Deltacloud API code style guidelines

General Ruby code guidelines

You may not like all rules presented here, but they work very well for me and have helped producing high quality code. Everyone is free to code however they want, write and follow their own style guides, but when you contribute to my code, please follow these rules:

Formatting

- Use ASCII (or UTF-8, if you have to).
- Use 2 space indent, no tabs.
- Use Unix-style line endings.
- Use spaces around operators, after commas, colons and semicolons, around { and before }.
- No spaces after (, [and before],).
- Use two spaces before statement modifiers (postfix: if/unless/while/until/rescue).
- Indent when as deep as case.
- Use an empty line before the return value of a method (unless it only has one line), and an empty line between defs.
- Use RDoc and its conventions for API documentation. Don't put an empty line between the comment block and the def.
- Use empty lines to break up a long method into logical paragraphs.
- Keep lines fewer than 80 characters.
- Avoid trailing whitespace.

Syntax:

- Use def with parentheses when there are arguments.
- Never use for, unless you exactly know why.
- Never use then.
- Use when x; ... for one-line cases.
- Use &&/|| for boolean expressions, and/or for control flow. (Rule of thumb: If you have to use outer parentheses, you are using the wrong operators.)
- Avoid multiline ?:, use if.
- Suppress superfluous parentheses when calling methods, but keep them when calling "functions", i.e. when you use the return value in the same line: x = Math.sin👍
  array.delete e
- Prefer {...} over do...end. Multiline {...} is fine: having different statement endings (} for blocks, end for if/while/...) makes it easier to see what ends where. But use do...end for "control flow" and "method definitions" (e.g. in Rakefiles and certain DSLs.) Avoid do...end when chaining.
- Avoid return where not required.
- Avoid line continuation () where not required.
- Using the return value of = is okay: if v = array.grep(/foo/) ...
- Use ||= freely.
- Use non-OO regexps (they won't make the code better). Freely use $=$, $0-9, $, $` and $' when needed.

Naming:

- Use snake_case for methods.
- Use CamelCase for classes and modules. (Keep acronyms like HTTP, RFC, XML uppercase.)
- Use SCREAMING_SNAKE_CASE for other constants.
- The length of an identifier determines its scope. Use one-letter variables for short block/method parameters, according to this scheme: a,b,c: any object
  d: directory names
  e: elements of an Enumerable
  ex: rescued exceptions
  f: files and file names
  i,j: indexes
  k: the key part of a hash entry
  m: methods
  o: any object
  r: return values of short methods
  s: strings
  v: any value
  v: the value part of a hash entry
  x,y,z: numbers

And in general, the first letter of the class name if all objects are of that type.

- Use _ or names prefixed with _ for unused variables.
- When using inject with short blocks, name the arguments |a, e| (mnemonic: accumulator, element)
- When defining binary operators, name the argument "other".
- Prefer map over collect, find over detect, find_all over select, size over length.

Comments:

- Comments longer than a word are capitalized and use punctuation. Use two spaces after periods.
- Avoid superfluous comments.

The rest:

- Write ruby -w safe code.
- Avoid hashes-as-optional-parameters. Does the method do too much?
- Avoid long methods.
- Avoid long parameter lists.
- Use def self.method to define singleton methods.
- Add "global" methods to Kernel (if you have to) and make them private.
- Avoid alias when alias_method will do.
- Use OptionParser for parsing complex command line options and ruby -s for trivial command line options.
- Write for 1.8, but avoid doing things you know that will break in 1.9.
- Avoid needless metaprogramming.

General:

- Code in a functional way, avoid mutation when it makes sense.
- Do not mutate arguments unless that is the purpose of the method.
- Do not mess around in core classes when writing libraries.
- Do not program defensively. (See http://www.erlang.se/doc/programming_rules.shtml#HDR11.)
- Keep the code simple.
- Don't overdesign.
- Don't underdesign.
- Avoid bugs.
- Read other style guides and apply the parts that don't dissent with this list.
- Be consistent.
- Use common sense.

(original source: https://raw.github.com/chneukirchen/styleguide/master/RUBY-STYLE)

Reviewing new drivers

- Driver name **should** not include special characters like "-" or "_"
- Each driver **should** include the valid_credentials? method
- Each driver **should** have the YAML config file stored in "config/drivers/new_driver.yaml"
- Driver **should** indicate which collections are supported by overriding the "supported_collections" method
- Each driver **must** have *state machine* defined
- Each state machine **must** define *RUNNING* and *PENDING*
- We prefer to have *new_client* method to return a client object for the back-end cloud provider
- You **should** keep all helper methods (eg. for converting outputs, states, etc..) as **private** methods
- You **must** wrap all calls to back-end cloud with "safely do"..."end" block to catch and convert all exceptions and then report them to the client in a safe way.
- You **must** catch all exceptions using "exceptions do" and assign them appropriate HTTP status code
- All back-end cloud related exceptions for errors **must** have the **502** code
- All driver related exceptions for errors **must** have the **500** code
- You **should** avoid monkey-patching of base models (like Instance, Image, etc...)
- Each driver **must** be *namespaced* as Deltacloud::Drivers::DRIVERNAME::DRIVERNAMEDriver
- Each driver method **should** call to back-end cloud just once
- Drivers **should** remain *stateless* if it's possible. Don't use YAML or other temporary files if it's not absolutely necessary
- Driver **must** include the ASF license header