H2O Web Server Cheatsheet

A concise guide to the H2O web server, covering configuration, key features, and useful commands.

Core Configuration

Basic Server Setup

H2O's configuration is primarily managed through a YAML file. The default location is usually `h2o.conf` or specified via the `-c` flag.

Example:

```
listen: 8080
hosts:
  "example.com":
    paths:
      "/":
        file.dir: /var/www/example.com
```

Explanation of main configuration directives:

- `listen`: Port to listen on.
- `hosts`: Defines virtual hosts.
- `paths`: Defines URL paths within a host.
- `file.dir`: Serves files from the specified directory.

TLS Configuration

`listen:`	Specifies TLS listener with cert and key:
	`listen: 443`
	`ssl:`
	`certificate-file: /path/to/cert.pem`
	`key-file: /path/to/key.pem`

`ocsp-stapling:`	Enable OCSP stapling:
	`listen: 443`
	`ssl:`
	`ocsp-stapling: on`

`http2-early-hints:`	Enables early hints for HTTP/2 `http2-early-hints: ON`

Proxying Configuration

`proxy.reverse:`	Reverse proxy to another server:
	`paths:`
	`"/":`
	`proxy.reverse: http://backend:3000`

`proxy.preserve-host:`	Preserves the original `Host` header. `proxy.preserve-host: ON`
`proxy.timeout.keep-alive:`	Sets the keep-alive timeout. `proxy.timeout.keep-alive: 60`

Advanced Features

Caching

H2O provides flexible caching mechanisms. You can configure cache based on file extensions or specific paths.

Example:

```
paths:
  "/static":
    cache:
      max-age: 3600
      public: on
```

This caches static content for an hour.

Explanation of caching directives:

- `max-age`: Cache expiration time in seconds.
- `public`: Allows caching by intermediate caches.
- `private`: Specifies that the response is only intended for a single user and must not be cached by shared caches.

Compression

`gzip:`	Enables Gzip compression. `gzip: ON`
`brotli:`	Enables Brotli compression (if available). `brotli: ON`

`file.mime.addtypes:`	Add custom MIME types for files.
	`file.mime.addtypes:`
	`text/custom: [.custom]`

Request Handling

`fastcgi.connect:`	Forward requests to a FastCGI server:
	`paths:`
	`"/app":`
	`fastcgi.connect: 127.0.0.1:9000`

`mruby.handler:`	Execute mruby scripts to handle requests.
	`paths:`
	`"/mruby":`
	`mruby.handler: /path/to/handler.rb`

Command-Line Usage

Basic Commands

`h2o -v` - Displays H2O version.
`h2o -c <config_file>` - Specifies the configuration file.
`h2o -m <number_of_process>` - Specifies the number of worker processes.
`h2o -d` - Runs H2O in daemon mode.
`h2o -t` - Test configuration file.

Example:

```
h2o -c h2o.conf
```

Starts H2O using the `h2o.conf` configuration file.

Signal Handling

`SIGTERM`	Graceful shutdown.
`SIGUSR1`	Reopens log files.
`SIGUSR2`	Initiates a graceful restart.

Debugging

`h2o -vvv`	Enables verbose logging for debugging purposes.
`h2o-error-log`	Check the error log file for any issues.

Modules and Handlers

Built-in Modules

H2O comes with a variety of built-in modules that extend its functionality. Here's a brief overview:

- `file`: Serves static files.
- `proxy`: Acts as a reverse proxy.
- `fastcgi`: Handles FastCGI connections.
- `mruby`: Executes mruby scripts.
- `redirect`: Handles URL redirects.

Custom Handlers

mruby handlers:	Allows executing mruby scripts for custom request handling.
	`paths:`
	`  "/custom":`
	`  mruby.handler:`
	`/path/to/handler.rb`

External handlers:	Use external programs to handle requests via HTTP or other protocols.

Security

`access-control:`	Configure access control rules based on IP addresses or other criteria.
	`paths:`
	`  "/admin":`
	`  access-control:`
	`  allow: [192.168.1.0/24]`
	`  deny: [0.0.0.0/0]`

`tls-ticket-key-file:`	Configures TLS ticket key for session resumption.
	`ssl:`
	`  tls-ticket-key-file:`
	`/path/to/tls-ticket.key`