Using encrypted priva HTTPS server

Prateek Nischal · Follow

4 min read · Oct 1, 2018

Listen Share

Yes, this is an Owl, protecting my traffic

Coming from a Java world (I am embarrassed enough), this hit me like a wall. Default golang `http.Server` does not have a way to accept private keys that are protected by a

passphrase. If you look up the documentation for `server.ListenAndServerTLS` , it needs 2 parameters, the certFile and keyFile strings, that represent the location of PEM encoded format for certificate and private key. There is no option to supply the password !!

Another weird thing I saw is there is field in the `http.Server` struct, it has a `tls.Config` parameter that too has a field to configure the `tls.Certificate` object and that is not just for Certificates but for PrivateKey as well. I thought, why does it need the parameters in `server.ListenAndServerTLS` when it already has it. I got into the messy part because I can't read documentation.

```
func (srv *Server) ServeTLS(l net.Listener, certFile, keyFile string)
error {
 // Setup HTTP/2 before srv.Serve, to initialize srv.TLSConfig
 // before we clone it and create the TLS Listener.
 if err := srv.setupHTTP2_ServeTLS(); err != nil {
  return err
 }

config := cloneTLSConfig(srv.TLSConfig)
 if !strSliceContains(config.NextProtos, "http/1.1") {
  config.NextProtos = append(config.NextProtos, "http/1.1")
 }

configHasCert := len(config.Certificates) > 0 || config.GetCertificate
!= nil
 if !configHasCert || certFile != "" || keyFile != "" {
  var err error
  config.Certificates = make([]tls.Certificate, 1)
  config.Certificates[0], err = tls.LoadX509KeyPair(certFile, keyFile)
  if err != nil {
   return err
  }
 }

 tlsListener := tls.NewListener(l, config)
 return srv.Serve(tlsListener)
}
```

It is first checking the TLSConfig that we might have supplied, if not then it would use the supplied cert and key file names.

It occurred to me, why not we try to build the `tls.Config.Certificate` with the encrypted key file that we have.

Let's get to some action

How do you load a plain cert and key files to build a Certificate Object

`tls.LoadX509KeyPair(certFile, keyFile string)` or `tls.X509KeyPair(certPEMBlock, keyPEMBlock []byte)`

The first one takes file names and second one takes PEM encoded blocks. First is no use, as it is the same as the Server interface. The second one looks interesting. Let's try to get the private key as a PEM block.

But first, What is PEM encoding. There is a very interesting and intimidating thing called ASN.1 that can be used to serialise structures and cryptographic implementation uses the DER encoding rules to store the ASN.1 structures.

A basic guide for layman can be found here: ASN.1 and surprisingly here: DER encoding for ASN.1

For now just think of those as serialization techniques and know that DER is a binary encoding scheme or it produces binary output. Which is ok to store on disk. But when you have to transfer it over emails, for which PEM (Privacy enhanced mails) was originally built, which is plain text, you need to make it a little simpler.

Here comes the base64 cannon. Another encoding scheme on top of an encoding scheme (I know, right !!). It can represent any binary data as printable characters. eg:

```
$ echo hello | xxd
00000000: 6865 6c6c 6f0a                           hello.

$ echo -ne "\x68\x65\x6c\x6c\x6f\x0a" | base64
aGVsbG8K

$ echo "aGVsbG8K" | base64 -D
hello
```

Yay ! So now you can transmit binary data in a printable format.

PEM is the fancy name for this base64 encoding of DER format. Try this for fun, it's pure pleasure :D

```
$ wget https://secure.globalsign.net/cacert/Root-R1.crt
$ cat Root-R1.crt | base64
MIIDdTCCAl2gAwIBAgILBAAAAAABFUtaw5QwDQYJKoZIhvcNAQEFBQAwVzELMAkGA1UEBh
MCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExEDAOBgNVBAsTB1Jvb3QgQ0ExGzAZ
BgNVBAMTEkdsb2JhbFNpZ24gUm9vdCBDQTAeFw05ODA5MDExMjAwMDBaFw0yODAxMjgxMj
AwMDBaMFcxCzAJBgNVBAYTAkJFMRkwFwYDVQQKExBHbG9iYWxTaWduIG52LXNhMRAwDgYD
VQQLEwdSb290IENBMRswGQYDVQQDExJHbG9iYWxTaWduIFJvb3QgQ0EwggEiMA0GCSqGSI
b3DQEBAQUAA4IBDwAwggEKAoIBAQDaDuaZjc6j40+Kfvvxi4Mla+pIH/EqsLmVEQS98GPR
4mdmzxzdzxtIK+6NiY6arymAZavpxy0Sy6scTHAHoT0KMM0VjU/43dSMUBUc71DuxC73/O
lS8pF94G3VNTCOXkNz8kHp1Wrjsok6Vjk4bwY8iGlbKk3Fp1S4bInMm/k8yuX9ifUSPJJ4
ltbcdG6TRGHRjcdGsnUOhugZitVtbNV4FpWi6cgKOOvyJBNPc1STE4U6G7weNLWLBYy5d4
ux2x8gkasJU26Qzns3dLlwR5EiUWMWea6xrkEmCMgZK9FGqkjWZCrXgzT/LCrBbBlDSgeF
59N89iFo7+ryUp9/k5DPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMBAf8EBT
ADAQH/MB0GA1UdDgQWBBRge2YaRQ2XyolQL30EzTSo//z9SzANBgkqhkiG9w0BAQUFAAOC
AQEA1nPnfE920I2/7LqivjTFKDK1fPxsnCwrvQmeU79rXqoRSLblCKOzyj1hTdNGCbM+w6
DjY1Ub8rrvrTnhQ7k4o+YviiY776BQVvnGCv04zcQLcFGUl5gE38NflNUVyRRBnMRddWQV
Df9VMOyGj/8N7yy5Y0b2qvzfvGn9LhJIZJrglfCm7ymPAbEVtQwdpf5pLGkkeB6zpxxxYu
7KyJesF12KwvhHhm4qxFYxldBniYUr+WymXUadDKqC5JlR3XC321Y9YeRq4VzW9v493kHM
B65jUr9TU/Qr6cf9tveCX4XSQRjbgbMEHMUfpIBvFSDJ3gyICh3WZlXi/EjJKSZp4A==

$ openssl x509 -in Root-R1.crt -inform DER -outform PEM
-----BEGIN CERTIFICATE-----
MIIDdTCCAl2gAwIBAgILBAAAAAABFUtaw5QwDQYJKoZIhvcNAQEFBQAwVzELMAkG
A1UEBhMCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExEDAOBgNVBAsTB1Jv
b3QgQ0ExGzAZBgNVBAMTEkdsb2JhbFNpZ24gUm9vdCBDQTAeFw05ODA5MDExMjAw
MDBaFw0yODAxMjgxMjAwMDBaMFcxCzAJBgNVBAYTAkJFMRkwFwYDVQQKExBHbG9i
YWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENBMRswGQYDVQQDExJHbG9iYWxT
aWduIFJvb3QgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDaDuaZ
jc6j40+Kfvvxi4Mla+pIH/EqsLmVEQS98GPR4mdmzxzdzxtIK+6NiY6arymAZavp
xy0Sy6scTHAHoT0KMM0VjU/43dSMUBUc71DuxC73/OlS8pF94G3VNTCOXkNz8kHp
1Wrjsok6Vjk4bwY8iGlbKk3Fp1S4bInMm/k8yuX9ifUSPJJ4ltbcdG6TRGHRjcdG
snUOhugZitVtbNV4FpWi6cgKOOvyJBNPc1STE4U6G7weNLWLBYy5d4ux2x8gkasJ
U26Qzns3dLlwR5EiUWMWea6xrkEmCMgZK9FGqkjWZCrXgzT/LCrBbBlDSgeF59N8
9iFo7+ryUp9/k5DPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMBAf8E
BTADAQH/MB0GA1UdDgQWBBRge2YaRQ2XyolQL30EzTSo//z9SzANBgkqhkiG9w0B
AQUFAAOCAQEA1nPnfE920I2/7LqivjTFKDK1fPxsnCwrvQmeU79rXqoRSLblCKOz
yj1hTdNGCbM+w6DjY1Ub8rrvrTnhQ7k4o+YviiY776BQVvnGCv04zcQLcFGUl5gE
38NflNUVyRRBnMRddWQVDf9VMOyGj/8N7yy5Y0b2qvzfvGn9LhJIZJrglfCm7ymP
AbEVtQwdpf5pLGkkeB6zpxxxYu7KyJesF12KwvhHhm4qxFYxldBniYUr+WymXUad
DKqC5JlR3XC321Y9YeRq4VzW9v493kHMB65jUr9TU/Qr6cf9tveCX4XSQRjbgbME
HMUfpIBvFSDJ3gyICh3WZlXi/EjJKSZp4A==
-----END CERTIFICATE-----
```

Look at both of the outputs, They are the same, with a better formatting though. And the

```
-----BEGIN CERTIFICATE-----
...
-----END CERTIFICATE-----
```

part is according to the RFC which tells about the type of the PEM block.

Coming back to our problem,

If we can get golang a PEM block that has the private key unencrypted, bingo. First let us get the PEM block in golang Native structs.

`encoding/pem` has a useful function: `Decrypt(data []bytes)(p *pem.Block, rest []byte)`. This function goes through the bytes and tries to find the marker of the PEM block and returns the first block as a pointer to `pem.Block`

```
-----BEGIN Type-----
Headers
base64-encoded Bytes
-----END Type-----
```

Let's try to see how to write this thing.

```go
package main

import (
	"crypto/tls"
	"crypto/x509"
	"encoding/pem"
	"fmt"
	"io/ioutil"
	"log"
	"net/http"
)

// Credits for server code: https://gist.github.com/denji/12b3a568f092ab951456

func main() {
	cert := "/path/to/cert.pem"
	mux := http.NewServeMux()
	mux.HandleFunc("/", func(w http.ResponseWriter, req *http.Request) {
		w.Header().Add("Strict-Transport-Security", "max-age=63072000; includeSubDomains")
		w.Write([]byte("It works !!\n"))
	})

	b, _ := ioutil.ReadFile(cert)
	var pemBlocks []*pem.Block
	var v *pem.Block
	var pkey []byte

	for {
		v, b = pem.Decode(b)
		if v == nil {
			break
		}
		if v.Type == "RSA PRIVATE KEY" {
			pkey = pem.EncodeToMemory(v)
		} else {
			pemBlocks = append(pemBlocks, v)
		}
	}
	c, _ := tls.X509KeyPair(pem.EncodeToMemory(pemBlocks[0]), pkey)

	cfg := &tls.Config{
		MinVersion:               tls.VersionTLS12,
		CurvePreferences:         []tls.CurveID{tls.CurveP521, tls.CurveP384, tls.CurveP25
		PreferServerCipherSuites: true,
		CipherSuites: []uint16{
```

```
                CipherSuites: []uint16{
                        tls.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
                        tls.TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
                        tls.TLS_RSA_WITH_AES_256_GCM_SHA384,
                        tls.TLS_RSA_WITH_AES_256_CBC_SHA,
                },
                Certificates: []tls.Certificate{c},
        }
        srv := &http.Server{
                Addr:         ":9000",
                Handler:      mux,
                TLSConfig:    cfg,
                TLSNextProto: make(map[string]func(*http.Server, *tls.Conn, http.Handler), 0),
        }

        log.Fatal(srv.ListenAndServeTLS("", ""))
}
```

srv.go hosted with ❤ by GitHub view raw

Not bad, but we still haven't reached the magic yet.

Another amazing function, a pair actually, that golang's crypto library provides is

- `IsEncryptedPEMBlock(b *pem.Block) bool`
- `DecryptPEMBlock(b *pem.Block, password []byte) ([]byte, error)`

Check if a PEM block is encrypted, if yes, decrypt it. Yes, it's that awesome.

All you need now is check if the detected Private key block is encrypted, If yes, decrypt it.

```
import (
        "crypto/tls"
        "crypto/x509"
        "encoding/pem"
        "io/ioutil"
        "log"
        "net/http"
)

func main() {
        cert := "/cert/with/encrypted/privatekey/cert.key"
        mux := http.NewServeMux()
        mux.HandleFunc("/", func(w http.ResponseWriter, req *http.Request) {
                w.Header().Add("Strict-Transport-Security", "max-age=63072000; includeSubDomains")
                w.Write([]byte("It works !!\n"))
        })

        b, _ := ioutil.ReadFile(cert)
        var pemBlocks []*pem.Block
        var v *pem.Block
        var pkey []byte

        for {
                v, b = pem.Decode(b)
                if v == nil {
                        break
                }
                if v.Type == "RSA PRIVATE KEY" {
                        if x509.IsEncryptedPEMBlock(v) {
                                pkey, _ = x509.DecryptPEMBlock(v, []byte("xxxxxxxxx"))
                                pkey = pem.EncodeToMemory(&pem.Block{
                                        Type:  v.Type,
                                        Bytes: pkey,
                                })
                        } else {
                                pkey = pem.EncodeToMemory(v)
                        }
                } else {
                        pemBlocks = append(pemBlocks, v)
                }
        }
        c, _ := tls.X509KeyPair(pem.EncodeToMemory(pemBlocks[0]), pkey)
```

```
        cfg := &tls.Config{
                MinVersion:               tls.VersionTLS12,
                CurvePreferences:         []tls.CurveID{tls.CurveP521, tls.CurveP384, tls.CurveP25
                PreferServerCipherSuites: true,
                CipherSuites: []uint16{
                        tls.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
                        tls.TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
                        tls.TLS_RSA_WITH_AES_256_GCM_SHA384,
                        tls.TLS_RSA_WITH_AES_256_CBC_SHA,
                },
                Certificates: []tls.Certificate{c},
        }
        srv := &http.Server{
                Addr:         ":9000",
                Handler:      mux,
                TLSConfig:    cfg,
                TLSNextProto: make(map[string]func(*http.Server, *tls.Conn, http.Handler), 0),
        }

        log.Fatal(srv.ListenAndServeTLS("", ""))
}
```

srv_enc.go hosted with ❤ by GitHub view raw

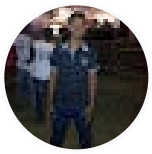
And, that was it.

If you would have seen the signature of the `tls.X509KeyPair(certPEMBlock, keyPEMBlock []byte)` all they need is a PEM block as bytes. And `pem.Block` already has a `Block.Byte` Value. What's this for ?

This is the DER encoding of the ASN.1 structure and we just don't need bytes in the function, but a well formed PEM block with all the markers and headers (yes, that took me a couple of hours and some facepalms to find out, as I read documentation as carefully as people think about their passwords.)

Now, how do you protect the password to the encrypted private key, is another problem altogether and my face already has enough finger marks.

Golang X509 Private Key Tls Server Encrypted



Follow

Written by Prateek Nischal

22 Followers

Software Engineer — infosec @WalmartlabsIndia

More from Prateek Nischal

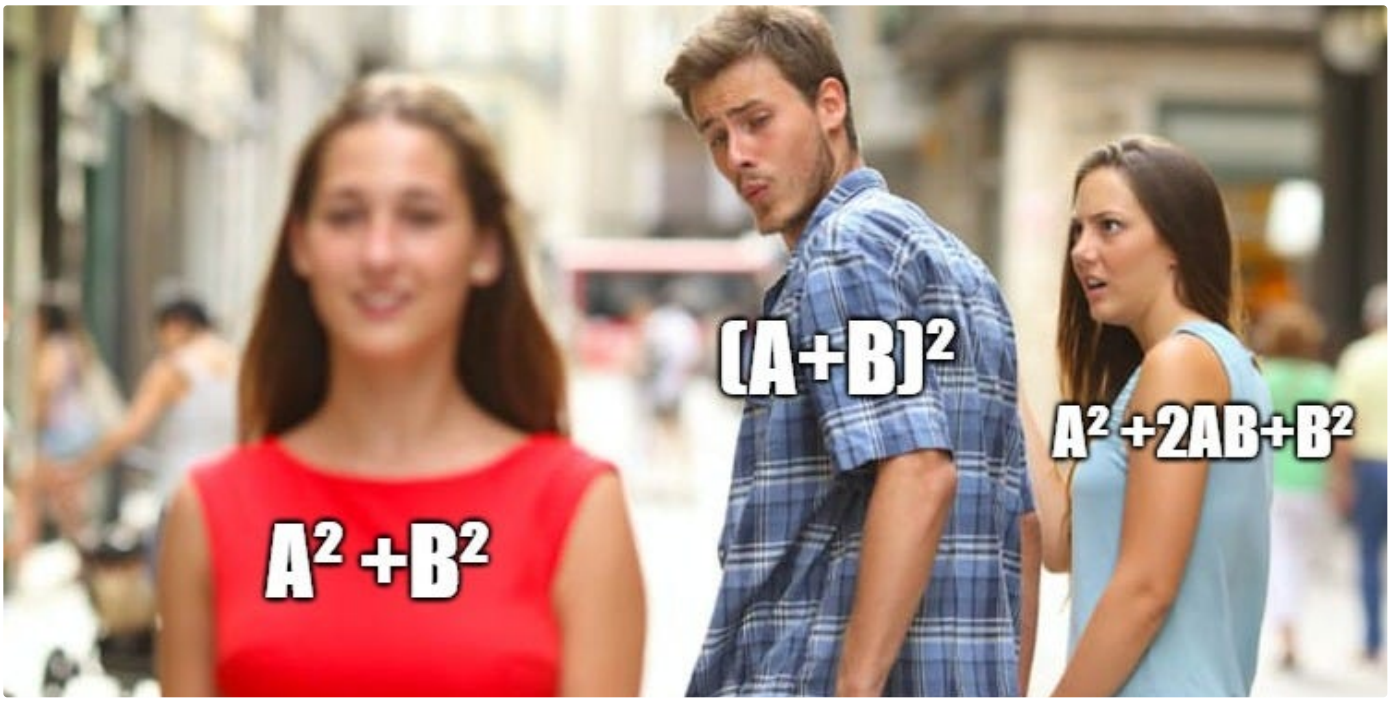
Prateek Nischal

Drunk president and the Nuclear launch—Shamir Secret Sharing—I

Suppose you work for a president who likes to flaunt their nuclear strength, but you understand the responsibility of owning a nuclear...

5 min read · Dec 9, 2018

16



Prateek Nischal

Drunk president and the nuclear launch—Shamir and the duel guy, Galois —III

Preface: Drunk president and the nuclear launch—Lagrange Decomposition of a function

7 min read · Dec 9, 2018

54

Prateek Nischal

Drunk president and the nuclear launch—Lagrange Decomposition of a function—II

Preface: Drunk president and the Nuclear launch—Shamir Secret Sharing

5 min read · Dec 9, 2018

5

Prateek Nischal in Walmart Global Tech Blog

Modding JJWT

JSON Web Tokens:

5 min read · Apr 27, 2017

17

See all from Prateek Nischal

Recommended from Medium

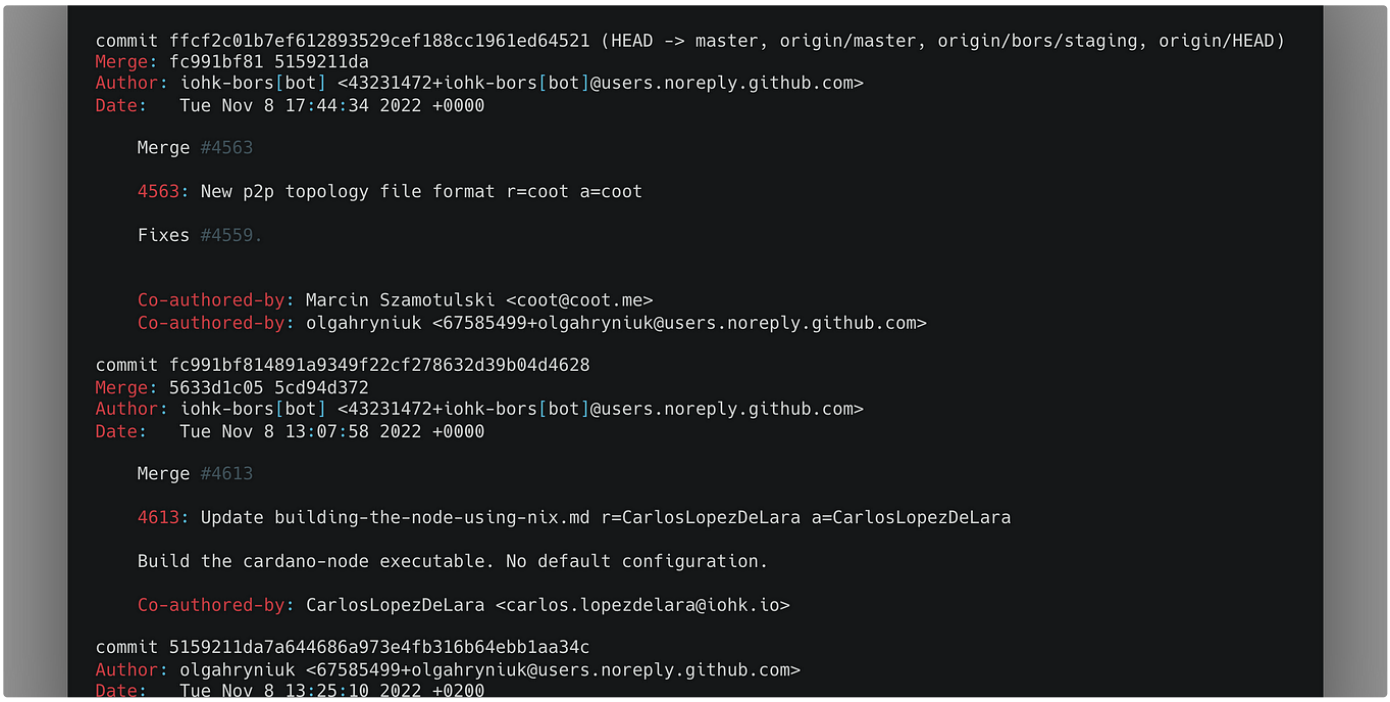
Prabhash

Secure gRPC APIs with OAuth2

Securing gRPC with OAuth2 end to end example with client and server using client credentials flow and Spring Boot Authorization Server.

· 6 min read · Mar 27

160



Jacob Bennett in Level Up Coding

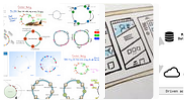
Use Git like a senior engineer

Git is a powerful tool that feels great to use when you know how to use it.

✦ · 4 min read · Nov 14, 2022

7.2K 75

Lists

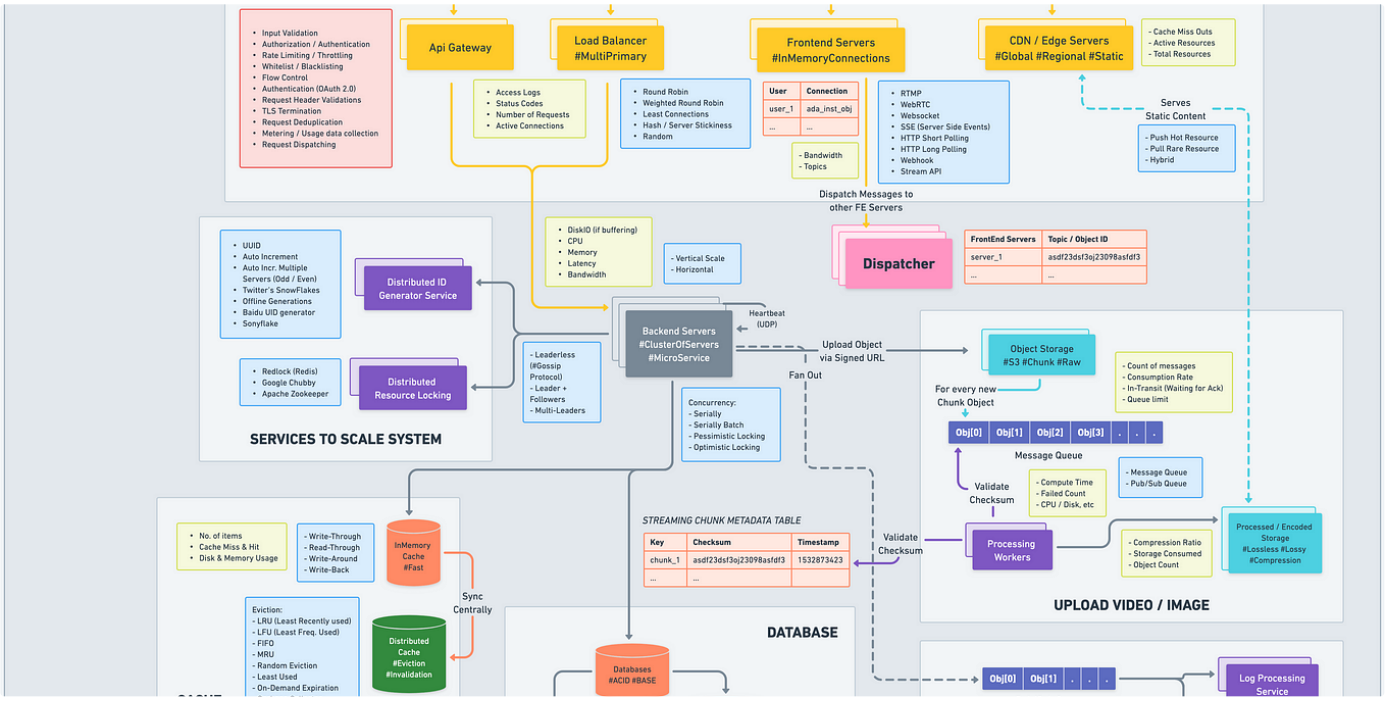


General Coding Knowledge

20 stories · 63 saves

Now in AI: Handpicked by Better Programming

249 stories · 31 saves



Love Sharma in ByteByteGo System Design Alliance

System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However, understanding the key concepts and components can make the...

✦ · 9 min read · Apr 20

5.8K 49

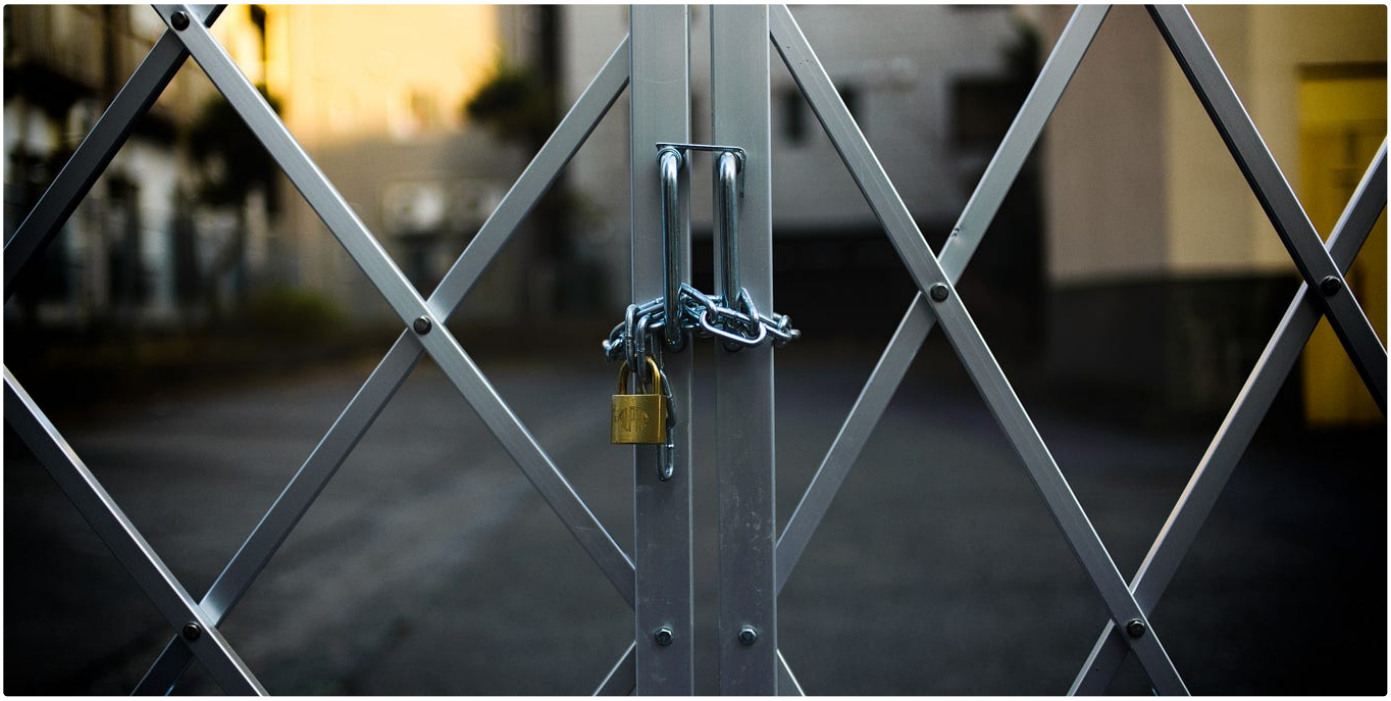
Unbecoming

10 Seconds That Ended My 20 Year Marriage

It's August in Northern Virginia, hot and humid. I still haven't showered from my morning trail run. I'm wearing my stay-at-home mom...

✦ · 4 min read · Feb 16, 2022

53K 830

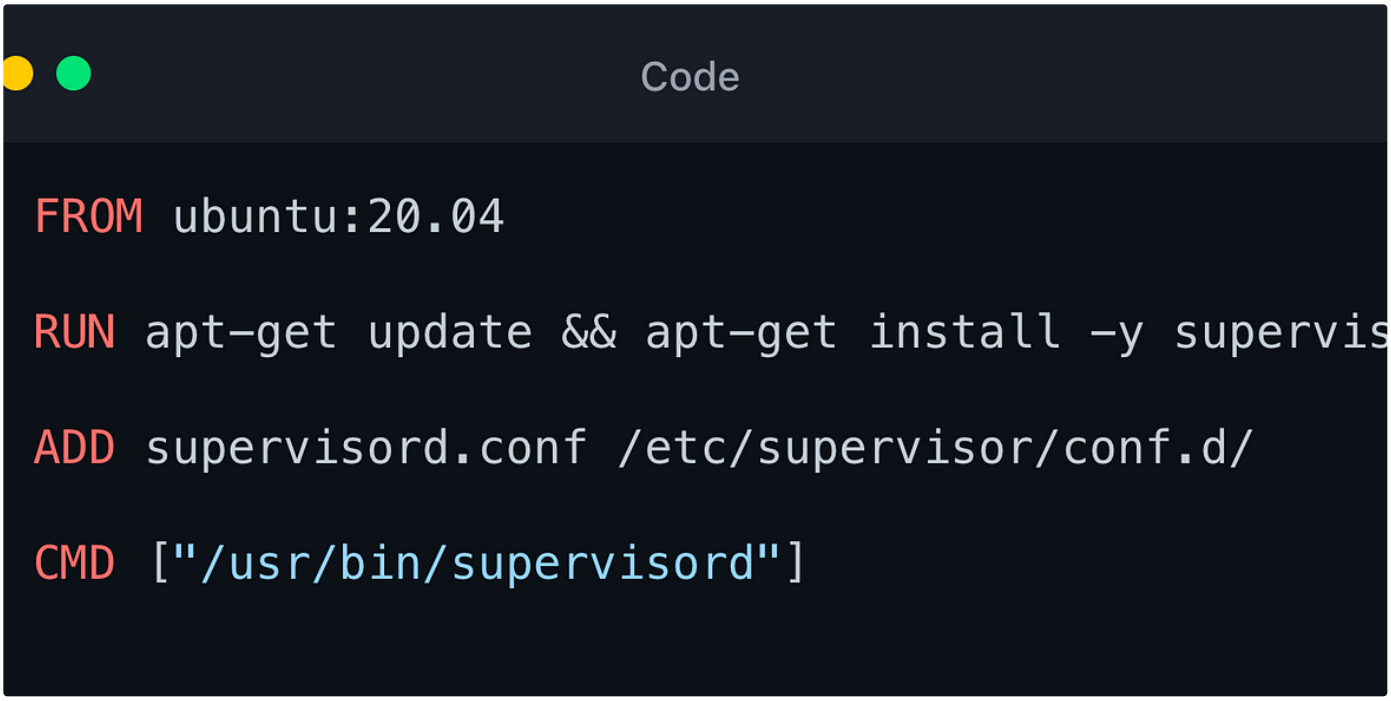


Noval Agung Prayogo in Level Up Coding

Golang Dockerfile for Project with Private Dependencies using HTTPS (without SSH)

Learn on how to dockerize Golang project with private modules/dependency using HTTPS only (without SSH)

· 4 min read · Feb 24



Ismat Babirli

Use Supervisor in Docker Container

Welcome to our blog post on using Supervisor in Docker containers. As more and more companies are adopting containerization for their...

· 12 min read · Jan 17

5

See more recommendations