1. Bitcoin CLI and Bitcoin daemon via Docker
2. Easy development with docker-compose
3. Allows us to easily rebuild when needed and start over
4. Platform agnostic
5. Exposes daemon to computer

```yaml
version: "3.4"
services:
  bitcoind:
    build: .
    ports:
      - "18444:18444"
      - "18443:18443"
    dns:
      - 8.8.8.8
```

```
FROM ubuntu:20.04

RUN apt-get update

RUN apt-get -yq install tar wget

RUN wget -O home/bitcoin-core.tar.gz https://bitcoin.org/bin/bitcoin-core-0.21.1/bitcoin-0.21.1-x86_64-linux-gnu.tar.gz

RUN cd home

RUN tar -xvzf home/bitcoin-core.tar.gz -C /home

ENTRYPOINT ["./home/bitcoin-0.21.1/bin/bitcoind","-regtest", "-rpcuser=user", "-rpcpassword=secret", "-rpcbind=0.0.0.0", "-rpcallowip=0.0.0.0/0", "-fallbackfee=0.00005"]
```

```
/bitcoin-0.21.1/bin/bitcoin-cli -regtest -rpcuser=user -rpcpassword=secret createwallet daniel

{
  "name": "daniel",
  "warning": ""
}
```

```
/bitcoin-0.21.1/bin/bitcoin-cli -regtest -rpcuser=user -rpcpassword=secret createwallet niels
{
  "name": "niels",
  "warning": ""
}
```

```
/bitcoin-0.21.1/bin/bitcoin-cli -regtest -rpcwallet=daniel -rpcuser=user -rpcpassword=secret -
generate 101
{
  "address": "bcrt1qxqgjpexeq05phgfx76y56thf6f2vaht8jg5xn5",
  "blocks": [
    "4e7298432767a172e075d7ed483d9f1c2fc18041522d0975011776cdec2924b6",
    "0b882bd9cc90947548f1a3972718400a8210489a70d65a1974056002b99ad6d1",
    "4d26dcd841e28671c283935be17ef56a27ddbeb566476fb1a99a4d16f0f347ec",
    "43477fe80d2a82d747886848aff27658e9e37bc585f252ba28050875e721ed7f",
    "4fc47ef293f52493b4fa6a7180b71775530c9fe694579bbb9eacbeb4568d7b84",
    "2cc17890f5363d3702acca50a9bb4e981388624aa458e7a38e6493adca23209b",
```

We generate two wallets using the bitcoin command-line utility.

Afterwards we then generate 101 blocks to daniel's wallet

```
using System;
using System.Text.Json.Serialization;

namespace Assignment1.Requests
{
    public class RpcRequest
    {
        [JsonPropertyName("jsonrpc")]
        public string JsonRpc { get; set; } = "1.0";

        [JsonPropertyName("id")]
        public string Id { get; set; } = Guid.NewGuid().ToString();

        [JsonPropertyName("method")]
        public string Method { get; set; }

        [JsonPropertyName("params")]
        public object[] Params { get; set; }


        public RpcRequest(string method, params object[] @params)
        {
            Method = method;
            Params = @params;
        }
    }
}
```

We represent requests to the bitcoin daemon using a C# class

And likewise represent a response using a class.

```
using System.Collections.Generic;
using System.Text.Json.Serialization;

namespace Assignment1.Response
{
    public class GetUnspentTransactions
    {
        [JsonPropertyName("result")]
        public List<Add> Result { get; set; }

        [JsonPropertyName("error")]
        public string Error { get; set; }

        [JsonPropertyName("id")]
        public string Id { get; set; }
    }

    public class Add
    {
        [JsonPropertyName("txid")]
        public string Id { get; set; }

        [JsonPropertyName("address")]
        public string Address { get; set; }

        [JsonPropertyName("label")]
        public string Label { get; set; }

        [JsonPropertyName("amount")]
        public decimal Amount { get; set; }
    }
}
```

A generic `GetRequest<T>(..)` method is created to communicate with our regtest RPC server.

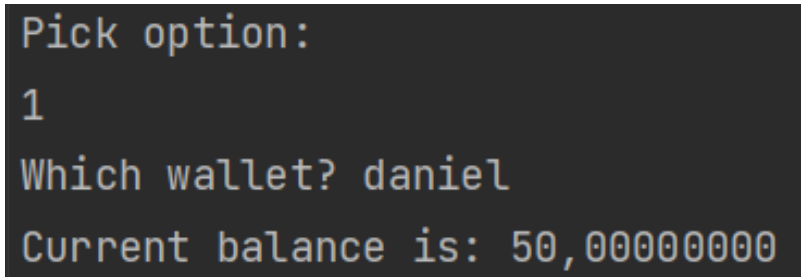
```
private static async Task<T> GetRequest<T>(RpcRequest request, string wallet = null)
        {
            using var httpClient = new HttpClient();
            using var message = new HttpRequestMessage(new HttpMethod("POST"),
                $"http://localhost:18443/" + (wallet == null ? "" : $"wallet/{wallet}"));
            var base64Authorization = Convert.ToBase64String(Encoding.ASCII.GetBytes("user:secret"));
            message.Headers.TryAddWithoutValidation("Authorization", $"Basic {base64Authorization}");

            message.Content = new StringContent(JsonSerializer.Serialize(request));
            var response = await httpClient.SendAsync(message);
            var readAsStringAsync = await response.Content.ReadAsStringAsync();

#if DEBUG
            Console.WriteLine(readAsStringAsync);
#endif

            var jsonResponse = JsonSerializer.Deserialize<T>(readAsStringAsync);

            return jsonResponse;
        }
```



Our CLI is an easy way to communicate with the RPC server and allows 4 different interactions

```
Pick option:
2
Which wallet? niels
New address is: bcrt1q7ddevwtdgcrrgxemvh7wd6yftmu59l9l8azk3s
```

```
Pick option:
3
From which wallet? daniel
Enter address to send to: bcrt1q7ddevwtdgcrrgxemvh7wd6yftmu59l9l8azk3s
Enter amount of BigBoi Coins to send: 25
The coins have been sent :)
```

Daniel has sent the coins but Niels has not received it :(

```
Pick option:
1
Which wallet? niels
Current balance is: 0,00000000
```

One block later

```
Pick option:
1
Which wallet? niels
Current balance is: 24,99999295
```

This is due to the current block being the same

Issues with the RPC server

1. Limitations of what commands you can run against it
2. Limited documentation available