



Building Zap - A look under the hood

Jack Mallers
October 24th, 2018



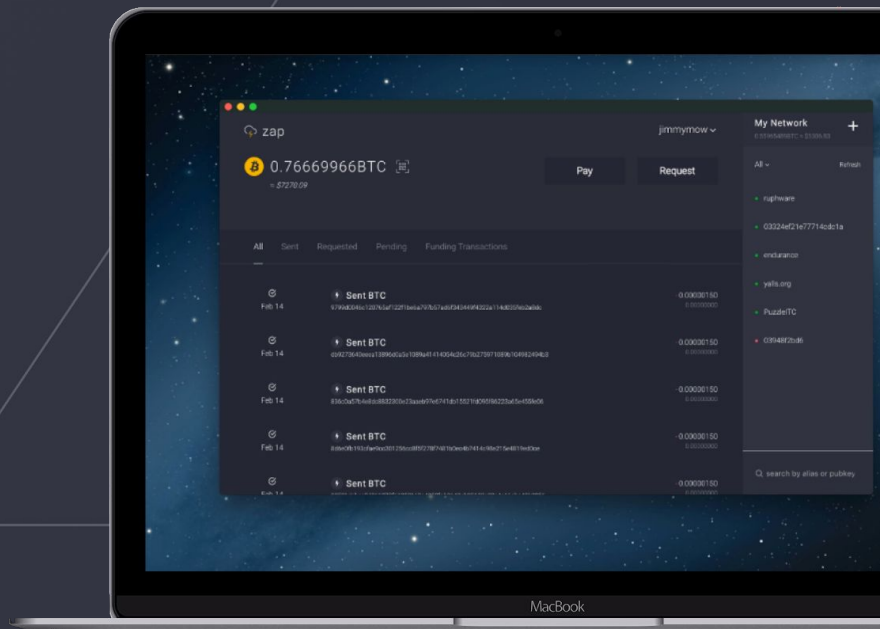
<https://github.com/LN-Zap>

- [zap-desktop](#)
- [zap-iOS](#)
- [swiftBTC](#)
- [docker-btc](#)
- [ln](#)
- [zapconnect](#)
- [zap-tutorials](#)
- [docker-ln](#)






Zap Desktop

- Why not React Native?
- Ensure code quality
- Automate workflows
- High level look
- LND / BTCD
- Backend components





React Native goals

- Allow a team to move faster 
- Only write code once for mobile 
- Improve upon the developer experience 



React Native

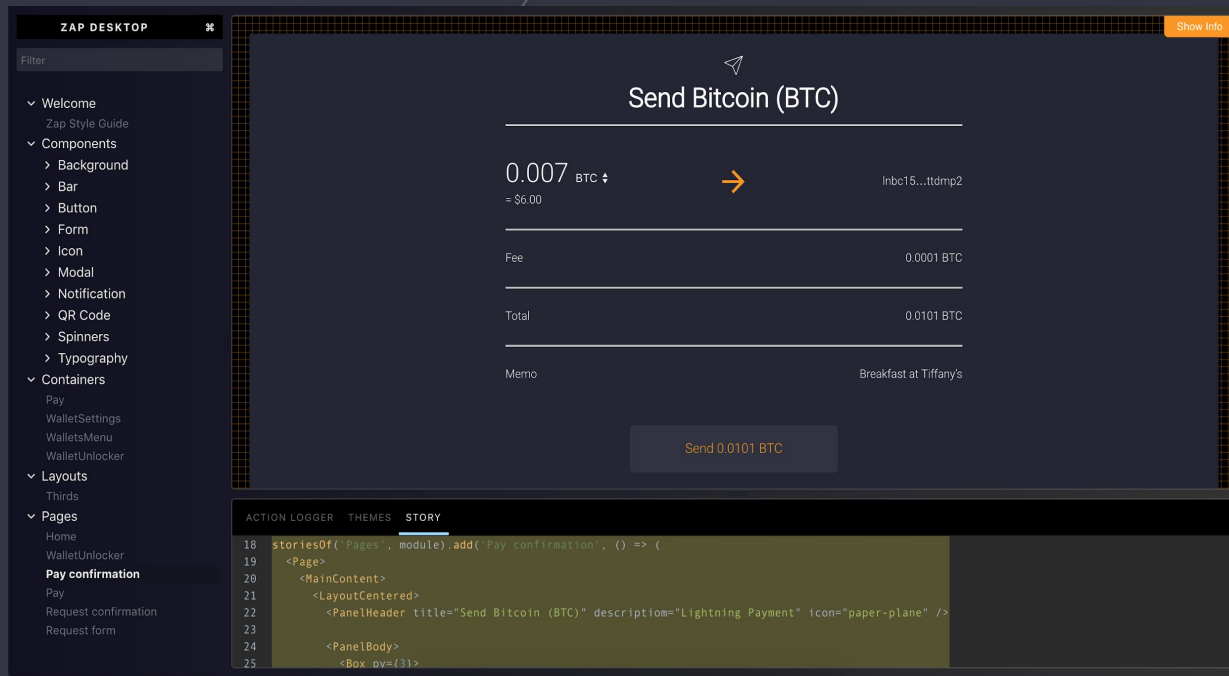
- Highly ambitious, immature and moves fast
- Still requires native development (Native Modules)
- Performance and app size not as good as native
- Attracting contributors is difficult



Ensure code quality

- Flowtype
- Eslint
- Stylelint
- Prettier
- Coveralls
- Storybook

<https://ln-zap.github.io/zap-desktop/?no-cache=1>





Your Wallets

Wallet #1


Wallet #2

More

Raspi Node

Digital Ocean Node

BTC Pay Server

 Create new wallet

Wallet #1

Launch now

Settings

Chain

☒ Bitcoin

☐ Litecoin

Autopilot

☒ On

Max number of Autopilot channels

5

Percentage of balance to use



60 %



Automated workflows

- Github
- Travis
- AppVeyor
- Electron-Builder
- GoReleaser



High level look

- Electron + React (Redux)
- Spawn local LND with neutrino + remote node + BTCPay
- Communicating with LND with gRPC



Electron and React

- Write code once for all platforms
- Strong community support
- More favorable and flexible for designers
- Reusable components for potential/future web apps



Zap + LND

shell python javascript

```
> var fs = require('fs');
> var grpc = require('grpc');
> var lnrpc = grpc.load('rpc.proto').lnrpc;
> process.env.GRPC_SSL_CIPHER_SUITES = 'HIGH+ECDSA'
> var-lndCert = fs.readFileSync('LND_DIR/tls.cert');
> var sslCreds = grpc.credentials.createSsl(lnCert);
> var macaroonCreds = grpc.credentials.createFromMetadataGenerator(function(args) {
    var macaroon = fs.readFileSync("LND_DIR/admin.macaroon").toString('hex');
    var metadata = new grpc.Metadata()
    metadata.add('macaroon', macaroon);
    callback(null, metadata);
  });
> var creds = grpc.credentials.combineChannelCredentials(sslCreds, macaroonCreds);
> var lightning = new lnrpc.Lightning('localhost:10009', creds);
> var request = {}
> lightning.getInfo(request, function(err, response) {
    console.log(response);
  })
{
  "identity_pubkey": <string>,
  "alias": <string>,
  "num_pending_channels": <uint32>,
  "num_active_channels": <uint32>,
  "num_peers": <uint32>,
  "block_height": <uint32>,
  "block_hash": <string>,
  "synced_to_chain": <bool>,
  "testnet": <bool>,
  "chains": <array string>,
  "uris": <array string>,
  "best_header_timestamp": <int64>,
  "version": <string>,
}
```



Zap + LND

```
class Lightning {  
  mainWindow: BrowserWindow  
  service: any  
 -lndConfig: LndConfig  
  subscriptions: LightningSubscriptionsType  
  fsm: StateMachine  
}
```



Zap + LND

```
this.fsm = new StateMachine({  
  init: 'ready',  
  transitions: [  
    { name: 'connect', from: 'ready', to: 'connected' },  
    { name: 'disconnect', from: 'connected', to: 'ready' },  
    { name: 'terminate', from: 'connected', to: 'ready' }  
  ],  
  methods: {  
    onBeforeConnect: this.onBeforeConnect.bind(this),  
    onBeforeDisconnect: this.onBeforeDisconnect.bind(this),  
    onBeforeTerminate: this.onBeforeTerminate.bind(this)  
  }  
})
```



LND + Neutrino

```
// Spawn lnd process.  
this.process = spawn(this.lndConfig.binaryPath, neutrinoArgs)  
.on('error', error => {  
  mainLog.debug('Neutrino process received "error" event with error: %s', error)  
  this.emit(ERROR, error, this.lastError)  
})  
.on('exit', (code, signal) => {  
  mainLog.debug(  
    'Neutrino process received "exit" event with code %s and signal %s',  
    code,  
    signal  
  )  
  this.process = null  
  this.emit(EXIT, code, signal, this.lastError)  
})
```



Remote node connections

```
startLnd({  
  type: connectionType,  
  string: connectionString,  
  host: connectionHost,  
  cert: connectionCert,  
  macaroon: connectionMacaroon  
})
```



BTCPay connection

```
{
  "configurations":
  [
    {
      "chainType": "Mainnet",
      "type": "grpc",
      "cryptoCode": "BTC",
      "host": "btcpay.nicolas-dorier.com",
      "port": 443,
      "ssl": true,
      "certificateThumbprint": null,
      "macaroon": "example_macaroon_here"
    }
  ]
}
```




Communicating with LND

- gRPC
- REST

[Back to Developer Site](#)

LND gRPC API Reference
GenSeed
InitWallet
UnlockWallet
ChangePassword
WalletBalance
ChannelBalance
GetTransactions
SendCoins
SubscribeTransactions
SendMany
NewAddress
SignMessage
VerifyMessage
ConnectPeer
DisconnectPeer
ListPeers
GetInfo
PendingChannels
ListChannels
ClosedChannels
OpenChannelSync
OpenChannel

LND gRPC API Reference

Welcome to the gRPC API reference documentation for LND, the Lightning Network Daemon.

This site features the API documentation for [Incl](#) (CLI), [Python](#), and [JavaScript](#) in order to communicate with a local `lnd` instance through gRPC. It is intended for those who already understand how to work with LND. If this is your first time or you need a refresher, you may consider perusing our LND developer site featuring a tutorial, resources and guides at dev.lightning.community.

The examples to the right assume that there is a local `lnd` instance running and listening for gRPC connections on port 10009. `LND_DIR` will be used as a placeholder to denote the base directory of the `lnd` instance. By default, this is `~/lnd` on Linux and `~/Library/Application Support/Lnd` on macOS.

At the time of writing this documentation, two things are needed in order to make a gRPC request to an `lnd` instance: a TLS/SSL connection and a macaroon used for RPC authentication. The examples to the right will show how these can be used in order to make a successful, secure, and authenticated gRPC request.

The original `rpc.proto` file from which the gRPC documentation was generated can be found [here](#).

Alternatively, the REST documentation can be found [here](#).

GenSeed

Simple RPC

GenSeed is the first method that should be used to instantiate a new `lnd` instance. This method allows a caller to generate a new `aezeed` cipher seed given an optional passphrase. If necessary to decrypt the cipherseed to expose the internal

shell python javascript

<https://dev.lightning.community>





```
case 'sendCoins':  
  // Transaction looks like { txid: String }  
  // { amount, addr } = data  
  walletController  
    .sendCoins(Ind, data)  
    .then(({ txid }) =>  
      event.sender.send('transactionSuccessful', { txid }))  
    )  
    .catch(error => {  
      log.error('error: ', error)  
      event.sender.send('transactionFailed', { error })  
    })  
  break  
case 'openChannel':  
  // Response is empty. Streaming  
  // { pubkey, localamt, pushamt }  
  channelController  
    .openChannel(Ind, event, data)  
    .then(channel => {  
      log.info('CHANNEL: ', channel)  
      event.sender.send('channelSuccessful', { channel })  
      return channel  
    })  
    .catch(error => log.error('openChannel:', error))
```





```
await this.openChannel({ pubkey, amount });
```



Managing LND + BTCD

- Continually track HEAD
- Manage our own LND fork
- Ind-binary
- Patch and test PRs that affect Zap



Our own LND fork

- Ability to bundle pre-built binaries with our codebase (developers don't need to set it up themselves)
- Ability to bundle binaries with experimental features enabled that are not available in the binaries that Lightning Labs creates
- Ability to bundle binaries with new / upcoming / unmerged PR's
- Ability to release updated versions of LND on our own schedule



Ind-binary

Install LND from npm ⚡

```
> npm install Ind-binary
```

```
> npx Ind --version
```

```
Ind version 0.5.0-beta commit=v0.5-beta-148-g6b19df162a161079ab794162b45e8f4c7bb8beec-dirty
```



Ind-binary

```
// package.json
"config": {
  "style_paths": "app/styles/*.scss app/components/**/*.scss app/components/**/*.js",
  "Ind-binary": {
    "binaryVersion": "0.5-beta-21-g25145acc",
    "binarySite": "https://github.com/LN-Zap/Ind/releases/download"
  }
}
```




Ind-binary

```
const IndBinary = require('Ind-binary')

function install(platform, arch, dest) {
  process.env.LND_BINARY_PLATFORM = platform
  process.env.LND_BINARY_ARCH = arch
  process.env.LND_BINARY_DIR = dest

  return IndBinary.install()
}

return install('darwin', 'amd64', 'resources/bin/mac/x64')
  .then(() => install('darwin', '386', 'resources/bin/mac/ia32'))
  .then(() => install('windows', 'amd64', 'resources/bin/win/x64'))
  .then(() => install('windows', '386', 'resources/bin/win/ia32'))
  .then(() => install('linux', 'amd64', 'resources/bin/linux/x64'))
  .then(() => install('linux', '386', 'resources/bin/linux/ia32'))
```



Ind-binary

```
> yarn package
```

```
"scripts": {  
  ...  
  "fetch-lnd": "node ./internals/scripts/fetch-lnd-for-packaging.js",  
  ...  
  "package": "npm run build && npm run fetch-lnd && build",  
  ...  
}
```



Backend components

- BTCD
- LND
- Docker
- Kubernetes
- Google Cloud
- All infrastructure components versioned and deployable with a single command



Thanks



https://twitter.com/ln_zap



<https://github.com/LN-Zap>



<https://zaphq.slack.com>



<https://crowdin.com/project/zap-desktop-test>