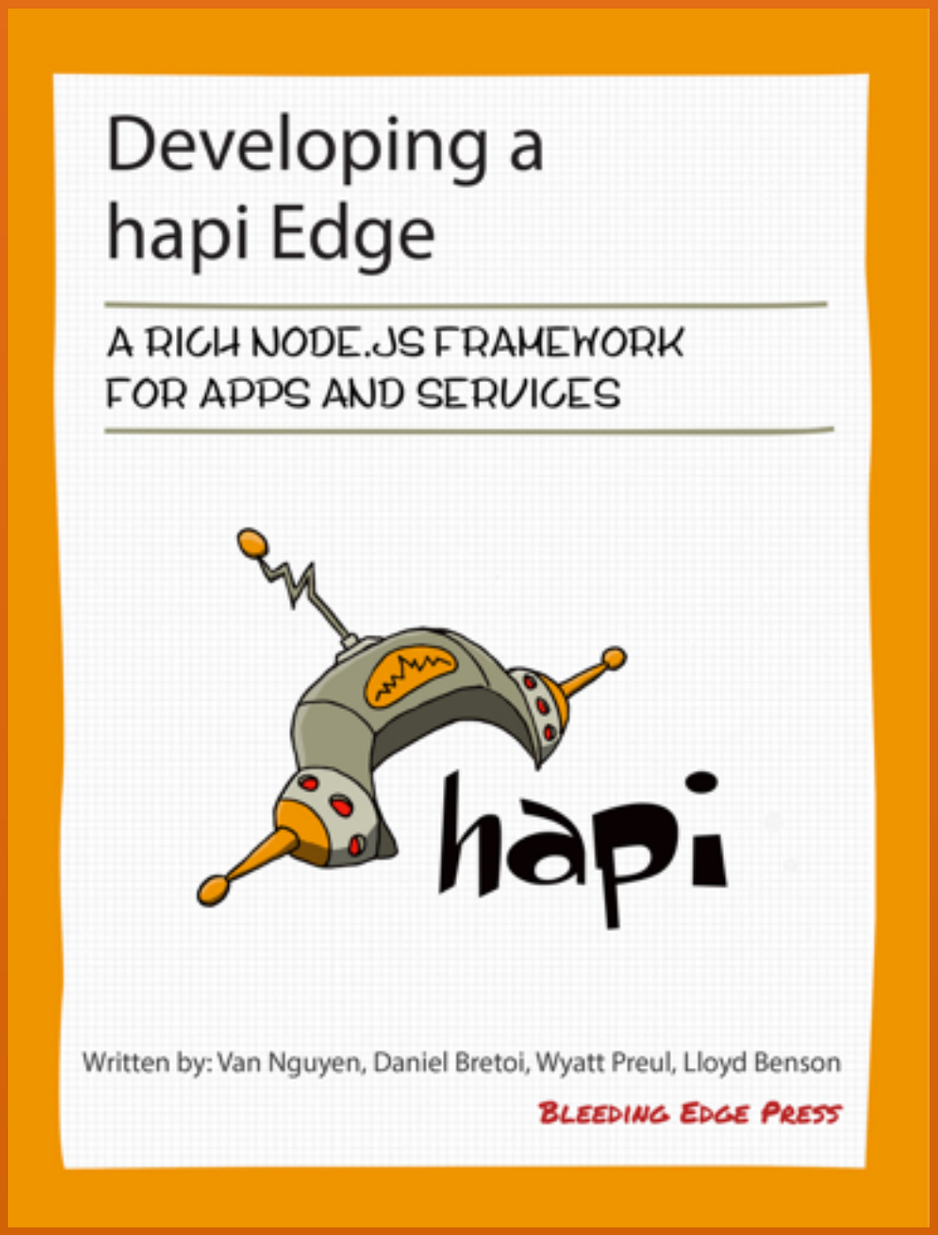
hapi workshop

npm install -g npm-clone &&
npm-clone install hapi-workshop

who we are

Wyatt Preul
- geek@github
- hapi for 3+ years

Lloyd Benson
- lloydbenson@github
- hapi for 3+ years



hapi?

- framework for building web apps/services
- solves boring stuff: caching, auth, validation, routing, CORS, load limiting, content-type processing… and more

differentiators

> designed for multi-team collaboration

> request lifecycle, not (blah, blah, next) =>

> community, composability, battle tested, fun…

production proven

&yet

clarify

beatsMUSIC

Disney

Code Winds Training HTML 5 JS

codio

GetHuman

Concrete

macy's

mozilla

npm

YAHOO!

Walmart

syllabus

1. community
2. server
3. routing
4. plugins
5. caching

   - break -

6. validation
7. testing
8. monitoring
9. deployment

code examples

- follows hapi style guide: hapijs.com/styleguide
- organized in folders by subject matter

additional resources

> hapijs.com

> hapi university - github.com/hapijs/university

> books - hapi in action & developing a hapi edge

> npm install makemehapi

> mentor program - hapijs.com/help

> gitter.im/hapijs/hapi

community

open source communities

- open source has many communities
- node already has many communities
- hapi is its own community and communities

hapi community values

> diversity

> empathy

> benevolent dictators

new contributors welcome

- always looking for new contributors
- always looking for new maintainers
- always expanding

encourage

- pull request over passivity
- test everything
- forks

miscellaneous

- BSD 3 clause license
- https://github.com/hapijs/discuss
- https://github.com/hapijs/contrib

server

server intro

- requires connection (example 1)
- server.start receives error (example 2)
- debug options (example 3)

advanced server

- many options: load, router, routes (example 4)
- multiple connections (example 5)

routing

request lifecycle

> onRequest - always called, creates reques obj

> lookup route, parse cookies

> onPreAuth

> auth, parse payload, auth payload

> onPostAuth

request lifecycle

- validate path, query, payload
- onPreHandler
- request pre's
- route handler executes
- onPostHandler
- validate response

request lifecycle

- onPreResponse
- send response
- response
- wait for tails
- tail

routing basics

- handler (example1)
- param (example2)

intermediate routing

- unlimited param (example3)
- limited param (example4)
- one or less param (example5)
- multiple routes (example6)

advanced routing

- deterministic (example7)
- put it together (example8)

server decorate

- request (example9)
- reply (example10)

```
> server
```

plugins

plugin interface

- register (server, options, next)
- attributes
- name, pkg, version
- dependencies
- multiple

user plugin

- access server interface (user-plugin)
- impact all servers registered (example1)

route plugin

- register routes (route-plugin)
- depend on user plugin
- register plugins in array (example2)

validation

overview

- object schema description language
- validator for javascript objects
- joi used useful in non-hapi projects
- built-in helpers for hapi
- response validation
- https://github.com/hapijs/joi

validation process

- 2 step process
- define schema
- then validate the schema

define schema

```
var Joi = require('joi');
var schema = {
    a: Joi.string()
};
```

validate schema

```
Joi.validate({ a: 'a string' }, schema, function (err, value) {

    if (!err) {
        console.log(JSON.stringify(value) + ' validated');
    }
});
```

notes on validating

- keys are optional by default
- strings are utf-8 encoded by default
- rules are defined in an additive fashion
- rules are evaluated in order after whitelist and blacklist checks

joi example

```
var Joi = require('joi');

var schema = Joi.object().keys({
    username: Joi.string().alphanum().min(3).max(30).required(),
    password: Joi.string().regex(/[a-zA-Z0-9]{3,30}/),
    accessToken: [Joi.string(), Joi.number()],
    birthyear: Joi.number().integer().min(1900).max(2013),
    email: Joi.string().email()
}).with('username', 'birthyear').without('password', 'accessToken');

var thing = { username: 'abc', birthyear: 1994 };
// err === null -> valid
Joi.validate(thing, schema, function (err, value) {

    if (!err) {
        console.log(JSON.stringify(value) + ' validated');
    }
});
```

simple joi follow along

```
cd validate
npm install
npm run simple
```

hapi validation (input) example

```
server.route({
    method: 'GET',
    path: '/hello',
    config: {
        handler: function (request, reply) {

            var message = '';
            if (request.query.id) {
                message = 'your id is ' + request.query.id;
            }
            if (request.query.username) {
                message = 'your username is ' + request.query.username;
            }
            return reply('hello ' + message);
        },
        validate: {
            query: Joi.object({
                id: Joi.number().min(5),
                username: Joi.string().alphanum().min(3).max(10)
            }).xor('id','username').required()
        }
    }
});

server.start(function () {

    console.log(server.info.uri + '/hello?id=4');
    console.log(server.info.uri + '/hello?id=5');
    console.log(server.info.uri + '/hello?username=5');
    console.log(server.info.uri + '/hello?username=ll');
    console.log(server.info.uri + '/hello?username=lloyd');
    console.log(server.info.uri + '/hello?username=lloyd&id=12345');
});
```

hapi validation follow along

```
cd validate
npm install
npm run hapi-validate
```

hapi response validation

```
server.route({
    method: 'GET',
    path: '/hello/{name}',
    config: {
        handler: function (request, reply) {

            return reply({ success: true });
        },
        validate: {
            // params, query, payload, headers
            params: {
                name: Joi.string().required()
            }
        },
        response: {
            // set percent rate
            sample: 0,
            schema: Joi.object().keys({
                success: Joi.boolean().required()
            })
        }
    }
});

server.start(function () {

    console.log(server.info.uri + '/hello/lloyd');
});
```

hapi response validation follow along

```
cd validate
npm install
npm run hapi-response
```

joi schema function by types

- any()
- array()
- boolean()
- date()
- func()
- number()
- object(schema)
- string()

joi helpful tips

```
// empty string
any.allow('')
// conditional
any.when('key',
  { is: 'val',
    then: Joi.required(),
    otherwise: Joi.allow('').optional()
  }
)
```

conclusion

Joi is an extremely powerful library. You can integrate it directly with hapi, but it can also be used on other non-hapi projects. It is useful for not only validating your inputs of params, query, headers, and payload but also your responses.

testing

lab overview

- lab is a command line test utility
- refactored mocha to handle most simple use cases
- https://github.com/hapijs/lab

code

- code is an assertion library
- direct rewrite of chai
- you can use chai with lab
- https://github.com/hapijs/code

whats wrong with chai?

> subset of functions (getting rid of browser complexity)

> chai is a mixture of functions and properties (easy to forget method)

> needed all functions (no missed assertions)

lab example

```
var Lab = require('lab');
var Code = require('code');

var lab = exports.lab = Lab.script();
var expect = Code.expect;
var describe = lab.describe;
var it = lab.it;

describe('simple', function () {

    it('returns true when 1 + 1 equals 2', function (done) {

        expect(1 + 1).to.equal(2);
        done();
    });
});
```

lab simple follow along

```
cd test
npm install
npm run test-simple
```

functions

```
Lab.expect(object).to.equal();
Lab.expect(object).to.not.equal();
Lab.expect(object).to.deep.equal();
Lab.expect(object).to.exist();
Lab.expect(object).to.not.exist();
```

CLI

```
-r - reporter (default console)
     console,html,junit,lcov,tap,json,clover
-m - individual test timeout in milliseconds (default 2s)
-t - minimum coverage threshold percentage (default 100%)
-g - global leak check (default)
-v - verbose
-i - individual tests (e.g. 1-2 or 1,3)
-p - run tests in parallel
-L - built-in hapijs linter
-a - assert library tallies assertions
```

package.json

```
"scripts": {
  "test": "lab -a code -r html -L -t 100 -m 10000"
},
```

linting

- hapijs projects enforce hapijs standards by default
- eslint by default
- can put in your own eslint rules
- tip: lab -d -L (just checks linting rules)

server.inject()

- uses shot module (https://github.com/hapijs/shot)
- injects itself in http layer without network stack
- no worrying about port conflicts

plugin example

```
exports.register = function (server, options, next) {

    // routes
    server.route({
        method: 'POST',
        path: '/hello',
        config: {
            handler: function (response, reply) {

                var obj = {
                    name: response.payload.name,
                    description: response.payload.description,
                    success: true
                };
                return reply(obj);
            }
        }
    });

    next();
};

exports.register.attributes = {

    pkg: require('../package.json')
};
```

prepareServer for plugin

```
internals.prepareServer = function (callback) {

    var server = new Hapi.Server();
    server.connection();
    server.register({
        register: require('..'),
        options: internals.defaults
    }, function (err) {

        expect(err).to.not.exist();
        callback(server);
    });
};
```

server.inject() example

```
describe('complex', function () {

    it('POST /hello', function (done) {

        internals.prepareServer(function (server) {

            var options = {
                method: 'POST',
                url: '/hello',
                payload: {
                    name: 'name',
                    description: 'description'
                }
            };
            server.inject(options, function (response) {

                expect(response.statusCode).to.equal(200);
                expect(response.result).to.exist();
                expect(response.result.name).to.equal('name');
                expect(response.result.description).to.equal('description');
                expect(response.result.success).to.be.true();
                done();
            });
        });
    });
});
```

lab complex follow along

```
cd test
npm install
npm run test-complex
```

lab full follow along

```
cd test
npm install
npm run test
```

multiple reporters

```
lab -a code -L -t 100 -m 10000
-r console -o stdout
-r console -o console.log
-r junit -o lab.xml
-r html -o lab.html
```

conclusion

Testing is a critical piece to writing quality software. We have shown how to leverage lab to help you with testing, code coverage, and linting. You can utilize server.inject() to bypass the network stack. Finally, you can use multiple reporters to integrate with your testing/ci frameworks.

monitoring

overview

- good is a process monitor that listens for event(s)
- maps to hapi server events
- loggers
- good-broadcast
- https://github.com/hapijs/good

good loggers

- good-console
- good-file
- good-http
- community and hapijs reporters

good event types

- response
- request (request.log)
- ops
- log (server.log)
- error
- wreck

good output types

> console - ops

```
141225/093015.900, [ops, event.tags], memory: 10Mb,
uptime (seconds): 1000,
load: [ 1.650390625, 1.6162109375, 1.65234375 ]
```

> console - error

```
141225/093015.900, [error, event.tags],
message: there was an error, stack: eventData.stack
```

good output types

> console - request

```
141225/093015.900, [request, event.tags],
data: {"message":"you made a request to a resource"}
```

> console - log

```
141225/093015.900, [log, event.tags],
data: you logged a message
```

good output types

> console - response

```
141223/164207.694, [response],
localhost: post /data {"name":"adam"} 200 (150ms) response
payload: {"foo":"bar","value":1}
```

additional good options

- opsInterval (default 15s)
- requestHeaders
- responsePayload
- requestPayload
- increases size and may impact security

good-console reporter

```
var Hapi = require('hapi');

var server = new Hapi.Server();
server.connection({ port: 8080, labels: ["api", "http"] });

server.route({
    method: 'GET',
    path: '/',
    handler: function (request, reply) {

        reply('hapi-workshop');
    }
});

var goodOptions = {
    "opsInterval": 5000,
    "reporters": [{
        "reporter": "good-console",
        "events": { "response": "*", "error": "*", "ops": "*" }
    }]
};

server.register({register: require('good'), options: goodOptions }, function(err) {

   if (err) {
       console.log('There was an err: ' + err);
   }
   server.start(function () { });
});
```

good-console follow along

```
cd monitor
npm install
npm run start-good-console
```

good-file example

```
server.route({
    method: 'GET', path: '/',
    handler: function (request, reply) {

        reply('hapi-workshop');
    }
});

var goodOptions = {
    opsInterval: 5000,
    reporters: [{
        reporter: 'good-file',
        events: { ops: '*' },
        config: './log/ops.log'
    }, {
        reporter: 'good-file',
        events: { response: '*' },
        config: './log/response.log'
    }, {
        reporter: 'good-file',
        events: { error: '*' },
        config: './log/error.log'
    }]
};

server.register({register: require('good'), options: goodOptions }, function(err) {

 // Handle error
   server.start(function () { });
});
```

good-file follow along

```
cd monitor
npm install
npm run start-good-file
```

good-http example

```
var options = {
  reporters: [{
    reporter: require('good-http'),
    events: { error: '*' },
    endpoint: 'http://prod.logs:3000',
    // events to hold before transmission
    threshold: 20,
    wreck: {
        headers: { 'x-api-key' : 12345 }
    }
  }]
};

server.register(require('good-http'), options, function (err) {

  if (!err) {
    // Plugin loaded successfully
  }
});
```

good-broadcast utility

- cli utility
- separates it out from your other process
- adds envelope to your files and sends to url

good-broadcast utility

```
var envelope = {
    schema: internals.schemaName,
    host: internals.host,
    appVer: internals.appVer,
    timestamp: Date.now(),
    events: log
};
```

good-broadcast example

```
{
    "url": "http://server/analytics",
    "log": "/log/response.log",
    "interval": 1000,
    "newOnly": true,
    "resumePath": "/log/responseIndex.tmp",
    "wait": 1000,
    "attempts": 1
}
```

```
$ broadcast -c broadcast.json
```

conclusion

You can get some great statistics about your system.
With a proper aggregate log tool like splunk or logstash, you can easily see what your app is doing.
This plugin gives you many "good" statistics for free.

deployment

overview

- rejoice
- glue
- confidence
- upstart/systemd

rejoice

- CLI utility for starting up hapi via a json config file
- replaces bin/hapi
- rejoice -c app.json
- based on composer (https://github.com/hapijs/glue)
- https://github.com/hapijs/rejoice

rejoice example

```
{
  "connections": [{
    "port": 8080,
    "labels": ["api", "http" ]
  }],
  "plugins": {
    "good": {
      "opsInterval": 5000,
      "reporters": [{
        "reporter": "good-console",
        "events": { "response": "*", "error": "*", "ops": "*" }
      }]
    }
  }
}
```

rejoice follow along

```
cd deploy
npm install
npm run start-simple
```

glue

- server composer for hapi
- code vs config style
- takes a json manifest
- more flexibility with entry points
- https://github.com/hapijs/glue

glue entry points

- compose()
- new Hapi.Server()
- preConnections()
- server.connection()
- prePlugins()
- server.register()
- callback()

glue example

```
var Glue = require('glue');
var Hapi = require('hapi');

var internals = {
    manifest: require('./simple.json')
};

Glue.compose(internals.manifest,
    { relativeTo: __dirname }, function (err, server) {

    if (err) {
        console.log('server.register err:', err);
    }
    server.start(function () {

        console.log('Server running at:', server.info.uri);
    });
});
```

glue follow along

```
cd deploy
npm install
npm run start-glue
```

confidence

- configuration document format
- foundation for A/B (not covered)
- useful when combined with rejoice

confidence example

```
"connections": [{
    "port": 8080,
    "labels": [ "api", "http" ]
}],
"plugins": {
    "$filter": "env",
    "$base": {
        "good": {
            "opsInterval": 5000,
            "requestHeaders": true
        }
    },
    "$default": {
        "good": {
            "reporters": [{
                "reporter": "good-file",
                "events": { "error": "*" },
                "config": "./log/error.log"
            }]
        }
    },
    "dev": {
        "good": {
            "requestPayload": true,
            "responsePayload": true,
            "reporters": [{
                "reporter": "good-console",
                "events": { "response": "*", "error": "*", "ops": "*" }
            }]
        }
    },
    "prod": {
        "good": {
            "reporters": [{
                "reporter": "good-file",
                "events": { "response": "*" },
                "config": "./log/response.log"
            }]
        }
```

confidence follow along

```
cd deploy
npm install
npm run gen-dev-cfg
npm run start-app
npm run gen-prod-cfg
npm run start-app
```

systemd

- toss in /etc/systemd/system
- sudo systemctl start/stop app

```
[Unit]
Description=app
After=syslog.target

[Service]
WorkingDirectory=/app/myapp
ExecStart=/path/to/node /path/to/rejoice -c /path/to/config
Restart=Always
User=appuser

[Install]
WantedBy=multi-user.target
```

upstart

- toss in /etc/init
- sudo stop/start app

```
start on runlevel [2345]
stop on runlevel [016]

respawn

script
  exec sudo -E -u username bash -c "/path/to/node /path/to/rejoice
            -c /path/to/hapi.cfg -p /path/to/modules >> /path/to/logfile 2&>1"
end script
```

conclusion

You can see how to utilize confidence and rejoice together to deploy your environment specific application.