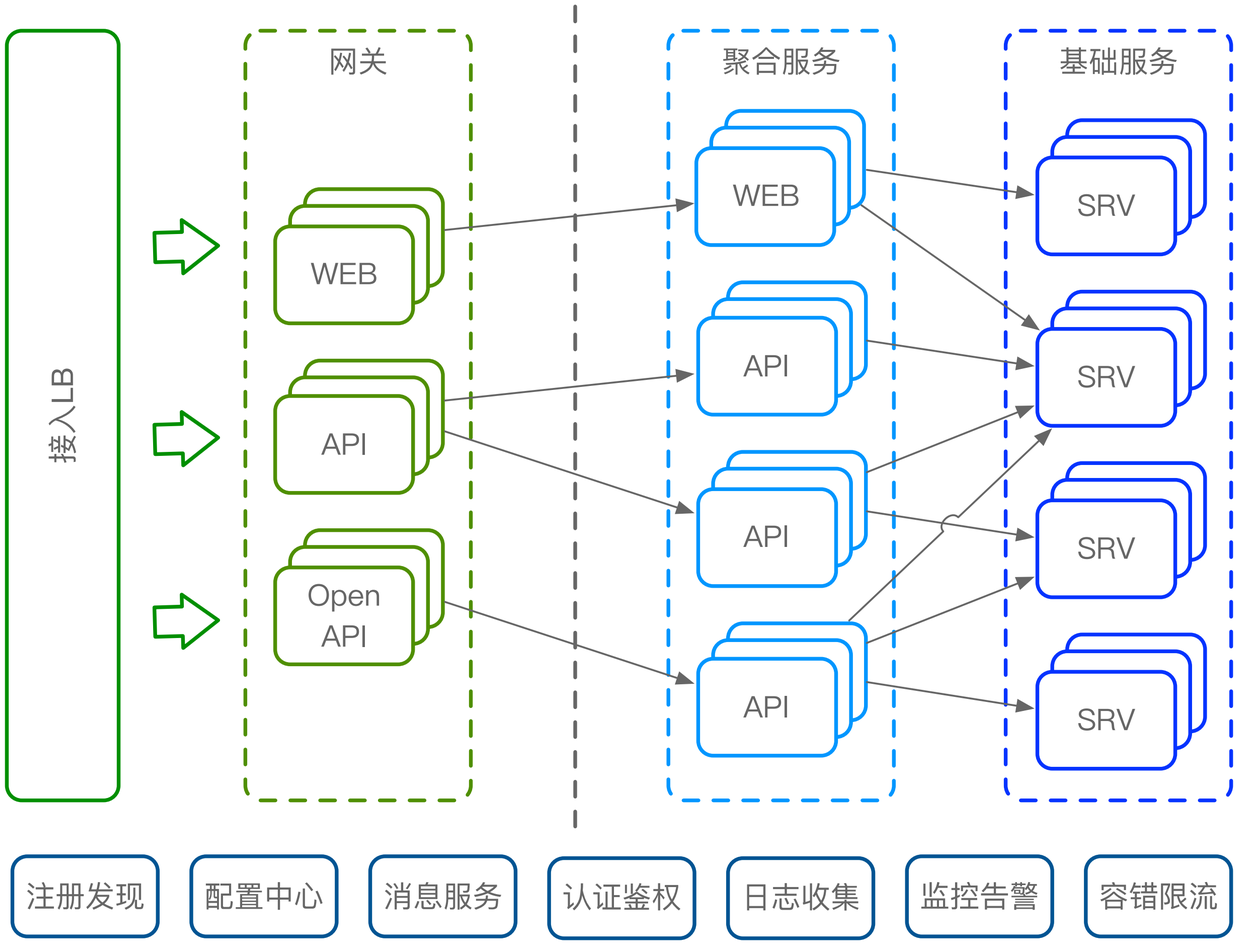
陈洪波

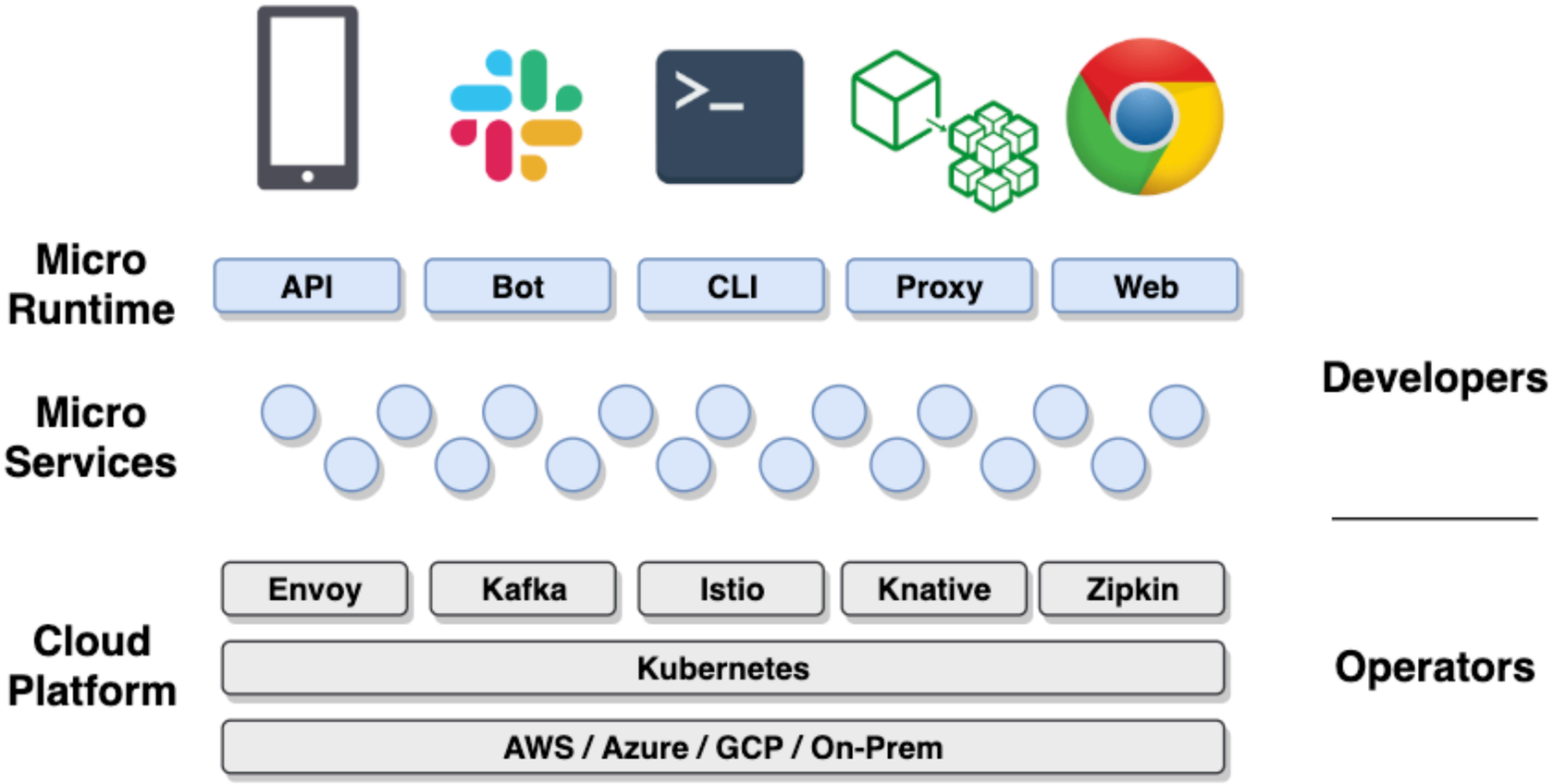
Micro框架与微服务实践

目录

微服务架构


接入LB
网关
WEB
API
Open API
聚合服务
WEB
API
API
API
基础服务
SRV
SRV
SRV
SRV
注册发现
配置中心
消息服务
认证鉴权
日志收集
监控告警
容错限流

Micro生态


Micro Runtime
API
Bot
CLI
Proxy
Web
Micro Services
Developers
Cloud Platform
Envoy
Kafka
Istio
Knative
Zipkin
Kubernetes
AWS / Azure / GCP / On-Prem
Operators

环境

- micro v1.11.0
- go-micro v1.11.3
- go-plugins v1.3.0
- go v1.12.5
- consul v1.2.0
- maxOS

工具安装

- micro
  - go get -u github.com/micro/micro
- protoc
  - brew install protobuf
- protoc-gen-go
  - go get -u github.com/golang/protobuf/{proto,protoc-gen-go}
- protoc-gen-micro
  - go get -u github.com/micro/protoc-gen-micro

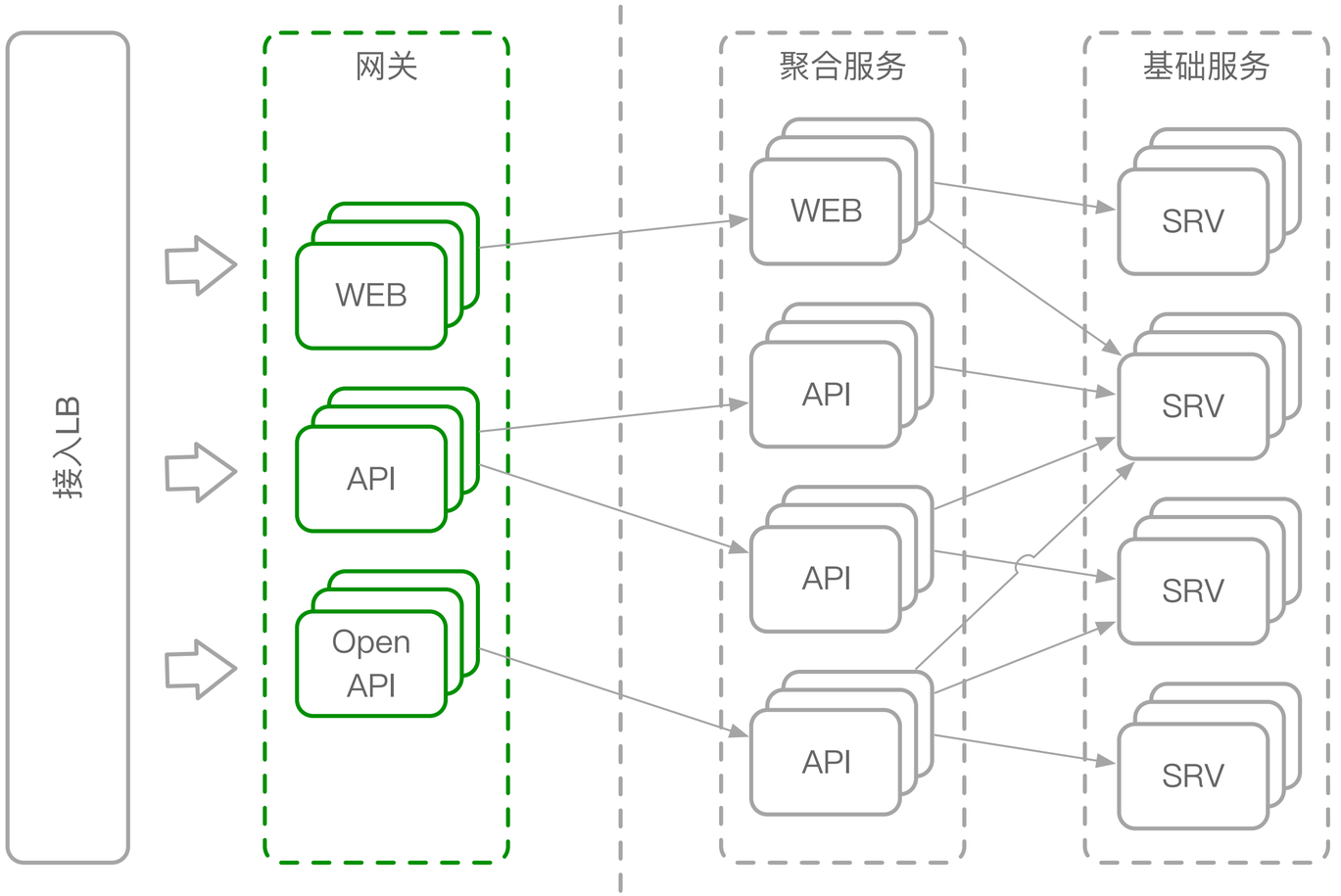
PROTOBUF

- 统一建模
- 定义服务边界
- 多语言代码生成

- https://developers.google.com/protocol-buffers/docs/overview

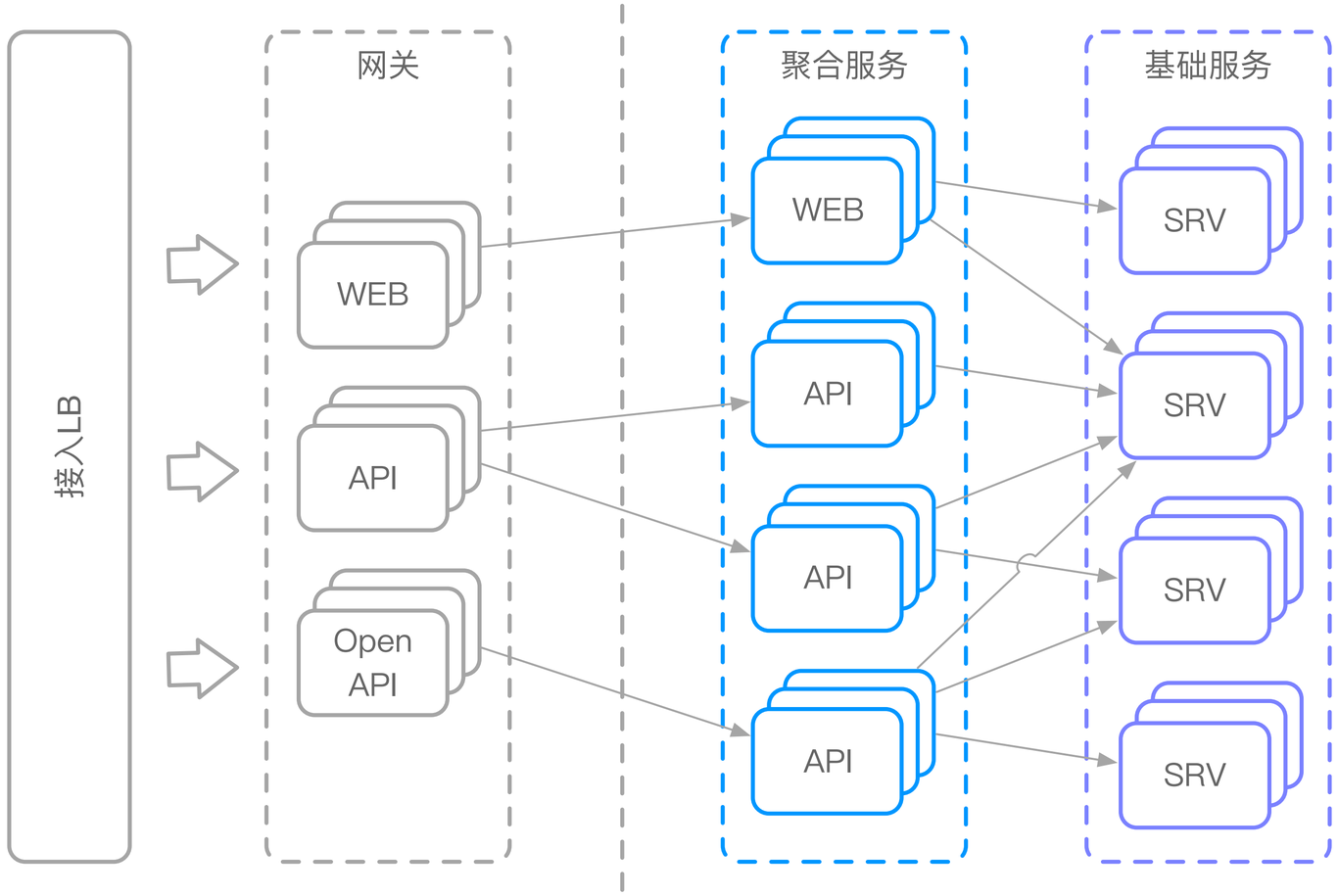
网关

- micro api
  - 默认端口8080
- micro web
  - 默认端口8082



TEMPLATE & NEW

- 服务模板
  - api
  - web
  - srv
  - fnc



其他

- health Query the health of a service
- stats Query the stats of a service
- list List items in registry
- get Get item from registry
- ……

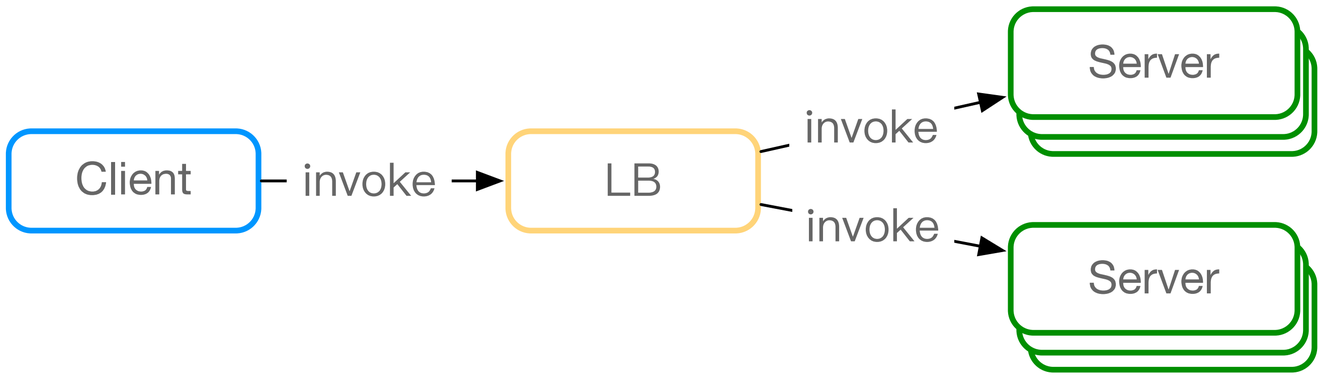
运行网关

- micro api
  - localhost:8080
- micro web
  - localhost:8082

- –enable_stats
- 其他
  - list
  - get
  - health
  - stats

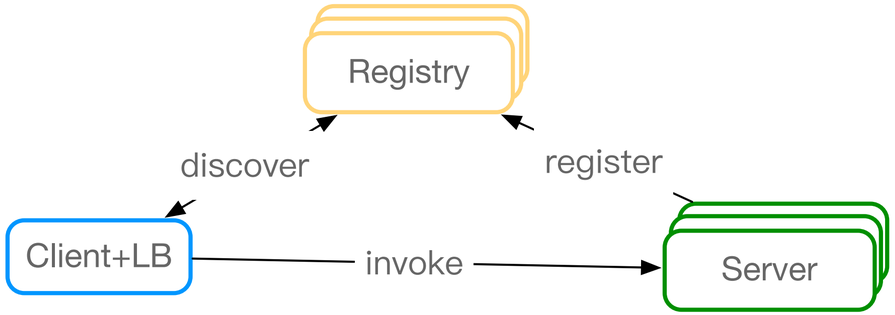
服务发现模式

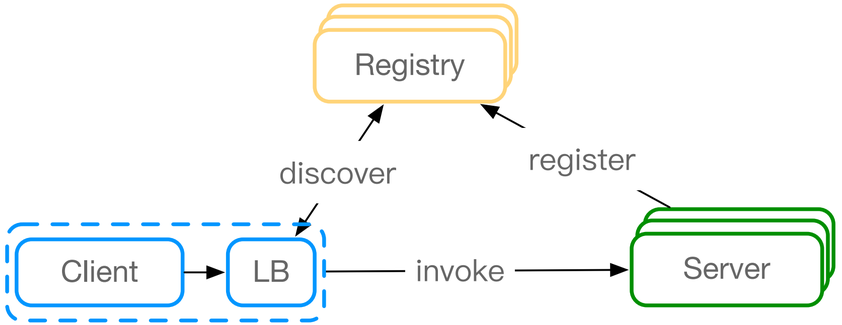
- 集中负载
  - 域名、URL转发策略
  - 服务分组管理



服务发现模式

- 分布式负载
  - 进程内
    - 多语言
    - Dubbo、Micro
  - 独立进程
    - 服务网格
    - Istio、Linkerd2





源码

registry.go

```go
type Registry interface {
   Init(...Option) error
   Options() Options
   Register(*Service, ...RegisterOption) error
   Deregister(*Service) error
   GetService(string) ([]*Service, error)
   ListServices() ([]*Service, error)
   Watch(...WatchOption) (Watcher, error)
   String() string
}

type Option func(*Options)

type RegisterOption func(*RegisterOptions)

type WatchOption func(*WatchOptions)
```

SERVICE

- 创建Service
  - NewServie
  - Init
  - Run

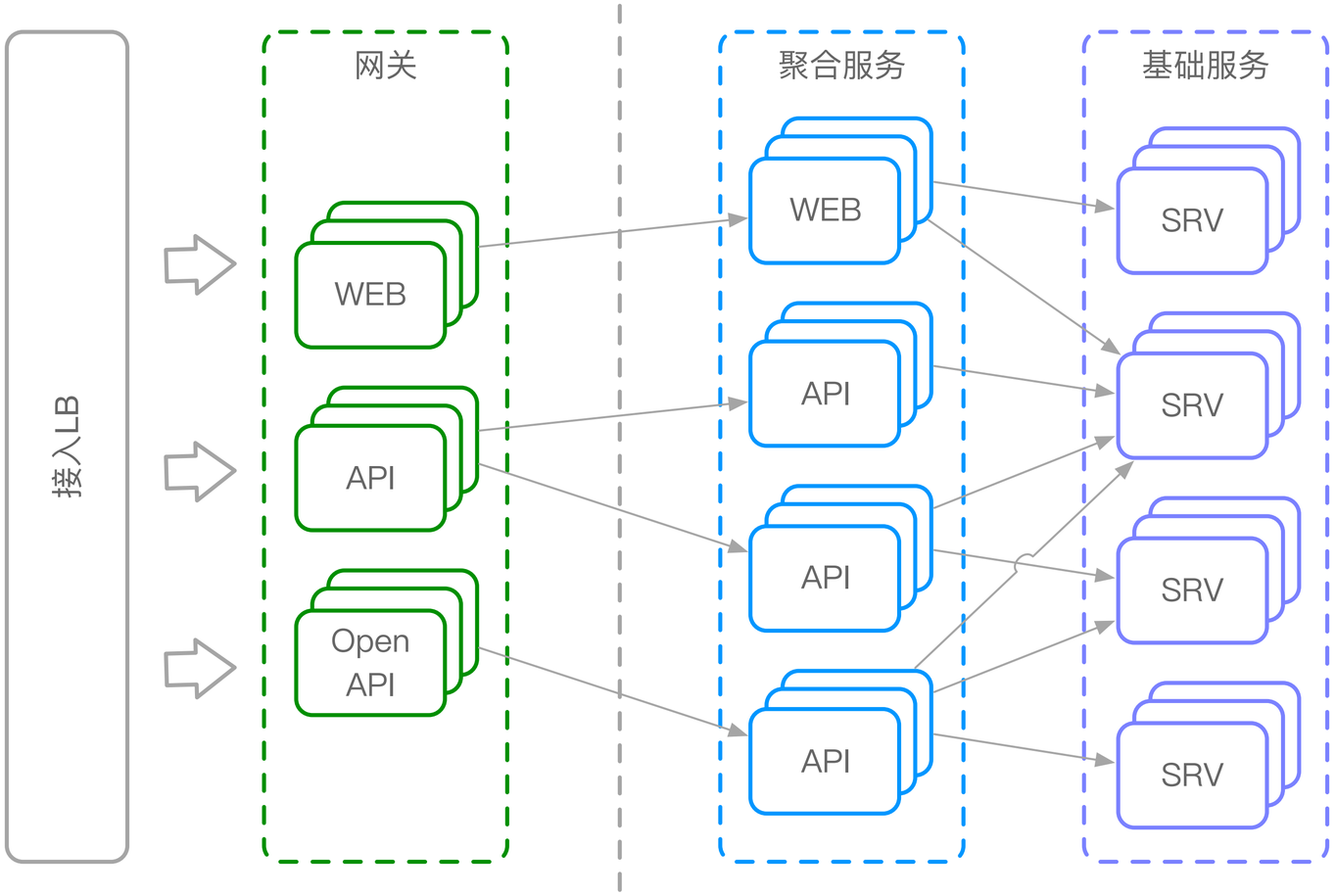

```go
package main

import (
        "github.com/micro/go-micro"
)

func main() {
        service := micro.NewService(
                micro.Name("helloworld"),
        )
        service.Run()
}
```

- 坑:Options顺序
  - server & client
    - broker
      - registry
    - transport
    - selector
    - name、version……

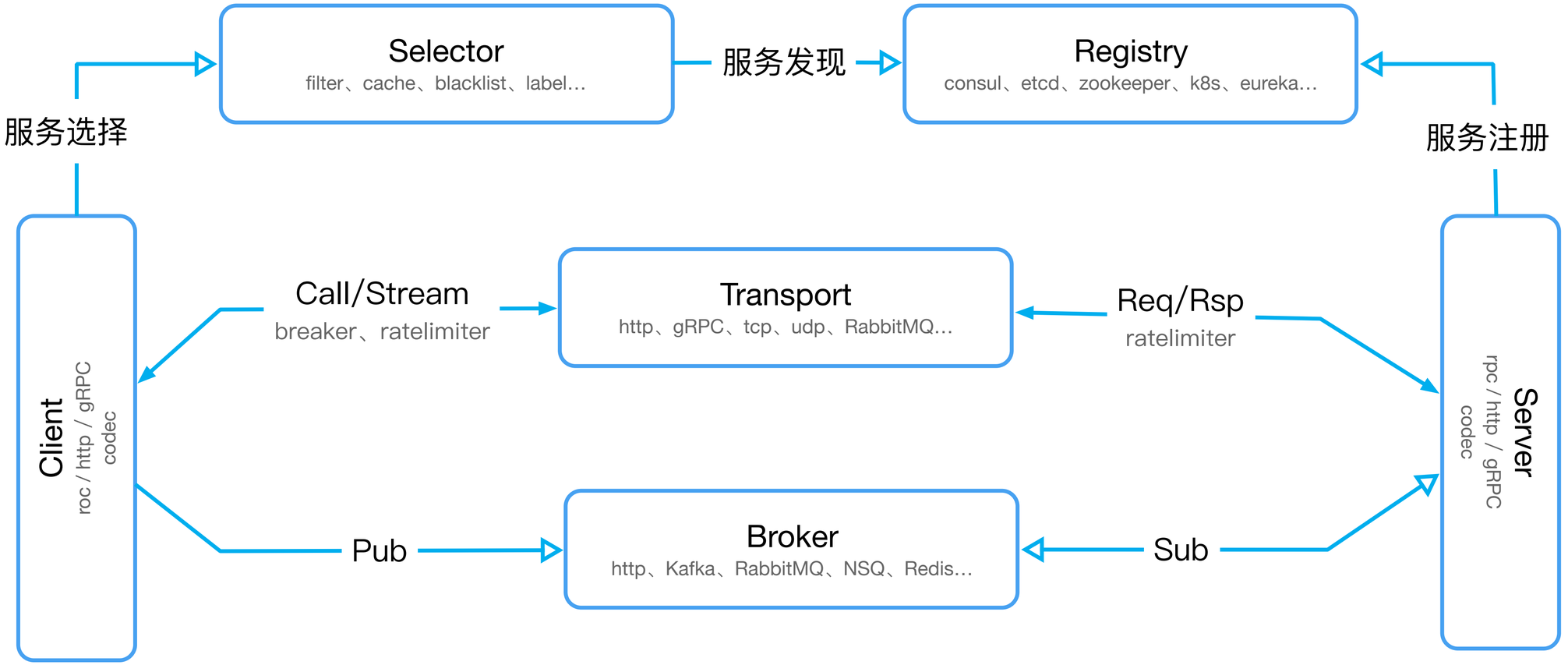
服务创建

- 创建api&srv
  - micro new --type api --alias example github.com/hb-go/micro-quick-start/example/api
  - micro new --type srv --alias example github.com/hb-go/micro-quick-start/example/srv
  - micro new --type web --alias example github.com/hb-go/micro-quick-start/example/web
- 代码生成
  - make proto
    - 或：protoc --proto_path=.:$GOPATH/src --go_out=. --micro_out=. proto/example/example.proto
  - github.com/micro/go-micro/api/proto/api.proto路径问题，替换目录或在src clone一份go-micro
- API endpoint定义
  - 不定义endpoint，REST请求参数异常
- curl http://127.0.0.1:8080/example/call
- curl -XPOST -H 'Content-Type: application/x-www-form-urlencoded' -d 'name=hobo' http://localhost:8080/example/call
- http://localhost:8082/example

SERVICE


接入LB
网关
WEB
API
Open
API
聚合服务
WEB
API
API
API
基础服务
SRV
SRV
SRV
SRV

服务框架


Selector
filter、cache、blacklist、label...
服务发现
Registry
consul、etcd、zookeeper、k8s、eureka...
服务选择
服务注册
Client
roc / http / gRPC
codec
Call/Stream
breaker、ratelimiter
Transport
http、gRPC、tcp、udp、RabbitMQ...
Req/Rsp
ratelimiter
Server
rpc / http / gRPC
codec
Pub
Broker
http、Kafka、RabbitMQ、NSQ、Redis...
Sub

SERVER & CLIENT

- rpc
  - transport
- grpc
- http

```
DefaultServers = map[string]func(...server.Option)
server.Server{
  "rpc":  server.NewServer,
  "mucp": smucp.NewServer,
  "grpc": sgrpc.NewServer,
}
```

```
DefaultClients = map[string]func(...client.Option)
client.Client{
  "rpc":  client.NewClient,
  "mucp": cmucp.NewClient,
  "grpc": cgrpc.NewClient,
}
```

CODEC

- 不同Server的codec支持有差别
- 多个服务不同编码
  - 自定义Client

- rpc内置编码

```
DefaultCodecs = map[string]codec.NewCodec{
  "application/grpc":         grpc.NewCodec,
  "application/grpc+json":    grpc.NewCodec,
  "application/grpc+proto":   grpc.NewCodec,
  "application/protobuf":     proto.NewCodec,
  "application/json":         json.NewCodec,
  "application/json-rpc":     jsonrpc.NewCodec,
  "application/proto-rpc":    protorpc.NewCodec,
  "application/octet-stream": raw.NewCodec,
}
```

TRANSPORT

- rpc协议支持不同transport

```
DefaultTransports = map[string]func(...transport.Option) transport.Transport{
  "memory": tmem.NewTransport,
  "http":   thttp.NewTransport,
  "grpc":   tgrpc.NewTransport,
}
```

自定义SERVER

- 默认 < 硬编码 < 命令行参数&ENV
- 选择插件
  - micro --client=grpc api
  - go run main.go --server=grpc --client=grpc
- 命令行指定Server，注意Name、Wrap等Options配置
- 强制硬编码?
  - Init()的cmd为once.Do()，可以先调用空Init()，再调用Init()自定义Server

自定义TRANSPORT

- 替换插件
  - micro --transport=tcp api
  - go run main.go plugin.go –transport=tcp
- HTTP vs TCP
  - hey -z 5s -c 50 “http://localhost:8080/example/call”

TRANSPORT + SERVER

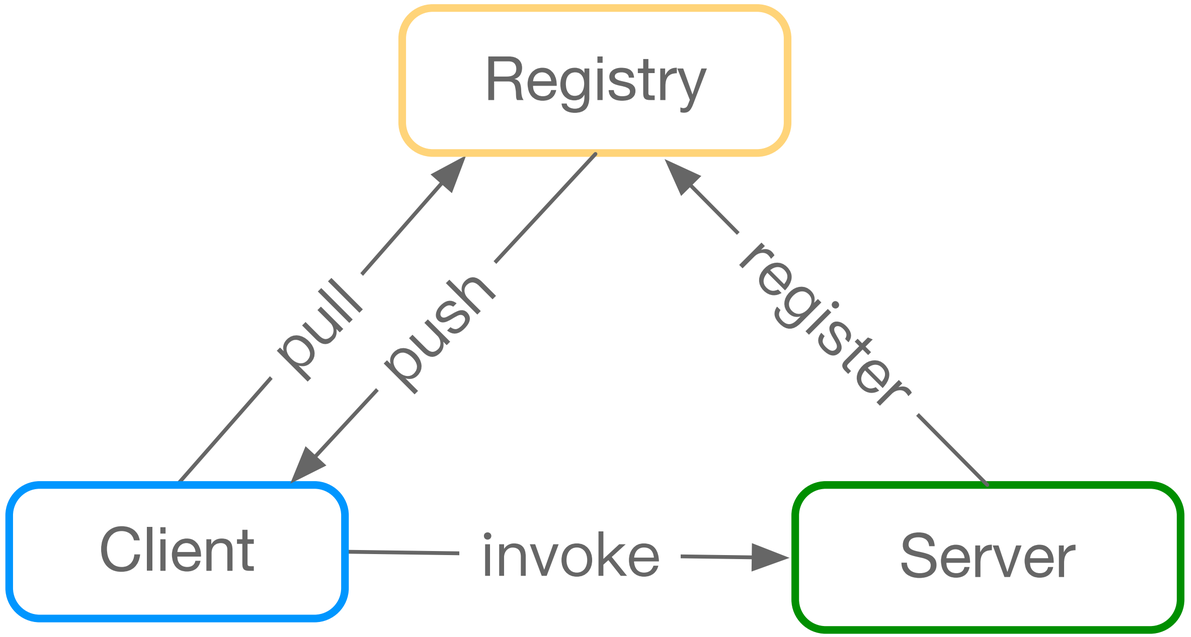
T+S	平均 (ms)	中位 (ms)	最大 (ms)	最小 (ms)	P90 (ms)	P99 (ms)	TPS
tcp+rpc	7.236	5.629	101.506	0.177	13.338	35.880	13192
grpc+rpc	8.668	7.964	101.280	0.251	12.744	21.672	11166
utp+rpc	11.824	11.600	53.183	0.204	15.575	21.334	8252
grpc+grpc	8.924	8.181	134.434	0.286	13.211	22.973	10845

CODEC

CODEC	平均 (ms)	中位 (ms)	最大 (ms)	最小 (ms)	P90 (ms)	P99 (ms)	TPS
grpc	3.937	2.979	90.004	0.180	7.184	19.355	12310
grpc+json	6.085	4.694	149.861	0.342	10.365	31.837	8000
protobuf	3.661	2.707	96.636	0.156	6.542	20.261	13150
json	5.402	4.122	122.360	0.225	9.186	30.474	8896
json-rpc	6.380	4.878	115.141	0.288	11.150	33.395	7631
proto-rpc	3.692	2.729	101.010	0.180	6.701	19.454	13041
bsonrpc	7.912	5.979	132.041	0.354	14.789	40.414	6145

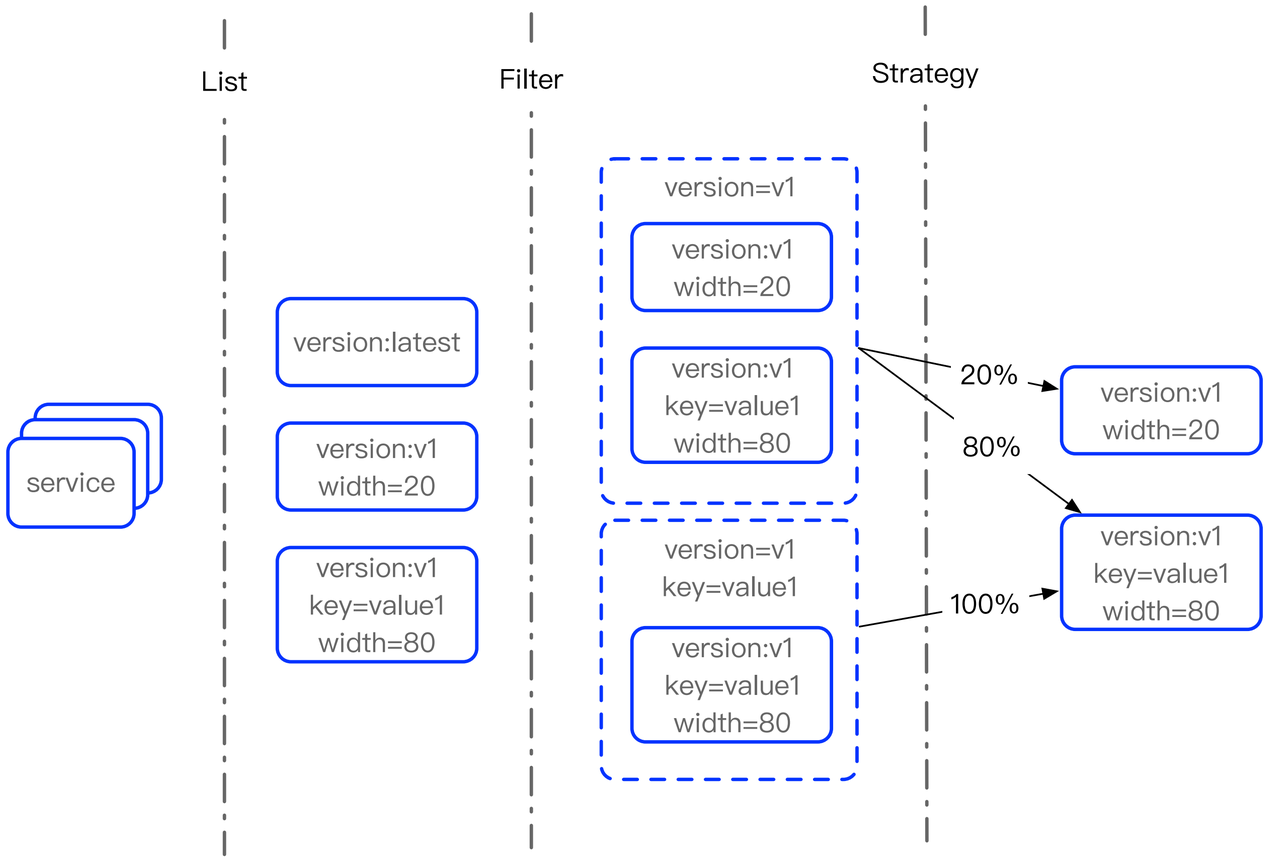
REGISTRY

- 注册中心
  - etcd、consul、mdns、k8s
- 接口
  - 注册、注销
  - 获取服务
  - Watch



SELECTOR

- List -> Filter -> Strategy
- 缓存 TTL
- 服务筛选
- 负载策略



注册中心

- Consul
  - consul agent -dev -advertise 127.0.0.1
  - http://localhost:8500
- micro --registry=consul api
- go run main.go –registry=consul

注册中心

- micro --registry=consul get service
- micro --registry=consul get service go.micro.api
  - go.micro.api-325ed78f-f37c-48bf-b8ff-3e3a822544c1 192.168.3.115 54173 server=rpc,registry=consul,protocol=mucp,transport=http,broker=http

服务筛选

- Filter

```
service.Init(
  func(o *micro.Options) {
    o.Client.Init(
      func(options *rc.Options) {
        options.CallOptions.SelectOptions = append(options.CallOptions.SelectOptions,
selector.WithFilter(selector.FilterVersion("v1")))
      },
    )
  },
)

client..WithSelectOption(
  selector.WithFilter(
    selector.FilterLabel("key", "value1"),
  ),
),
```

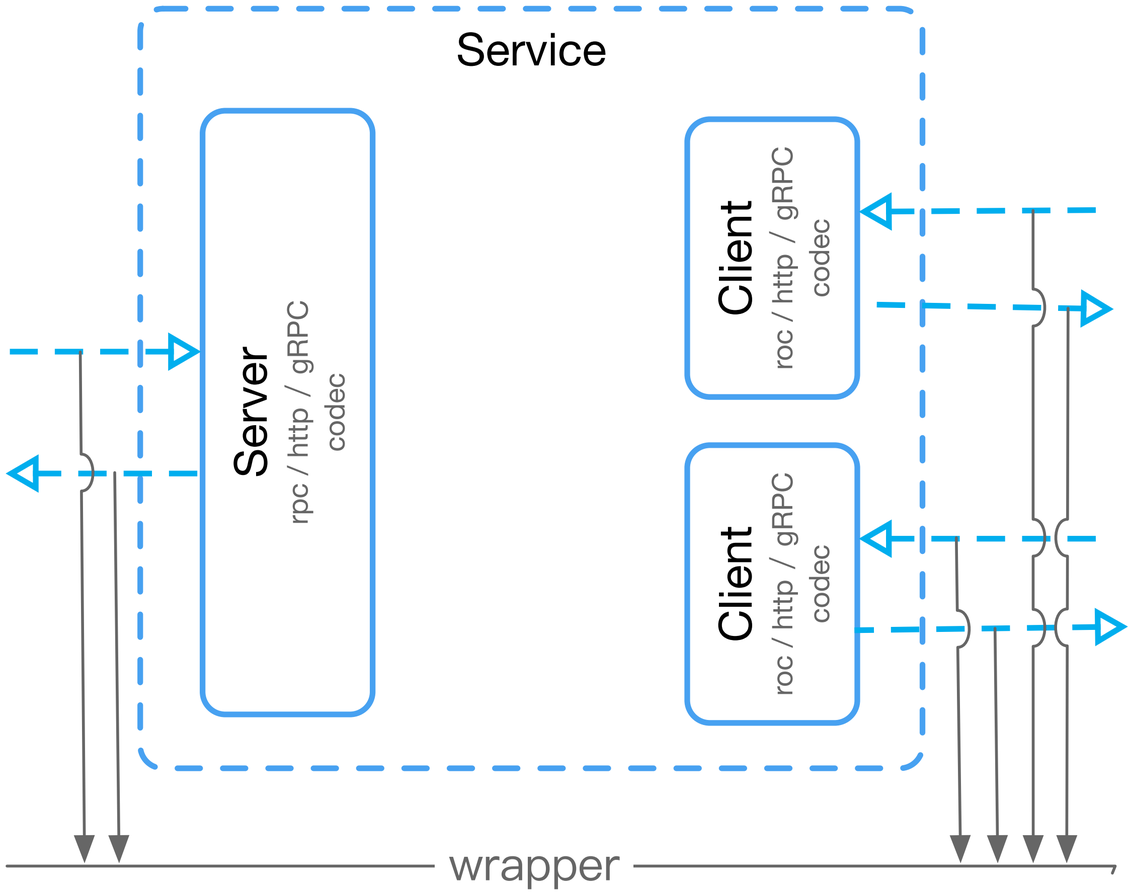
version	label
latest	-
v1	-
v1	key=value1

BROKER

- 略

WRAPPER

- Client
  - Call
  - 装饰Client
- Server
  - Handler
  - Subscriber
  - Stream
- github.com/micro/go-plugins/wrapper/*



自定义WRAPPER

- Context注入

```
service.Init(
  // create wrap for the Account srv client
  micro.WrapHandler(client.AccountWrapper(service)),
)
```

```
// Client returns a wrapper for the AccountClient
func AccountWrapper(service micro.Service) server.HandlerWrapper {
  client := account.NewAccountService("go.micro.srv.account", service.Client())

  return func(fn server.HandlerFunc) server.HandlerFunc {
    return func(ctx context.Context, req server.Request, rsp interface{}) error {
      ctx = context.WithValue(ctx, accountKey{}, client)
      return fn(ctx, req, rsp)
    }
  }
}
```

自定义WRAPPER

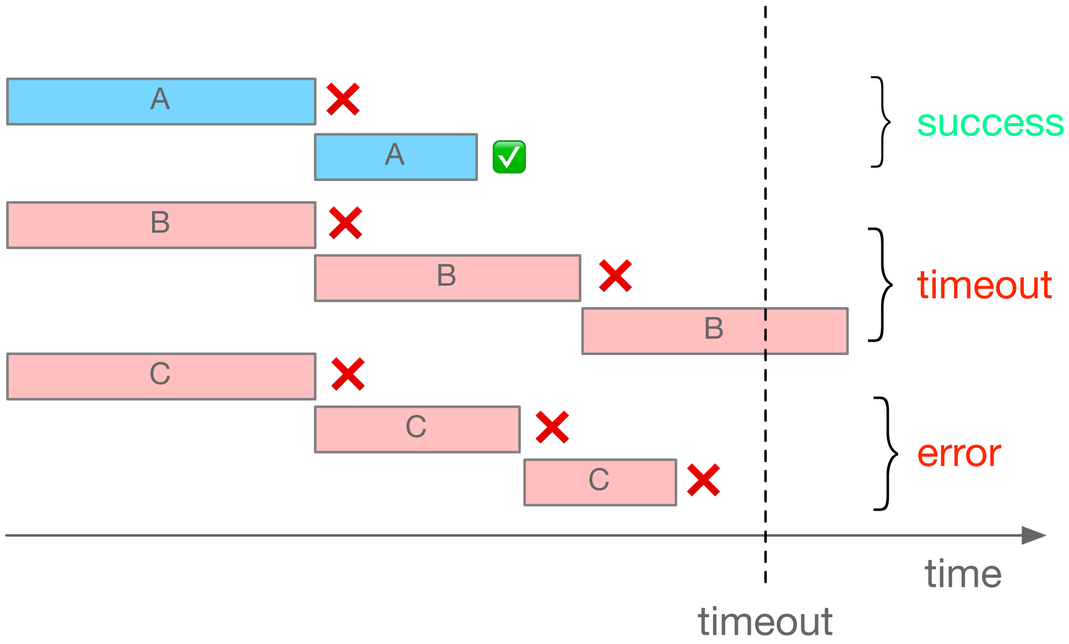
- 监控

```
service.Init(
  micro.WrapHandler(prometheus.NewHandlerWrapper()),
)

go func() {
  ls, err := net.Listen("tcp", ":9091")
  if err == nil {
    err := http.Serve(ls, promhttp.Handler())
    if err != nil {
      panic(err)
    }
  } else {
    panic(err)
  }
}()
```

- http://localhost:9091

重试 & 超时

- 分布式系统假设所有调用都可能失败
- 超时是多次重试累计时长
- 幂等



重试 & 超时

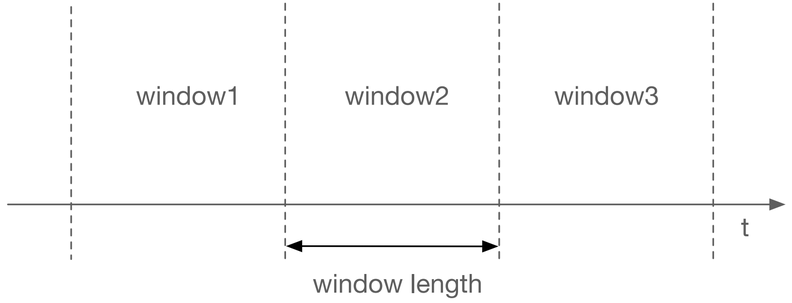
- Client Option

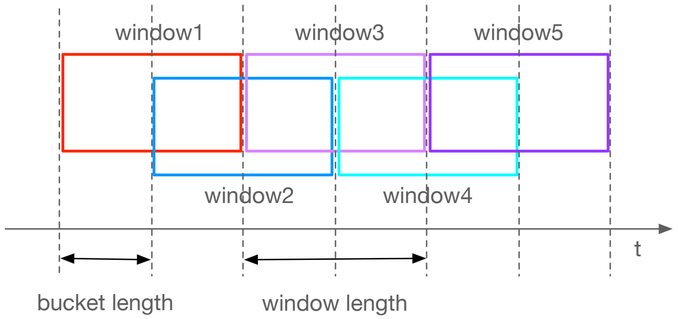
```
service.Init(
  func(o *micro.Options) {
    o.Client.Init(
      rc.Retries(3),
      rc.RequestTimeout(time.Millisecond*100),
    )
  },
)
```

- SRV注入错误
  - 1/3错误
  - 延时3s

限流 & 熔断 & 降级

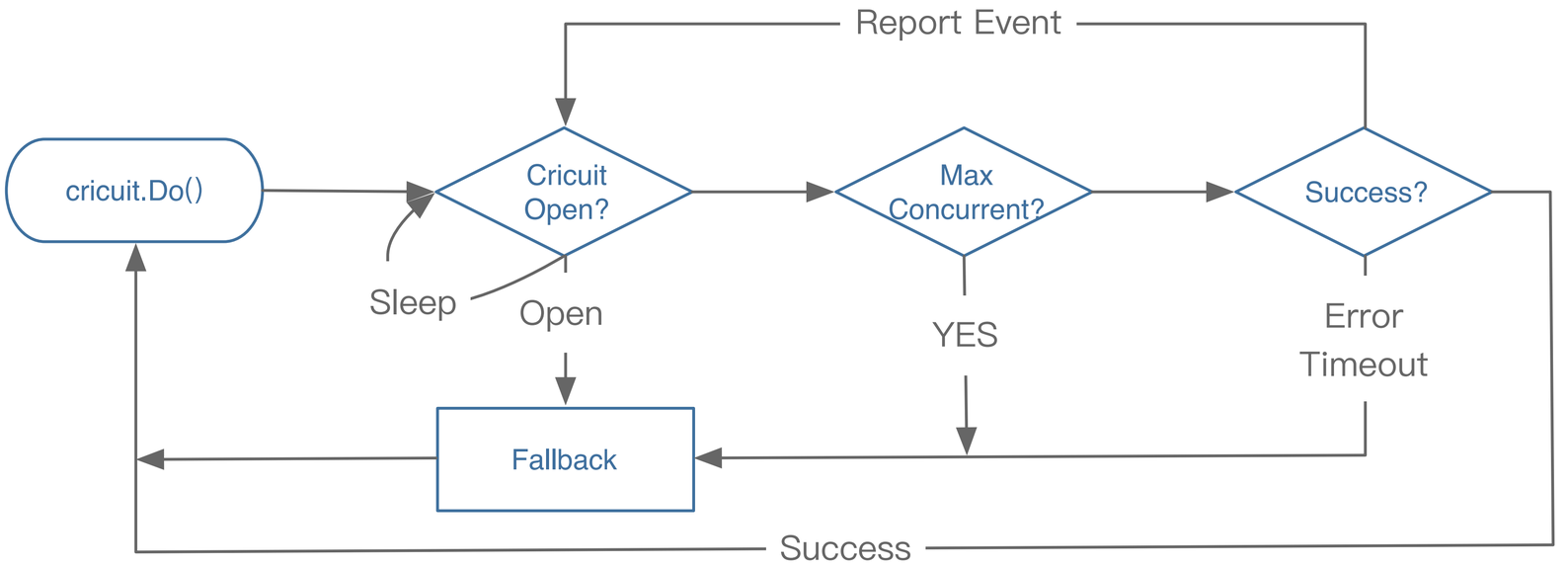
- 限流
  - Server / Client限流，注意**限流对象**，如:共享Client
  - Wait / FallFast
  - 分布式集中限流，如:OpenAPI限流





限流 & 熔断 & 降级

- 熔断
  - Service + Endpoint



限流 & 熔断 & 降级

- Wrap Client & Handler

```
service.Init(
  micro.WrapClient(NewClientWrapper()),
  micro.WrapClient(ratelimit.NewClientWrapper(10)),
  micro.WrapHandler(ratelimit.NewHandlerWrapper(10)),
)
```

- Wrap Client不要放在client.Init()
  - Client的wrappers只在new方法中有效，或者service.WrapClient()

```
type clientWrapper struct {
    client2.Client
}

func (c *clientWrapper) Call(ctx context.Context, req client2.Request, rsp
interface{}, opts ...client2.CallOption) error {
    return hystrix.Do(req.Service()+"."+req.Endpoint(), func() error {
        if cir, ok, _ := hystrix.GetCircuit(req.Service() + "." + req.Endpoint());
ok {
            log.Logf("circuit: %v %v", cir.Name, cir.AllowRequest())
        } else {
            log.Logf("circuit: %v %v", cir.Name, cir.AllowRequest())
        }

        return c.Client.Call(ctx, req, rsp, opts...)
    }, func(err error) error {
        log.Logf("fallback: %v", err)
        return err
    })
}

// NewClientWrapper returns a hystrix client Wrapper.
func NewClientWrapper() client2.Wrapper {
    return func(c client2.Client) client2.Client {
        return &clientWrapper{c}
    }
}
```

GO-MICRO内置插件

- go-micro/config/cmd/cmd.go

```
DefaultBrokers = map[string]func(...broker.Option)
broker.Broker{
   "http":   http.NewBroker,
   "memory": memory.NewBroker,
   "nats":   nats.NewBroker,
}
```

```
DefaultClients = map[string]func(...client.Option)
client.Client{
   "rpc":  client.NewClient,
   "mucp": cmucp.NewClient,
   "grpc": cgrpc.NewClient,
}
```

```
DefaultRegistries = map[string]func(...registry.Option)
registry.Registry{
   "consul": consul.NewRegistry,
   "gossip": gossip.NewRegistry,
   "mdns":   mdns.NewRegistry,
   "memory": rmem.NewRegistry,
}
```

```
DefaultSelectors = map[string]func(...selector.Option)
selector.Selector{
   "default": selector.NewSelector,
   "dns":     dns.NewSelector,
   "cache":   selector.NewSelector,
   "router":  router.NewSelector,
   "static":  static.NewSelector,
}
```

```
DefaultServers = map[string]func(...server.Option)
server.Server{
   "rpc":  server.NewServer,
   "mucp": smucp.NewServer,
   "grpc": sgrpc.NewServer,
}
```

```
DefaultTransports = map[string]func(...transport.Option)
transport.Transport{
   "memory": tmem.NewTransport,
   "http":   thttp.NewTransport,
   "grpc":   tgrpc.NewTransport,
}
```

GO-MICRO自定义插件

- github.com/micro/go-plugins/*

```
func init() {
  cmd.DefaultTransports["tcp"] = NewTransport
}
```

- micro --transport=tcp api
- go run main.go plugin.go –transport=tcp

MICRO自定义插件

- github.com/micro/go-plugins/micro/*

```
type Plugin interface {
  // Global Flags
  Flags() []cli.Flag
  // Sub-commands
  Commands() []cli.Command
  // Handle is the middleware handler for HTTP requests. We pass in
  // the existing handler so it can be wrapped to create a call chain.
  Handler() Handler
  // Init called when command line args are parsed.
  // The initialised cli.Context is passed in.
  Init(*cli.Context) error
  // Name of the plugin
  String() string
}
```

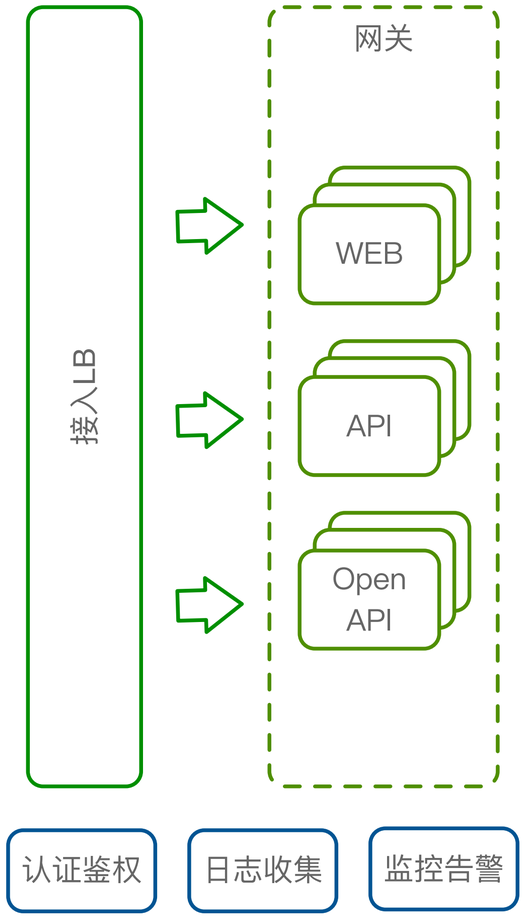
MICRO自定义PLUGIN

```
func init() {
 api.Register(metrics.NewPlugin())
 web.Register(metrics.NewPlugin())
}
```

- micro --registry=consul api --metrics=prometheus
- http://localhost:8080/metrics

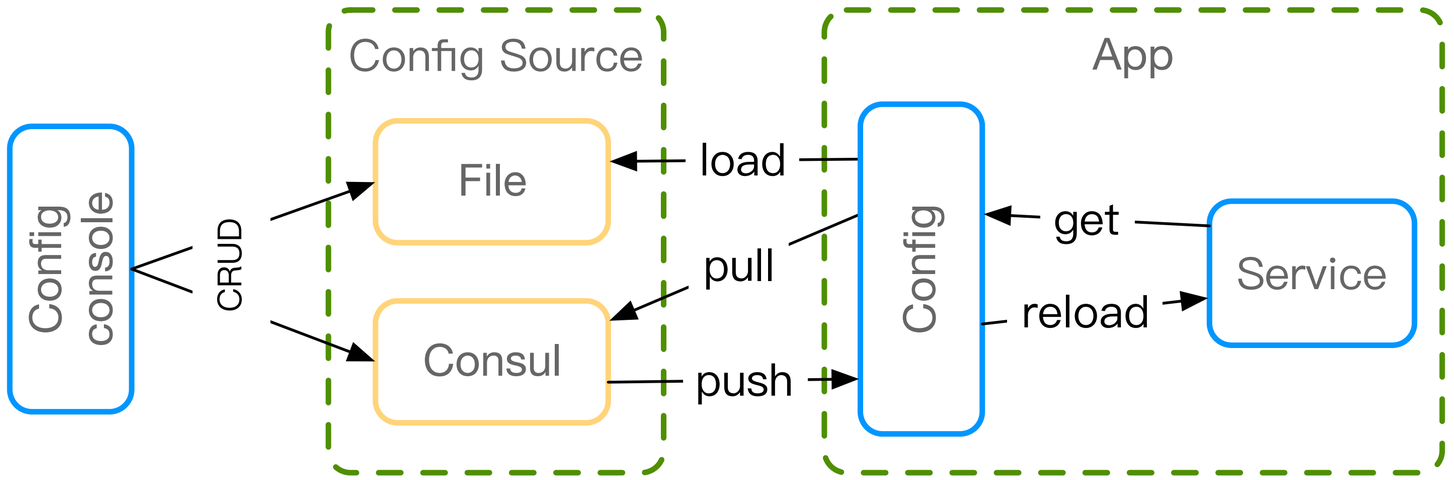
MICRO API/WEB

- 负载注册
- 认证安全 JWT SSO
- 访问控制 ACL RBAC
- 流控
- 日志监控
- ……



CONFIG

- 没有标准
  - 格式、管理
- 管理困难
  - 修改慢、难
- 安全审计
  - 问题追溯



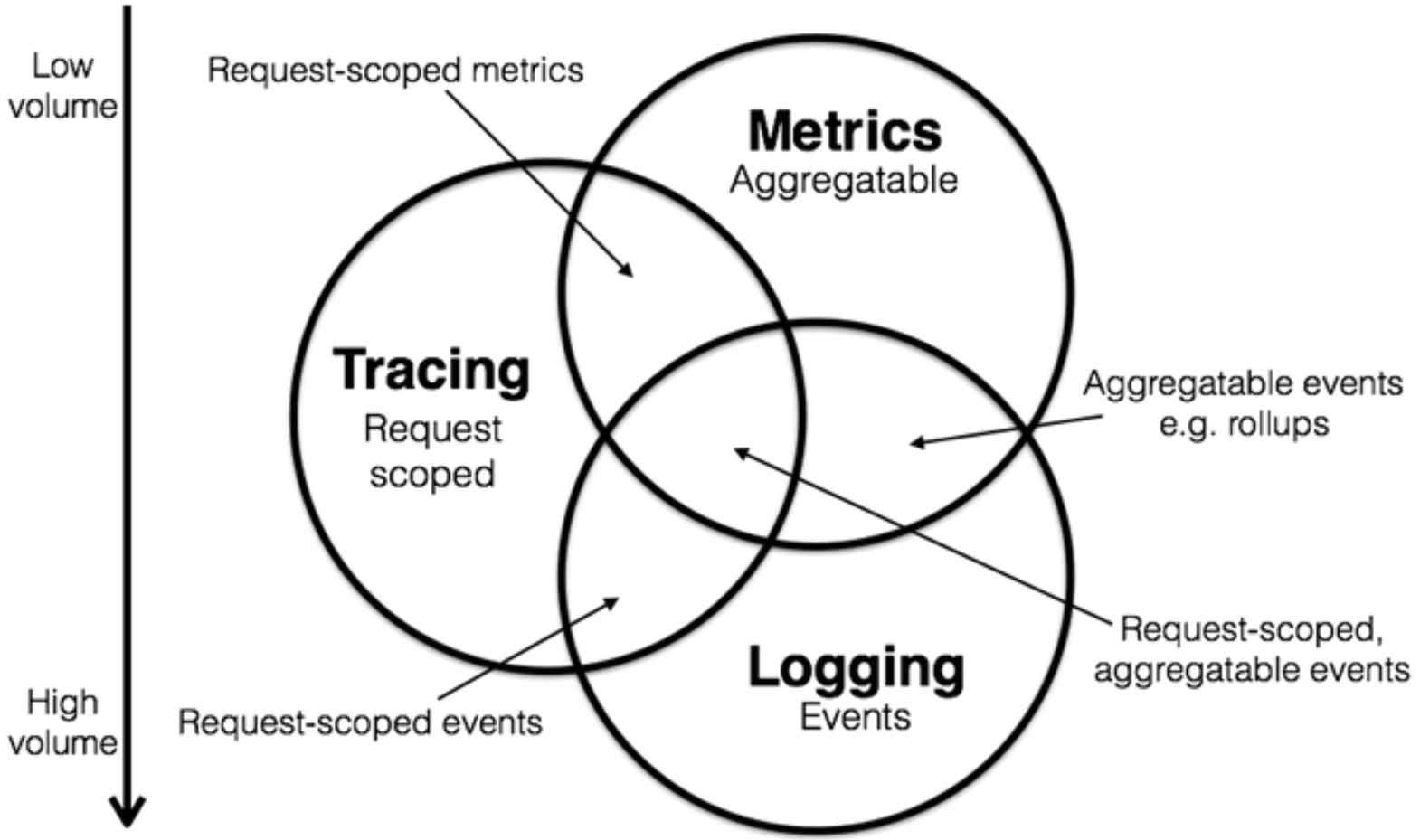
- 需要业务能够动态更新
- Push/Pull获取配置，结合使用

配置

- Consul

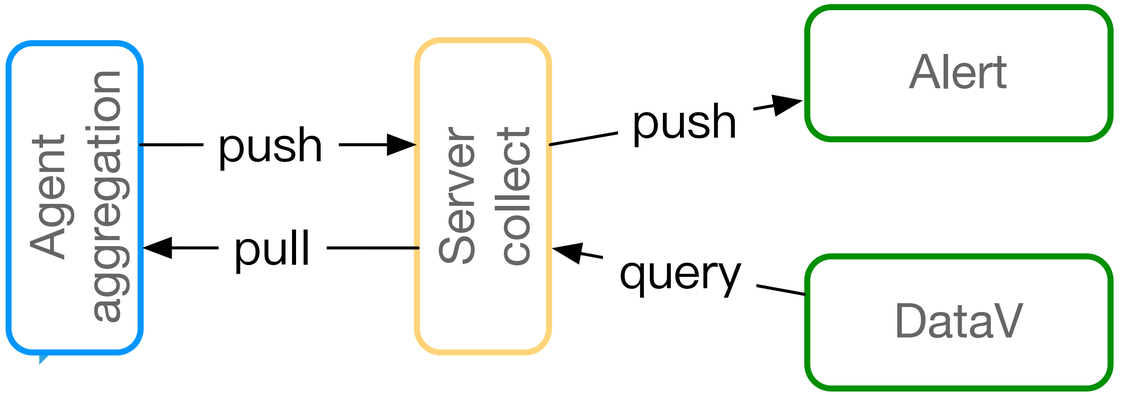
```
func conf() config.Watcher {
    consulSource := consul.NewSource(
        consul.WithPrefix("/micro/config"),
        consul.StripPrefix(true),
    )
    // 加载配置
    err := config.Load(consulSource)
    if err != nil {
        log.Logf("config load error: %v", err)
    } else {
        log.Logf("config data: %v", string(config.Bytes()))
    }

    // 监控动态配置
    w, err := config.Watch("key1")
    if err == nil {
        go func() {
            for {
                if v, err := w.Next(); err == nil {
                    log.Logf("config : %v", string(v.Bytes()))
                } else {
                    log.Log("config error: %v", err)
                    return
                }
            }
        }()

        return w
    } else {
        log.Logf("config watch error: %v", err)
    }

    return nil
}
```

LOGGING METRICS TRACING



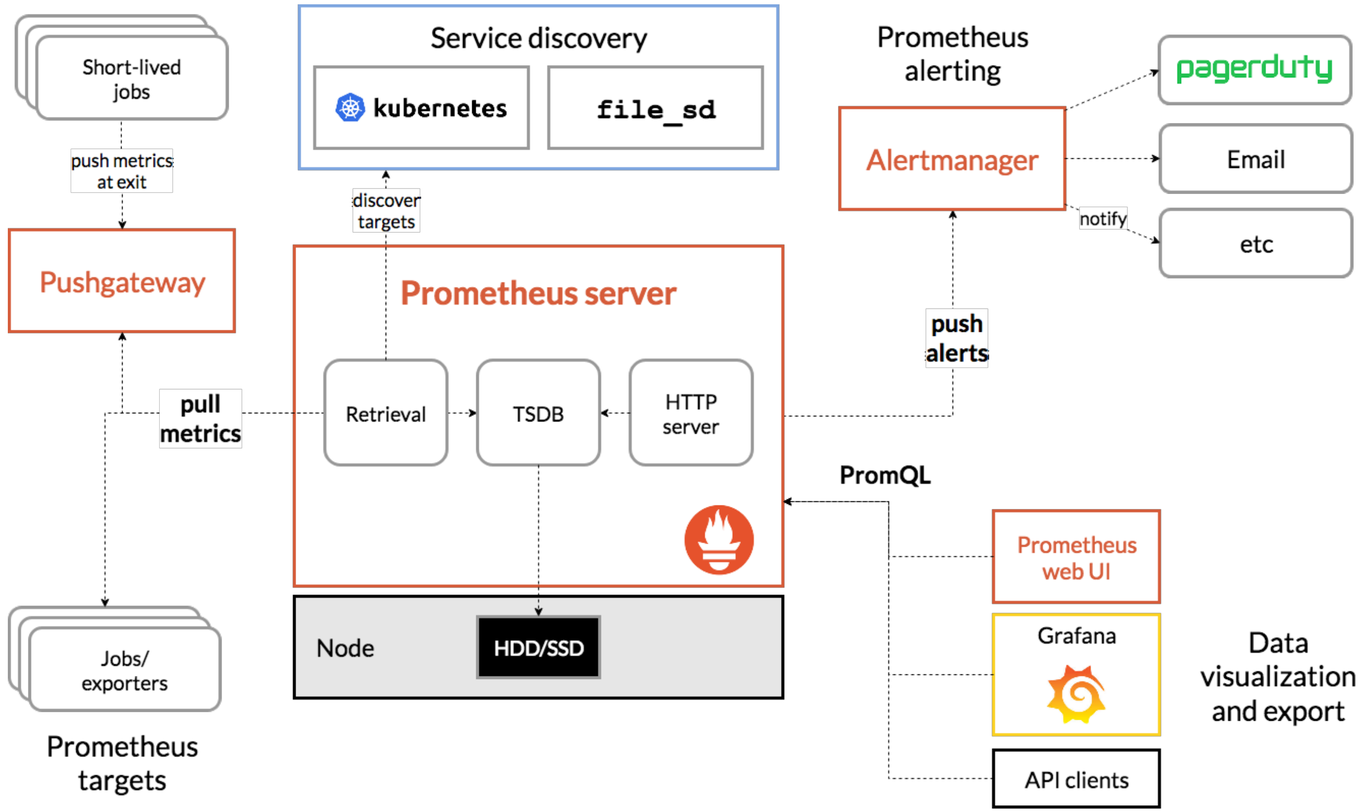
METRICS

- 业务
  - 激活、注册、下单……
- 应用
  - QPS、时延
- 系统
  - CPU、内存
- 其他
  - 向上客户端，向下系统、基础设施
- Push / Pull



PROMETHEUS

- Prometheus + Grafana
  - ARMS
- Elastic



网关监控

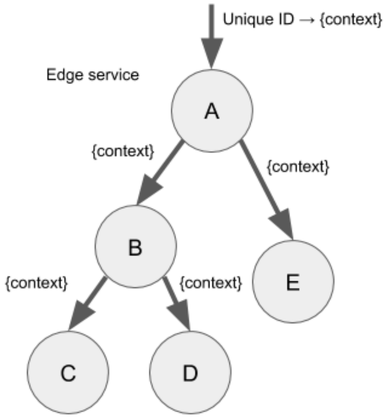
```
func init() {
   api.Register(plugin.NewPlugin(
      plugin.WithHandler(
         func(h http.Handler) http.Handler {
            ……
         }),
   ))
}
```

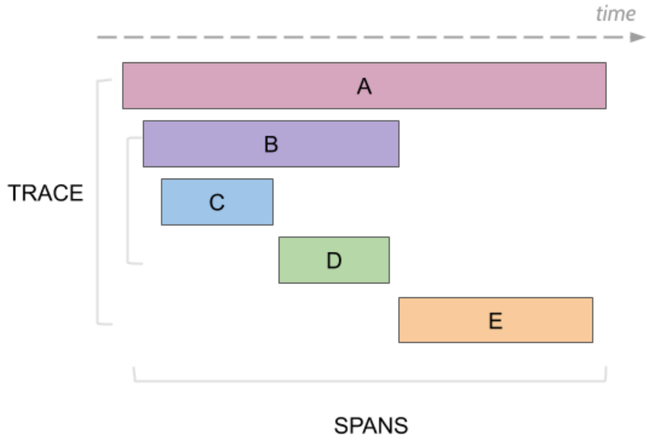
- http://localhost:8080/metrics
- docker run -d --name prometheus -p 9090:9090 -v ~/tmp/prometheus.yml:/etc/prometheus/prometheus.yml prom/prometheus

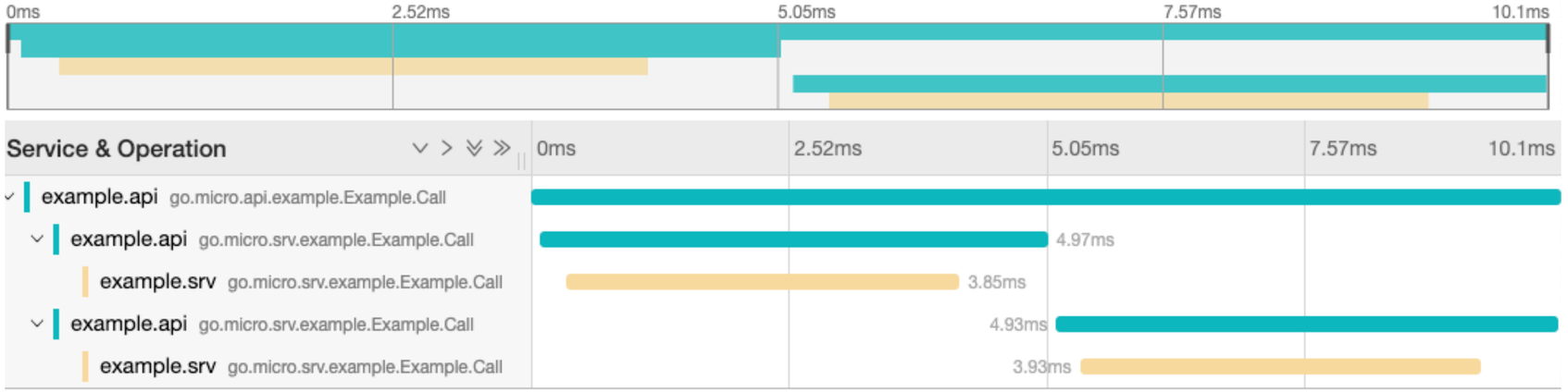
```
- job_name:       'micro'
  # Override the global default and scrape targets from this job every 5 seconds.
  scrape_interval: 5s
  metrics_path: '/metrics'
  static_configs:
    - targets: ['192.168.3.115:8080']
      labels:
        group: 'demo'
```

TRACING

- Wrap
  - Call
  - Handler

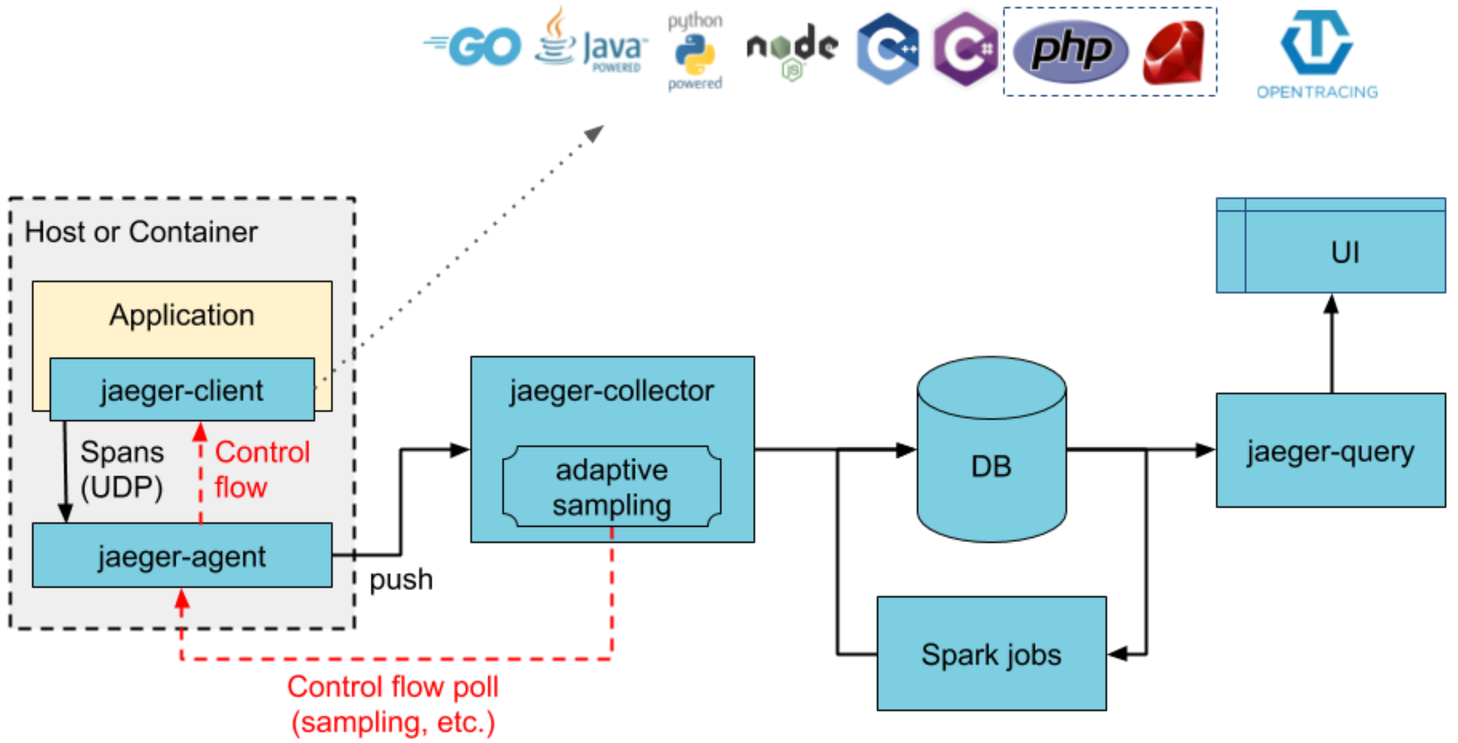






TRACING

- Opentracing
  - Jaeger
    - SLS
  - Elastic
- Opencensus



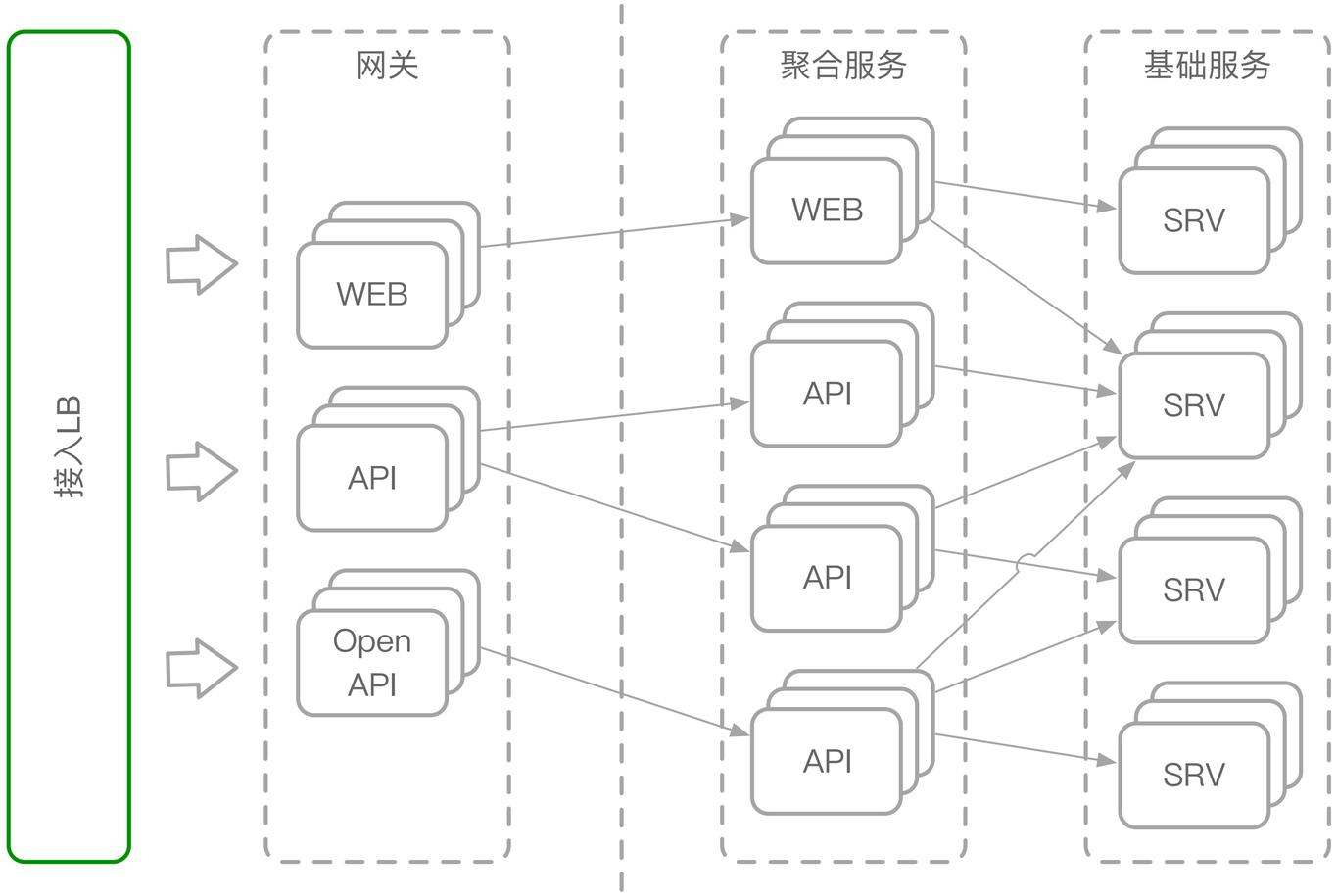
链路追踪

```
t, closer, err := tracer.NewJaegerTracer("account.api", "127.0.0.1:6831")
if err != nil {
   log.Fatalf("opentracing tracer create error:%v", err)
}
defer closer.Close()

service := micro.NewService(
   micro.WrapCall(opentracing.NewCallWrapper(t)),
   micro.WrapHandler(opentracing.NewHandlerWrapper(t)),
)
```

- docker run -d --name=jaeger -e COLLECTOR_ZIPKIN_HTTP_PORT=9411 -p5775:5775/udp -p6831:6831/udp -p6832:6832/udp -p5778:5778 -p16686:16686 -p14268:14268 -p9411:9411 jaegertracing/all-in-one:latest
- http://localhost:16686
- 熔断、重试等Tracing的变化
- 并发优化、合并请求

K8S

- 自定义micro
  - k8s注册中心
  - tcp transport
  - ……
- k8s RBAC
- 网关LB



部署

- 打包Linux镜像
- K8S RBAC
  - Deployment指定ServiceAccount

```
serviceAccountName: micro-services
```

- curl -XPOST -H 'Content-Type: application/x-www-form-urlencoded' -d 'name=hobo' http://192.168.39.147:30001/example/call

hbchen.com

Thanks!