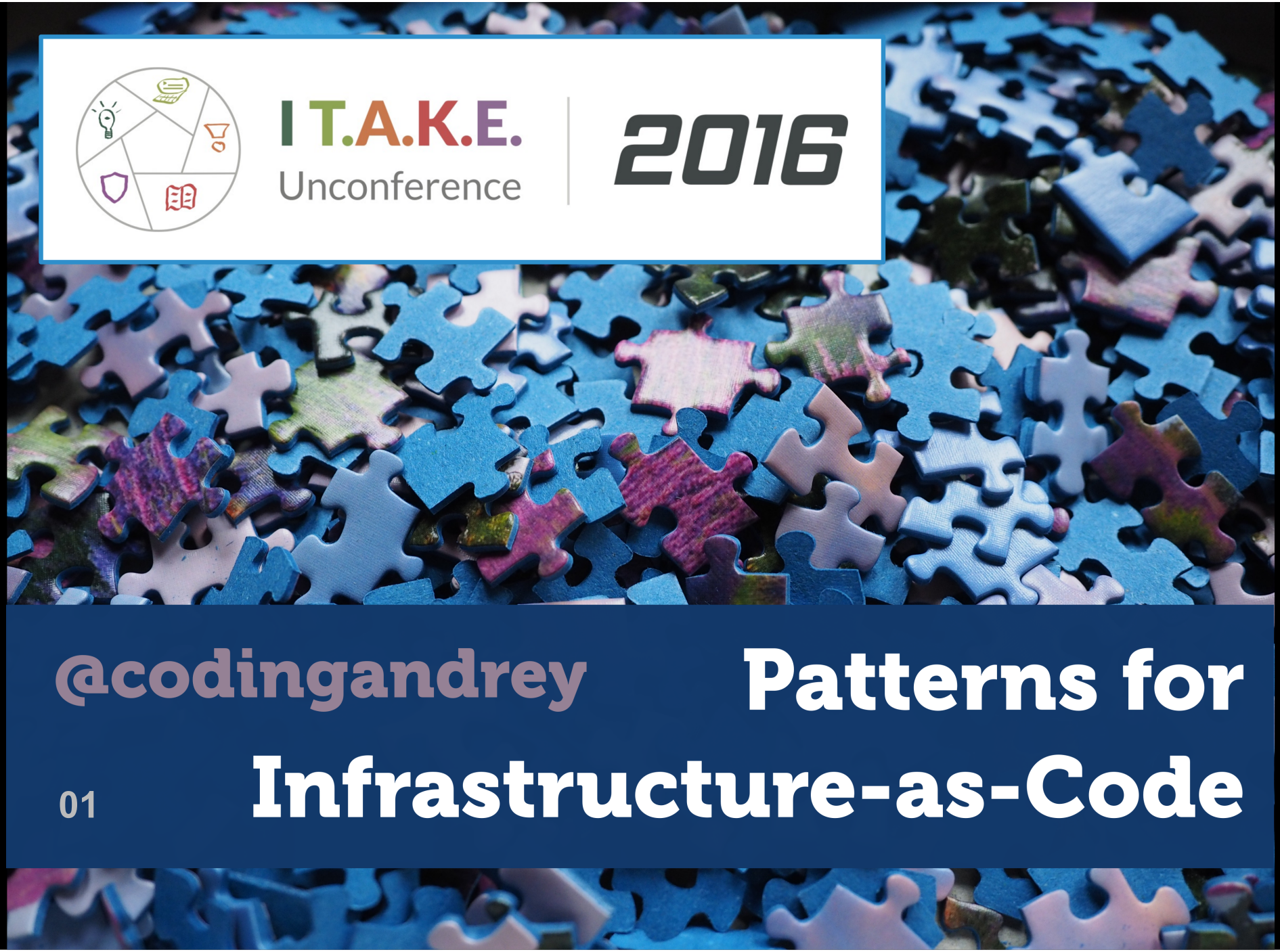

I T.A.K.E.
Unconference
2016
@codingandrey
Patterns for
Infrastructure-as-Code
01

About me

Andrey Adamovich

- Java/Groovy developer, clean coder
- DevOps guy, automation junkie
- Co-organizer of @latcraft and @devternity
- Coach at @devchampions
- Twitter: @codingandrey

What this talk is about?

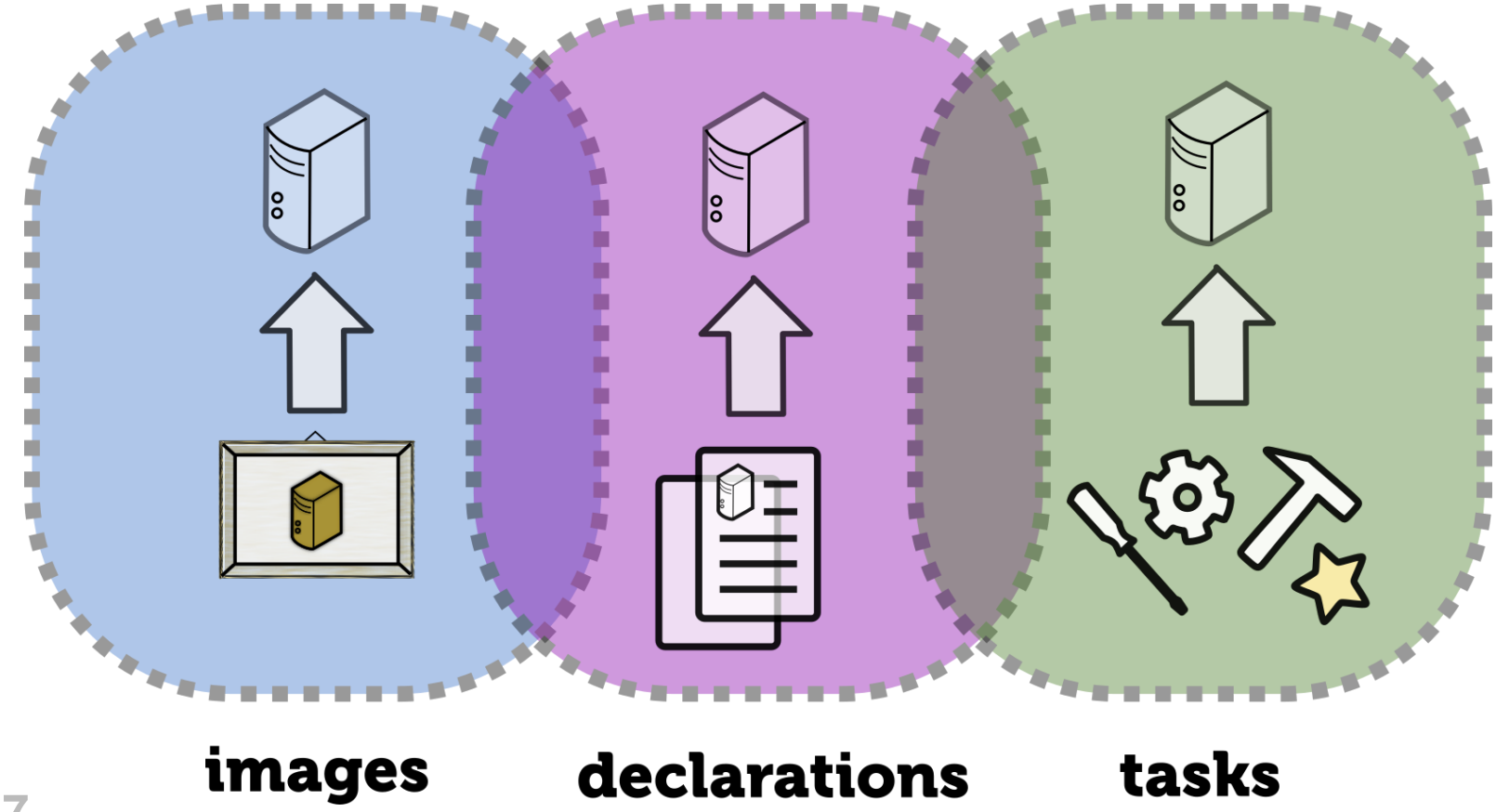
Well...

- Collection of patterns (and anti-patterns) for representing your infrastructure-as-code.
- Work in progress (never done).
- Feedback is more than welcome!

Infrastructure-as-code

Infrastructure-as-Code *(IaC) is a type of IT infrastructure that operations teams can automatically manage and provision through code, rather than using a manual process. Infrastructure-as-Code is sometimes referred to as programmable infrastructure.*

Images, declarations, tasks



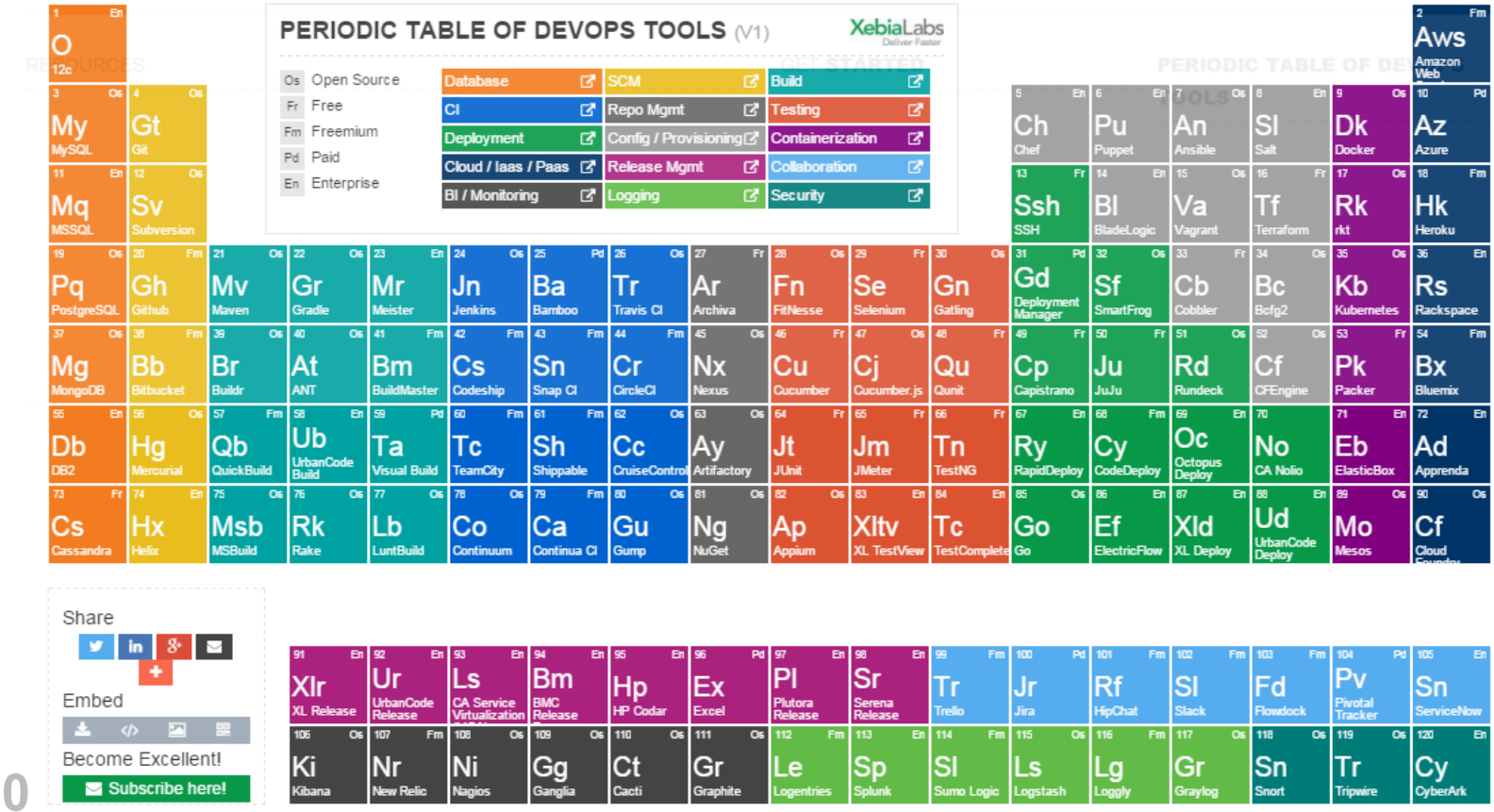
IaC players

- Image defitions
- Provisioning declarations
- Automation scripts

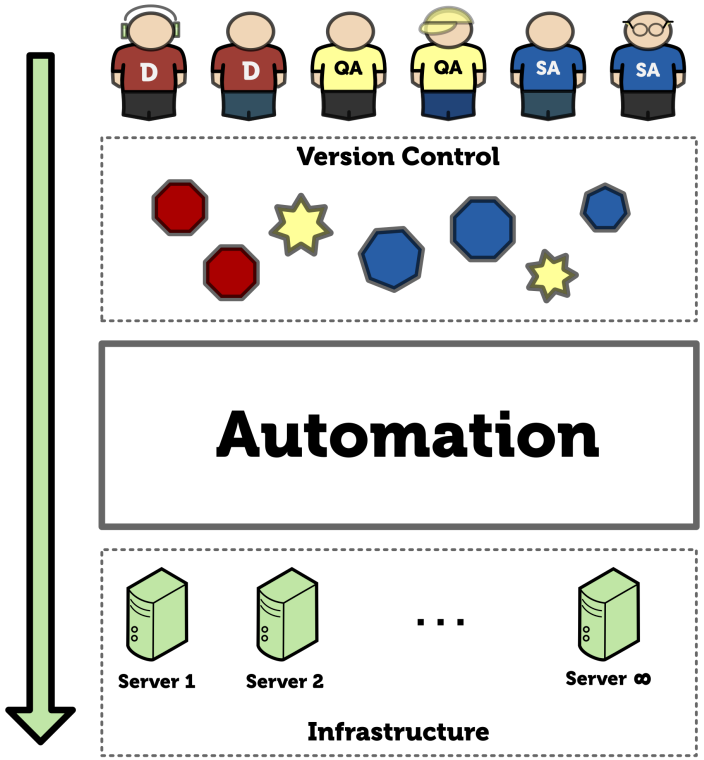
Tools

- Image creation/management: Packer, Docker, AWS AMI/EC2 etc.
- State declarations: Puppet, Chef, Ansible etc.
- Automation blocks: SSH, API + `<your favorite scripting language>`

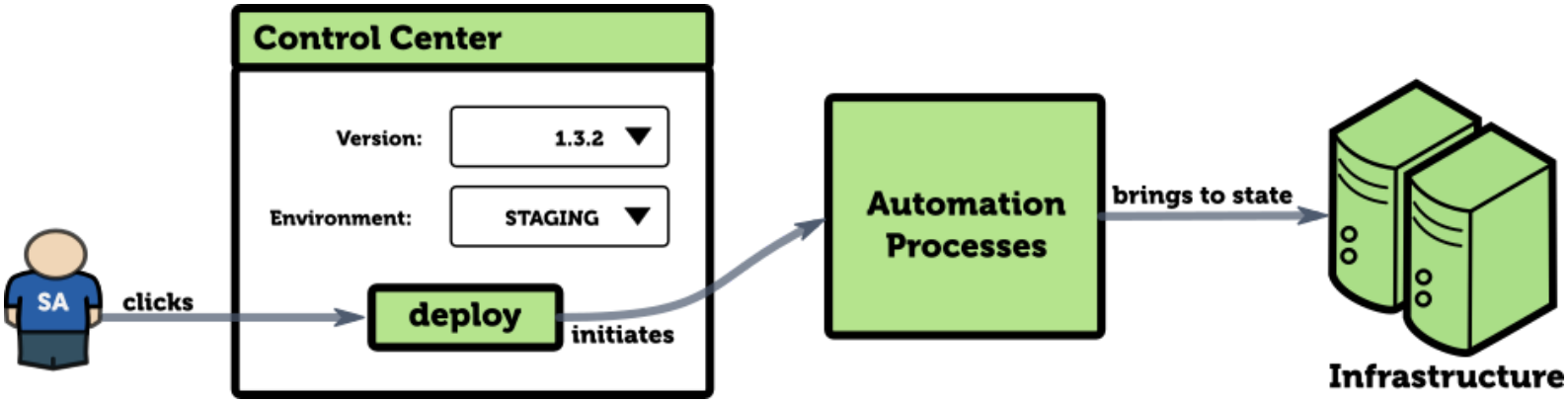
Periodic table of DevOps



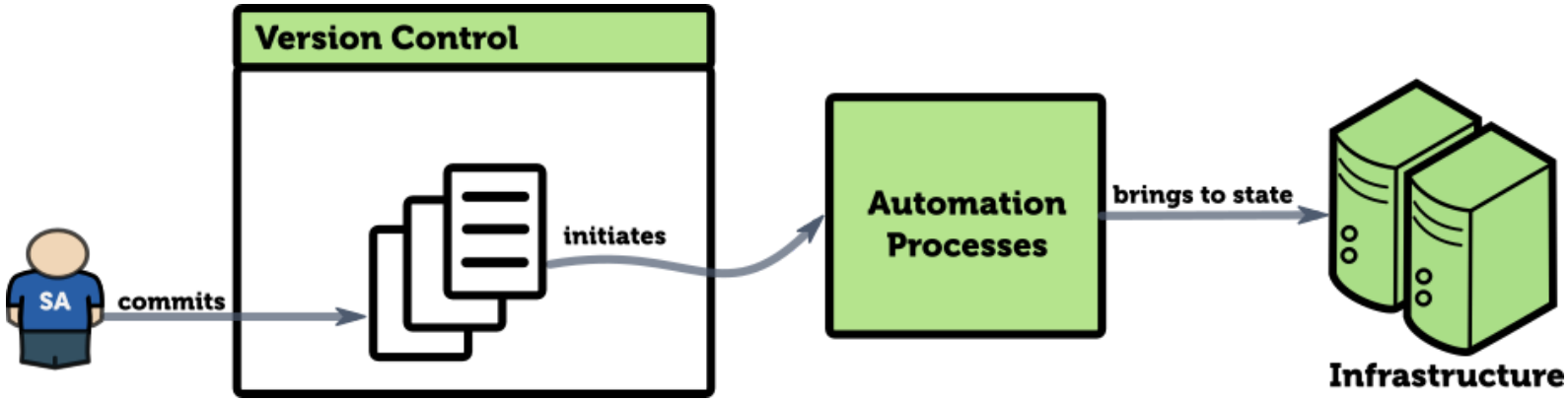
Everything is code!



Code "deployment" (one click away)



Code "deployment" (one commit away)



Let's see some patterns!


Images

Anti-pattern: Golden Image

- Manually crafted base infrastructure server image that nobody dares or knows how to change.

Pattern: Reproducible Images

- Operating system distributions (`*.iso`).
- Base provider images.
- Packer can create images for many virtualization software and cloud providers.
- Docker can build and package containers as images for distribution.


Secrets

Pattern: Secret Isolating

- Everything is code, but secrets are not!
- Secrets should reside in a separate location!
- Secrets should be injected on the very last stage of "deploying" your code.
- In this way, the actual code still remains sharable.

Pattern: Encrypted Secrets

- Shared secrets must be encrypted!
- Well, all stored secrets must be encrypted!
- Decryption password is shared through a different channel.

Encryption options

- Encrypt hard drives
- Encrypt files in version control
- Use vault service

Encryption options: GPG

- GPG/PGP:

```
01. > cd <path-to-your-repo>/
02. > gpg --encrypt sensitive_file
03. > git add sensitive_file
04. > git commit -m 'Add encrypted version of a sensitive file'
```

Encryption options: Transcrypt

- OpenSSL + Transcrypt (https://github.com/elasticdog/transcrypt):

```
> cd <path-to-your-repo>/
> transcrypt
> echo 'sensitive_file  filter=crypt diff=crypt' >> .gitattributes
> git add .gitattributes sensitive_file
> git commit -m 'Add encrypted version of a sensitive file'
```

Vault services

- Generic: Vault from HashiCorp
- Chef: encrypted data bags
- Puppet: hiera-gpg, hiera-eyaml
- Ansible: ansible-vault

Anti-pattern: Postponing Secret Isolation

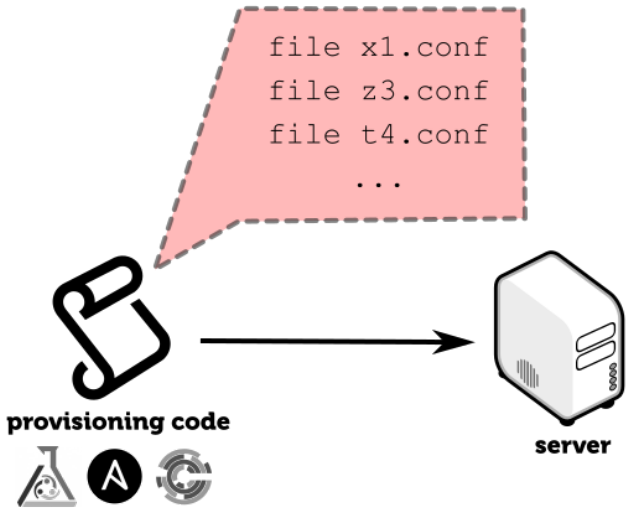
- "It's OK for now" does not really work!
- It creates a culture of security being not so important!
- It may alienate your Dev and Ops teams, because they can't share code due to hard-coded secrets!

Code organization

Anti-pattern: "Fancy-File-Copying"

- To configure package X, you keep all configuration files it needs within your "code".
- You use provisioning tool abstractions to copy every single file onto the target system.

Anti-pattern: "Fancy-File-Copying"



Example: nginx

- nginx.conf
- mime.conf
- servers.conf
- params.conf
- nginx.pp | nginx.rb | nginx.yml

Example: nginx

```
01. file { 'servers.conf': ... }
02. file { 'mime.conf': ... }
03. file { 'nginx.conf': ... }
04. file { 'params.conf': ... }
05. ...
```

Example: nginx

```
01. - template: src=servers.conf ...
02. - template: src=mime.conf ...
03. - template: src=nginx.conf ...
04. - template: src=params.conf ...
05. ...
```

Hiding abstractions

- Well, there are much simpler ways to copy files.
- You actually hide your intent and the goal of your configuration.
- *File* and *package* are not always the right abstractions.

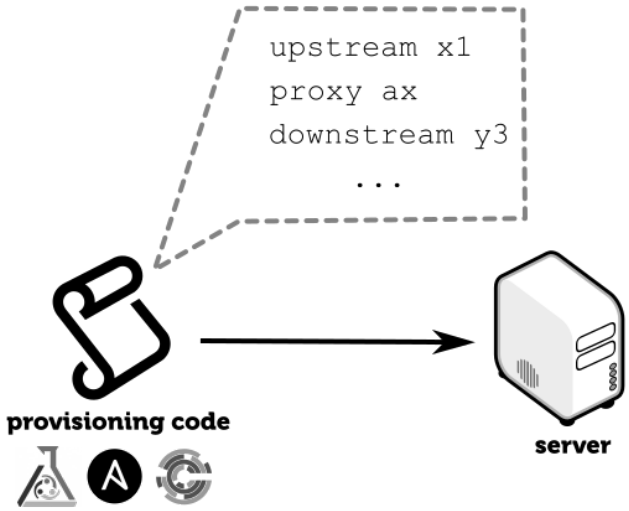
Example: nginx

- Upstream Server
- Virtual Host
- Static Directory
- System Wide Setting

Pattern: Infrastructure Component DSL

- Nobody knows your domain better than you!
- You can write your own DSL or you can leverage existing tools.
- The main thing is to group infrastructure configuration into reusable components.
- That's what we do with application code, that's what we should do with IaC!

Pattern: Infrastructure Component DSL



Example: pseudo-code

```
system1 {
  http_proxy {
    cache=true
    business_app {
      param1=A
    }
  }
  database {
    memory=3GB
  }
```

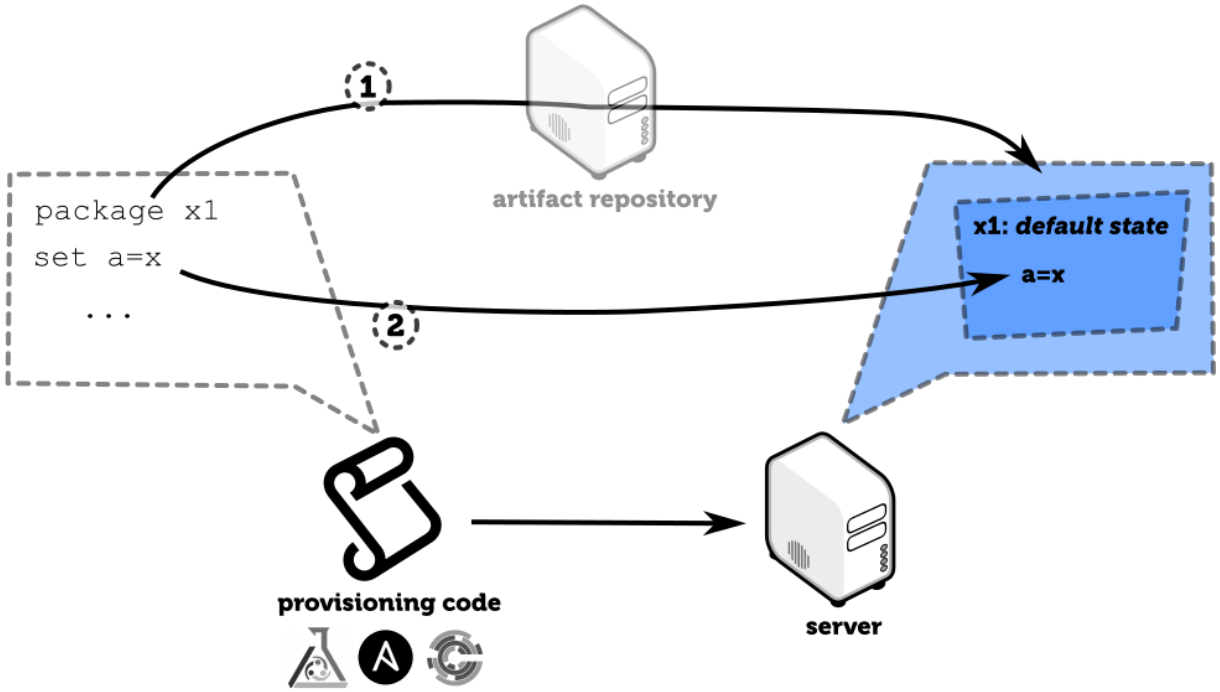
DSL

- Bash, Perl, Python, Groovy,... anything works.
- Though, Puppet, Chef, Ansible provide facilities to define and group abstractions.

Pattern: Incremental Configuration

- Many packages will already be on the system in their default state.
- Instead of duplicating default state in your code, you can only define an incremental change.

Pattern: Incremental Configuration



Examples

- Disallow root access on the system
- Set SELinux into permissive mode
- Set default caching timeout in Nginx

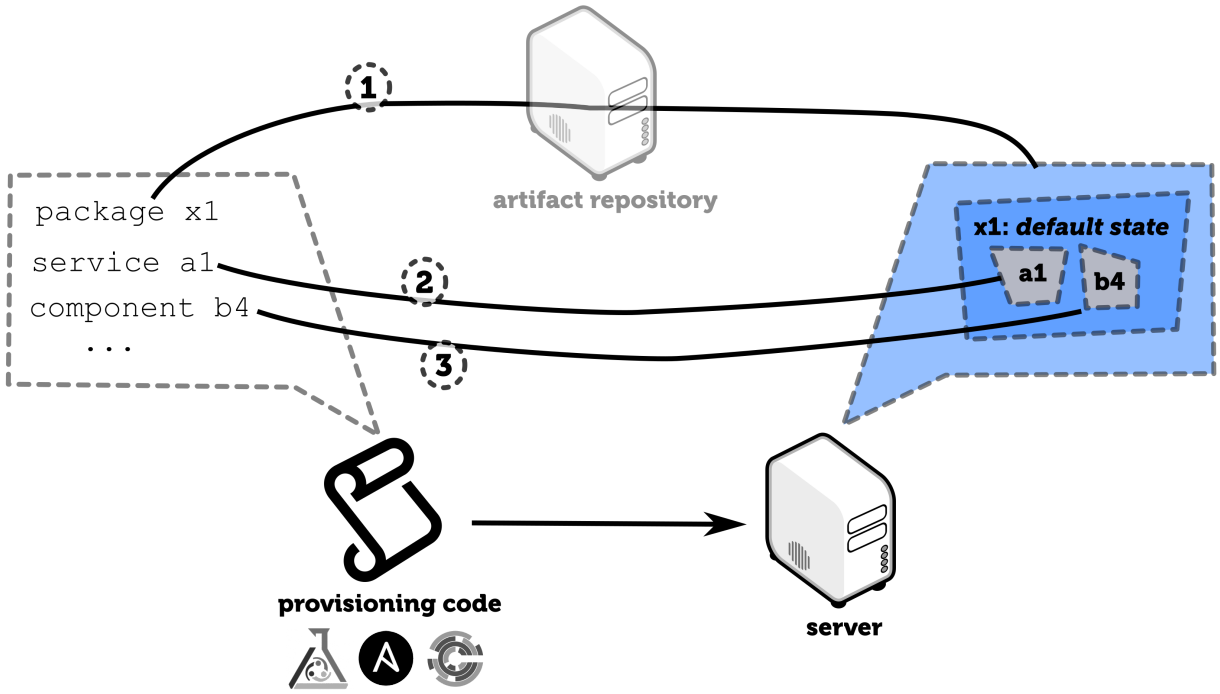
Tooling examples

- Generic: sed, perl, regular expressions
- Puppet: file_line, augeas
- Ansible: lineinfile, replace
- Chef: ruby

Pattern: Configuration Composition

- Compose your configuration of several template calls or API call blocks.
- Expose abstractions through configuration blocks.

Pattern: Configuration Composition



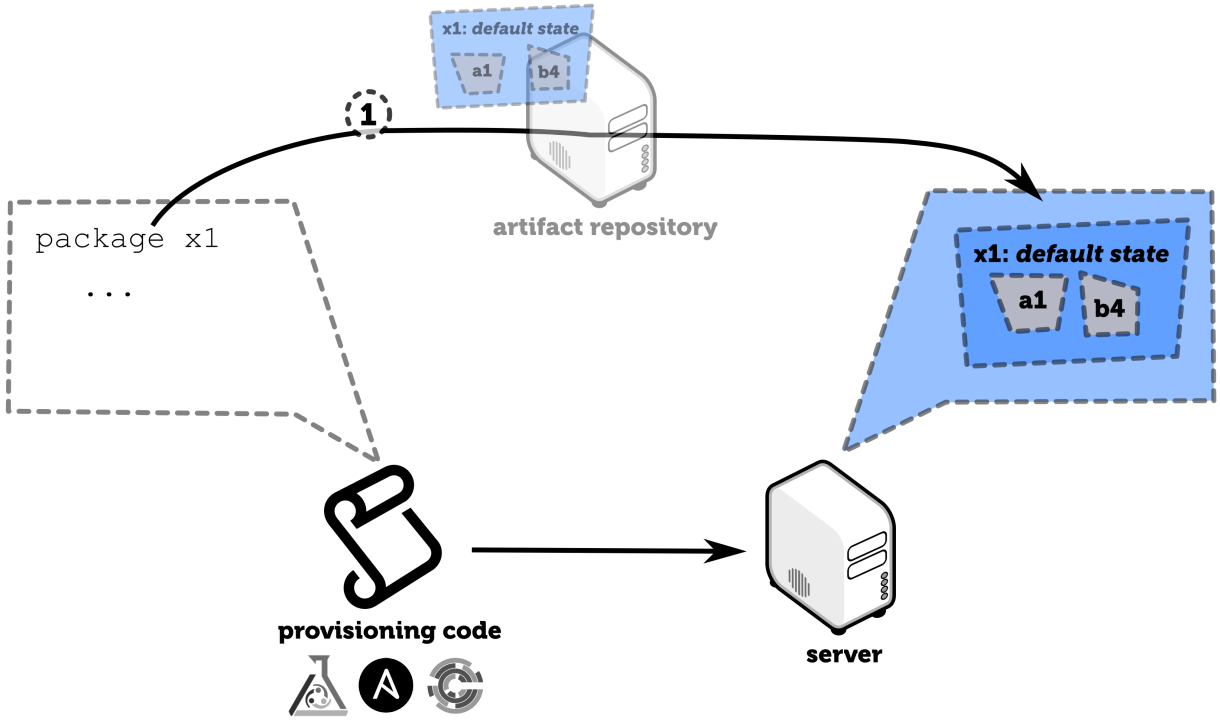
Tooling examples

- Puppet: concat module
- Ansible: assemble module
- Chef: partials

Pattern: Extra-Packaging Code

- Package your application in the most approriate format that is ready for the most hassle-free deployment.
- Publish it to artifact repository (Maven, RubyGems, Yum, Apt...).
- Artifact repository serves as a layer of isolation between pipelines.
- Reduces amount of code needed on later stages of configuration management.

Pattern: Extra-Packaging Code



Application can be packaged differently

1. jar|gem|pyc|...
2. tar.gz|tar.bz2|zip|...
3. rpm|deb|msi|...
4. server|container image

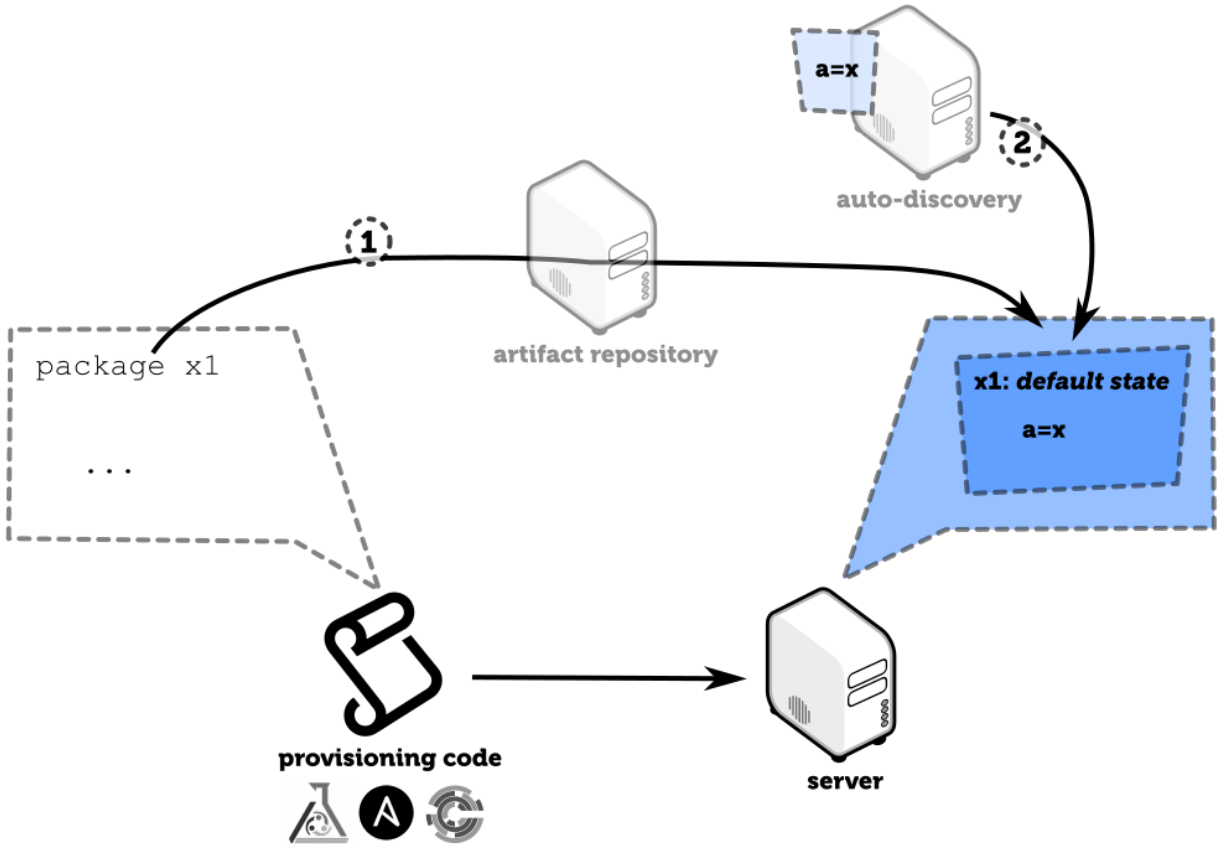
Anti-pattern: Data as Code

- Data has different lifecycle. It's more dynamic.
- Data changes more often than code.
- Example 1: use your provisioning tool to define organization users.
- Example 2: manifest that lists all your 500 servers.

Pattern: Configuration Discovery

- Part or all of system configuration is distributed through auto-discovery mechanism.
- This cleans your IaC from storing specifics. Define keys instead of values.
- Basically, "convention over configuration" for your cluster.

Pattern: Configuration Discovery



Tooling examples

- Etcd (https://github.com/coreos/etcd)
- Eureka (https://github.com/Netflix/eureka)
- ZooKeeper (https://zookeeper.apache.org/)
- Consul (http://www.consul.io/)

Pattern: Configuration Data Source

- Useful when number of managed items exceeds certain amount.
- Data file (Text, Excel, etc.)
- Database (PuppetDB etc.)
- Infrastructure Service API

Pattern: Infrastructure Query

- Language or API that allows to query your infrastructure state (real time or last available report).
- Examples: AWS EC2 API, PuppetDB, MCollective, Salt

Pattern: Environment Template

- Define template from which you can create a fully working environment.
- It gives scaling.
- It gives isolation.
- It gives flexibilty.

Tooling examples

- Vagrant
- AWS Cloud Formation
- Terraform
- Docker and Docker Compose
- Kubernetes API


Code Quality

Anti-pattern: Not Treating IaC as Code

- Code must be in **Version Control**.
- Lack of experience with new tool may require **Code Reviews**.
- Yes, there are tools for **Static Code Analysis** even for IaC products.
- Unit testing does not make a lot of sense for IaC, but **Integration Testing** does.
- Applying all the above techniques gives the best QA result for any code.

Testing IaC

- Serverspec (http://serverspec.org/)
- BATS (https://github.com/sstephenson/bats)

Anti-pattern: Ignoring Styling Guidelines

- Each tool/language out there has one.
- Nobody canceled clean code teachings.
- Reading, writing and eventually merging code is always easier if people follow the same formatting and styling.

Static code analysis tools

- shellcheck (https://github.com/koalaman/shellcheck)
- yaml-lint (https://github.com/Pryz/yaml-lint)
- Puppet Lint (http://puppet-lint.com/)
- Ansible Lint (https://github.com/willthames/ansible-lint)
- FoodCritic (http://www.foodcritic.io/)


Side effects

Pattern: Metrics as Code

- Metrics that your application provides evolve with your application.
- New components, new endpoints, new KPIs...
- Keep monitoring configuration close to the code!
- Or make it auto-discoverable and visible!

Configuring and collecting metrics

- Monitoring software has configuration files and/or an API that can be programmed.
- There a plenty of libraries that allow making monitoring a built-in feature of your application.

Examples: Java

- DropWizard Metrics
- Hystrix
- StageMonitor

Pattern: Control Panel as Code

- Repeatable things live well in scripts.
- Scripts can (and will) be well executed by your CI server (or any other UI you can build around your automation).
- Effectively, that server becomes your "control panel".
- Keep configuration of your "control panel" in version control.

Example: Jenkins

- Jenkins API (https://jenkinsapi.readthedocs.org/en/latest/)
- Job DSL (https://github.com/jenkinsci/job-dsl-plugin)
- Gradle plugin (https://github.com/ghale/gradle-jenkins-plugin)

Example: RunDeck

- RunDeck API (http://rundeck.org/2.5.3/api/index.html)
- RunDeck Command Line (http://rundeck.org/docs/man1/index.html)

Anti-pattern: Private Fork of a Community Module

- There is a lot of code out there.
- Private fork may work as a short-term solution.
- Do not keep your updates only to yourself. Share them back.

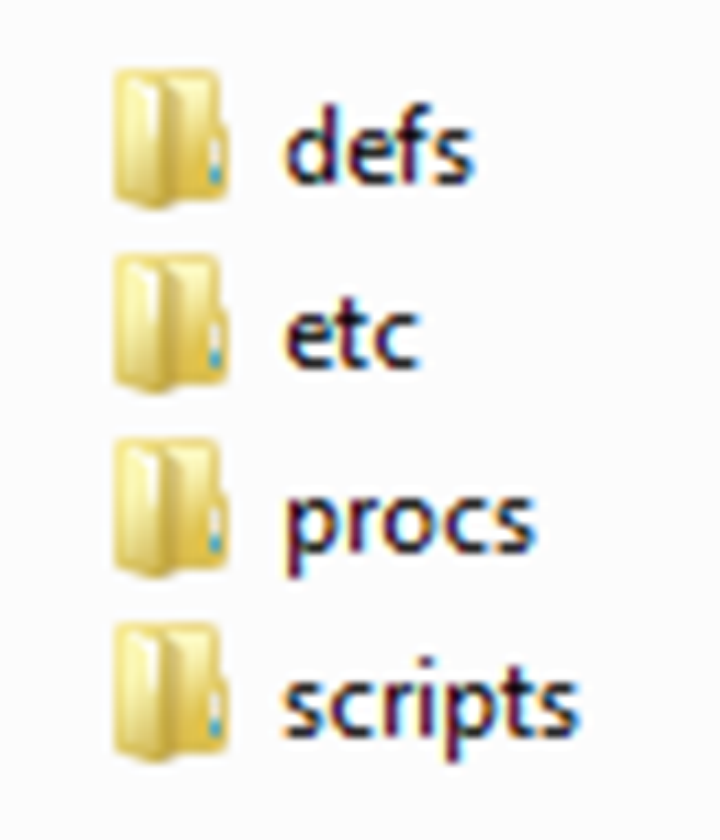
Pattern: Community Module Wrapper

- It's better to create a wrapper.
- This simplifies upgrades.
- And tracebilty.

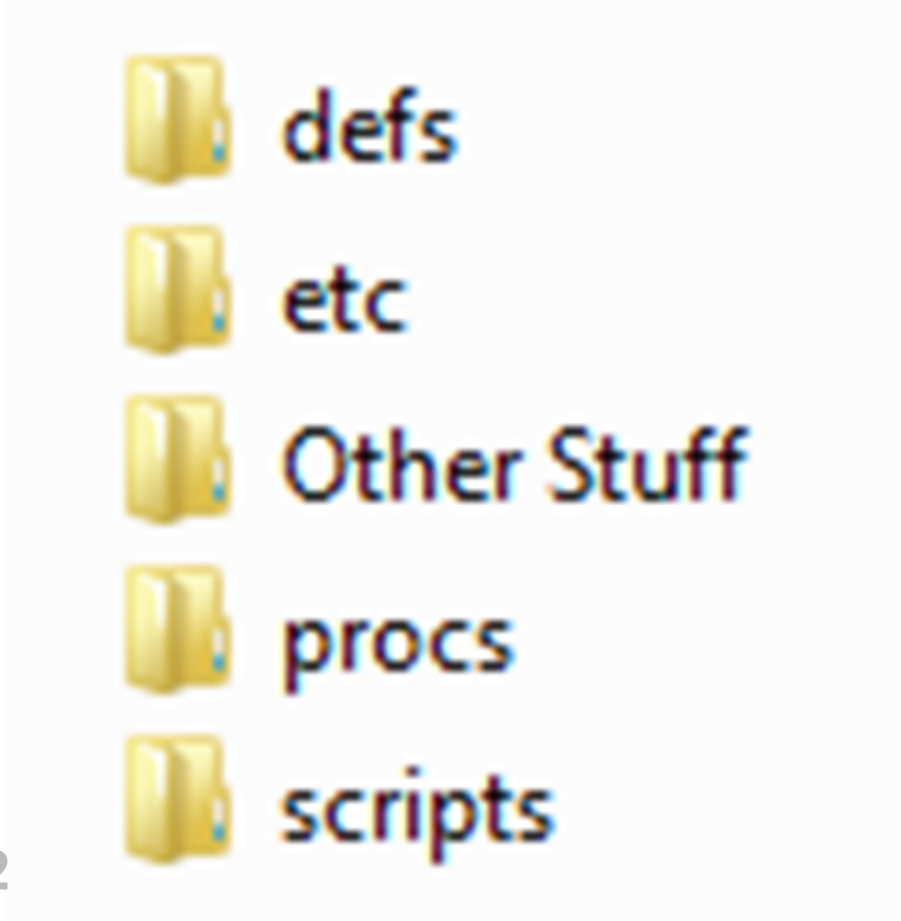
Anti-pattern: "Other Stuff"

- Team members do not fully understand the logic behind code organization.
- They still are eager to contribute, but when they actually do, they break it.

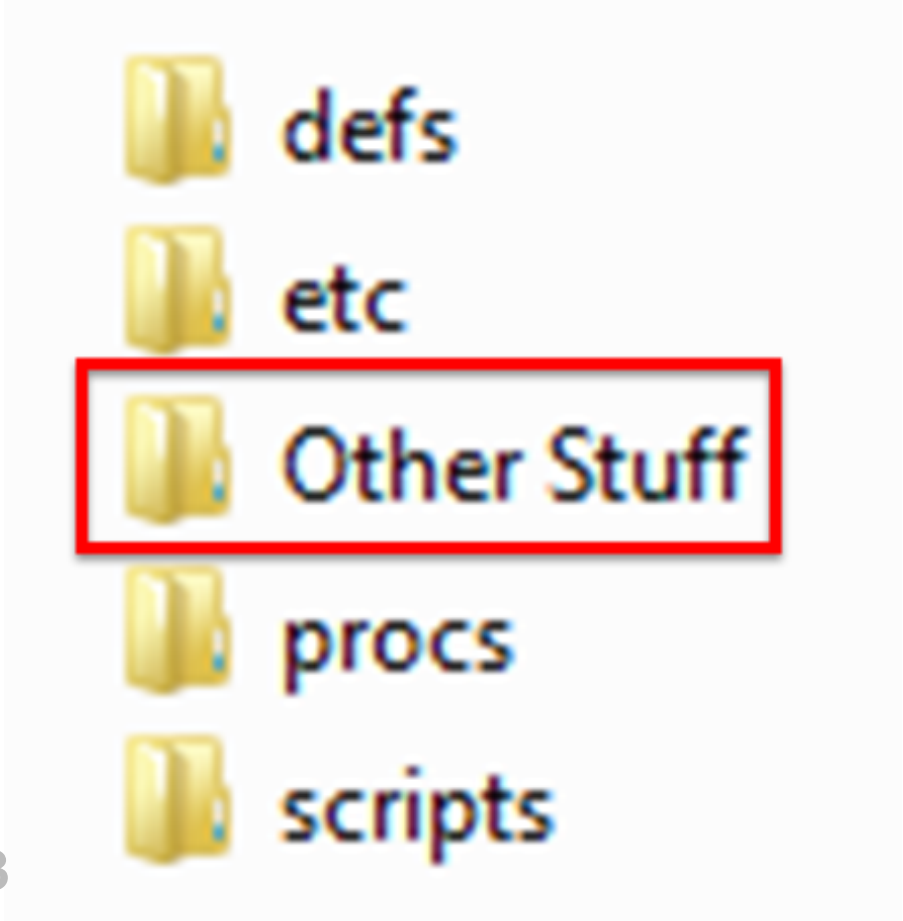
Example: "Other Stuff"



Example: "Other Stuff"



Example: "Other Stuff"



Anti-pattern: Big Ball of Mud

- Well, it's possible to create mess out of anything.

Pattern: Automation over Documentation

- It's quite common that Ops team have been given or have created a bunch of documents describing procedures for system operations.
- Code can do better!
- It happens that writing those documents take as much time as writing and testing code that implement the same guide lines.
- Automating procedures can reduce the amount of documentation needed or eliminate the documentation completely.

Conclusion

Summary of anti-patterns I

- Anti-pattern: Golden Image
- Anti-pattern: Postponing Secret Isolation
- Anti-pattern: "Fancy-File-Copying"
- Anti-pattern: Data as Code

Summary of anti-patterns II

- Anti-pattern: Not Treating IaC as Code
- Anti-pattern: Ignoring Styling Guidelines
- Anti-pattern: Private Fork of a Community Module
- Anti-pattern: "Other Stuff"
- Anti-pattern: Big Ball of Mud

Summary of patterns I

- Pattern: Reproducible Images
- Pattern: Secret Isolating
- Pattern: Encrypted Secrets
- Pattern: Infrastructure Component DSL
- Pattern: Incremental Configuration
- Pattern: Configuration Composition
- Pattern: Extra-Packaging Code

Summary of patterns II

- Pattern: Configuration Discovery
- Pattern: Configuration Data Source
- Pattern: Infrastructure Query
- Pattern: Metrics as Code
- Pattern: Control Panel as Code
- Pattern: Community Module Wrapper
- Pattern: Environment Template
- Pattern: Automation over Documentation

That's it!

Thank you!

Questions?

Feedback is welcome!

Share your patterns:

- Write a blog post!
- Share a tweet with @codingandrey or #iacpatterns.
- Or just write me to andrey@aestasit.com.