Best Practices

Best Practice 1: Write IaC programs for people, not computers

Sub Practice Category	Atomic Practices							
	Independent		Ansible		Chef		Puppet	
	Name	Sources	Name	Sources	Name	Sources	Name	Sources
Make names consistent, distinctive, and meaningful			Name tasks (10) Use variables in task names (1) Omit superfluous information in task names (1) Name variables consistently (2) Prefix variables with context (8)	S48 S50 S55 S52 S51 S57 S46 S53 S54	Name elements uniformly for their system and component (1) Name attributes after the primary recipe (1) Name cookbook with context (2) Name the data bag based on the names of the cookbook use it (1) Always use uppercase for environment names (1) Prefix private recipes with underscore (1) Qualify attributes with cookbook's namespace (1)	S66 S43 S41	Name profiles according to the technology stack being modeled (2) Profiles applicable to a specific OS may be named accordingly (1) Name roles with node/machine type (1) Do not name Roles after specific technologies (1) Name modules after technologies being managed by them (1) Names in resource type definitions must have a unique variable (1) Only use numbers, lowercase letters, and underscores for variable names (1) Explicitly specify absolute namespaces for variables (1)	S38 S39 S67
Make code style and formatting consistent			Use native YAML syntax and conventions (8) Consistent quoting (2)	S46, S48 S55 S52 S54 S50	Use standard layout and content organization for each element (1) Use Ruby Style for Ruby-specific Code (3)	S41 S66 S45	Follow the standard order of information when structuring classes and defined resource types (1) Use the recommended spacing, indentation, and whitespace for manifests (1) Define Array/Hash with multiple elements on multiple lines (1) Use backslash as an escape character (1) Explicitly specify iteration over arrays or hashes (1)	S67
Make parameters, their types, and defaults explicit			Specify module defaults in tasks (1) Always mention module state (3)	S51 S54 S46			Metadata should include hard module dependencies (1) Readme file should include soft dependencies (1) Must define metadata for each module (2) Include data types of input parameters (1)	S38 S67
Use conditionals properly					Conditionals (on node attributes) with many branches (2) Do not use the name of an environment in conditionals (1)	S44 S63 S41	Avoid complex conditional expressions (1) Avoid mixing conditionals with resource declarations (1) Include defaults for case statements and selectors (1)	S67

Best Practice 2: Don't Repeat Yourself (or Others)

Sub Practice Category	Atomic Practices							
	Independent		Ansible		Chef		Puppet	
	Name	Sources	Name	Sources	Name	Sources	Name	Sources
Reuse code instead of rewriting it	Configuration Composition (1)	S28 S19	Wrapper roles (1) Use Ansible Galaxy to find and share Roles (5)	S50 S49 S55	Wrapper cookbooks (6) Create separate lightweight cookbooks for	S42 S41 S40	Use a single profile wrapper module defining all profiles (1)	S39 S38 S37

	Configuration Discovery (1) Environment Template (2)			S54 S51	sharing attributes (1) Use a base cookbook for configurations relevant for each node (1) Use providers instead of recipes for writing reusable cookbooks (1) Use a private common recipe to share resources among sub-recipes (1) Copy and use external attributes (1) Use a separate attribute file for used external attributes (1) Keep derived attributes in a separate attribute file (1) Avoid derived attributes in attribute files (1) Publish cookbooks to Supermarket with Stove (1)	S63 S44 S66	Use sub-profiles for specializations (2) Hide sub-profiles with a super profile (2) Do not define classes and defined resource types within other classes or defined resource types (1) Wrap resources to add new functionality to an existing resource (1) Use task plans to orchestrate multiple tasks into a workflow (1) Write puppet code for installing task prerequisites or dependencies (1) Minimize reimplementation of platform independent logic by reusing Puppet resources and facter (1) Use the archive module to copy a large directory structure into place on an agent (1)	
Modularize IaC programs	Modularity (5)	S22 S18 S8 S28	Decompose top-level playbooks by their roles (1) Separate infrastructure configuration from application deployment (1) Use tags only for speeding and debugging play execution (1) Use roles to group related tasks (4) Parameterized roles (2) Different inventory for different environments (4) Logically group variables (3) Group definitions of hosts based on their roles (1)	S46 S47 S49 S57 S55	Generic cookbooks (1) Role cookbooks (2) Lightweight Application cookbooks (1) Recipe per managed component (1) Do not use the default recipe (1)	S44 S40 S43 S41	Modularize Puppet code with roles and profiles (3) Parameterized profiles (2) Use role hierarchy for declaring multilayer profiles (1) Prefer role-per-node (1) Roles shall not contain resources (1) Parameterized tasks (1) Split modules into public and private classes and defined types to separate configurable behaviors from protected behaviors (1) Keep component modules generic and cohesive (2) Use a single class for basic modules (1) Use install config-service pattern for complex modules (1) Use separate files for all classes and resource type definitions (1) Parameterized classes (1)	S38 S37 S39 S20 S67
Select the right modules for the job and use it correctly			Use task-specific modules instead of general modules (6) Use template or copy modules instead of lineinfile or blockinfile modules (3)	S49 S48 S55 S51			Use recursive file permissions resource type for managing permissions, owner, and/or group (1) Use recursive file resource type (recurse =>true) only for managing a small number of files (1)	S38 S67
Reuse the tools that the community use					Use Berkshelf to manage dependencies (2) Use Knife to create data bags (1)	S44 S41 S42	Use puppet development kit (4) Use a supported Puppet architecture, including a Puppet master (1) Use Librarian-puppet tool to manage modules (2)	S38 S38 S62 S61

Best Practice 3: Let the IaC tools do the work

Sub Practice Category	**Atomic Practices**			
	Independent	**Ansible**	**Chef**	**Puppet**

	Name	Sources	Name	Sources	Name	Sources	Name	Sources
Codify everything	Codify Everything (4)	S22 S09 S08 S14						
Package applications for deployment	Package application for deployment (1)	S28						
Do not violate immutability and reproducibility of your infrastructure	Immutable Infrastructure (5) Reproducible Image (1)	S21 S31 S22 S10 S28						
Do not violate idempotence of IaC programs			Don't skipping tasks (1) Do not use non-idempotent modules (1)	S57 S49	Declarative recipes (2) Use guards instead of if statements for delaying resource action execution to runtime (1)	S63 S41	Imperative logic in classes (1) imperative tasks managing ad-hoc configurations (1)	S38

Best Practice 4: Make incremental changes

Sub Practice Category	Atomic Practices							
	Independent		Ansible		Chef		Puppet	
	Name	Sources	Name	Sources	Name	Sources	Name	Sources
Use a version control system	Versioning Everything (16) Incremental Configuration (1)	S14 S29 S22 S4 S6 S7 S13 S61 S19 S23 S26 S27 S28 S32 S33 S8 S28	Use semantic versioning (1)	S46	Semantic versioning (1), Prefer optimistic over pessimistic version constraints (1) Never decrement the version of a cookbook (1)	S41	Use semantic versioning (1)	S67
Favor versionable functionalities					Avoid setting attributes in roles (1) Do not use roles for setting a run list (recipes) of node (2)	S42 S44 S40		

Best Practice 5: Prevent avoidable mistakes

Sub Practice Category	Atomic Practices							
	Independent		Ansible		Chef		Puppet	
	Name	Sources	Name	Sources	Name	Sources	Name	Sources
Use the correct quoting style			Minimal Consistent quoting (2)	S50 S57	Use single-quoted strings for values not being interpolated (1) Use double quoted strings for variables used in whitespace arrays (1)	S66 S41	Consistent and appropriate quoting of string values (1) All resource names or titles must be quoted (1)	S67
Avoid unexpected behaviors whenever possible			Handle errors with Built-ins handlers (1) Verify service state (1)	S57 S54	Always define a :remove action to clean up resources created by the provider (1) Always specify resource action in each item of data-bag collection (1)	S41	Use one input method for receiving input data (1) Use task metadata to validate inputs and control its execution (1)	S67
Use proper values			Use a valid 4-digit octal value or symbol for a File Mode parameter (1)	S46	Do not use hyphen in cookbook and custom resource names (1) Never use hyphens in the names of cookbooks with resource providers (1) Specify the file mode as a quoted 3- 5 character	S66	Specify File mode always as a 4-digits octal value (numeric notation) or a string (symbolic notation) (2)	S67 S38

					string of its valid octal mode (1) Do not use static Unix Style paths (1)			

Best Practice 6: Plan for unavoidable mistakes

Sub Practice Category	Atomic Practices							
	Independent		Ansible		Chef		Puppet	
	Name	Sources	Name	Sources	Name	Sources	Name	Sources
Write tests as you code	Testing (12)	S28 S7 S9 S22 S6 S61 S27 S17	Test Roles with an emulated environment (3)	S50 S54 S57	Always test against the latest version of Chef (1) Create unit and integration tests for recipes and resources with ChefSpec and InSpec (1)	S41	All classes, defined types, and custom extensions should have associated testing (1) Use dry run mode (2)	S38 S61 S62
Do not ignore errors			Do not ignore failed Tasks (1)	S57				
Use off-the-shelf testing libraries					Test cookbooks with Test Kitchen (1) Create unit and integration tests for recipes and resources with ChefSpec and InSpec (1)	S41		
Monitors your environment	Infrastructure Query Language (1) Metrics as Code (1)	S28	Use tags debugging play execution (1)	S47			Use the puppet metrics collector module to collect metrics (1)	S38

Best Practice 7: Document little but well

Sub Practice Category	Atomic Practices							
	Independent		Ansible		Chef		Puppet	
	Name	Sources	Name	Sources	Name	Sources	Name	Sources
Code as documentation	Code as Documentation (7)	S22 S7 S20 S28 S9						
Use document templates			Use role documentation templates (2)	S50 S51			Use hash comments (1) Use Documenting modules with Puppet Strings (1)	S67

Best Practice 8: Organize repositories well

Sub Practice Category	Atomic Practices							
	Independent		Ansible		Chef		Puppet	
	Name	Sources	Name	Sources	Name	Sources	Name	Sources
Modularize repositories	Decentralized Repository (1)	S32			Git repository per each cookbook (1)	S41	Use separate repositories for complex component modules (1) Use a separate profile repository to delegate profile management to application development teams (1) Put the Hiera data at the module level in a separate repository (1)	S38

Use standard folder structures			Use the standard Ansible project structure (3) Use the standard role directory structure (2)	S46 S51 S57 S54	Use the standard Chef directory structure (1) Use the standard Chef cookbook folder structure (1)	S41	Use standard Puppet repository layout and content (2) Use standard Puppet module layout (1)	S38 S67

Best Practice 9: Separate configuration data from code

Sub Practice Category	Atomic Practices							
	Independent		Ansible		Chef		Puppet	
	Name	Sources	Name	Sources	Name	Sources	Name	Sources
Use configuration datasource	Configuration Data Source (1)	S28			Use a specific recipe to refer to information from a data bag (1) Minimize data bag lookups (1)	S41	Use Hiera to separate configuration data from code (3) Use Hiera data to set parameter defaults of classes (1) Use Hiera lookups for separate private data from code in classes (1) Use Hiera to assign classes to nodes (1) Avoid using calls to Hiera functions in public modules (1) Do not do Hiera lookups in component modules (1) Use Hiera lookups and business logic within a role only for dynamic (1)	S67 S38 S58
Modularize configuration data			Different inventory for different environments (4) Group definitions of hosts based on their roles (1)	S46 S57 S55			Use Hiera hierarchy only for addressing operating system differences (2)	S38 S37
Select data sources wisely					Do not use data bags to store environment specific data (2) Use data bags instead of attributes for storing global data (2) Avoid using data bags in public cookbooks (1) Prefer environment files over databags (1) Use environment files for cookbook versions and Run lists (1) Use only one entry in the run list for each node (1)	S41 S43		
Use configuration templates	Configuration template (1)	S28	Configuration file template (3)	S49 S55	Avoid using the node object within configuration templates (1) Never refer to attributes from multiple cookbooks in a configuration template (1)	S41	Use configuration templates (2) Use template reading functions that can validate the templates being read (1)	S51 S38 S67

Best Practice 10: Write secure code

Sub Practice Category	Atomic Practices							
	Independent		Ansible		Chef		Puppet	
	Name	Sources	Name	Sources	Name	Sources	Name	Sources
Isolate secrets from code	Secret Isolation (3)	S28	Vaults for Storing Secrets (3)	S55 S53 S51	Prefer secret isolation with a Vault (3)	S63 S41	Separate secrets from code with Hiera (2)	S61 S62
Protect your data at rest	Encrypted Secret (1)	S28			Use data bags for data to be encrypted (1) Use service discovery tools instead of hard-coded IP addresses (1)	S41		
Use facts from trusted sources							Use facts from the trusted facts array ($trusted) (1)	S67 S38

							Refer to facts using the $facts hash (2)	
Use standard secure coding practices			Use sudo only where necessary (1) Secure logging (1)	S57			Explicitly state if task parameters are sensitive (1), Use the secure coding practices supported by the implementation language of tasks (1)	S67

Bad Practices

Bad Practice 1: Violation of IaC principles

Sub Practice Category	Atomic Practices							
	Independent		Ansible		Chef		Puppet	
	Name	Sources	Name	Sources	Name	Sources	Name	Sources
Not letting IaC do its work	Not treating IaC as Code (1) Fancy configuration file copying (1) Automation over documentation (1) Data as code (1)	S28	Overuse of comments (1) Using general modules shell, command, raw, and script instead task specific modules (2)	S65 S54			Configuration data in code (1)	S34
Violating Idempotence			Non Idempotent roles (2) Using imperative modules such as shell and commands (2)	S65 S54			Imperative logic in classes (1) Imperative tasks managing ad-hoc configurations (1)	S38
Violating immutability and reproducibility	Non-reproducible image (1) Non reproducible environments (2)	S28 S31 S6						

Bad Practice 2: Not writing IaC programs for people

Sub Practice Category	Atomic Practices							
	Independent		Ansible		Chef		Puppet	
	Name	Sources	Name	Sources	Name	Sources	Name	Sources
Violation of naming and styling conventions	Ignoring styling guidelines (1)	S28	Mixing YAML syntax styles (1) Superfluous information in task names (1)	S65 S51			Roles named after specific technologies (1) Profiles named after generic server types (1)	S38
Favor complexity			Restart services without using a handler (1) Chaining handlers (1)	S54	Complex branching logic (1) Numerous resource notifications (1)	S63		
Insufficient modularity			Single one-size-fits-all playbook (2) Including business logic in the playbooks (3) Multiple responsibilities roles (3) Per-host fine grained variables in inventories (1)	S47 S50, S48, S47 S65	God recipes (2) Conditionals (on node attributes) with many branches (2) Customize all the things using attributes (1)	S63 S43	Everything in general Manifests (2) Profiles containing resources from component modules (1)	S34 S38

Bad Practice 3: Improper project organization

Sub Practice Category	Atomic Practices							
	Independent		Ansible		Chef		Puppet	
	Name	Sources	Name	Sources	Name	Sources	Name	Sources
Improper Repository							Everything in separate repositories (2) Monolithic modules repository (2) Using data-in-modules	S38 S36 S30 S34

							in the control repository (1)	
Improper Version Control	Private fork of a community module (1) Version control by copy and paste (1)	S28 S18			Changing a community cookbook (2) Setting a server's run list in a role (1) Using role attributes (1)	S40 S63 S42	Roles containing resources (1) Using multiple Puppet environments for a single infrastructure (1)	S38 S34

Bad Practice 4: Insecure configure data and coding practices

Sub Practice Category	Atomic Practices							
	Independent		Ansible		Chef		Puppet	
	Name	Sources	Name	Sources	Name	Sources	Name	Sources
Hardcoding Information	Postponing secret isolation (1)	S28	Hardcoding variables (1)	S57			Hardcoding variables (1)	S58
Not using built-in security tools and mechanisms correctly					Storing secrets as Encrypted data bags (1)	S63	Using facts from the unsecure facts array (1) Developing an autosign system from scratch (1)	S38