Local microservices

From Cloud to Bare-metal with Go

Motivation

Problems that I am trying to solve

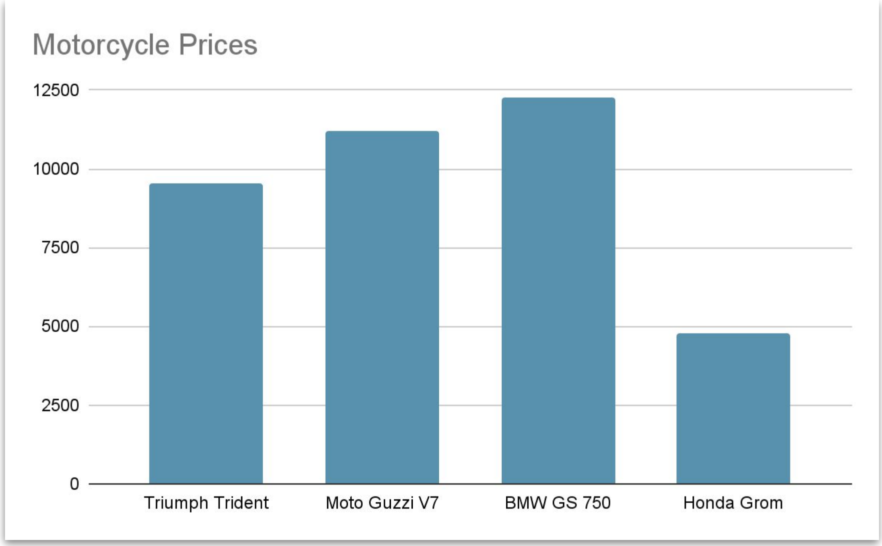
Avoid starvation, buy burgundy wine, travel, buy a motorcycle.

Limitations:

- I am allergic to authority.
- I am aging.
- I like making my own decisions and take consequences.
- I can not trust young people to pay my pension.
- I have huge ego and this is why all the previous points are starting with “I”.

Prices of notable motorcycles

Triumph Trident 660:	9545.00 Euro
BMW 750 GS:	12250.00 Euro
Moto Guzzi V7 Stone:	11200.00 Euro
Honda Grom:	4799.00 Euro



The real price of a motorcycle*

- You have paid off your mortgage; 5*10^5
- Bought a vacation house to get your vitamin D, read and write books; 2*10^5
- Collected enough money to send your kid(s) to college; 2*10^5
- Collected enough to sustain your family in case of your fatality; 1.7*10^6
- Motorcycle and gear; 1.6*10^4

Total: 2.616*10^6

~2 616 000.00 Euro

* if you are a responsible person && you are married && (have || plan kids)

Solution

Build a business

Money is the measure of how much value you brought to the market. So I need to bring some value to the market to get money.

Game Rules:

♠ Under my control;

♦ Substantial entry barrier;

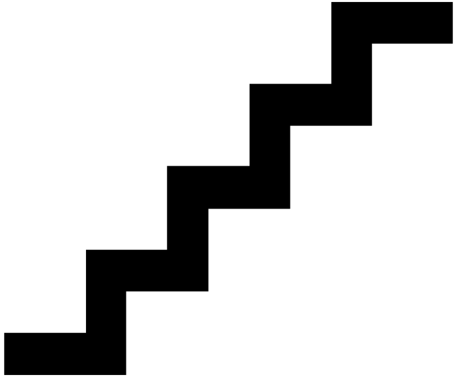
♣ Separated from my time;

♥ Has enough scale potential;

Stair Step approach

Build simple product first, then use resources and knowledge to build something more complicated.

1. Info product;
2. Course;
3. Addon or plugin;
4. SaaS, PaaS, etc.



Let’s build a ~~SaaS~~ Plugin

After some investigation I have settled on Shopify marketplace.

The plan:

1. Build a few apps;
2. See which one will get traction;
3. Invest most of the time in the one that gets traction;
4. …
5. Motorcycle;

Value proposition of my e-commerce applications

App #1: Use automation to generate boring documents that are required by law.

- Do not pay your employees to for editing spreadsheets, humans are slow;
- Pay me instead, my app is cheaper, 10 cents per document VS salary;
- Price is usage based, pay as you go;

App #2: Create reminders that have relevant context.

- Follow leads;
- Save on customer acquisition;
- Be a reliable partner;

Implementation V1

Initial stack

- Cloud Run;
- NoSQL Cloud DB;
- PubSub;
- Cloud Tasks;
- Cloud Logs;
- Cloud Scheduler;
- Cloud Redis;
- Cloud Storage;
- Secret management;
- Error reporting;
- gRPC;
- CI/CD;
- Terraform;

Google Cloud

Shared code

Should I build libraries or should I build services?

SOA & Libraries

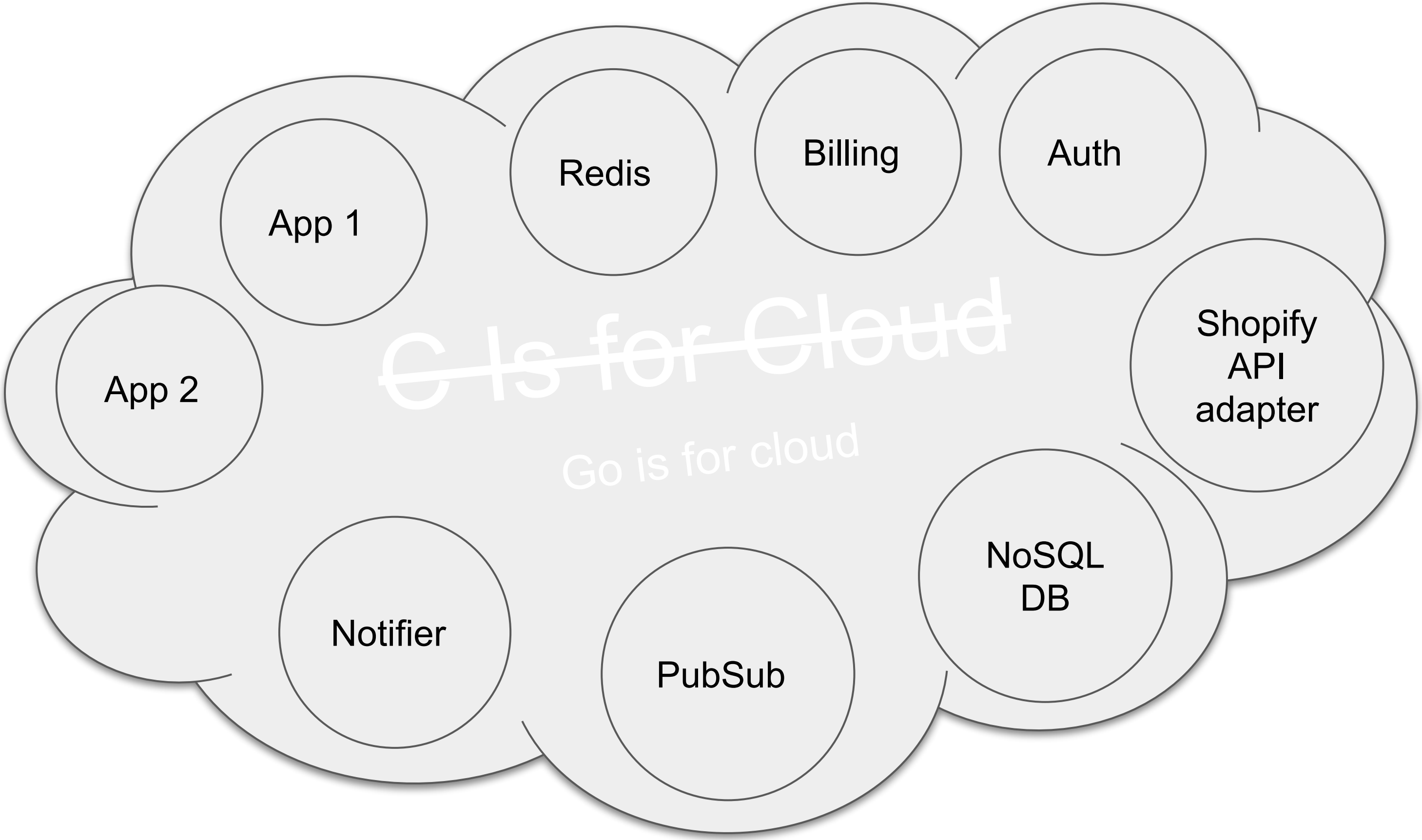
Services to define shared behavior:

- Auth
- Billing
- Notifications

Libraries to define shared data structures:

- Logger
- Errors (`sdk.ErrorNotFound`)
- Common HTTP handlers (webhooks, middleware)
- Protobuf

Initial architecture



Got ready for an overnight success

What if I will **build it and they will come**?

Millions of them.

- Autoscaling;
- High availability;
- Multi-region;

I have paid high price for it - time

Major issues:

- Managing infrastructure requires a lot of time;
- Cloud always evolves, older APIs become deprecated;

Minor issues:

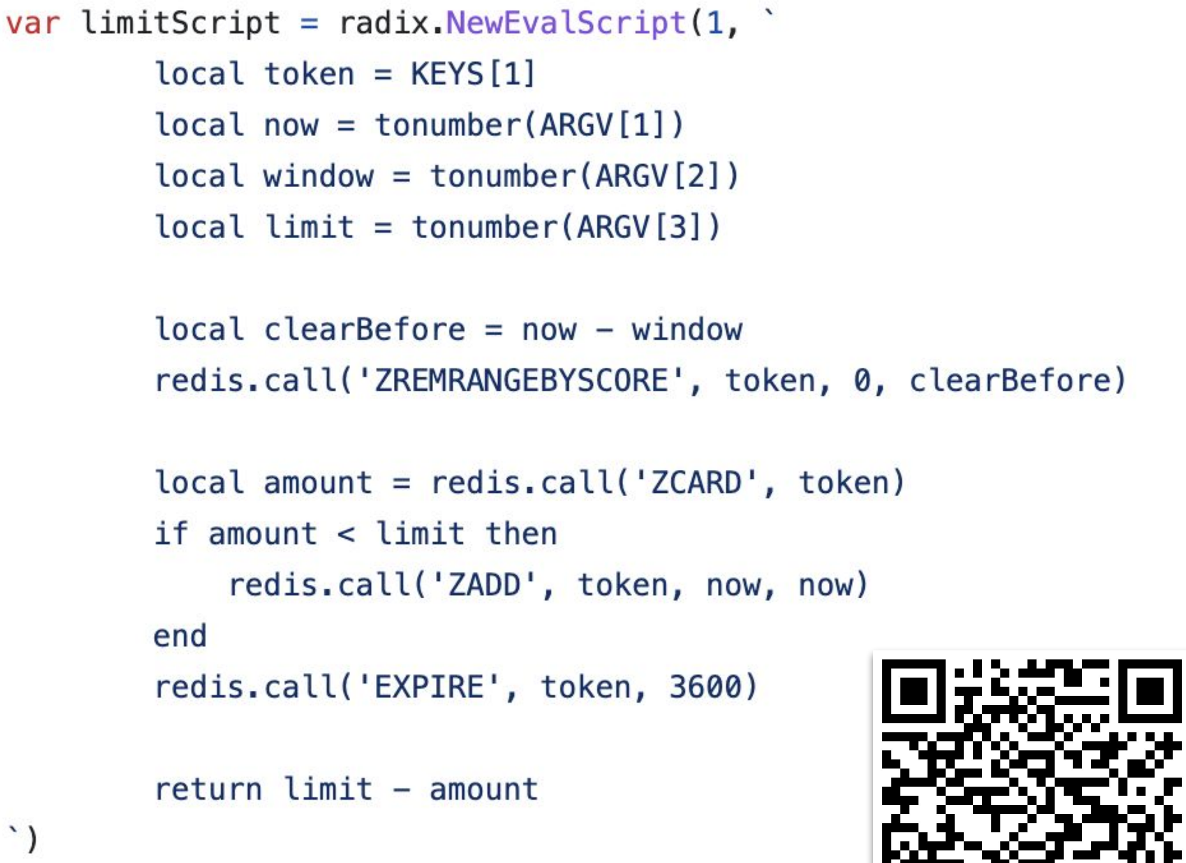
- Stateless means extracting and sharing the state in some other place;
- Multiple instances means eventual networking issues;
- Serverless means high latency;

State management

Multiple instances need to comply to a global rate limit.

Tiny rate-limit library solves that, but I would rather not have it.

The real problem is that I am forced to rely on probability.



```
var limitScript = radix.NewEvalScript(1, `
        local token = KEYS[1]
        local now = tonumber(ARGV[1])
        local window = tonumber(ARGV[2])
        local limit = tonumber(ARGV[3])

        local clearBefore = now - window
        redis.call('ZREMRANGEBYSCORE', token, 0, clearBefore)

        local amount = redis.call('ZCARD', token)
        if amount < limit then
            redis.call('ZADD', token, now, now)
        end
        redis.call('EXPIRE', token, 3600)

        return limit - amount
`)
```

Implementation V2

If I would start it from scratch, how would I do it

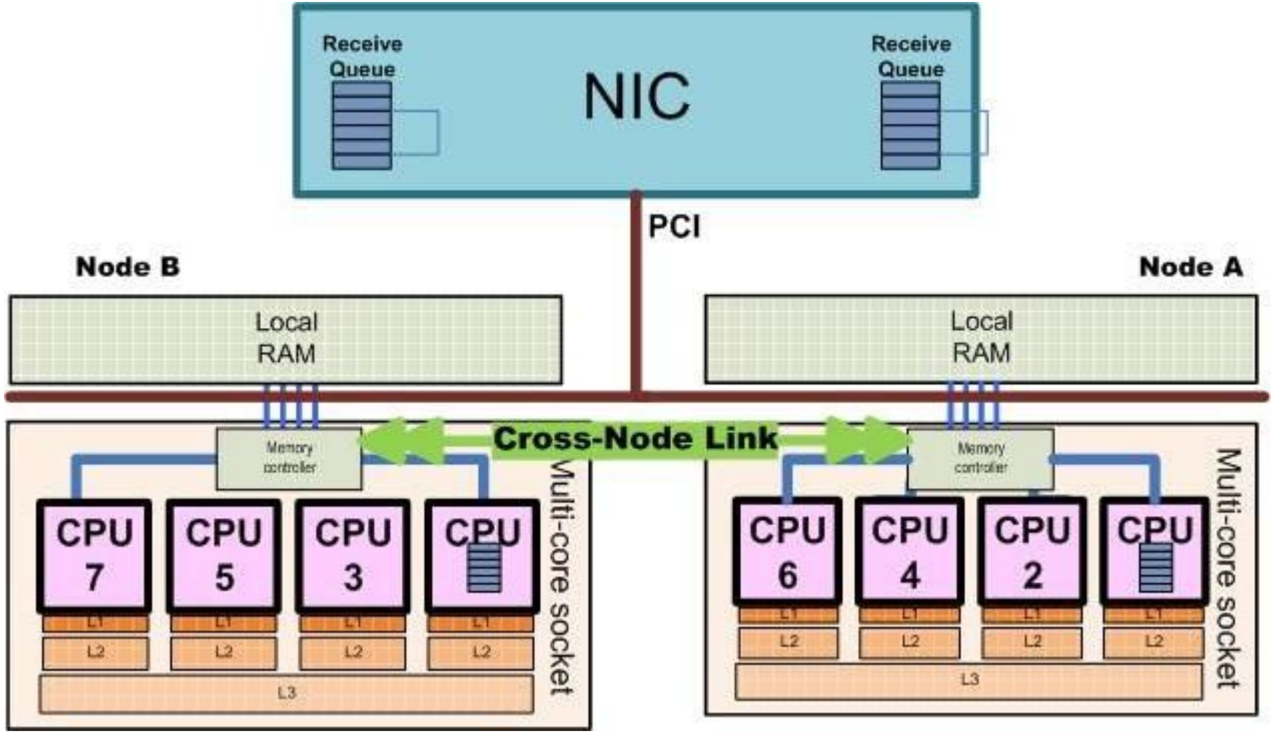
- Single server;
- Service oriented architecture;
- Single state;
- Vertical scaling;
- Boring stack (Go, gRPC, etc);
- Simple deployments;
- Less networking;

Focus on building the core product.

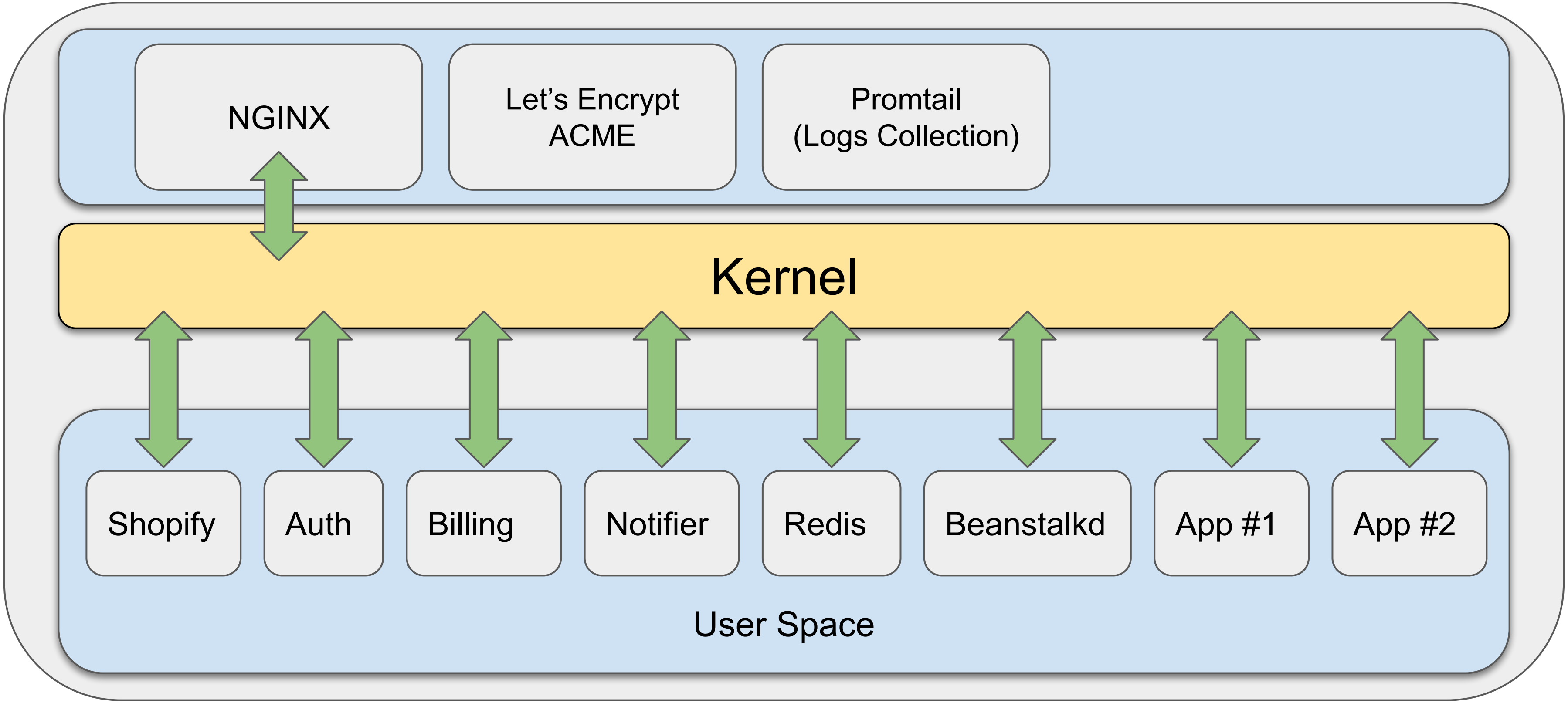
New stack

- gRPC;
- SQLite;
- Beanstalkd;
- Redis;
- Nginx;
- Pull instead of Push;
- Config files;
- Systemd;
- Tailscale;
- Grafana + Loki + Victoria Metrics;
- Ansible;
- CI/CD;

Modern server already looks like a cluster of nodes



New architecture

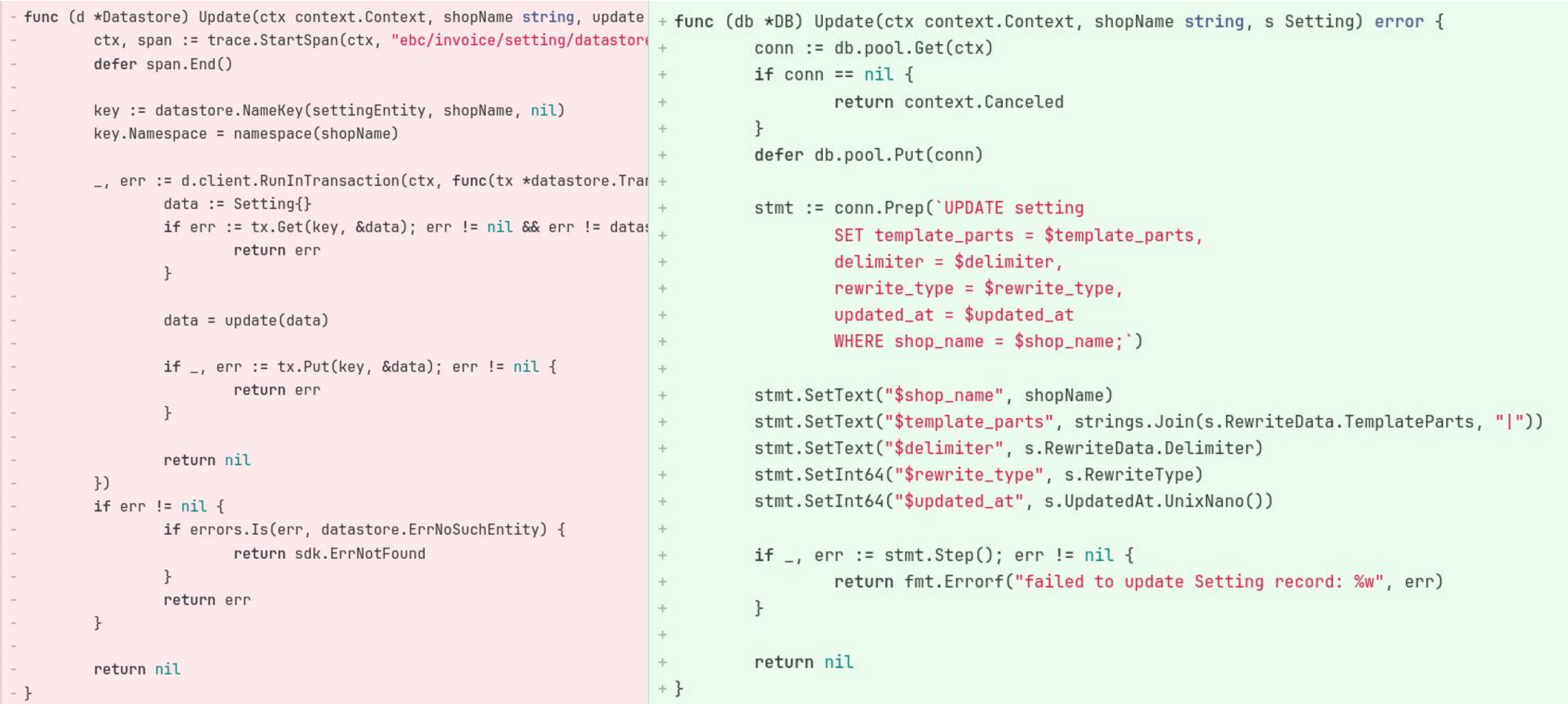


Let’s rewrite the code

Good news, apps logic stays the same.

- Rewrite DB code: SQL instead of proprietary DB API;
- Rewrite PubSub code: Beanstalkd client instead of PubSub client;
- Unix Domain Sockets instead of HTTPS over TCP;

SQL instead of client library call

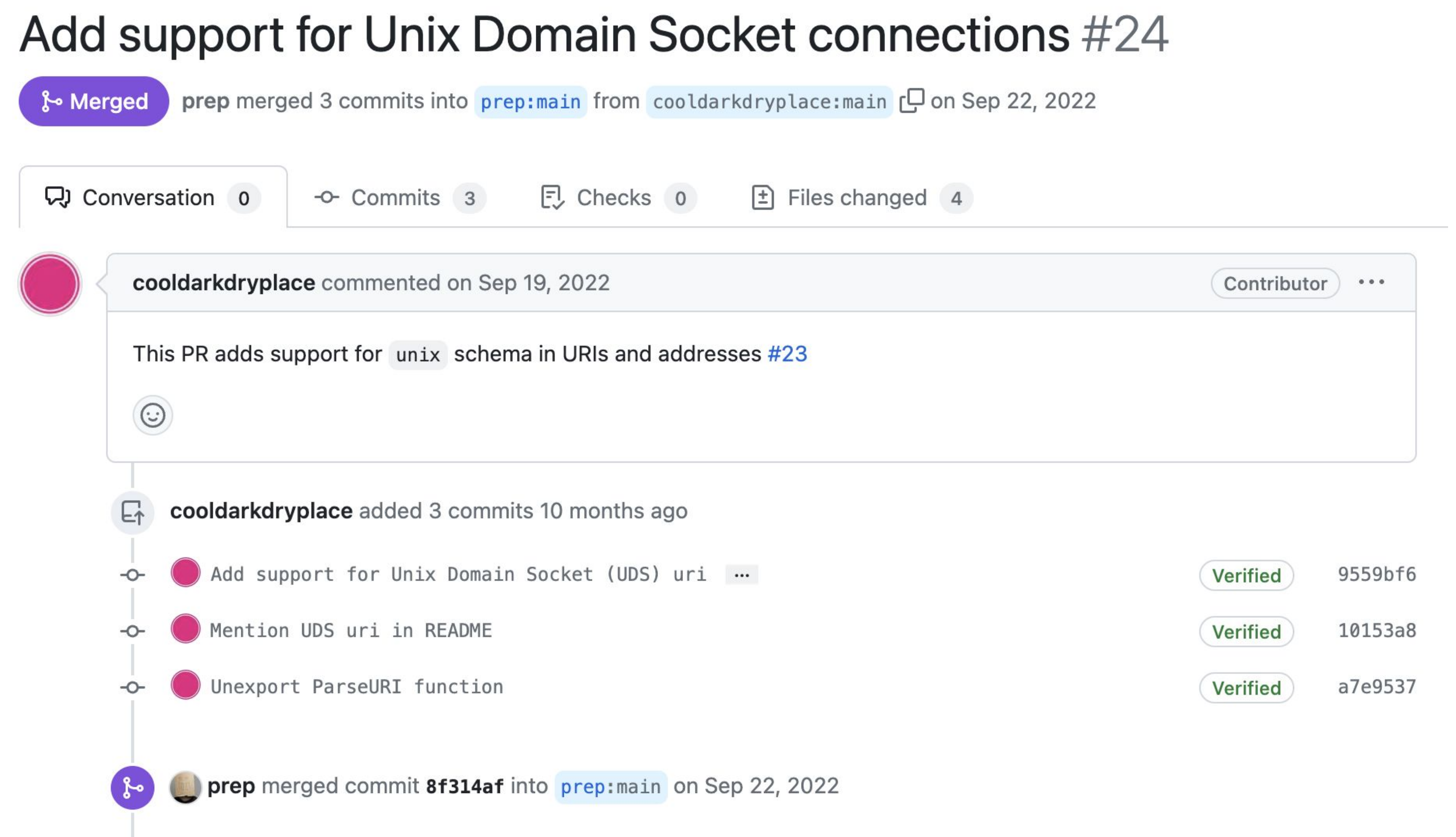


```diff
- func (d *Datastore) Update(ctx context.Context, shopName string, update
-         ctx, span := trace.StartSpan(ctx, "ebc/invoice/setting/datastore
-         defer span.End()
-
-         key := datastore.NameKey(settingEntity, shopName, nil)
-         key.Namespace = namespace(shopName)
-
-         _, err := d.client.RunInTransaction(ctx, func(tx *datastore.Tran
-                 data := Setting{}
-                 if err := tx.Get(key, &data); err != nil && err != datas
-                         return err
-                 }
-
-                 data = update(data)
-
-                 if _, err := tx.Put(key, &data); err != nil {
-                         return err
-                 }
-
-                 return nil
-         })
-         if err != nil {
-                 if errors.Is(err, datastore.ErrNoSuchEntity) {
-                         return sdk.ErrNotFound
-                 }
-                 return err
-         }
-
-         return nil
- }
```

```diff
+ func (db *DB) Update(ctx context.Context, shopName string, s Setting) error {
+         conn := db.pool.Get(ctx)
+         if conn == nil {
+                 return context.Canceled
+         }
+         defer db.pool.Put(conn)
+
+         stmt := conn.Prep(`UPDATE setting
+                 SET template_parts = $template_parts,
+                 delimiter = $delimiter,
+                 rewrite_type = $rewrite_type,
+                 updated_at = $updated_at
+                 WHERE shop_name = $shop_name;`)
+
+         stmt.SetText("$shop_name", shopName)
+         stmt.SetText("$template_parts", strings.Join(s.RewriteData.TemplateParts, "|"))
+         stmt.SetText("$delimiter", s.RewriteData.Delimiter)
+         stmt.SetInt64("$rewrite_type", s.RewriteType)
+         stmt.SetInt64("$updated_at", s.UpdatedAt.UnixNano())
+
+         if _, err := stmt.Step(); err != nil {
+                 return fmt.Errorf("failed to update Setting record: %w", err)
+         }
+
+         return nil
+ }
```

Beanstalk instead of PubSub

A very trivial change is displayed here. Blah, blah, blah. Next slide please.



Add support for Unix Domain Socket connections #24

prep merged 3 commits into `prep:main` from `cooldarkdryplace:main` on Sep 22, 2022

cooldarkdryplace commented on Sep 19, 2022

This PR adds support for `unix` schema in URIs and addresses #23

cooldarkdryplace added 3 commits 10 months ago

```
Add support for Unix Domain Socket (UDS) uri …    Verified    9559bf6
Mention UDS uri in README                         Verified    10153a8
Unexport ParseURI function                        Verified    a7e9537
```

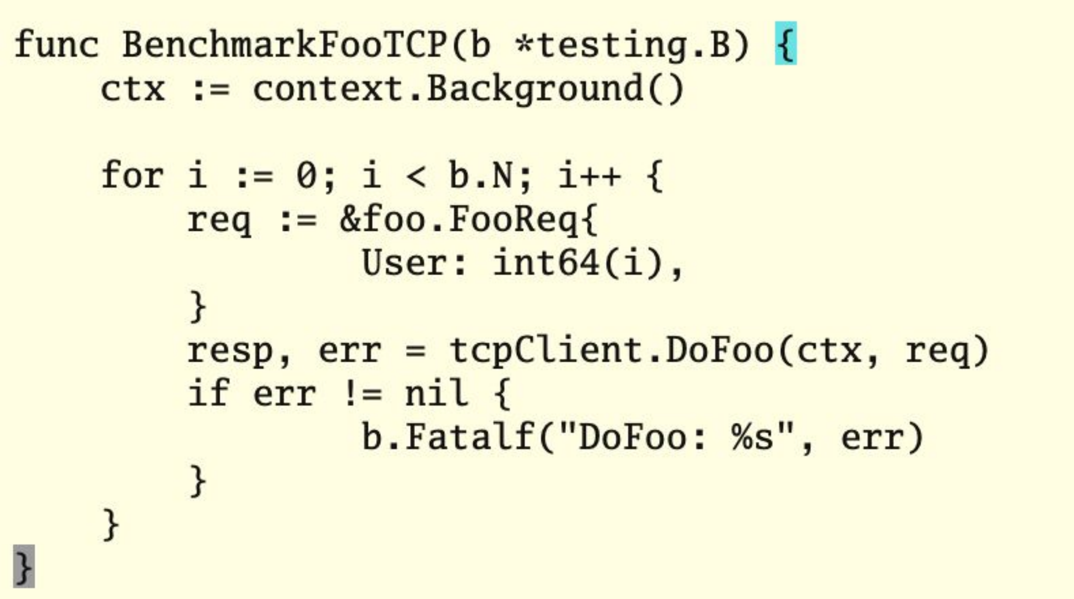
prep merged commit **8f314af** into `prep:main` on Sep 22, 2022

Unix Domain Sockets

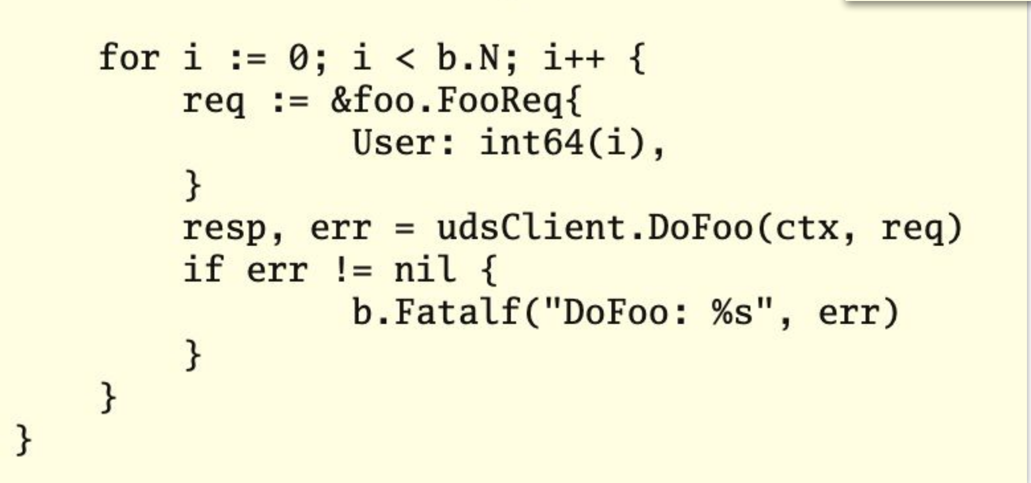
- All the communication happens within Linux kernel;
- Linux permission model applies to sockets;
- Go makes it easy to start using sockets;

```
conn, err := net.Dial("unix", "/var/myapp.sock")
if err != nil {
    log.Fatal(err)
}
```

UDS VS TCP (loopback)

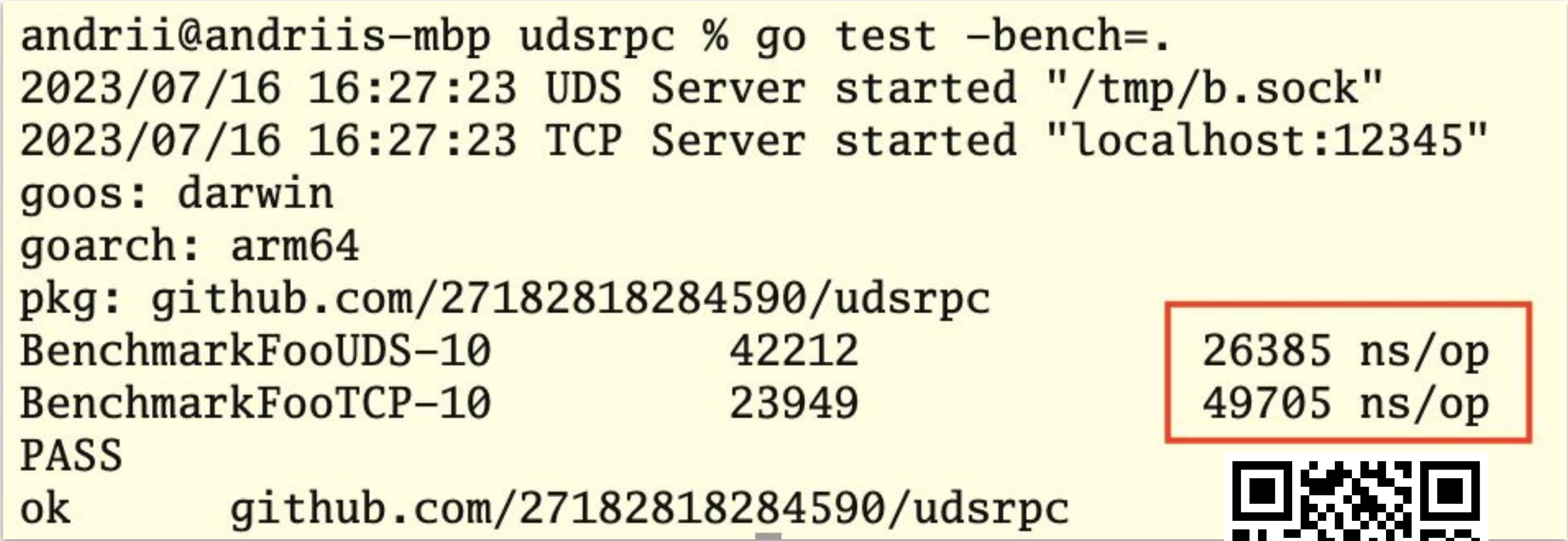


```go
func BenchmarkFooTCP(b *testing.B) {
    ctx := context.Background()

    for i := 0; i < b.N; i++ {
        req := &foo.FooReq{
                User: int64(i),
        }
        resp, err = tcpClient.DoFoo(ctx, req)
        if err != nil {
                b.Fatalf("DoFoo: %s", err)
        }
    }
}
```



```go
func BenchmarkFooUDS(b *testing.B) {
    ctx := context.Background()

    for i := 0; i < b.N; i++ {
        req := &foo.FooReq{
                User: int64(i),
        }
        resp, err = udsClient.DoFoo(ctx, req)
        if err != nil {
                b.Fatalf("DoFoo: %s", err)
        }
    }
}
```

Benchmark results



```
andrii@andriis-mbp udsrpc % go test -bench=.
2023/07/16 16:27:23 UDS Server started "/tmp/b.sock"
2023/07/16 16:27:23 TCP Server started "localhost:12345"
goos: darwin
goarch: arm64
pkg: github.com/27182818284590/udsrpc
BenchmarkFooUDS-10           42212              26385 ns/op
BenchmarkFooTCP-10           23949              49705 ns/op
PASS
ok      github.com/27182818284590/udsrpc
```

https://github.com/27182818284590/udsrpc

What do I get by paying higher price of TCP?

Location independence (inter-machine connectivity)

- MTU-size datagrams (maximum transmission unit)
- Flow control - (throttles if needed)
- Acknowledgements (reliable delivery)

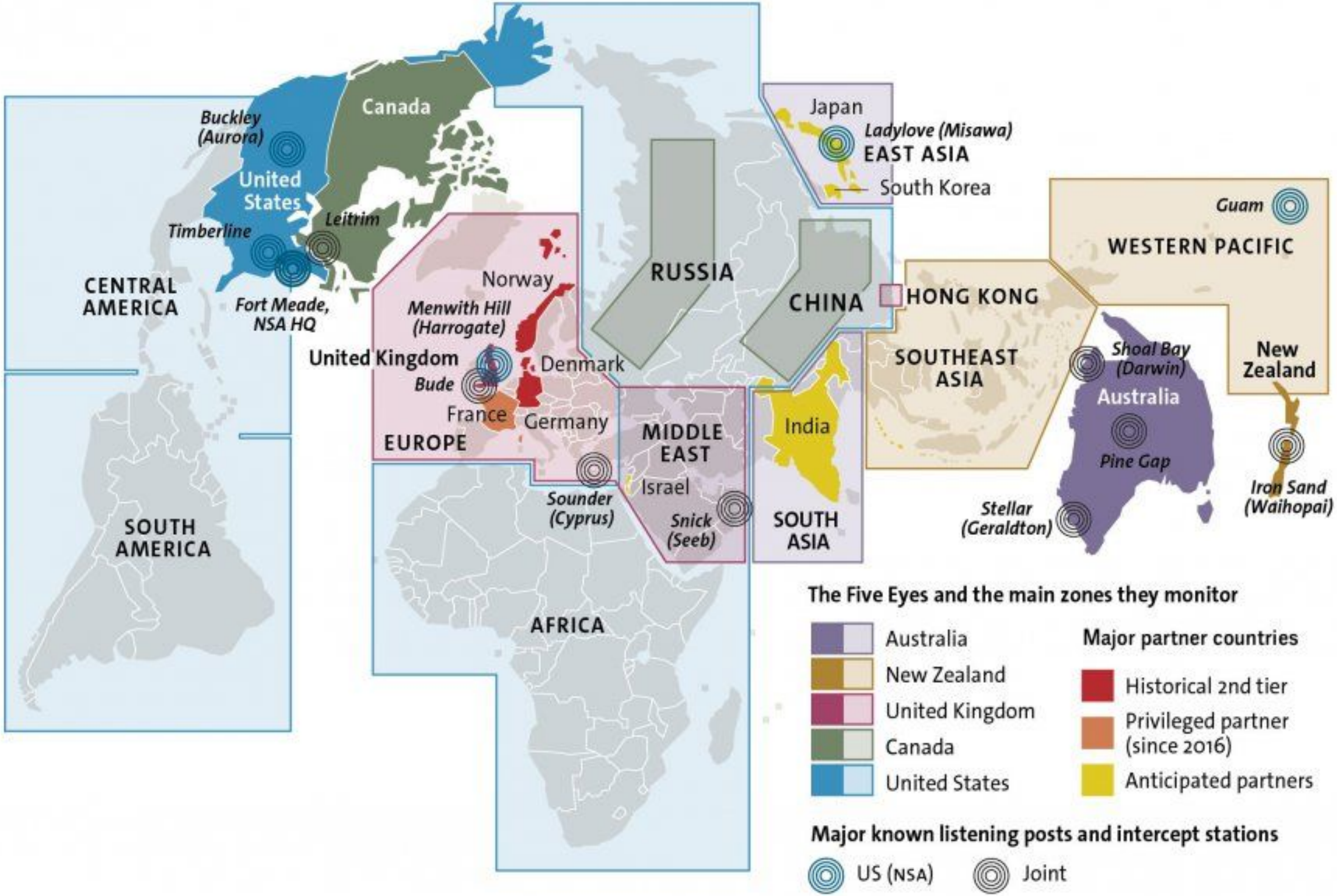
Serve HTTPS with Nginx

```
server {
    listen 443 ssl;
    ...
    location / {
        proxy_pass http://sock/;
    }
}
upstream sock {
    server unix:/var/{{service_name}}/sock;
}
```

Results

What changed

- It is more reliable. I was able to go on vacation without any incidents.
- I have more control over my business, If necessary I can migrate to other service providers.
- It is cheaper.
- It is easier to sell.
- Nothing to put in my CV (may be a good thing)


Canada
Buckley (Aurora)
United States
Timberline
Leitrim
Fort Meade, NSA HQ
CENTRAL AMERICA
SOUTH AMERICA
Norway
Menwith Hill (Harrogate)
United Kingdom
Bude
Denmark
France
Germany
EUROPE
AFRICA
Sounder (Cyprus)
MIDDLE EAST
Israel
Snick (Seeb)
RUSSIA
CHINA
HONG KONG
Japan
Ladylove (Misawa)
EAST ASIA
South Korea
Guam
WESTERN PACIFIC
SOUTHEAST ASIA
India
SOUTH ASIA
Shoal Bay (Darwin)
Australia
Pine Gap
Stellar (Geraldton)
New Zealand
Iron Sand (Waihopai)
The Five Eyes and the main zones they monitor
Australia
New Zealand
United Kingdom
Canada
United States
Major partner countries
Historical 2nd tier
Privileged partner (since 2016)
Anticipated partners
Major known listening posts and intercept stations
US (NSA)
Joint

What do I need to do next?

- Marketing;
- Litestream for backups;
- Disaster recovery;
- Synthetic client for SLA monitoring;
- Integrations with other e-commerce;
- Marketing;
- Marketing;
- Marketing;
- Marketing;
- Marketing;
- Marketing;