Implementation of the server-client solution for storing KV data, lightweight analog of the Redis (https://redis.io/).

Minimal requirements:

Golang version: 1.11.1.

Code style:

1. "**go vet**" has not to have errors on project
2. "**goimports**" has not to have errors on project
3. "**golint**" has not to have errors on project

Distribution:

1. The solution has to be shipped as a docker container with both server and client binaries.
2. The server is a default entrypoint for the docker container.
3. There have to be a README file describe how to build and run server and client.

Building:

1. The building of the final docker container has to be done inside the special container which includes all the needed binaries for the successfull build (go, goimports, golint, ...).
2. The project has to include the Makefile with at least the following targets:
   - "**build**" - compiles the binaries for server and client and creates the ready-to-use docker container. This is a default target.
   - "**test**" - runs the Unit Tests and Integration Tests for the project. Fails if any test do not pass.
   - "**check**" - runs subsequently "go vet", "goimports", "golint" on the project. Fails if any errors occur.
   - "**run**" - starts the server with default configuration

Testing:

1. The coverage have to be more than 50%.
2. The coverage have to be measured by running "make test" command and save the output to the "coverage.out" file.
3. The "coverage.out" file have to contain joined coverage over the packages.

Functional:

1. The implementation has to be done on pure Go (only standard libraries allowed).
2. Protocol should be text-based.
3. Keys and values are utf8-encoded strings.
4. "GET", "SET" and "DEL" methods are implemented.

Commands description (example: https://redis.io/commands):

- "SET" - updates one key at a time with the given value.
- "GET" - returns tuple of the value and the key state. The state either present or absent.
- "DEL" - removes one key at a time and returns the state of the resource. The state either ignored or absent. A key should be ignored if it does not exist.

Server requirements:

1. It has to have at least the following starting options:
   - "-m, --mode" The possible storage option ("disk")
   - "-p, --port" The port for listening on (default 9090)

Client requirements:

1. The client has to be interactive (after running the cli command user has to be able to pass the commands)
2. It has to have at least the following starting options:
   - "-p, --port" The port to connect to the server (default 9090)
   - "-h, --host" The host to connect to the server (default 127.0.0.1)

Optional requirements (in addition to the minimal requirements):

Functional:

1. "KEYS" command is implemented.
2. "PUBLISH/SUBSCRIBE" commands are implemented.

Commands description (example: https://redis.io/commands):

1. "KEYS" - returns all keys matching pattern. Pattern could include '*' symbol which matches zero or more characters.
2. "SUBSCRIBE" - subscribes the client on specified channel.
3. "PUBLISH" - sends the message to the specified channel.

Server requrements:

1. Add the following starting options:
   - "-m, --mode" The possible storage option ("memory")
   - "-v, --verbose" Verbose mode, full log of the client requests.

Client requirements:

1. Add TAB completion of the commands in cli.
2. Add the following options:
   - "--dump" Dump the whole database to the JSON format on STDOUT (example: '[{"key": "YOUR_KEY", "value": "YOUR_VALUE"}]').
   - "--restore" Restore the database from the dumped file.