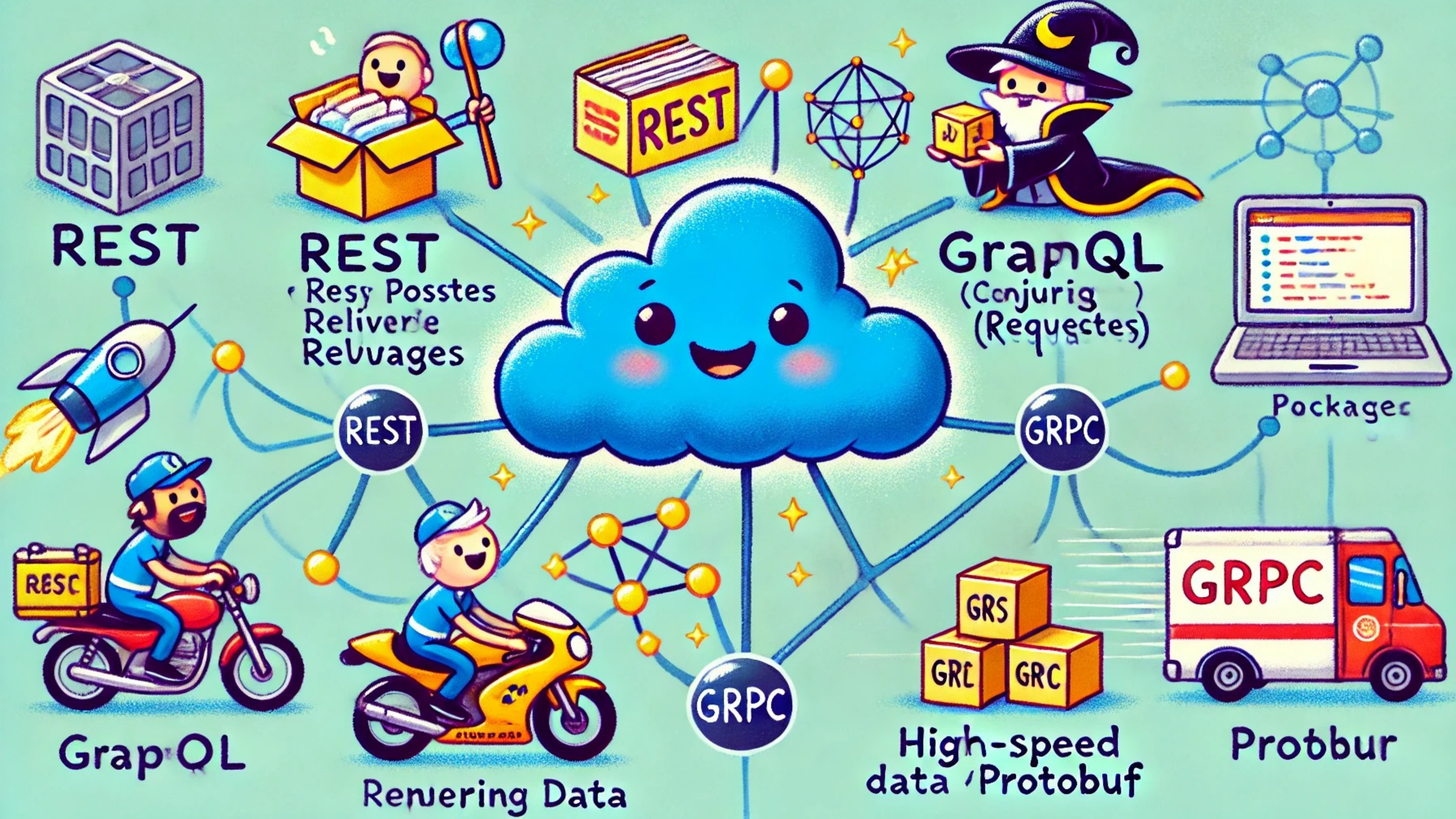
Cloud Computing Technologies

DAT515 - Fall 2024

Cloud APIs

Prof. Hein Meling

University
of Stavanger


REST
REST
REST
GraphQL
(Conjuring
(Requestes)
Packages
REST
GRPC
GRPC
GRS
GRC
GRC
GRPC
GraphQL
Rendering Data
High-speed
data Protobuf
Protobuf

Cloud APIs

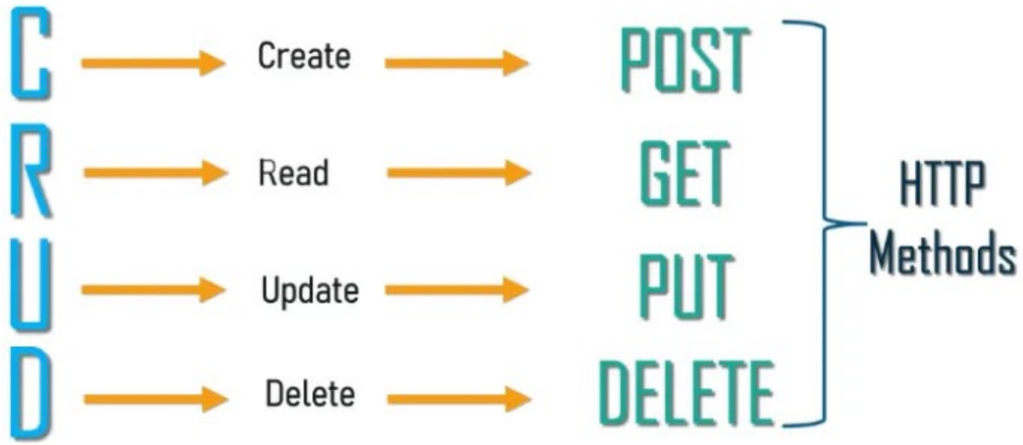
- REST
- GraphQL
- gRPC

REST APIs

Representational State Transfer

REST APIs

- REST is a web standards-based architecture that uses HTTP
- **Stateless**: Each request from client to server must contain all necessary information
- **CRUD** operations
- Operates on resources identified by URIs
- Widely supported and simple to use



Routing Requests

Most Basic Web Server

```go
package main

import (
    "fmt"
    "net/http"
)

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintf(w, "Hello, you've requested: %s\n", r.URL.Path)
    })

    http.ListenAndServe(":80", nil)
}
```

Web Server with Gorilla Mux

```go
package main

import (
    "fmt"
    "net/http"

    "github.com/gorilla/mux"
)

func main() {
    r := mux.NewRouter()

    r.HandleFunc("/books/{title}/page/{page}", func(w http.ResponseWriter, r *http.Request) {
        vars := mux.Vars(r)
        title := vars["title"]
        page := vars["page"]

        fmt.Fprintf(w, "You've requested the book: %s on page %s\n", title, page)
    })

    http.ListenAndServe(":80", r)
}
```

Web Server with Gorilla Mux

```
r.HandleFunc("/books/{title}", CreateBook).Methods("POST")
r.HandleFunc("/books/{title}", ReadBook).Methods("GET")
r.HandleFunc("/books/{title}", UpdateBook).Methods("PUT")
r.HandleFunc("/books/{title}", DeleteBook).Methods("DELETE")
```

Web Server with Stdlib

```
mux := http.NewServeMux()
server := NewTaskServer()

mux.HandleFunc("POST /task/", server.createTaskHandler)
mux.HandleFunc("GET /task/", server.getAllTasksHandler)
mux.HandleFunc("DELETE /task/", server.deleteAllTasksHandler)
mux.HandleFunc("GET /task/{id}/", server.getTaskHandler)
mux.HandleFunc("DELETE /task/{id}/", server.deleteTaskHandler)
mux.HandleFunc("GET /tag/{tag}/", server.tagHandler)
mux.HandleFunc("GET /due/{year}/{month}/{day}/", server.dueHandler)
```

Web Server with Stdlib

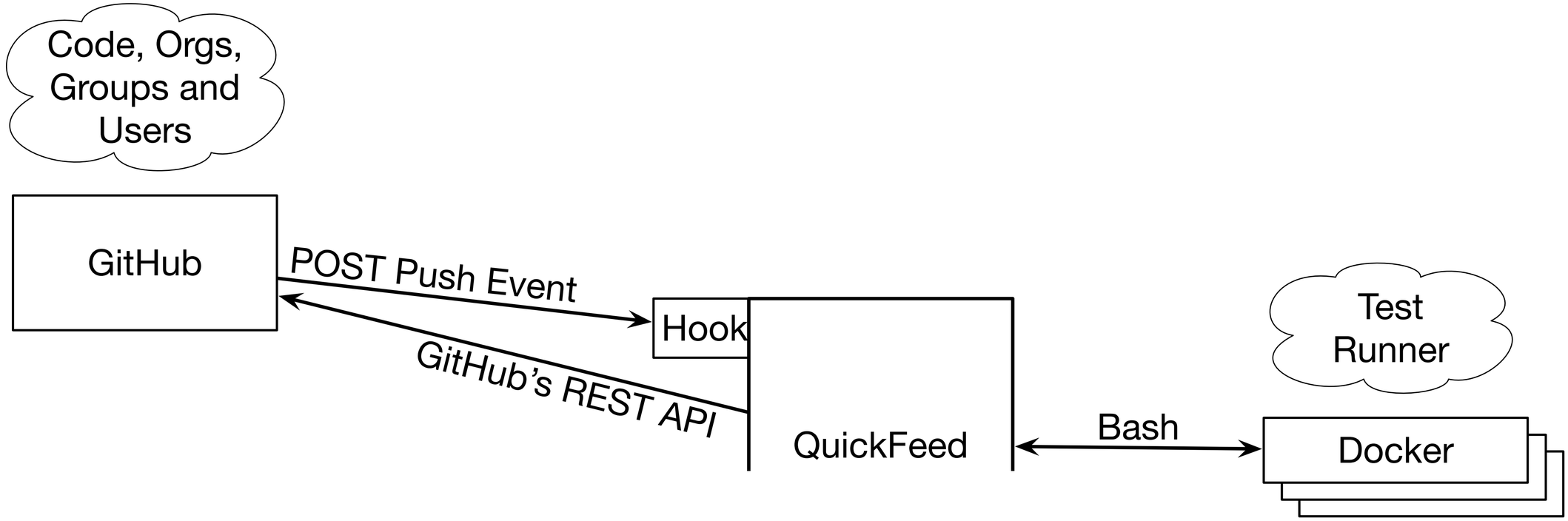
```
func (ts *taskServer) getTaskHandler(w http.ResponseWriter, req *http.Request) {
  log.Printf("handling get task at %s\n", req.URL.Path)

  id, err := strconv.Atoi(req.PathValue("id"))
  if err != nil {
    http.Error(w, "invalid id", http.StatusBadRequest)
    return
  }

  task, err := ts.store.GetTask(id)
  if err != nil {
    http.Error(w, err.Error(), http.StatusNotFound)
    return
  }

  renderJSON(w, task)
}
```

QuickFeed Overview



QuickFeed Web Server Router

```go
// RegisterRouter registers http endpoints for authentication API and scm provider webhooks.
func (s *QuickFeedService) RegisterRouter(tm *auth.TokenManager, authConfig *oauth2.Config, public string) *http.ServeMux {
    // Serve static files.
    router := http.NewServeMux()
    assets := http.FileServer(http.Dir(public + "/assets")) // skipcq: GO-S1034
    dist := http.FileServer(http.Dir(public + "/dist"))     // skipcq: GO-S1034

    router.Handle("/", http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        http.ServeFile(w, r, public+"/assets/index.html")
    }))
    paths, handler := s.NewQuickFeedHandler(tm)
    router.Handle(paths, controller(handler, streamTimeout))

    router.Handle(auth.Assets, http.StripPrefix(auth.Assets, assets))
    router.Handle(auth.Static, http.StripPrefix(auth.Static, dist))
    // Register auth endpoints.
    callbackSecret := rand.String()
    router.HandleFunc(auth.Auth, auth.OAuth2Login(s.logger, authConfig, callbackSecret))
    router.HandleFunc(auth.Callback, auth.OAuth2Callback(s.logger, s.db, tm, authConfig, callbackSecret))
    router.HandleFunc(auth.Logout, auth.OAuth2Logout())

    // Register hooks.
    ghHook := hooks.NewGitHubWebHook(s.logger, s.db, s.scmMgr, s.runner, s.bh.Secret, s.streams, tm)
    router.HandleFunc(auth.Hook, ghHook.Handle())

    return router
}
```

```go
const (
    Cookie                = "cookie"
    CookieName            = "auth"
    SetCookie             = "Set-Cookie"
    tokenExpirationTime   = 15 * time.Mi
    cookieExpirationTime = 24 * time.Ho
    alg                   = "HS256"

    githubUserAPI = "https://api.github

    // Routes
    Auth     = "/auth/"
    Teacher  = "/auth/teacher/"
    Callback = "/auth/callback/"
    Logout   = "/logout"
    Hook     = "/hook/"
    Assets   = "/assets/"
    Static   = "/static/"
)
```

QuickFeed Web Server Router

```go
// Handle take POST requests from GitHub, representing Push events
// associated with course repositories, which then triggers various
// actions on the QuickFeed backend.
func (wh GitHubWebHook) Handle() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        payload, err := github.ValidatePayload(r, []byte(wh.secret))
        if err != nil {
            wh.logger.Errorf("Error in request body: %v", err)
            w.WriteHeader(http.StatusUnauthorized)
            return
        }
        defer r.Body.Close()

        event, err := github.ParseWebHook(github.WebHookType(r), payload)
        if err != nil {
            wh.logger.Errorf("Could not parse github webhook: %v", err)
            w.WriteHeader(http.StatusBadRequest)
            return
        }
        wh.logger.Debug(qlog.IndentJson(event))
        switch e := event.(type) {
        case *github.PushEvent:
            commitID := e.GetHeadCommit().GetID()
            wh.logger.Debugf("Received push event: %s", commitID)
            if wh.dup.Duplicate(commitID) {
                wh.logger.Debugf("Ignoring duplicate push event: %s", commitID)
                return
            }
```

Push Event from GitHub

```
2024-08-23T09:20:30.741+0200    DEBUG    hooks/github.go:66      {
  "ref": "refs/heads/main",
  "commits": [
    {
      "message": "Siste gang forhåpentilgvis",
      "author": {
        "name": "Christopher Liebich Kolle",
        "email": "cl.kolle@stud.uis.no",
        "username": "CKolle"
      },
      "url": "https://github.com/dat240-2024/CKolle-labs/commit/b411f6719158d095cff00b1edfc5d331663e8c62",
      "distinct": true,
      "id": "b411f6719158d095cff00b1edfc5d331663e8c62",
      "tree_id": "76bd3ecbea490d4b86c50e8d4ddee41f863f6ad0",
      "timestamp": "2024-08-23T09:20:26+02:00",
      "committer": {
        "name": "Christopher Liebich Kolle",
        "email": "cl.kolle@stud.uis.no",
        "username": "CKolle"
      },
      "modified": [
        "Lab1/UiS.Dat240.Lab1/Program.cs",
        "Lab1/UiS.Dat240.Lab1/TestSubmissions.cs"
      ]
    }
  ],
  "before": "3c2112adbbf2c9368f3946602522cdec0abbc1f1",
  "after": "b411f6719158d095cff00b1edfc5d331663e8c62",
  "created": false,
  "deleted": false,
  "forced": false,
  "compare": "https://github.com/dat240-2024/CKolle-labs/compare/3c2112adbbf2...b411f6719158",
  "repository": {
```

GitHub's REST API

GitHub REST Examples

GitHub Docs | Version: Free, Pro, & Team | Search GitHub Docs

← Home

REST API

API Version: 2022-11-28 (latest)

- Migrations
- Organizations
- Packages
- Pages
- Projects (classic)
- Pull requests
- Rate limit
- Reactions
- Releases
- Repositories
  - Autolinks

The REST API is now versioned. For more information, see "About API versioning."

REST API /

REST API endpoints for repositories

Use the REST API to create, manage and control the workflow of public and private GitHub repositories.

REST API endpoints for repositories

- List organization repositories
- Create an organization repository
- Get a repository
- Update a repository
- Delete a repository

https://docs.github.com/en/rest/repos?apiVersion=2022-11-28

GitHub REST Examples

REST API endpoints for repositories

Use the REST API to manage repositories on GitHub.

gh api /orgs/dat515-2024/repos

List organization repositories

Lists repositories for the specified organization.

> **Note**
>
> In order to see the `security_and_analysis` block for a repository you must have admin permissions for the repository or be an owner or security manager for the organization that owns the repository. For more information, see "Managing security managers in your organization."

Fine-grained access tokens for "List organization repositories"

This endpoint works with the following fine-grained token types:

- GitHub App user access tokens
- GitHub App installation access tokens
- Fine-grained personal access tokens

The fine-grained token must have the following permission set:

- "Metadata" repository permissions (read)

Code samples for "List organization repositories"

Request example

GET /orgs/{org}/repos

cURL | JavaScript | GitHub CLI

```
# GitHub CLI api
# https://cli.github.com/manual/gh_api

gh api \
  -H "Accept: application/vnd.github+json" \
  -H "X-GitHub-Api-Version: 2022-11-28" \
  /orgs/ORG/repos
```

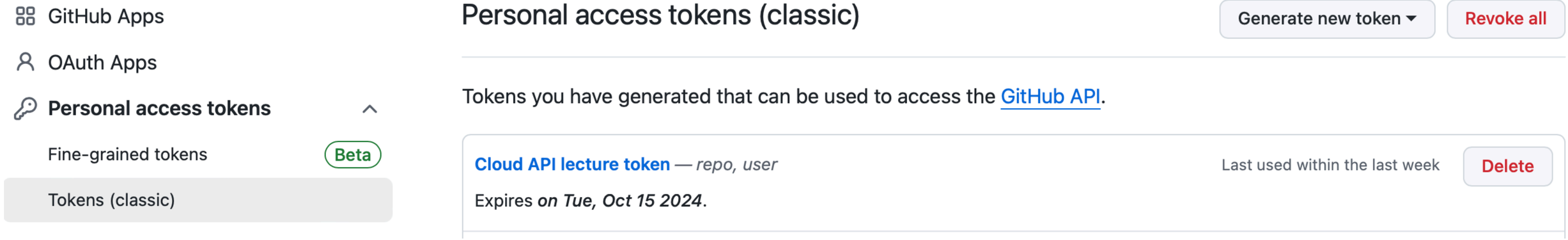
Response

Example response | Response schema

```
Status: 200
```


```
[
  {
    "id": 841558548,
    "node_id": "R_kgDOMikqFA",
    "name": "info",
    "full_name": "dat515-2024/info",
    "private": false,
    "owner": {
      "login": "dat515-2024",
      "id": 178194774,
      "node_id": "O_kgDOCp8JVg",
      "avatar_url": "https://avatars.githubusercontent.com/u/178194774?v=4",
      "gravatar_id": "",
      "url": "https://api.github.com/users/dat515-2024",
      "html_url": "https://github.com/dat515-2024",
      "followers_url": "https://api.github.com/users/dat515-2024/followers",
      "following_url": "https://api.github.com/users/dat515-2024/following{/other_user}",
      "gists_url": "https://api.github.com/users/dat515-2024/gists{/gist_id}",
      "starred_url": "https://api.github.com/users/dat515-2024/starred{/owner}{/repo}",
      "subscriptions_url": "https://api.github.com/users/dat515-2024/subscriptions",
      "organizations_url": "https://api.github.com/users/dat515-2024/orgs",
      "repos_url": "https://api.github.com/users/dat515-2024/repos",
      "events_url": "https://api.github.com/users/dat515-2024/events{/privacy}",
      "received_events_url": "https://api.github.com/users/dat515-2024/received_events",
      "type": "Organization",
      "site_admin": false
    },
```

GitHub REST Examples



Navigate to **Settings**

- Developer Settings
- Personal Access Tokens
- Fine-grained tokens or Tokens (classic)
- Generate new token
- Save token string securely

GitHub Go API Examples

```go
// createRepository creates a new repository or returns an existing repository with the given name.
func (s *GithubSCM) createRepository(ctx context.Context, opt *CreateRepositoryOptions) (*Repository, error) {
    const op Op = "createRepository"
    m := M("failed to create repository")
    if !opt.valid() {
        return nil, E(op, m, fmt.Errorf("missing fields: %+v", *opt))
    }

    // check that repo does not already exist for this user or group
    repo, resp, err := s.client.Repositories.Get(ctx, opt.Owner, opt.Repo)
    if repo != nil {
        s.logger.Debugf("CreateRepository: found existing repository (skipping creation): %s: %v", opt.Repo, repo)
        return toRepository(repo), nil
    }
    // error expected with response status code to be 404 Not Found
    if resp != nil && resp.StatusCode != http.StatusNotFound {
        s.logger.Errorf("CreateRepository: get repository %s returned unexpected status %d: %v", opt.Repo, resp.Sta
    }

    // repo does not exist, create it
    s.logger.Debugf("CreateRepository: creating %s", opt.Repo)
    repo, _, err = s.client.Repositories.Create(ctx, opt.Owner, &github.Repository{
        Name:    github.String(opt.Repo),
        Private: github.Bool(opt.Private),
    })
    if err != nil {
        return nil, E(op, M("failed to create repository %s/%s", opt.Owner, opt.Repo), err)
    }
    s.logger.Debugf("CreateRepository: successfully created %s/%s", opt.Owner, opt.Repo)
    return toRepository(repo), nil
}
```

GitHub Go API Examples

```go
// Get fetches a repository.
//
// GitHub API docs: https://docs.github.com/rest/repos/repos#get-a-repository
//
//meta:operation GET /repos/{owner}/{repo}
func (s *RepositoriesService) Get(ctx context.Context, owner, repo string) (*Repository, *Response, error) {
    u := fmt.Sprintf("repos/%v/%v", owner, repo)
    req, err := s.client.NewRequest("GET", u, nil)
    if err != nil {
        return nil, nil, err
    }

    // TODO: remove custom Accept header when the license support fully launches
    // https://docs.github.com/rest/licenses/#get-a-repositorys-license
    acceptHeaders := []string{
        mediaTypeCodesOfConductPreview,
        mediaTypeTopicsPreview,
        mediaTypeRepositoryTemplatePreview,
        mediaTypeRepositoryVisibilityPreview,
    }
    req.Header.Set("Accept", strings.Join(acceptHeaders, ", "))

    repository := new(Repository)
    resp, err := s.client.Do(ctx, req, repository)
    if err != nil {
        return nil, resp, err
    }

    return repository, resp, nil
}
```

GitHub Go API

- go-github is a Go client library for accessing the GitHub API v3
  - https://github.com/google/go-github
- `go get github.com/google/go-github/v64`
- `import “github.com/google/go-github/v64/github"`

QuickFeed's SCM Interface

```
// SCM is the source code management interface for managing courses and users.
type SCM interface {
    GetOrganization(context.Context, *OrganizationOptions) (*qf.Organization, error)
    GetRepositories(context.Context, string) ([]*Repository, error)
    RepositoryIsEmpty(context.Context, *RepositoryOptions) bool

    CreateCourse(context.Context, *CourseOptions) ([]*Repository, error)
    UpdateEnrollment(context.Context, *UpdateEnrollmentOptions) (*Repository, error)
    RejectEnrollment(context.Context, *RejectEnrollmentOptions) error
    DemoteTeacherToStudent(context.Context, *UpdateEnrollmentOptions) error
    CreateGroup(context.Context, *GroupOptions) (*Repository, error)
    UpdateGroupMembers(context.Context, *GroupOptions) error
    DeleteGroup(context.Context, *RepositoryOptions) error

    Clone(context.Context, *CloneOptions) (string, error)
    AcceptInvitations(context.Context, *InvitationOptions) (string, error)

    CreateIssue(context.Context, *IssueOptions) (*Issue, error)
    UpdateIssue(context.Context, *IssueOptions) (*Issue, error)
    GetIssue(context.Context, *RepositoryOptions, int) (*Issue, error)
    GetIssues(context.Context, *RepositoryOptions) ([]*Issue, error)
    DeleteIssue(context.Context, *RepositoryOptions, int) error
    DeleteIssues(context.Context, *RepositoryOptions) error
```

Joke time

What's the most frustrating part of cloud API development?

When the response says '200 OK,' but your brain says '404 Not Found.'

GraphQL

Query language for APIs and runtime to execute those queries

- Allows clients to request **only the data they** need
  - No over-fetching or under-fetching
- Single endpoint: `/graphql`
- Supports nested queries and mutations
- Developed by Facebook for their news feed
  - (now used widely across services)

GraphQL Example

Query

```
{
  hero {
    name
    friends {
      name
    }
  }
}
```

Response

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {"name": "Luke Skywalker”},
        {"name": "Han Solo”},
        {"name": "Leia Organa”}
      ]
    }
  }
}
```

Who Uses GraphQL?

- GitHub's API v4
  - https://github.com/shurcooL/githubv4
- Twitter, Shopify, Netflix, Airbnb, PayPal

QuickFeed Uses GitHub's GraphQL API

```go
func (s *GithubSCM) DeleteIssue(ctx context.Context, opt *RepositoryOptions, issueNumber int) error {
    var q struct {
        Repository struct {
            Issue struct {
                ID githubv4.ID
            } `graphql:"issue(number:$issueNumber)"`
        } `graphql:"repository(owner:$repositoryOwner,name:$repositoryName)"`
    }
    variables := map[string]interface{}{
        "repositoryOwner": githubv4.String(opt.Owner),
        "repositoryName":  githubv4.String(opt.Repo),
        "issueNumber":     githubv4.Int(issueNumber),
    }
    if err := s.clientV4.Query(ctx, &q, variables); err != nil {
        return err
    }

    var m struct {
        DeleteIssue struct {
            Repository struct {
                Name string
            }
        } `graphql:"deleteIssue(input:$input)"`
    }
    input := githubv4.DeleteIssueInput{
        IssueID: q.Repository.Issue.ID,
    }
    return s.clientV4.Mutate(ctx, &m, input, nil)
}
```

Criticisms Against GraphQL

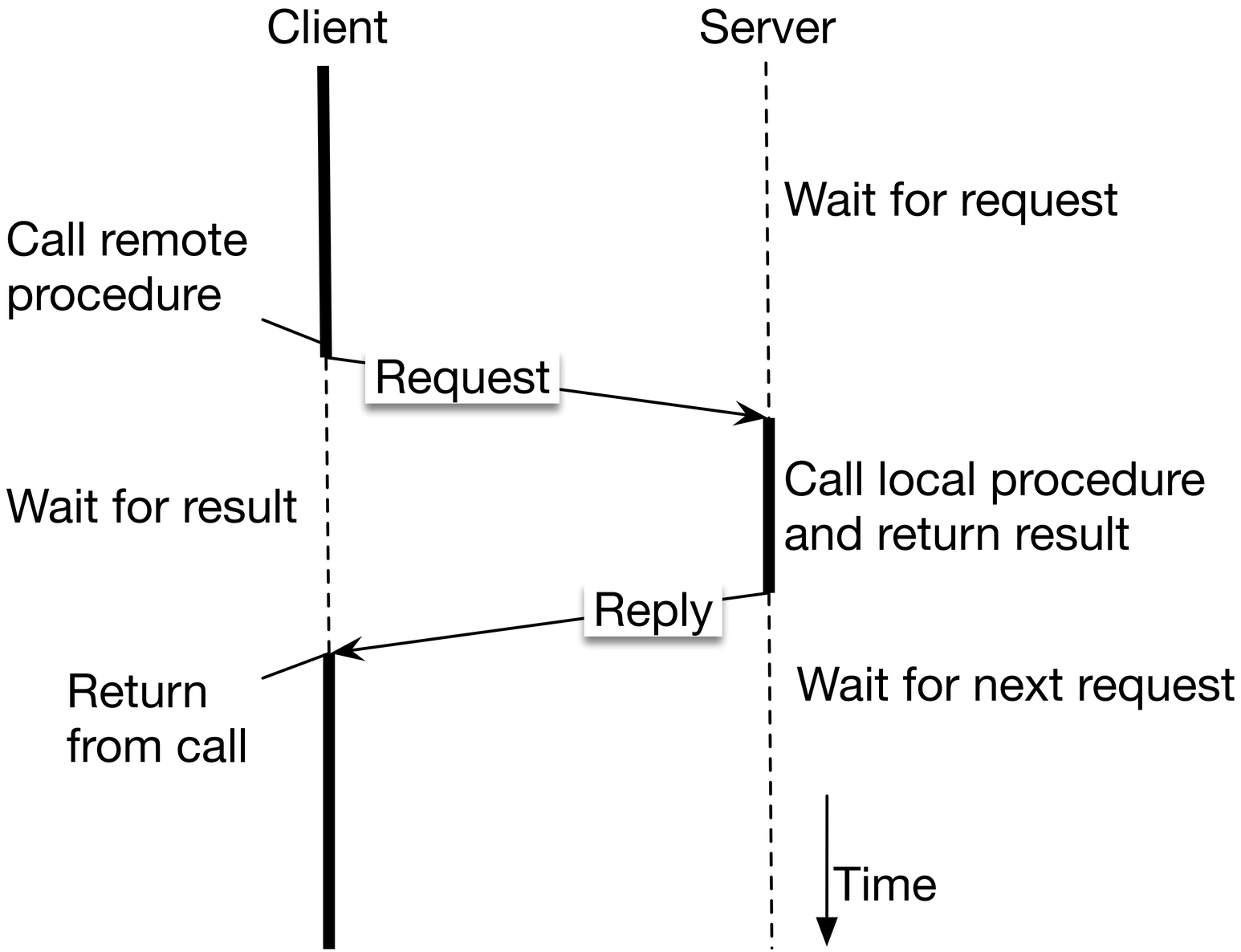
- Complex queries can result in **performance bottlenecks**
  - Servers must fetch and assemble from multiple data sources
  - (server-side over-fetching)
- Caching Challenges
  - Caching each REST resource is easy — has a unique URL (`GET /users/1`)
  - In GraphQL, traditional caching mechanisms become less effective

Criticisms Against GraphQL

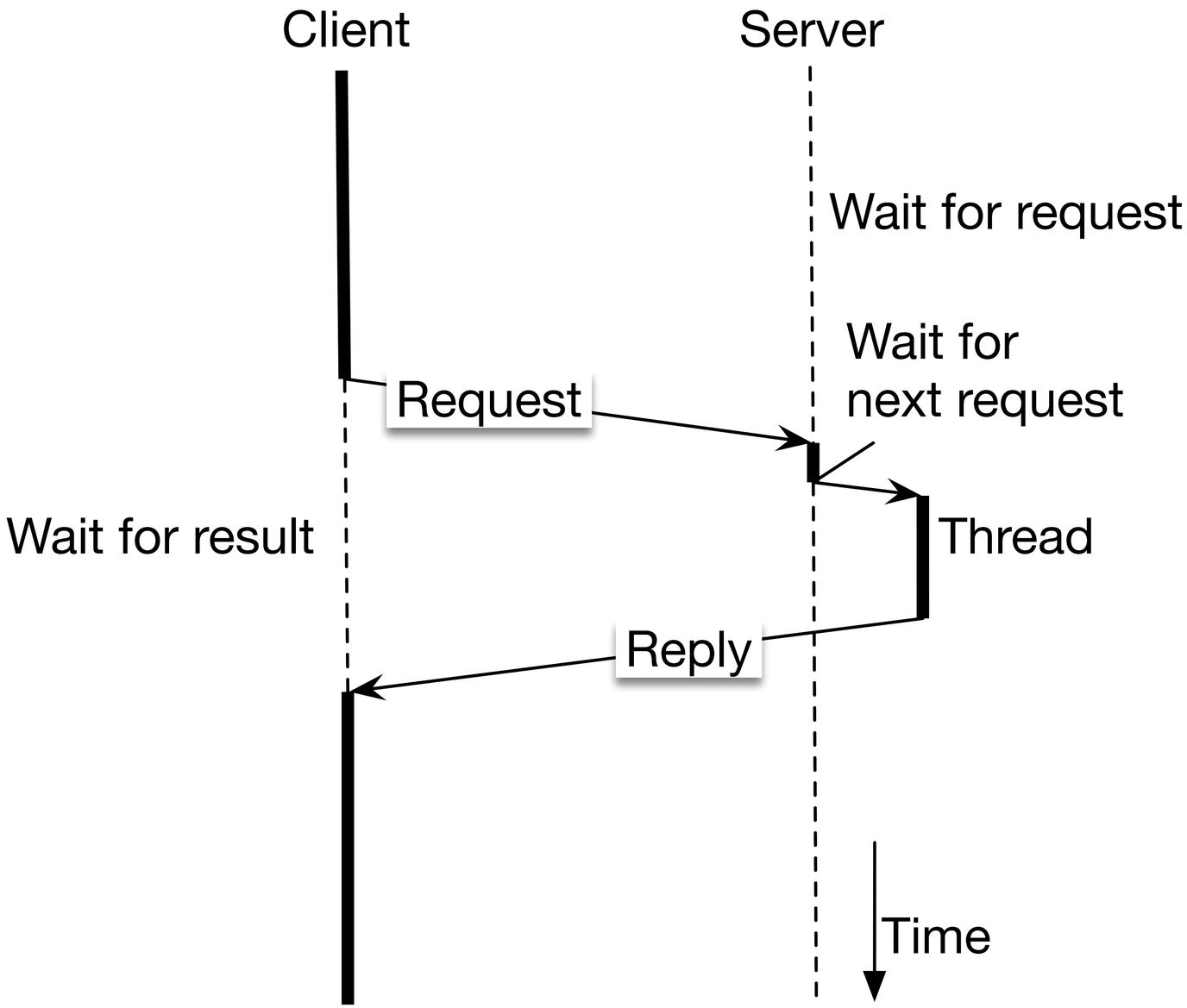
- For simple applications, GraphQL is **overkill**
- The **learning curve** for GraphQL can be steep
  - Must learn concepts like schemas, resolvers, and queries/mutations
- GraphQL queries can request **any combination of fields**
  - Managing **authorization** at the field level **is complex**
  - Easier to introduce **vulnerabilities**

Remote Procedure Call

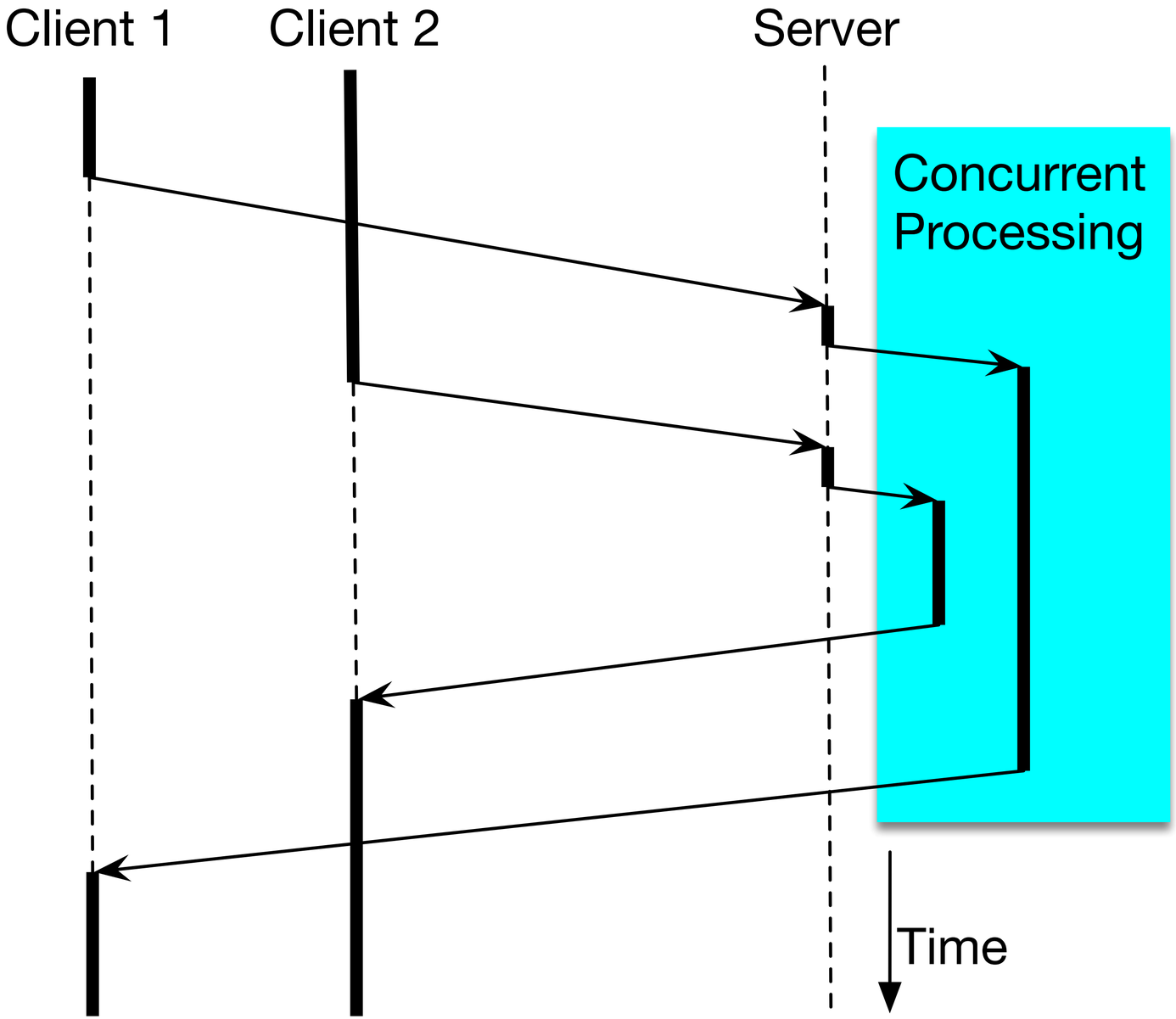
Remote Procedure Call



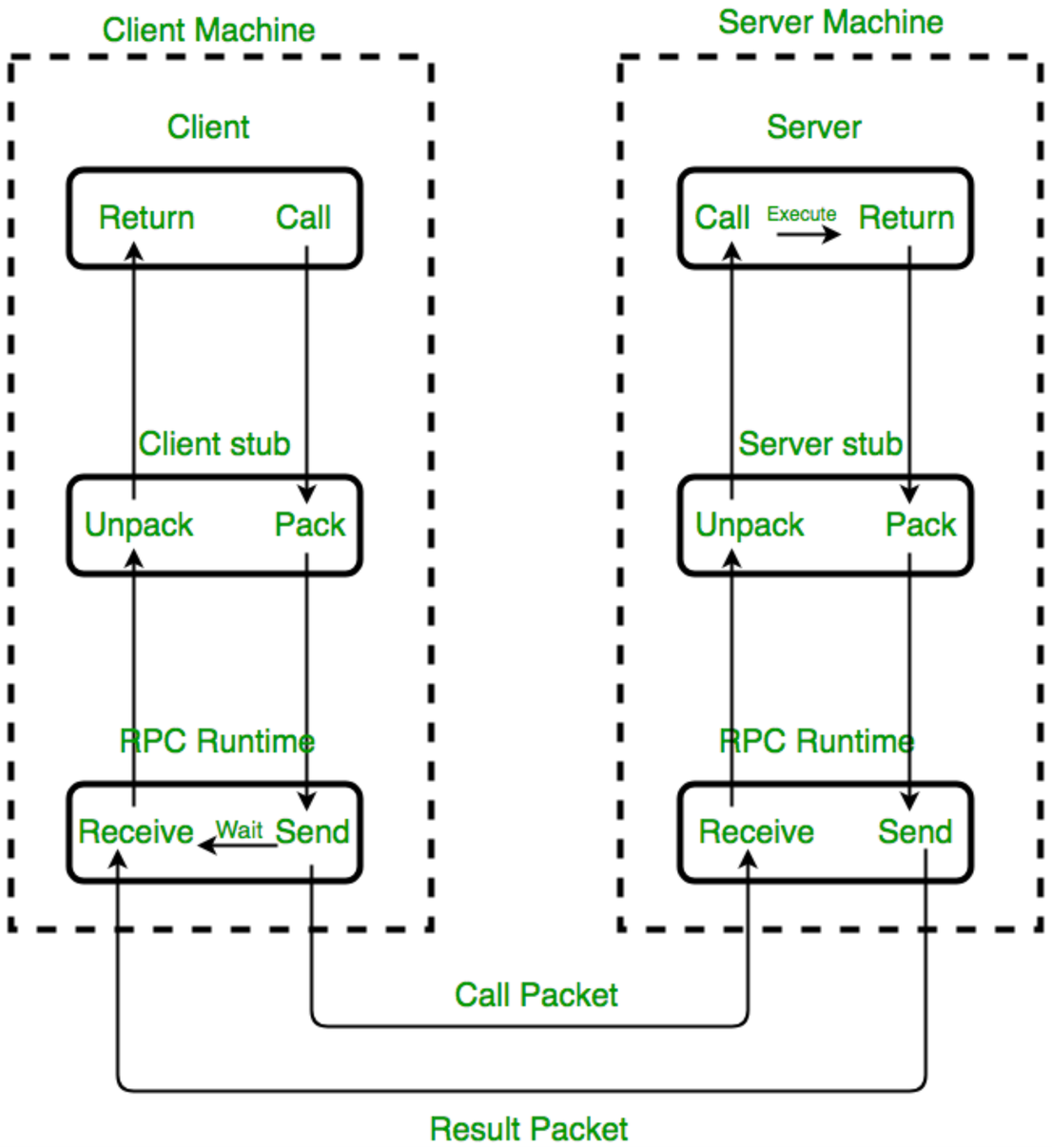
Remote Procedure Call



Remote Procedure Call



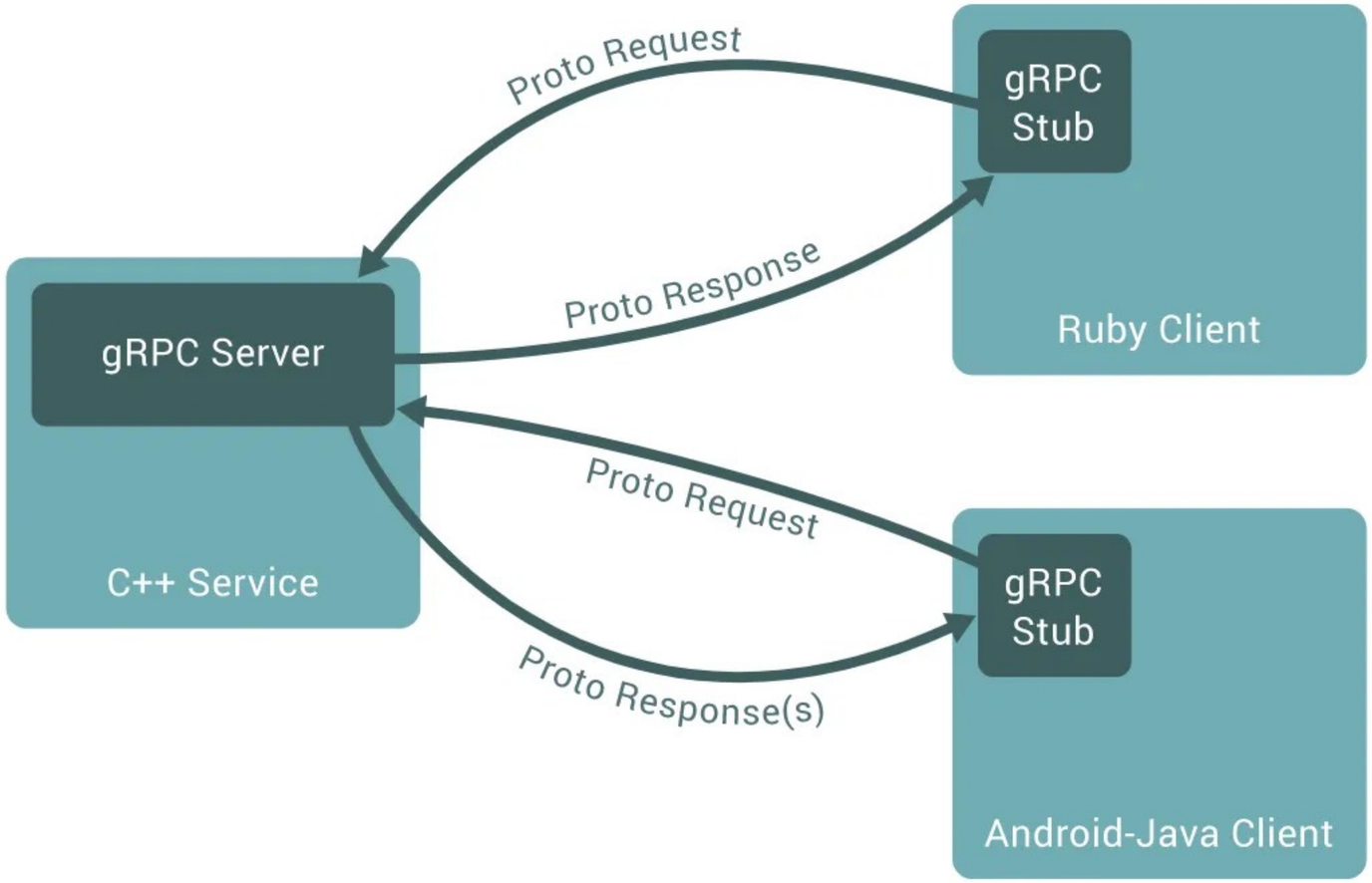
Remote Procedure Call



Google's gRPC Remote Procedure Call

gRPC Remote Procedure Calls

gRPC is a recursive acronym



gRPC Remote Procedure Calls

High-performance RPC framework

- Uses Protocol Buffers (protobuf) as its Interface Definition Language (IDL)
- Supports bi-directional streaming
- Uses HTTP/2 for transport, enabling multiplexed streams
- Strongly typed contracts between services with auto-generated code
- Widely used in microservices and distributed systems

Protocol Buffers (protobuf)

Interface Definition Language (IDL)

- Specifies RPC methods and message content

```
service HelloService {
  rpc SayHello (HelloRequest) returns (HelloResponse);
}

message HelloRequest {
  string greeting = 1;
}

message HelloResponse {
  string reply = 1;
}
```

gRPC Remote Procedure Calls

Four kinds of service methods

- Unary RPCs
  - Client sends a single request to the server
  - Get single response back
  - (like a normal function call)

```
rpc SayHello(HelloRequest) returns (HelloResponse);
```

gRPC Remote Procedure Calls

Four kinds of service methods

- Server streaming RPCs
  - Client sends a request to the server
  - Gets a stream to read a sequence of messages back
  - Client reads from the stream until there are no more messages
  - (guaranteed message ordering within an individual RPC call)

```
rpc LotsOfReplies(HelloRequest) returns (stream HelloResponse);
```

gRPC Remote Procedure Calls

Four kinds of service methods

- Client streaming RPCs
  - Client sends a sequence of messages server using a stream
  - Client waits for the server to read the messages and return a response
  - (guaranteed message ordering within an individual RPC call)

```
rpc LotsOfGreetings(stream HelloRequest) returns (HelloResponse);
```

gRPC Remote Procedure Calls

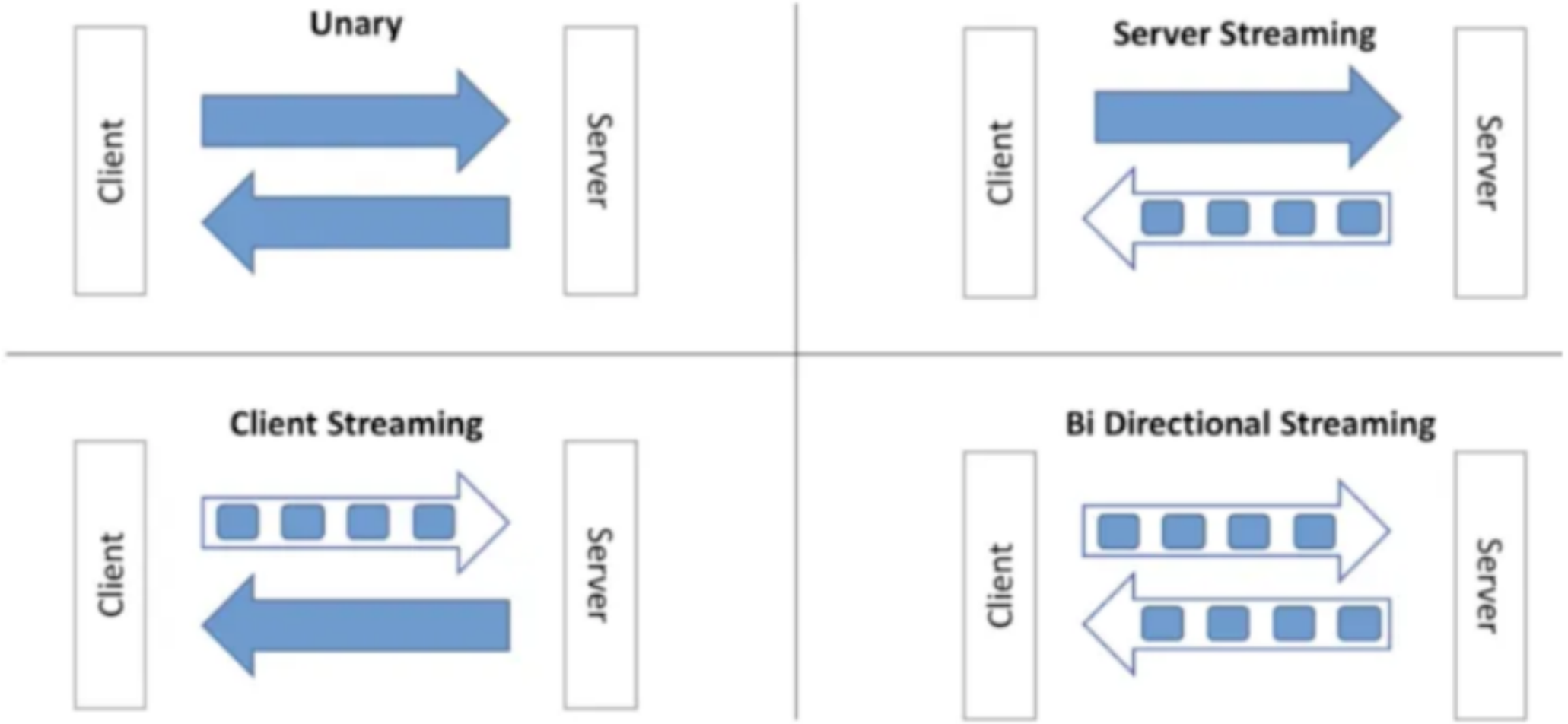
Four kinds of service methods

- Bidirectional streaming RPCs
  - Both sides send a sequence of messages using a read-write stream
  - The two streams operate independently
    - Clients and servers can read and write in whatever order they like
  - (order of messages in each stream is preserved)

```
rpc BidiHello(stream HelloRequest) returns (stream HelloResponse);
```

gRPC Remote Procedure Calls

Four kinds of service methods



Getting Started with Protobuf

Installing the toolchain

- brew install protobuf
- protoc --version
  - Can generate code for: C++, C#, Python, …
- Go requires a plugin:
- go install google.golang.org/protobuf/cmd/protoc-gen-go@latest
- go install google.golang.org/grpc/cmd/protoc-gen-go-grpc@latest

Joke time

Why is gRPC so fast?

Because it skips the small talk and gets straight to the point with binary protobufs!

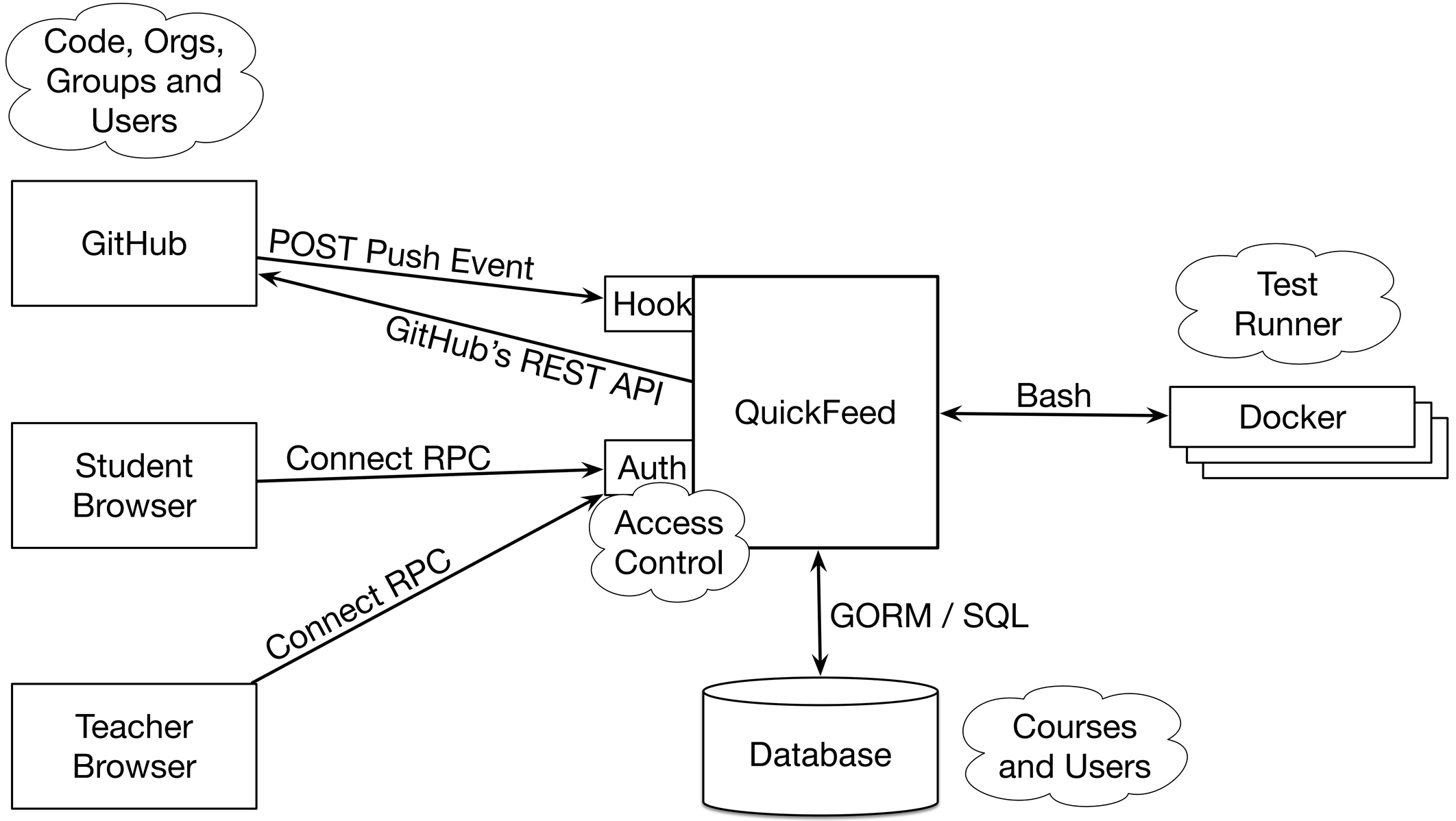
Buf's Connect RPC

Connect RPC

Alternative to Google's gRPC implementation

- Set of libraries to build browser and **gRPC-compatible** HTTP APIs
- Generates code idiomatic and type-safe for
  - Marshaling, routing, compression, and content-type negotiation
- Supported programming languages
  - Go, JavaScript/TypeScript, Java/Kotlin, Python, Swift, and Rust

QuickFeed Overview



QuickFeed Service Spec

```
syntax = "proto3";
package qf;
option go_package = "github.com/quickfeed/quickfeed/qf";
import "qf/types.proto";
import "qf/requests.proto";

service QuickFeedService {
    rpc GetUser(Void) returns (User) {}
    rpc GetUsers(Void) returns (Users) {}
    rpc UpdateUser(User) returns (Void) {}

    rpc GetGroup(GroupRequest) returns (Group) {}
    rpc GetGroupsByCourse(CourseRequest) returns (Groups) {}
    rpc CreateGroup(Group) returns (Group) {}
    rpc UpdateGroup(Group) returns (Group) {}
    rpc DeleteGroup(GroupRequest) returns (Void) {}

    rpc GetCourse(CourseRequest) returns (Course) {}
    rpc GetCourses(Void) returns (Courses) {}
    rpc UpdateCourse(Course) returns (Void) {}
    rpc UpdateCourseVisibility(Enrollment) returns (Void) {}

    rpc GetAssignments(CourseRequest) returns (Assignments) {}
    rpc UpdateAssignments(CourseRequest) returns (Void) {}

    rpc GetEnrollments(EnrollmentRequest) returns (Enrollments) {}
    rpc CreateEnrollment(Enrollment) returns (Void) {}
    rpc UpdateEnrollments(Enrollments) returns (Void) {}

    rpc GetSubmission(SubmissionRequest) returns (Submission) {}
    rpc GetSubmissions(SubmissionRequest) returns (Submissions) {}
    rpc GetSubmissionsByCourse(SubmissionRequest) returns (CourseSubmissions) {}
    rpc UpdateSubmission(UpdateSubmissionRequest) returns (Void) {}
    rpc UpdateSubmissions(UpdateSubmissionsRequest) returns (Void) {}
    rpc RebuildSubmissions(RebuildRequest) returns (Void) {}

    rpc CreateBenchmark(GradingBenchmark) returns (GradingBenchmark) {}
    rpc UpdateBenchmark(GradingBenchmark) returns (Void) {}
    rpc DeleteBenchmark(GradingBenchmark) returns (Void) {}

    rpc CreateCriterion(GradingCriterion) returns (GradingCriterion) {}
    rpc UpdateCriterion(GradingCriterion) returns (Void) {}
    rpc DeleteCriterion(GradingCriterion) returns (Void) {}

    rpc CreateReview(ReviewRequest) returns (Review) {}
    rpc UpdateReview(ReviewRequest) returns (Review) {}

    rpc GetOrganization(Organization) returns (Organization) {}
    rpc GetRepositories(CourseRequest) returns (Repositories) {}
    rpc IsEmptyRepo(RepositoryRequest) returns (Void) {}
    rpc SubmissionStream(Void) returns (stream Submission) {}
}
```

QuickFeed Service Spec

```
syntax = "proto3";
package qf;
option go_package = "github.com/quickfeed/quickfeed/qf";
import "qf/types.proto";
import "qf/requests.proto";

service QuickFeedService {
    rpc GetUser(Void) returns (User) {}
    rpc GetUsers(Void) returns (Users) {}
    rpc UpdateUser(User) returns (Void) {}

    rpc GetGroup(GroupRequest) returns (Group) {}
    rpc GetGroupsByCourse(CourseRequest) returns (Groups) {}
    rpc CreateGroup(Group) returns (Group) {}
    rpc UpdateGroup(Group) returns (Group) {}
    rpc DeleteGroup(GroupRequest) returns (Void) {}

    rpc GetCourse(CourseRequest) returns (Course) {}
    rpc GetCourses(Void) returns (Courses) {}
    rpc UpdateCourse(Course) returns (Void) {}
    rpc UpdateCourseVisibility(Enrollment) returns (Void) {}

    rpc GetAssignments(CourseRequest) returns (Assignments) {}
    rpc UpdateAssignments(CourseRequest) returns (Void) {}

    rpc GetEnrollments(EnrollmentRequest) returns (Enrollments) {}
    rpc CreateEnrollment(Enrollment) returns (Void) {}
    rpc UpdateEnrollments(Enrollments) returns (Void) {}
```

```
message CourseSubmissions {
    map<uint64, Submissions> submissions = 1;
}

message ReviewRequest {
    uint64 courseID = 1;
    Review review   = 2;
}

message CourseRequest {
    uint64 courseID = 1;
}

message GroupRequest {
    uint64 courseID = 1;
    uint64 userID   = 2;
    uint64 groupID  = 3;
}
```

QuickFeed Service Spec

```
rpc GetSubmission(SubmissionRequest) returns (Submission) {}
rpc GetSubmissions(SubmissionRequest) returns (Submissions) {}
rpc GetSubmissionsByCourse(SubmissionRequest) returns (CourseSubmissions) {}
rpc UpdateSubmission(UpdateSubmissionRequest) returns (Void) {}
rpc UpdateSubmissions(UpdateSubmissionsRequest) returns (Void) {}
rpc RebuildSubmissions(RebuildRequest) returns (Void) {}
```

```
message SubmissionRequest {
    enum SubmissionType {
        ALL   = 0;  // fetch all submissions
        USER  = 1;  // fetch all user submissions
        GROUP = 2;  // fetch all group submissions
    }
    uint64 CourseID     = 1;
    uint64 AssignmentID = 2;  // only used for user and group submissions
    oneof FetchMode {
        uint64 UserID       = 3;  // fetch single user's submissions with build info
        uint64 GroupID      = 4;  // fetch single group's submissions with build info
        uint64 SubmissionID = 5;  // fetch single specific submission with build info
        SubmissionType Type = 6;  // fetch all submissions of given type without build info
    }
}
```

QuickFeed Service Implementation

```go
// GetCourse returns course information for the given course.
func (s *QuickFeedService) GetCourse(_ context.Context, in *connect.Request[qf.CourseRequest]) (*connect.Response[qf.Course], error) {
    course, err := s.db.GetCourse(in.Msg.GetCourseID(), false)
    if err != nil {
        s.logger.Errorf("GetCourse failed: %v", err)
        return nil, connect.NewError(connect.CodeNotFound, errors.New("course not found"))
    }
    return connect.NewResponse(course), nil
}

// GetCourses returns a list of all courses.
func (s *QuickFeedService) GetCourses(_ context.Context, _ *connect.Request[qf.Void]) (*connect.Response[qf.Courses], error) {
    courses, err := s.db.GetCourses()
    if err != nil {
        s.logger.Errorf("GetCourses failed: %v", err)
        return nil, connect.NewError(connect.CodeNotFound, errors.New("no courses found"))
    }
    return connect.NewResponse(&qf.Courses{
        Courses: courses,
    }), nil
}
```

QuickFeed Service Implementation

```go
// UpdateCourse changes the course information details.
func (s *QuickFeedService) UpdateCourse(ctx context.Context, in *connect.Request[qf.Course]) (*connect.Response[qf.Void], error) {
    scmClient, err := s.getSCM(ctx, in.Msg.ScmOrganizationName)
    if err != nil {
        s.logger.Errorf("UpdateCourse failed: could not create scm client for organization %s: %v", in.Msg.ScmOrganizationName, err)
        return nil, connect.NewError(connect.CodeNotFound, err)
    }
    if err = s.updateCourse(ctx, scmClient, in.Msg); err != nil {
        s.logger.Errorf("UpdateCourse failed: %v", err)
        if ctxErr := ctxErr(ctx); ctxErr != nil {
            s.logger.Error(ctxErr)
            return nil, ctxErr
        }
        if ok, parsedErr := parseSCMError(err); ok {
            return nil, parsedErr
        }
        return nil, connect.NewError(connect.CodeInvalidArgument, errors.New("failed to update course"))
    }
    return &connect.Response[qf.Void]{}, nil
}
```

QuickFeed Service Implementation

```go
// updateCourse updates an existing course.
func (s *QuickFeedService) updateCourse(ctx context.Context, sc scm.SCM, request *qf.Course) error {
    // ensure the course exists
    _, err := s.db.GetCourse(request.ID, false)
    if err != nil {
        return err
    }
    // ensure the organization exists
    org, err := sc.GetOrganization(ctx, &scm.OrganizationOptions{ID: request.ScmOrganizationID})
    if err != nil {
        return err
    }
    request.ScmOrganizationName = org.GetScmOrganizationName()
    return s.db.UpdateCourse(request)
}
```

Connect RPC

- Connect RPC libraries are committed to being 100% spec conformant with
  - gRPC and gRPC-web protocols
  - TypeScript

Frontend using gRPC-Web

```
export const getGroup = async ({ state, effects }: Context, enrollment: Enrollment): Promise<void> => {
    const response = await effects.api.client.getGroup({ courseID: enrollment.courseID, groupID: enrollment.groupID })
    if (response.error) {
        return
    }
    state.userGroup[enrollment.courseID.toString()] = response.message
}
```

Frontend using gRPC-Web

```
export const createGroup = async ({ state, actions, effects }: Context, group: CourseGroup): Promise<void> => {
    const check = validateGroup(group)
    if (!check.valid) {
        actions.alert({ text: check.message, color: Color.RED, delay: 10000 })
        return
    }

    const response = await effects.api.client.createGroup({
        courseID: group.courseID,
        name: group.name,
        users: group.users.map(userID => new User({ ID: userID }))
    })

    if (response.error) {
        return
    }

    state.userGroup[group.courseID.toString()] = response.message
    state.activeGroup = null
}
```

Buf Command

Buf is a tool for Protobuf API management

- Formatting
- Linting
- Breaking change detection
- Code generation

https://buf.build/

brew install bufbuild/buf/buf

Buf Schema Registry

Stores and Manages Protobuf files

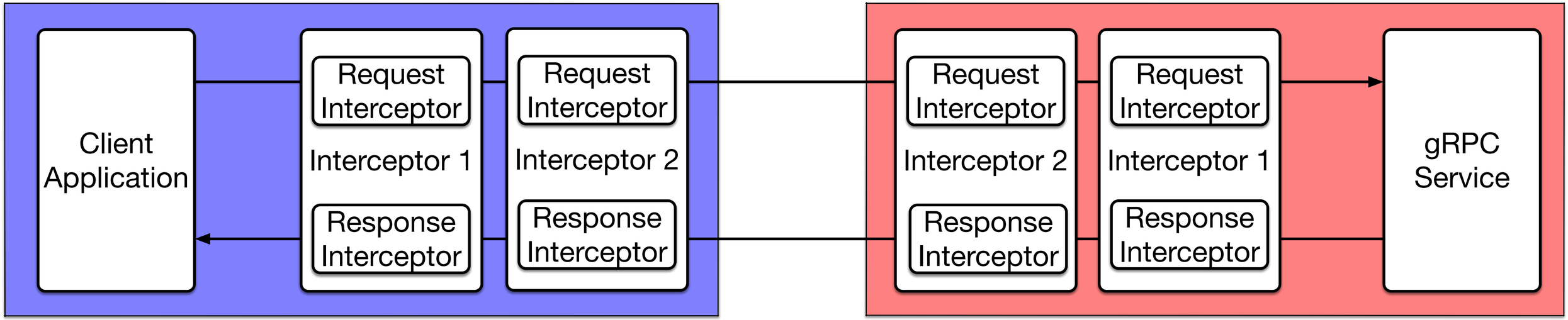
- Versioned modules
- Browseable UI
- Dependency management
- API validation
- Generated documentation
- Extensible plugin system

https://buf.build/

brew install bufbuild/buf/buf

gRPC / Connect RPC Interceptors

Interceptors



QuickFeed's Interceptors

```go
func (s *QuickFeedService) NewQuickFeedHandler(tm *auth.TokenManager) (string, http.Handler) {
    interceptors := connect.WithInterceptors(
        interceptor.NewMetricsInterceptor(),
        interceptor.NewValidationInterceptor(s.logger),
        interceptor.NewTokenAuthInterceptor(s.logger, tm, s.db),
        interceptor.NewUserInterceptor(s.logger, tm),
        interceptor.NewAccessControlInterceptor(tm),
        interceptor.NewTokenInterceptor(tm),
    )
    return qfconnect.NewQuickFeedServiceHandler(s, interceptors)
}
```

QuickFeed's Interceptors

```go
// WrapUnary returns a new unary server interceptor that validates requests
// that implements the validator interface.
// Invalid requests are rejected without logging and before it reaches any
// user-level code and returns an illegal argument to the client.
// Further, the response values are cleaned of any remote IDs.
// In addition, the interceptor also implements a cancellation mechanism.
func (v *ValidationInterceptor) WrapUnary(next connect.UnaryFunc) connect.UnaryFunc {
    return connect.UnaryFunc(func(ctx context.Context, request connect.AnyRequest) (connect.AnyResponse, error) {
        if request.Any() != nil {
            if err := validate(v.logger, request.Any()); err != nil {
                // Reject the request if it is invalid.
                return nil, err
            }
        }
        resp, err := next(ctx, request)
        if err != nil {
            // Do not return the message to the client if an error occurs.
            // We log the error and return an empty response.
            v.logger.Errorf("Method '%s' failed: %v", request.Spec().Procedure, err)
            v.logger.Errorf("Request Message: %T: %v", request.Any(), request.Any())
            return nil, err
        }
        clean(resp.Any())
        return resp, err
    })
}
```

QuickFeed's Interceptors

```go
func validate(logger *zap.SugaredLogger, req interface{}) error {
    if v, ok := req.(validator); ok {
        if !v.IsValid() {
            return connect.NewError(connect.CodeInvalidArgument, errors.New("invalid payload"))
        }
    } else {
        // just logging, but still handling the call
        logger.Debugf("message type %T does not implement validator interface", req)
    }
    return nil
}

func clean(resp interface{}) {
    if resp != nil {
        if v, ok := resp.(idCleaner); ok {
            v.RemoveRemoteID()
        }
    }
}
```

API Standardization

OpenAPI Specification (OAS)

Defines a standard, language-agnostic interface to HTTP APIs

- Allow both humans and computers to
  - Discover and understand the capabilities of the service
  - Without access to source code or documentation
- Definition can be used to generate
  - Documentation, code, testing tools etc
- https://spec.openapis.org/oas/v3.1.0.html

Vendor Specific APIs

Google APIs Explorer

- https://developers.google.com/apis-explorer/
- https://developers.google.com/discovery
- https://developers.google.com/api-client-library
- https://github.com/googleapis/google-api-go-client
- https://github.com/googleapis/google-api-go-client/blob/main/GettingStarted.md

Cloud APIs

- Google
  - Google Compute Engine, AutoML API, Google Calendar API, Cloud Filestore API, Cloud Firestore API,…
- Amazon APIs
  - AWS Cloud Control, EventBridge, Amazon S3
- Azure APIs
  - Azure Communication Services, Azure Cognitive Services, and Azure API Management

Accessing Amazon S3 Storage

Using AWS Go API

- https://aws.github.io/aws-sdk-go-v2/docs/getting-started/

```go
package main

import (
    "context"
    "log"
    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
)

func main() {
    // Load the Shared AWS Configuration (~/.aws/config)
    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Fatal(err)
    }

    // Create an Amazon S3 service client
    client := s3.NewFromConfig(cfg)

    // Get the first page of results for ListObjectsV2 for a bucket
    output, err := client.ListObjectsV2(context.TODO(), &s3.ListObjectsV2Input{
        Bucket: aws.String("my-bucket"),
    })
    if err != nil {
        log.Fatal(err)
    }

    log.Println("first page results:")
    for _, object := range output.Contents {
        log.Printf("key=%s size=%d", aws.ToString(object.Key), object.Size)
    }
}
```

Questions?

References

- https://grpc.io/
- https://buf.build/docs/introduction
- https://buf.build/