[GCP]

- bigquery

배송 서비스

[AWS]

- redis
- msk
- rds

[GCP]

- cloud function
- places api
- gcs

인증 서비스

[AWS]

- memcached
- sqs
- rds

[GCP]

- bigquery

추천 서비스

[AWS]

- s3
- lambda

[GCP]

- pub/sub
- dataflow
- vertex AI



AWS Tag Editor ([resourcegroupstaggingapi](#))

Resource search results (20)

Choose up to 500 resources for which you want to edit tags.

Export 20 resources to CSV

Manage tags of selected resources

Q


Filter resources

< 1 > ⚙


<input type="checkbox"/>	Identifier	Tag: Name	Service	Type	Region	Tags
<input type="checkbox"/>	dev-lambda-demo-with-another-role	(not tagged)	Lambda	Function	ap-northeast-2	-
<input type="checkbox"/>	dev-lambda-demo	(not tagged)	Lambda	Function	ap-northeast-2	-
<input type="checkbox"/>	i-03608b008e875084b	client	EC2	Instance	ap-northeast-2	1
<input type="checkbox"/>	vpc-0185d0bc577ecd22	(not tagged)	EC2	VPC	ap-northeast-2	-
<input type="checkbox"/>	vpc-0eff1536039f66207	(not tagged)	EC2	VPC	ap-northeast-2	3
<input type="checkbox"/>	vpc-0c42812dfd0cab852	(not tagged)	EC2	VPC	ap-northeast-1	-
<input type="checkbox"/>	vpc-073f5324cd4d5c54d	(not tagged)	EC2	VPC	ap-southeast-1	-
<input type="checkbox"/>	vpc-0a075f9fd2ecd7e3c	(not tagged)	EC2	VPC	ap-south-1	-
<input type="checkbox"/>	vpc-0bbbca8ea1217a24f	(not tagged)	EC2	VPC	ap-southeast-2	-
<input type="checkbox"/>	vpc-0874cf19d7ac7202c	(not tagged)	EC2	VPC	us-west-1	-
<input type="checkbox"/>	vpc-076cdfc9f59eccb8f	(not tagged)	EC2	VPC	us-west-2	-
<input type="checkbox"/>	vpc-03c52c8b00da67f77	(not tagged)	EC2	VPC	us-east-2	-
<input type="checkbox"/>	vpc-0eec08fbfb1c630c4	(not tagged)	EC2	VPC	ca-central-1	-



GCP Asset Inventory (cloud asset)


Asset Inventory 

OVERVIEW **RESOURCE** IAM POLICY

Filter results [CLEAR ALL](#)  Results 1-24 of 24 [DOWNLOAD CSV](#)

Resource type

- ☒ serviceusage.Service 20
- ☐ iam.ServiceAccountKey 4
- ☐ iam.ServiceAccount 3
- ☒ storage.Bucket 3
- ☐ logging.LogBucket 2
- ☐ logging.LogSink 2
- ☒ bigquery.Dataset 1
- ☐ cloudbilling.ProjectBillingInfo 1
- ☐ cloudresourcemanager.Project 1

 **Filter** Example: myInstanc* compute@developer.gserviceaccount.com

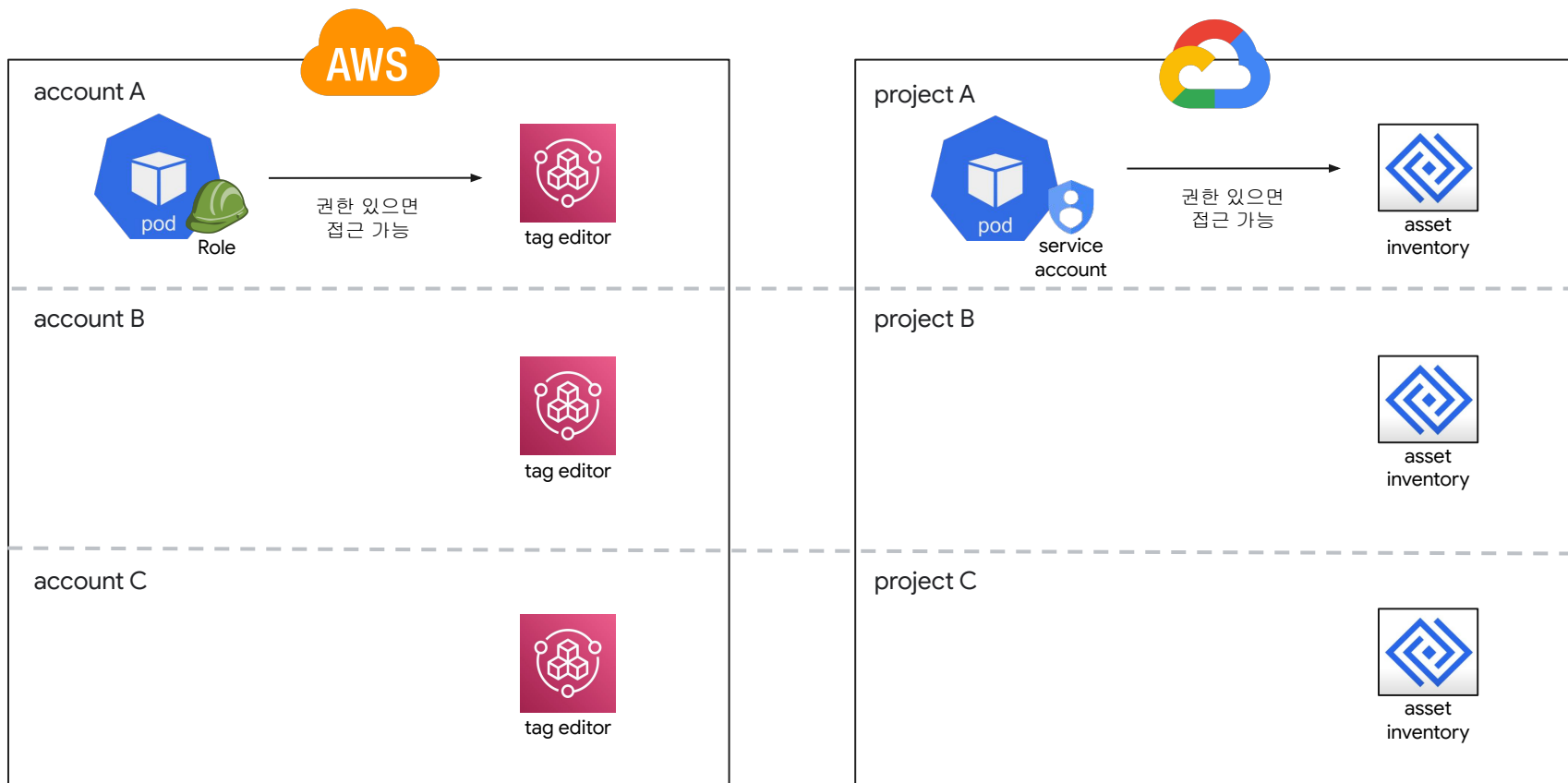
Display name ↓	Resource type	Location	Labels	KMS keys	Network tags
test-iam-per-bucket	storage.Bucket	asia-northeast3			
test-gcs-devfest	storage.Bucket	us			
sts.googleapis.com	serviceusage.Service	global			
storage.googleapis.com	serviceusage.Service	global			
storage-component.googleapis.com	serviceusage.Service	global			
storage-api.googleapis.com	serviceusage.Service	global			
sql-component.googleapis.com	serviceusage.Service	global			
serviceusage.googleapis.com	serviceusage.Service	global			
servicemanagement.googleapis.com	serviceusage.Service	global			
monitoring.googleapis.com	serviceusage.Service	global			
logging.googleapis.com	serviceusage.Service	global			



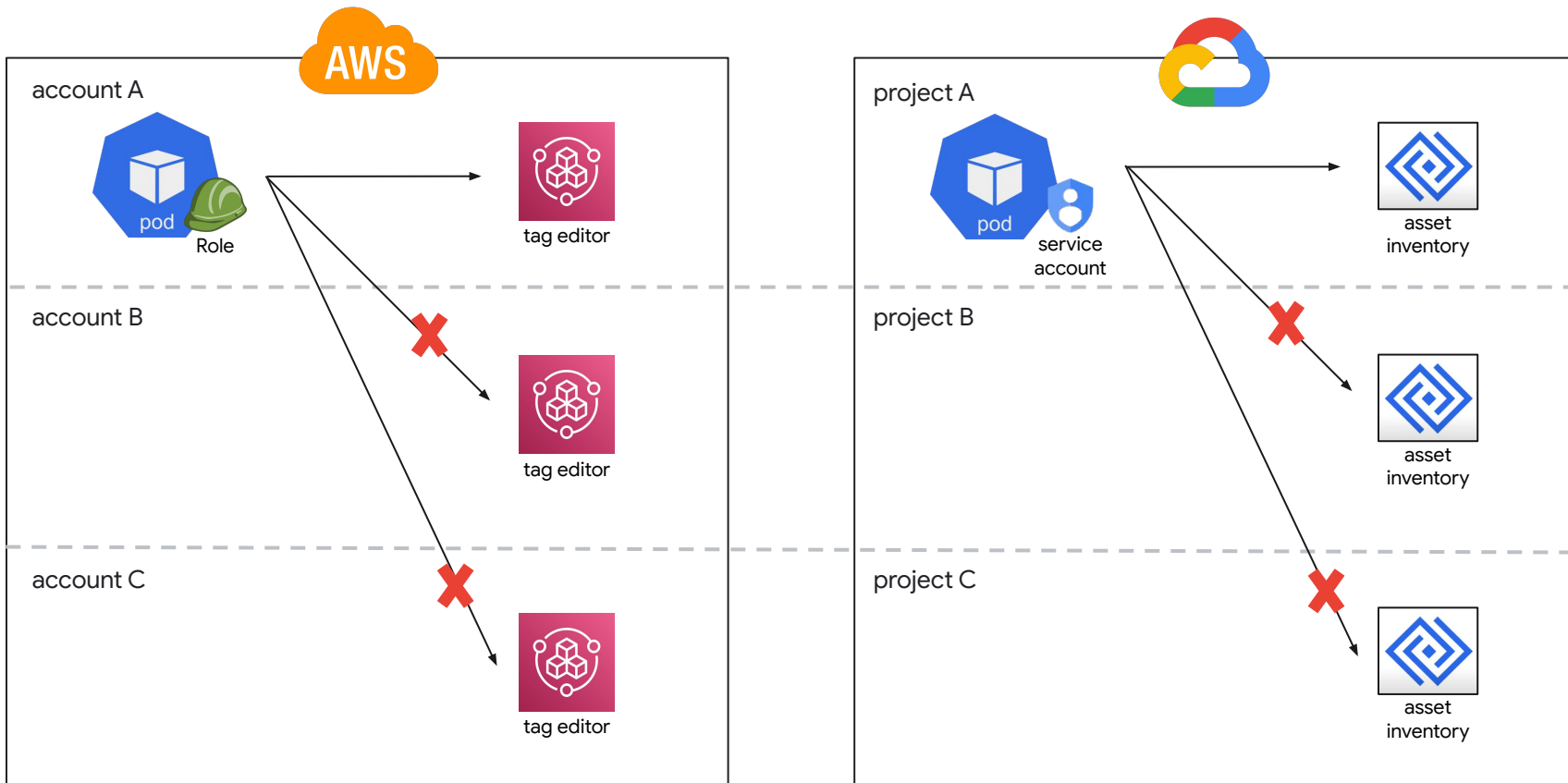
aws multi account와 gcp multi project 환경에서의 인증과 권한



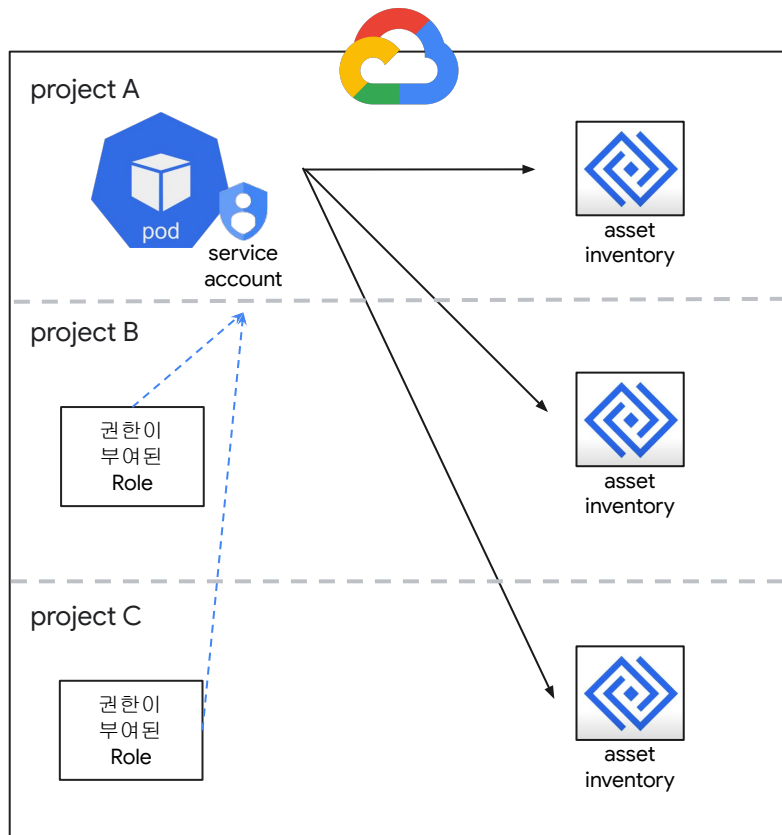
같은 account/project에 있는 api를 사용해야 하는 경우



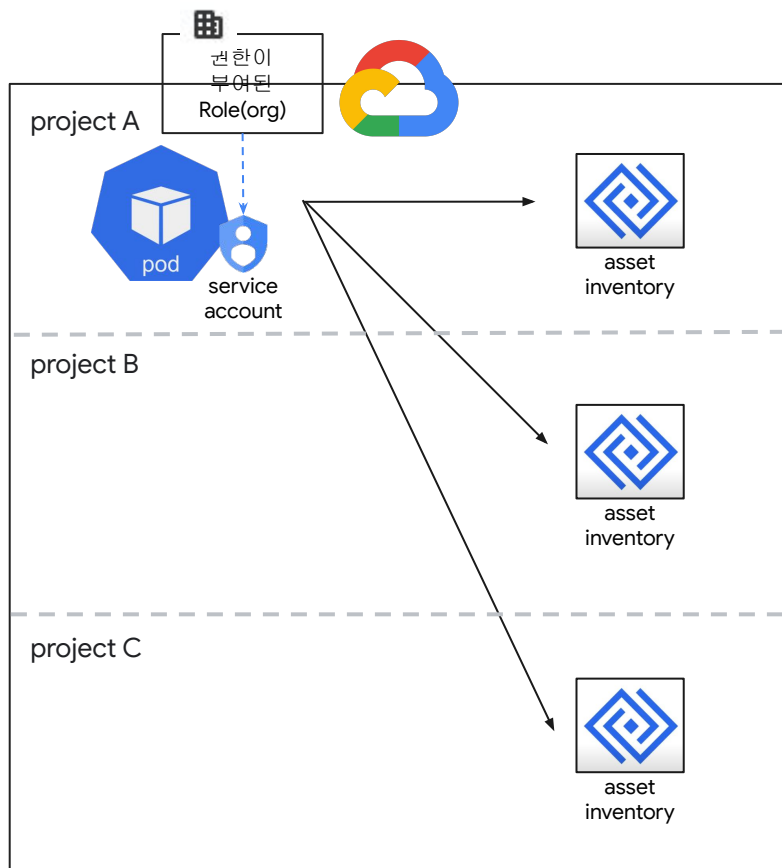
다른 account/project에 있는 api를 사용해야 하는 경우



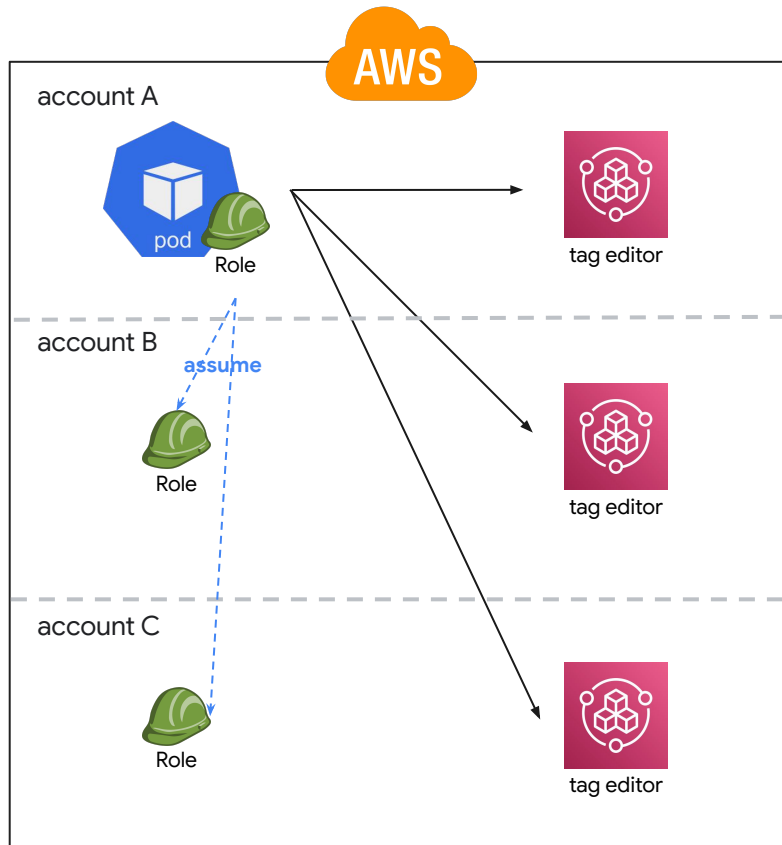
multi projects에 있는 api를 사용해야 하는 경우 (GCP)



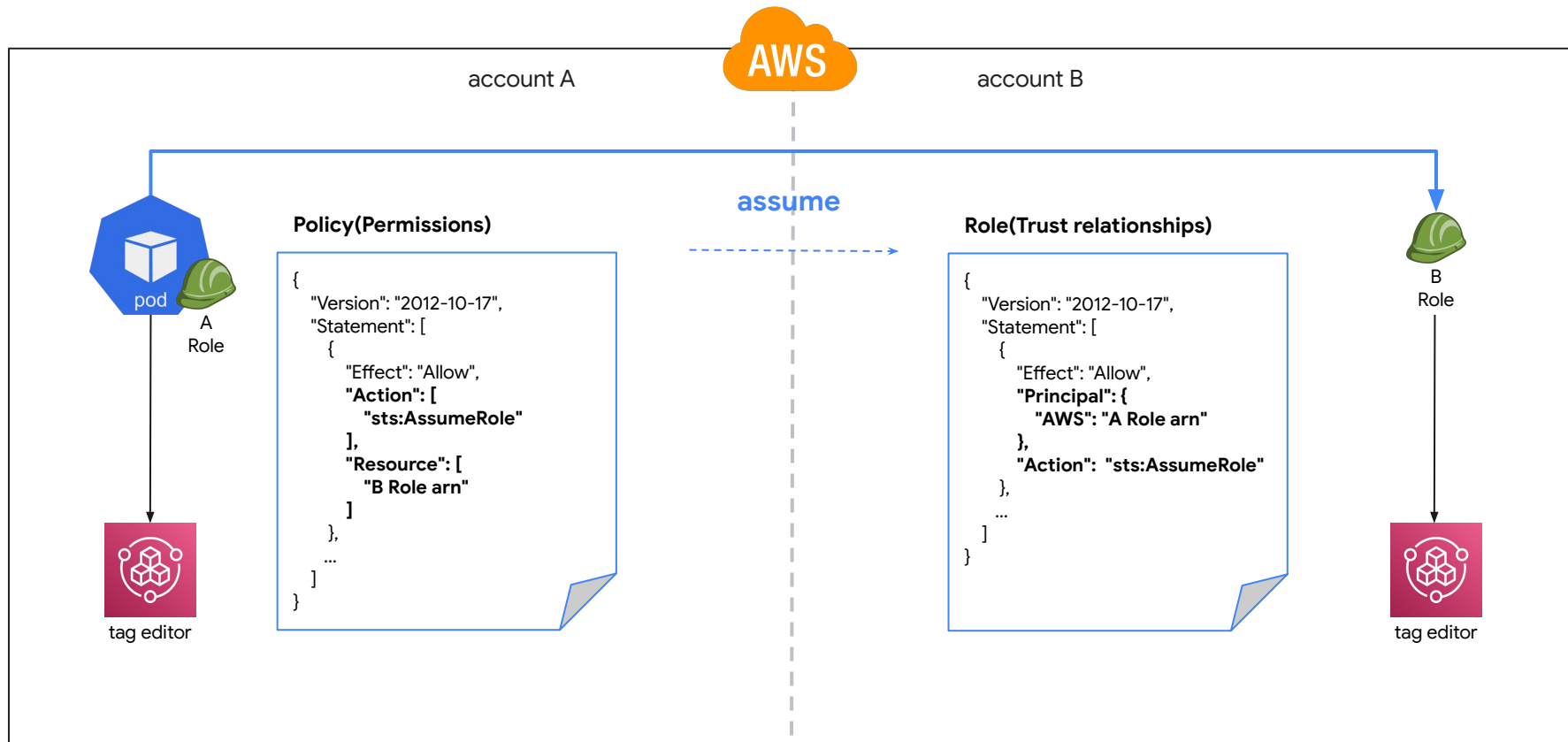
multi projects에 있는 api를 사용해야 하는 경우 (GCP)



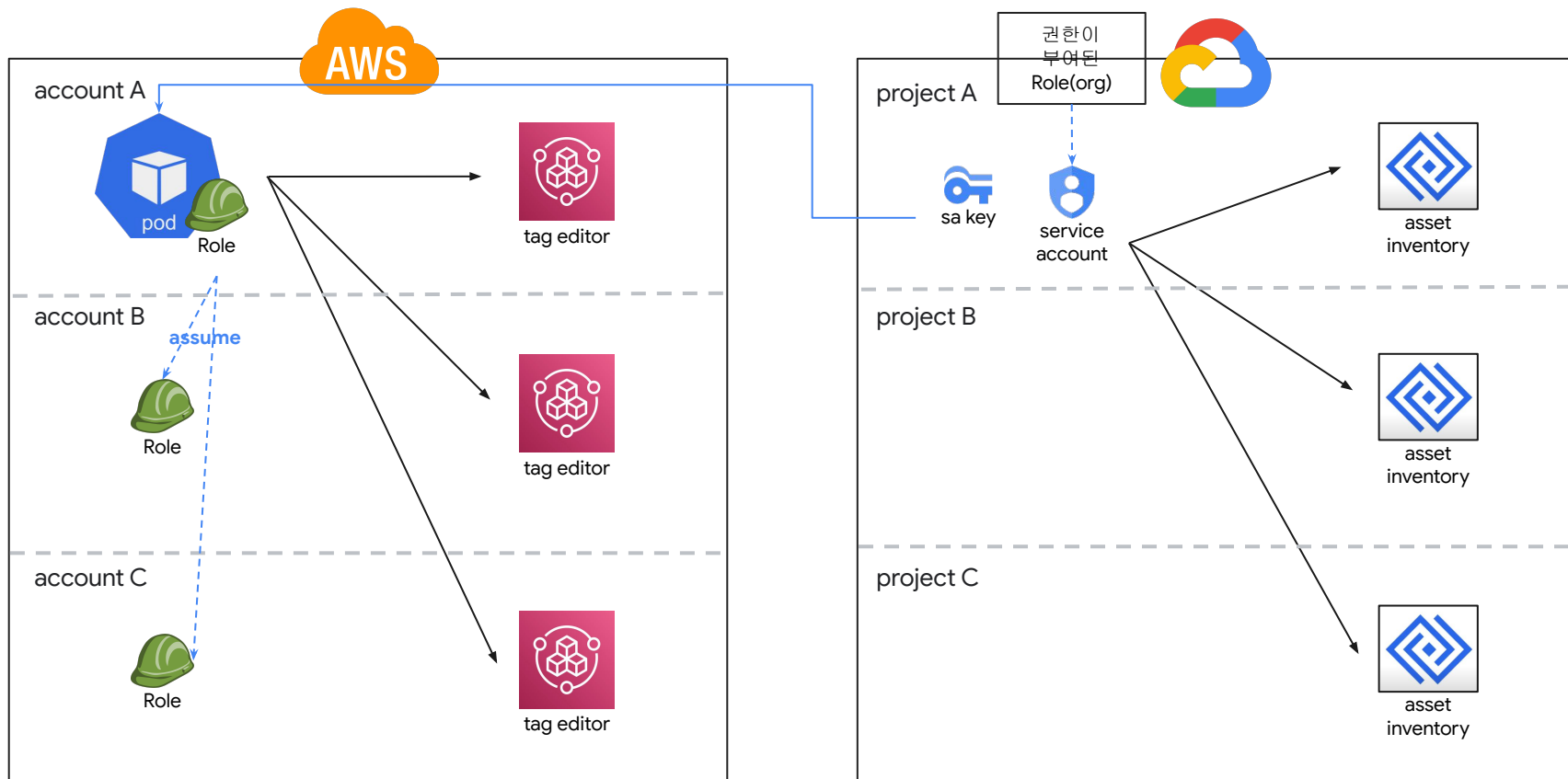
multi accounts에 있는 api를 사용해야 하는 경우 (AWS)



multi accounts에 있는 api를 사용해야 하는 경우 (AWS)



하나의 application에서 multi provider & multi accounts/projects 접근



코드 예제

account A

AWS

```
func makeAssumedClient(accountID string)
(*resourcegroupstaggingapi.Client, error) {
    // aws config에 assumed client용 credentials 설정
    stsClient := sts.NewFromConfig(cfg)
    arn := "arn:aws:iam::" + accountID + ":" + ${account B의 Role}
    assumed := stscreds.NewAssumeRoleProvider(stsClient, arn)
    cfg.Credentials = aws.NewCredentialsCache(assumed)

    // assume한 resourcegroupstaggingapi의 client 생성
    return resourcegroupstaggingapi.NewFromConfig(*cfg), nil
}
```

account B



Role

project A



```
func makePubsubClient(projectID string, base64Cred string)
*pubsub.Client {
    // project B에도 권한이 있는 sa key를 인코딩한 값(base64Cred)으로 B
    project의 pubsub client 생성
    client, err := newPubsubClient(projectID, base64Cred)
    if err != nil {
        panic(err)
    }

    return client
}
```

project B



service
account



코드 예제

AWS

goroutine으로 각 account(aws)/project(gcp)별 aws tag editor api와 gcp cloud asset api의 client를 만들면,
account와 project의 수가 많더라도, 성능의 큰 문제 없이 resource의 metadata들을 불러올 수 있어요.

```
func makeAssumedClient(accountID string) (*resourcegroupstaggingapi.Client, error) {  
    // aws config에 assumed client용 credentials 설정  
    stsClient := sts.NewFromConfig(cfg)  
    arn := "arn:aws:iam::" + accountID + ":" + ${account B의 Role}  
    assumed := stscreds.NewAssumeRoleProvider(stsClient, arn)  
    cfg.Credentials = aws.NewCredentialsCache(assumed)  
  
    // assume한 resourcegroupstaggingapi의 client 생성  
    return resourcegroupstaggingapi.NewFromConfig(*cfg), nil  
}
```



go magic()

project A

```
func makePubsubClient(projectID string, base64Cred string) (*pubsub.Client {  
    // project B에도 권한이 있는 sa key를 인코딩한 값(base64Cred)으로 B  
    // project의 pubsub client 생성  
    client, err := newPubsubClient(projectID, base64Cred)  
    if err != nil {  
        panic(err)  
    }  
    return client  
}
```

account B



Role

project B



service
account



GCP cloud asset api vs GCP other services' api



Cloud Asset api의 단점

1. AWS Tag Editor와는 다르게 **Label Update** 기능이 없습니다.
 - Tag/Label을 업데이트할 수 있어야, 각 리소스의 메타데이터를 관리할 수 있습니다.
 - AWS와 다르게 GCP는 cloud asset api를 통해 리소스의 Label을 업데이트를 할 수 없기 때문에 **각 서비스별 api에서 Label Update 기능을 이용**해야 합니다.
2. 한 번에 불러올 수 있는 **Resource**의 수가 최대 **500개**이다.
 - **cloud asset api**로 불러올 수 있는 리소스는 최대 500개([pageToken을 활용](#))이기 때문에, 전체 리소스를 불러올 경우 꽤 많은 지연이 발생할 수 있습니다.
 - 예를 들어, 하루에도 몇 번씩 실행되는 dataflow jobs, 조직 내에 있는 수많은 Bigquery Table들을 다 더하면 회사 리소스가 몇 만개가 넘어가는 것은 어렵지 않은 일입니다.



Cloud Asset api 대신 각 서비스별 api의 Label 조회, 수정 기능 활용

1. cloud asset api에 없는 **Label Update** 기능을 수행할 수 있어요.

- 각 서비스들(gcs, bigquery, pub/sub, vm 등)은 각각의 개별 api에서도 대부분 Label Update기능을 제공하고 있어요.
- 여러 gcp 리소스에 대한 Update 요청이 있으면 각 서비스별 api를 병렬적으로 실행시켜서 리소스의 메타데이터를 변경해요.

2. 전부 확인한 것은 아니지만, 각 서비스별 api는 cloud asset api에 비해 한 번에 불러올 수 있는 리소스 개수가 더 많았어요.

- 일반적으로 리소스 개수가 많은 dataflow의 경우, 한 프로젝트에서 3천 개 이상의 job들을 조회할 수도

```
1 SELECT
2   t1.table_catalog project_id, t1.table_schema dataset_id, t1.table_name, 'US' location, t2.option_value labels
3 FROM
4   'region-US'.INFORMATION_SCHEMA.TABLES t1
5 LEFT JOIN
6   (SELECT
7     table_catalog, table_schema, table_name, option_name, option_value
8   FROM
9     'region-US'.INFORMATION_SCHEMA.TABLE_OPTIONS
10    WHERE option_name = 'labels') t2
11 ON t1.table_catalog = t2.table_catalog AND t1.table_schema = t2.table_schema AND t1.table_name = t2.table_name
```

Query results

JOB INFORMATION	RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH	PREVIEW
Row	project_id	dataset_id	table_name	location	labels
1					null
2					null
3					null
4					null
5					null
6					null
7					null

query 메타데이터 정보들을 한 번에 볼 수 있는
물론 리소스 메타데이터 조회 기능만
dataset과 Table을 한 번에 조회할 수 있어요.
필요하고, 프로젝트에 리소스가 많지 않을
경우에는 cloud asset api가 더 좋은
옵션이 될 수 있어요.



동시성 프로그래밍을 통해 gcp asset inventory 대체해보기

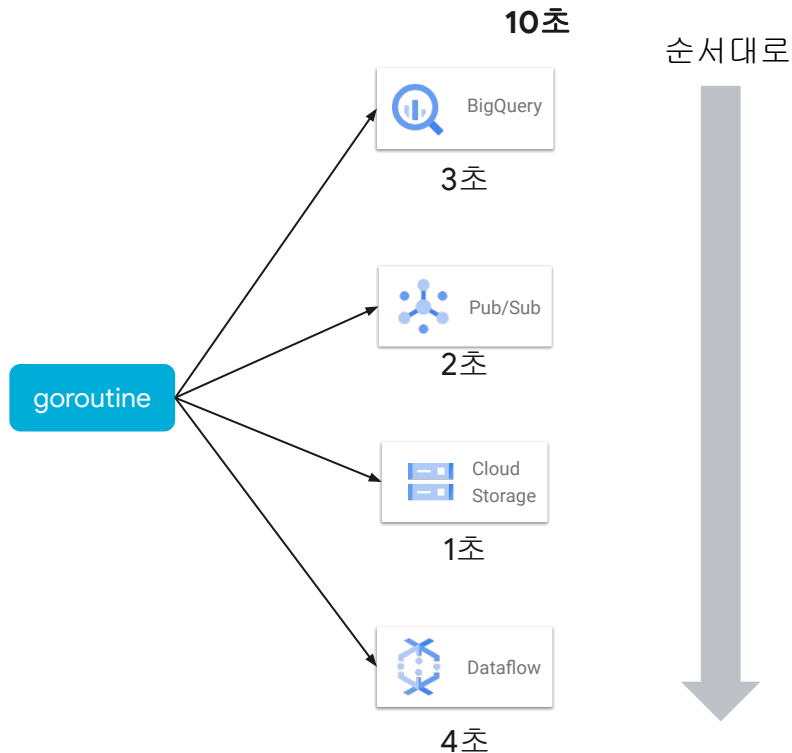


goroutine과 channel을 통해 동시성 프로그래밍 활용

여러 서비스별 api를 순서대로 호출하면, 지원하는 리소스가 많을수록 성능상에 이슈가 발생할 수 있어요.

pseudo code

```
MakeGCPResources(...) {  
    var gcpResources []*CustomGCPResourceType  
    a := bigqueryClient.GetResources()  
    b := pubsubClient.GetResources()  
    c := gcsClient.GetResources()  
    d := dataflow.GetResources()  
    gcpResources = append(gcpResources, a...)  
    gcpResources = append(gcpResources, b...)  
    gcpResources = append(gcpResources, c...)  
    ...  
}
```

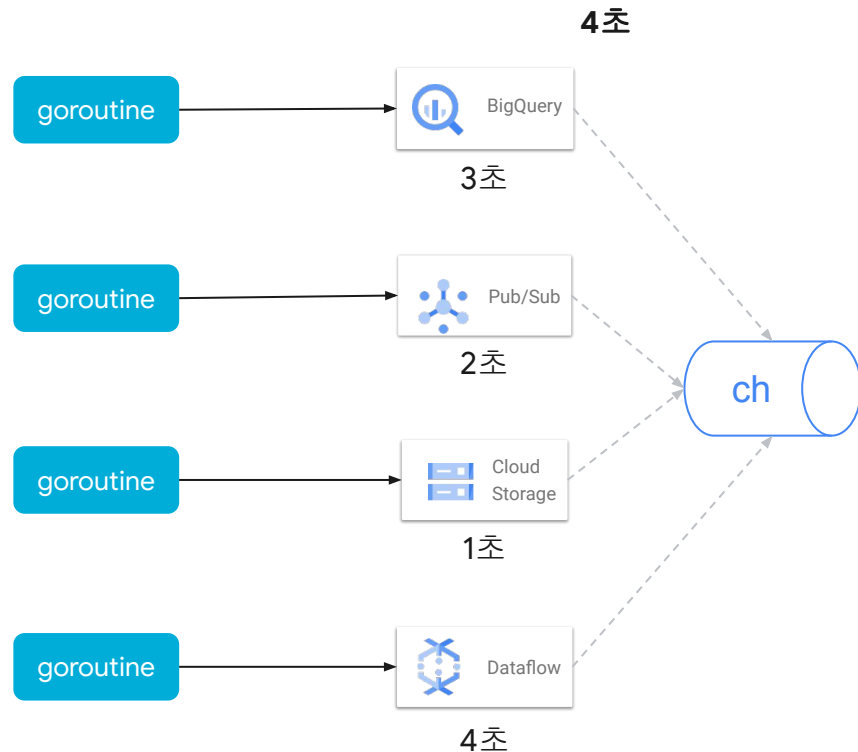


goroutine과 channel을 통해 동시성 프로그래밍 활용

goroutine으로 각 서비스별 api를 병렬로 호출하고,
이를 channel(고루틴 메세지 큐)에 쌓으면 지원하는 리소스가 늘어나도 성능 이슈가 발생하지 않아요.

pseudo code

```
MakeGCPResources(...) {  
    var gcpResources []*CustomGCPResourceType  
  
    switch resourceName {  
    case "BigQuery":  
        gcpResources = bigqueryClient.GetResources()  
    case "PubSub":  
        gcpResources = pubsubClient.GetResources()  
    case "GoogleCloudStorage":  
        gcpResources = gcsClient.GetResources()  
    case "Dataflow":  
        gcpResources = d := dataflow.GetResources()  
    }  
    ch <- gcpResources  
    ...  
}
```



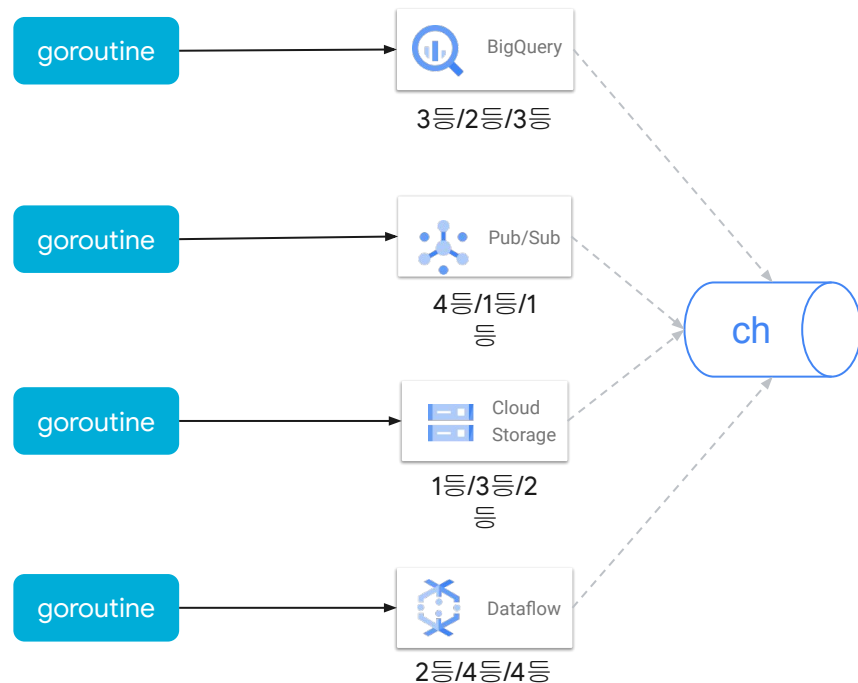
goroutine과 channel을 통해 동시성 프로그래밍 활용

실제로 goroutine으로 실행한 서비스별 api 호출 함수는 동시성 프로그래밍으로 실행되고, 함수가 종료되면 채널에 데이터를 전송해줘요. 그래서 아래 함수들의 순서가 실행할 때마다 다를 수 있어요.

```
시작
GCS 메타데이터 호출하고 채널에 넣기 완료
Dataflow 메타데이터 호출하고 채널에 넣기 완료
BigQuery 메타데이터 호출하고 채널에 넣기 완료
PubSub 메타데이터 호출하고 채널에 넣기 완료
```

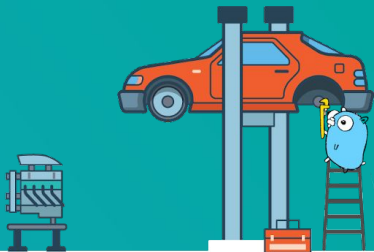
```
시작
PubSub 메타데이터 호출하고 채널에 넣기 완료
BigQuery 메타데이터 호출하고 채널에 넣기 완료
GCS 메타데이터 호출하고 채널에 넣기 완료
Dataflow 메타데이터 호출하고 채널에 넣기 완료
```

```
시작
PubSub 메타데이터 호출하고 채널에 넣기 완료
GCS 메타데이터 호출하고 채널에 넣기 완료
BigQuery 메타데이터 호출하고 채널에 넣기 완료
Dataflow 메타데이터 호출하고 채널에 넣기 완료
```



“

참고



- <https://cloud.google.com/blog/products/identity-security/improve-visibility-with-four-cloud-asset-inventory-features?hl=en>
- <https://cloud.google.com/bigquery/docs/information-schema-intro>
- <https://cloud.google.com/asset-inventory>
- <https://pkg.go.dev/google.golang.org/api/cloudasset/v1>
- <https://pkg.go.dev/github.com/aws/aws-sdk-go-v2/service/resourcegroupstaggingapi>

“



Q&A

GDG Cloud Korea Organizer 김효민

”

