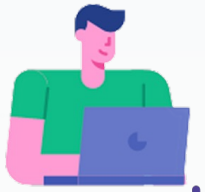
浙江大学
ZHEJIANG UNIVERSITY

Vodis

基于VOLO-RPC的Mini-Redis实现

Vodis 架构



CLIENT

VOLO

SERVER

TRANSACTIONS

AOF BUFF

CLUSTER

MASTER

MASTER

MASTER

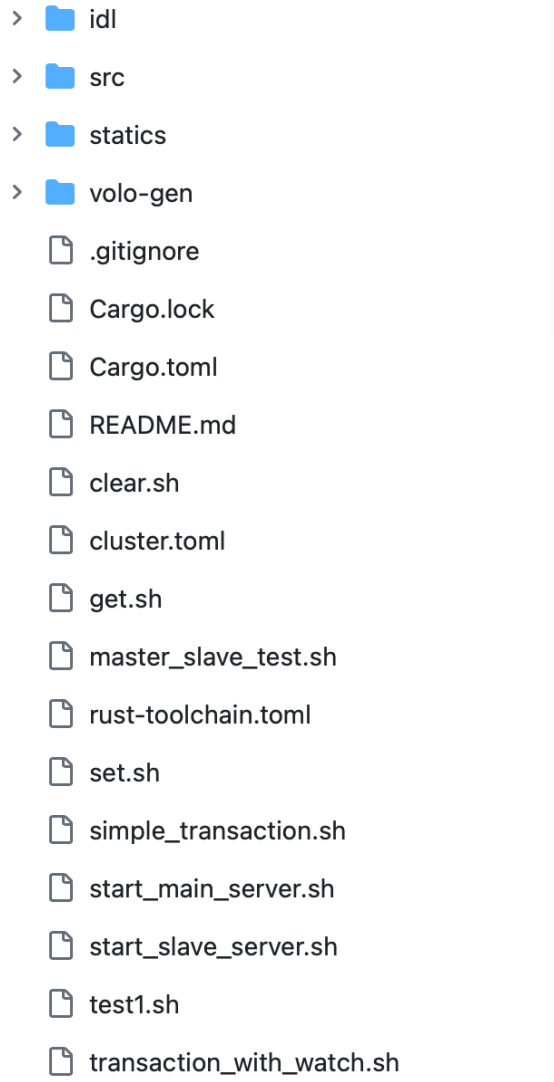
SLAVE

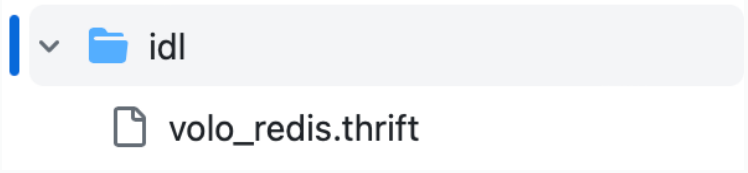
SLAVE

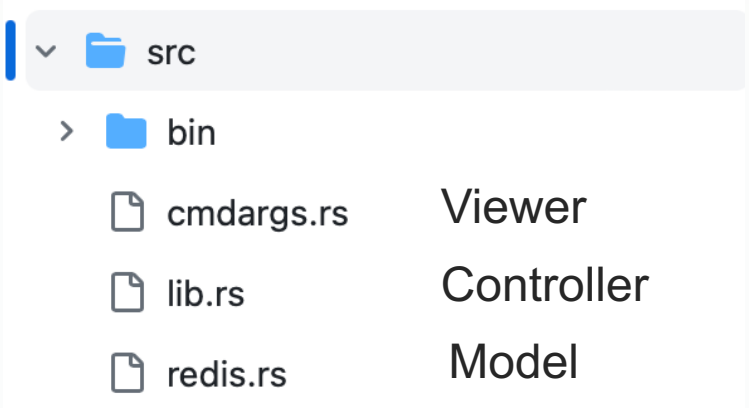
SLAVE

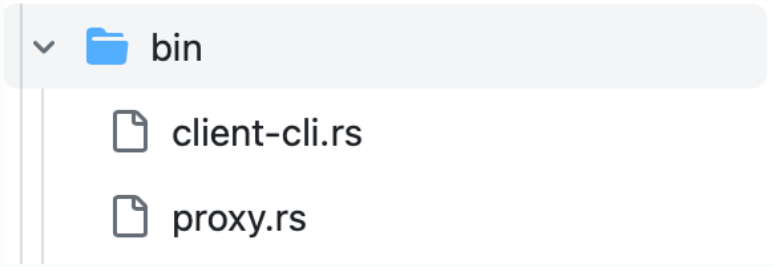
AOF FILE

文件结构


idl
src
statics
volo-gen
.gitignore
Cargo.lock
Cargo.toml
README.md
clear.sh
cluster.toml
get.sh
master_slave_test.sh
rust-toolchain.toml
set.sh
simple_transaction.sh
start_main_server.sh
start_slave_server.sh
test1.sh
transaction_with_watch.sh


idl
volo_redis.thrift


src
bin
cmdargs.rs
lib.rs
redis.rs
Viewer
Controller
Model


bin
client-cli.rs
proxy.rs

亮点:活用语言特性

```rust
async fn subscribe(handle: String) -> ! {
    loop {
```

```rust
pub fn get_client(addr: impl Into<Address>) -> volo
    volo_gen::volo::redis::ItemServiceClientBuilder
```

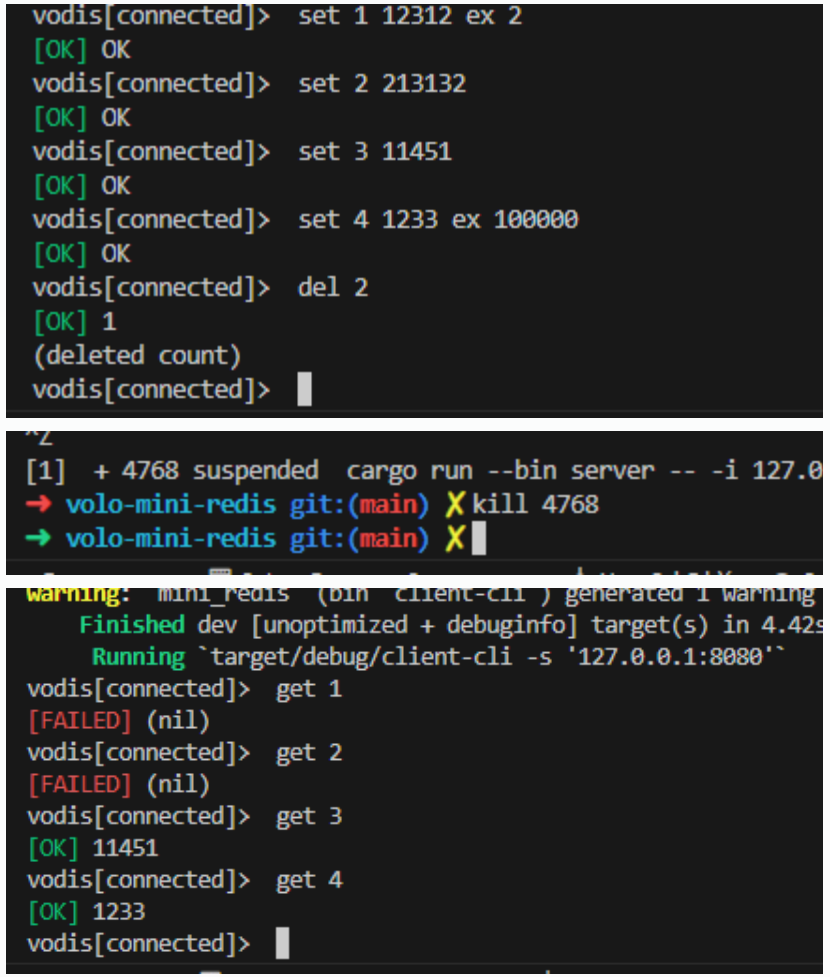
```rust
tokio::spawn(future: async move {
    //AOF线程，实现everysec策略
```

1.AOF （Append-only File）实现持久化 （30分）

1.对于一个已经存在的 log file（格式自定，文件自行准备），在 redis server 启动的时候可以根据该文件重建redis 数据。

2.提供一个测试脚本:

Client 发送一些 Set 操作之后，重启 Server，然后再用 client 进行 Get 调用，可以访问到之前的数据



浙江大学 ZHEJIANG UNIVERSITY

1.AOF （Append-only File）实现持久化 （30分）

亮点：

1.AOF支持Expire time；

2.AOF支持两种写盘策略：

混合模式(低负载）——定时(1s)刷盘

高负载——缓冲区满刷盘

```
vodis[connected]> set 1 12312 ex 2
[OK] OK
vodis[connected]> set 2 213132
[OK] OK
vodis[connected]> set 3 11451
[OK] OK
vodis[connected]> set 4 1233 ex 100000
[OK] OK
vodis[connected]> del 2
[OK] 1
(deleted count)
vodis[connected]>
```

```
[1]  + 4768 suspended  cargo run --bin server -- -i 127.0
➜ volo-mini-redis git:(main) X kill 4768
➜ volo-mini-redis git:(main) X
```

```
warning: mini_redis (bin "client-cli") generated 1 warning
    Finished dev [unoptimized + debuginfo] target(s) in 4.42s
     Running `target/debug/client-cli -s '127.0.0.1:8080'`
vodis[connected]> get 1
[FAILED] (nil)
vodis[connected]> get 2
[FAILED] (nil)
vodis[connected]> get 3
[OK] 11451
vodis[connected]> get 4
[OK] 1233
vodis[connected]>
```

2.Redis 主从架构（30分）

1.定义一个配置文件的格式，可以从该配置文件启动多个 redis 实例，并且区分哪些是主节点，哪些是从节点。

2.从节点对于 Set 操作返回错误 。

3.对主节点进行 Set 操作，可以通过从节点通过 Get 获取到设置的数据。

亮点：

1.支持一主多从，从上加从

2.支持集群间事务(通过主为从分发的uuid实现指令来源区分)

```
    Finished dev [unoptimized + debuginfo] target(s) in 2.76s
     Running `target\debug\client-cli.exe -s 127.0.0.1:8080`
vodis[connected]>  get 1
[OK] 4617238123
vodis[connected]>  set 1 12312312
[OK] OK
vodis[connected]>
```

```
warning: `mini_redis` (bin "client-cli") generated 1 warning (run `cargo fix --
    Finished dev [unoptimized + debuginfo] target(s) in 0.32s
     Running `target\debug\client-cli.exe -s 127.0.0.1:8888`
vodis[connected]>  get 1
[OK] 4617238123
vodis[connected]>  get 1
[OK] 12312312
vodis[connected]>
```

```
warning: `mini_redis` (lib) generated 8 warnings (run `cargo fix --lib -p mini_redis`
    Finished dev [unoptimized + debuginfo] target(s) in 0.30s
     Running `target\debug\server.exe -i 127.0.0.1 -p 8888 --slaveof 127.0.0.1:8080 -
Sync send.
2023-09-15T02:35:29.126315Z  INFO volo_thrift::server: [VOLO] server start at: Tcp(Tc
0.0.1:8888, socket: 344 }) } })
2023-09-15T02:35:29.714687Z  INFO mini_redis: Serialize success!!!
xxx
COMMAND SAVED!!
propagate to 0 clients
xxx
COMMAND SAVED!!
propagate to 0 clients
```

3.Redis Cluster （40分）

1.实现一个 Redis Proxy，可以将 Set 和 Get 请求转发到其他的 Redis 实例上进行处理。

2.定义一个配置文件的格式，可以从该配置文件启动多个 redis 实例和一个 redis proxy。

3.Redis Proxy 可以根据 Get 和 Set 目标的 key 将请求分发到不同的 redis 实例进行处理。

测试样例

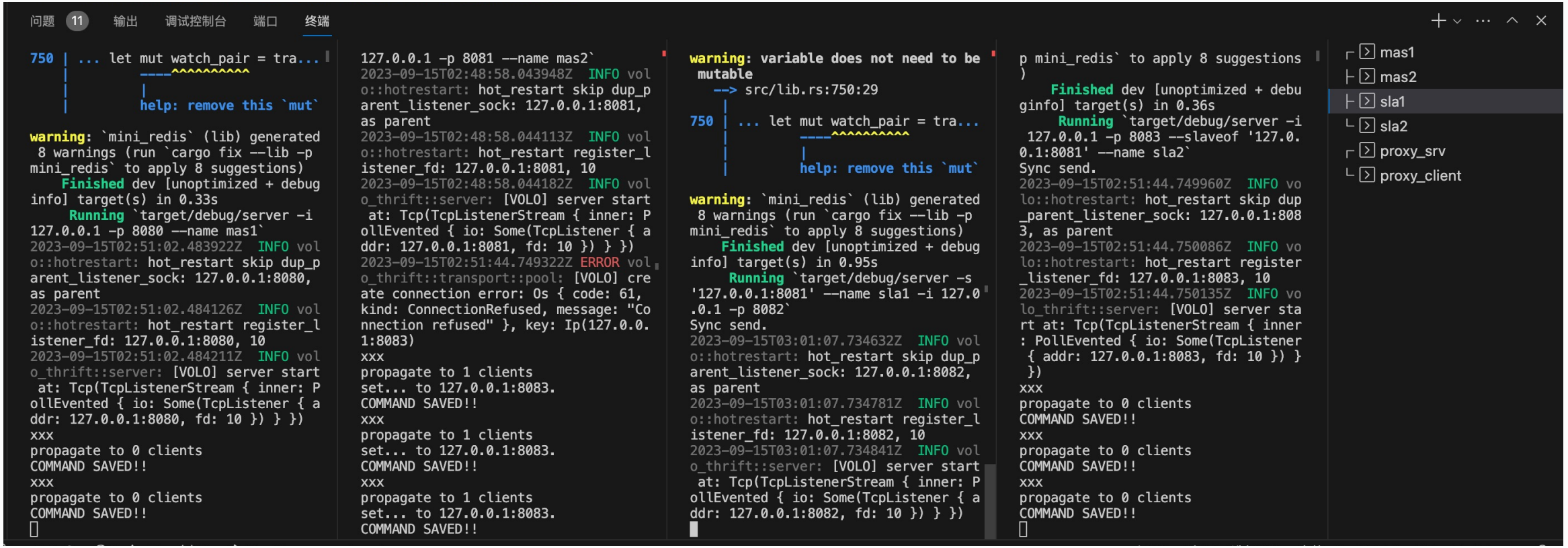
3.Redis Cluster （40分）

8080 主1-8082从1
8081 主2-8083从2
proxy：8888
get/set经过proxy→主→从 多层转发。

```
70    cargo run --bin proxy -- --cfg cluster.toml # 挂载到proxy服务器

问题 11   输出   调试控制台   端口   终端

    |         |
    |         help: remove this `mut`

warning: variable does not need to be mutable
   --> src/lib.rs:744:21
    |
744 | ...et mut transaction = tran...
    |       ----^^^^^^^^^^^
    |       |
    |       help: remove this `mut`

warning: variable does not need to be mutable
   --> src/lib.rs:750:29
    |
750 | ... let mut watch_pair = tra...
    |         ----^^^^^^^^^^
    |         |
    |         help: remove this `mut`

warning: `mini_redis` (lib) generated 8 warnings (run `cargo fix --lib -p mini_
redis` to apply 8 suggestions)
    Finished dev [unoptimized + debuginfo] target(s) in 0.41s
     Running `target/debug/server -i 127.0.0.1 -p 8888 --name proxy`
2023-09-15T02:53:43.875671Z  INFO volo::hotrestart: hot_restart skip dup_parent
_listener_sock: 127.0.0.1:8888, as parent
2023-09-15T02:53:43.875885Z  INFO volo::hotrestart: hot_restart register_listen
er_fd: 127.0.0.1:8888, 10
2023-09-15T02:53:43.875980Z  INFO volo_thrift::server: [VOLO] server start at:
Tcp(TcpListenerStream { inner: PollEvented { io: Some(TcpListener { addr: 127.0
.0.1:8888, fd: 10 }) } })

warning: `mini_redis` (bin "proxy") generated 2 warnings (run `cargo fix --bin
 "proxy"` to apply 1 suggestion)
    Finished dev [unoptimized + debuginfo] target(s) in 0.48s
     Running `target/debug/proxy -a '127.0.0.1:8888' --masters '127.0.0.1:8080
' --masters '127.0.0.1:8081'`
vodis[cluster:proxy]>  set a a
2023-09-15T02:55:11.473978Z  INFO proxy: proxyed to 127.0.0.1:8081.
[OK] OK
vodis[cluster:proxy]>  set b b
2023-09-15T02:55:14.461754Z  INFO proxy: proxyed to 127.0.0.1:8081.
[OK] OK
vodis[cluster:proxy]>  set c c
2023-09-15T02:55:16.848329Z  INFO proxy: proxyed to 127.0.0.1:8081.
[OK] OK
vodis[cluster:proxy]>  set d d
2023-09-15T02:55:19.449357Z  INFO proxy: proxyed to 127.0.0.1:8080.
[OK] OK
vodis[cluster:proxy]>  set e e
2023-09-15T02:55:21.979445Z  INFO proxy: proxyed to 127.0.0.1:8080.
[OK] OK
vodis[cluster:proxy]>  get nope
[FAILED] (nil)
2023-09-15T02:55:27.422097Z  INFO proxy: (by Bloom)
vodis[cluster:proxy]>  get e
2023-09-15T02:55:30.849526Z  INFO proxy: proxyed to 127.0.0.1:8080.
[OK] e
vodis[cluster:proxy]>  get a
2023-09-15T02:55:33.921922Z  INFO proxy: proxyed to 127.0.0.1:8081.
[OK] a
vodis[cluster:proxy]>

mas1
mas2
sla1
sla2
proxy_srv
proxy_client
```

3.Redis Cluster （40分）

8080 主1-8082从1
8081 主2-8083从2
proxy：8888
get/set经过proxy→主→从 多层转发。



Graceful exit （10分）

收到退出信号时可以等待当前正在处理的连接执行结束之后再退出。

模拟处理服务过程

```
match _req.cmd {
    RedisCommand::Ping => {
        tokio::time::sleep(Duration::from_secs(10)).await;
        if let Some(arg: Vec<FastStr>) = _req.args {
```

Ping后等待响应

```
vodis[connected]>  ping
```

捕获Ctrl- C后进入等待

```
0.0.1:8080, socket: 324 }) } })
2023-09-15T02:45:09.686504Z  INFO volo_thrift::server: [VOLO] received
2023-09-15T02:45:09.686527Z  INFO mini_redis: Ctrl-C received, shutting
```

等待服务处理完后退出

```
2023-09-15T02:54:00.345424Z  INFO volo_thrift::server: [VOLO] server start at: Tcp(TcpListenerStream
0.0.1:8080, socket: 336 }) } })
2023-09-15T02:54:16.984492Z  INFO volo_thrift::server: [VOLO] received signal, gracefully exiting no
2023-09-15T02:54:16.984538Z  INFO mini_redis: Ctrl-C received, shutting down
2023-09-15T02:54:23.809971Z  INFO mini_redis: New requests rejected whe shutting down.
2023-09-15T02:54:23.809971Z  INFO mini_redis: shutdown finally
```

Transactions （20分）

1.MULTI EXEC 命令实现。

2.WATCH 命令实现。

```
vodis[connected]>  multi
[OK] M8Qpi
vodis[transaction]>  watch 1
[OK] OK
vodis[transaction]>  get 1
[OK] 1
vodis[transaction]>  set 1 1
[OK] OK
vodis[transaction]>  get 1 -t M8Qpi
[OK] OK
vodis[transaction]>  set 2 -t M8Qpi
error: the following required arguments were not provided:
  <VALUE>

Usage: set --transaction-id <TRANSACTION_ID> <KEY> <VALUE> [TTYPE] [TNUM]

For more information, try '--help'.
vodis[transaction]>  set 2 1 -t M8Qpi
[OK] OK
vodis[transaction]>  exec
[OK] 1
[OK] OK
vodis[connected]>  get 1
[OK] 1
vodis[connected]>  get 2
[OK] 1
vodis[connected]>
```

```
vodis[connected]>  multi
[OK] c9GEc
vodis[transaction]>  watch 1
[OK] OK
vodis[transaction]>  set 1 1145
[OK] OK
vodis[transaction]>  exec
2023-09-15T02:28:23.821340Z ERROR client_cli: Application(ApplicationError { kind: Appli
vodis[transaction]>
```

其他实现细节

异步sync：

基于云函数rpc实现“远程interrupt”的效果；

布隆过滤器：

减少缓存击穿与读放大；

基于状态机的分支处理；

可扩展性：

低耦合，可以简单从大作业要求修改到真实redis工作模型；

包装完善的命令行工具。

```
❯ ./target/debug/client-cli.exe
vodis[connected]>  help
Usage: mini_redis <COMMAND>

Commands:
  ping       ping the server
  set        set a key-value pair
  get        get a value by key, nil if key not exist
  del        delete a key-value pair, error if key not exist
  exit       quit the client
  publish    publish a message to a channel
  subscribe  subscribe a channel
  watch      watch a transaction
  multi      start a transaction
  exec       execute a transaction
  help       Print this message or the help of the given subcommand(s)

Options:
  -h, --help     Print help
  -V, --version  Print version
vodis[connected]>
```

```
❯ ./target/debug/server.exe -h
Usage: server.exe [OPTIONS] --ip <IP> --port <port>

Options:
  -n, --name <NAME>              Server name, used in default AOF file name. Will be randomly chosen if not provided
  -a, --aof <FILE>               Optional AOF file path
  -s, --slaveof <Master IP:PORT>  Mark this Vodis instance as a slave
  -i, --ip <IP>
  -p, --port <port>
  -c, --cluster...               Sets a custom config file
      --pre-run <PRE_RUN>        Execute provided commands after initialization
  -h, --help                     Print help
  -V, --version                  Print version
```