# The Zen of Ansible

Tim Appnel
Sr. Product Manager, Red Hat Ansible
GitHub: tima
Matrix: @tima:ansible.im

## About me

10+ years of experience with Ansible as a contributor, customer, consultant, evangelist, product manager, and “jack of all trades.”

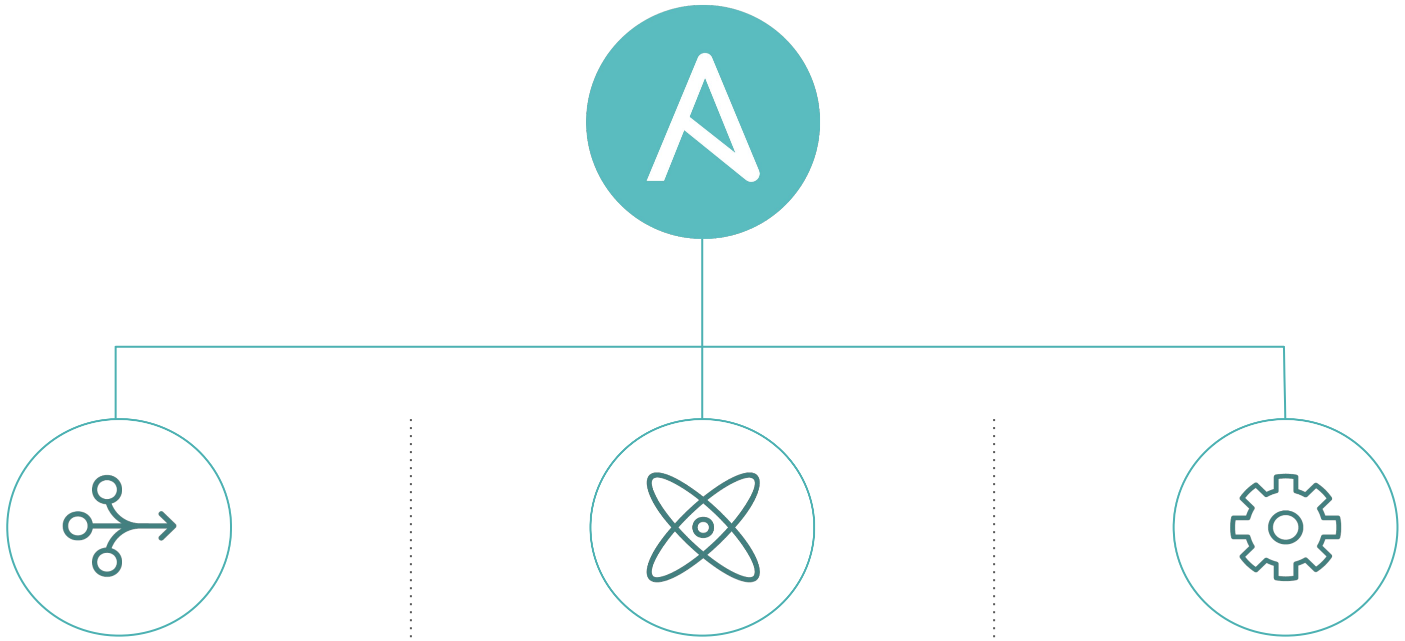
The synchronize module in Ansible is all my fault. (Sorry engineering)

## About this talk

Spiritual successor of the “Ansible best practices” talk first presented at Red Hat Summit June 2016 and later at dozens of events over the years

Inspired by “The Zen of Python” by Tim Peters

# The Ansible way



**Simple**

Human-readable language with quick adoption

**Powerful**

Universal language spanning multiple IT domains

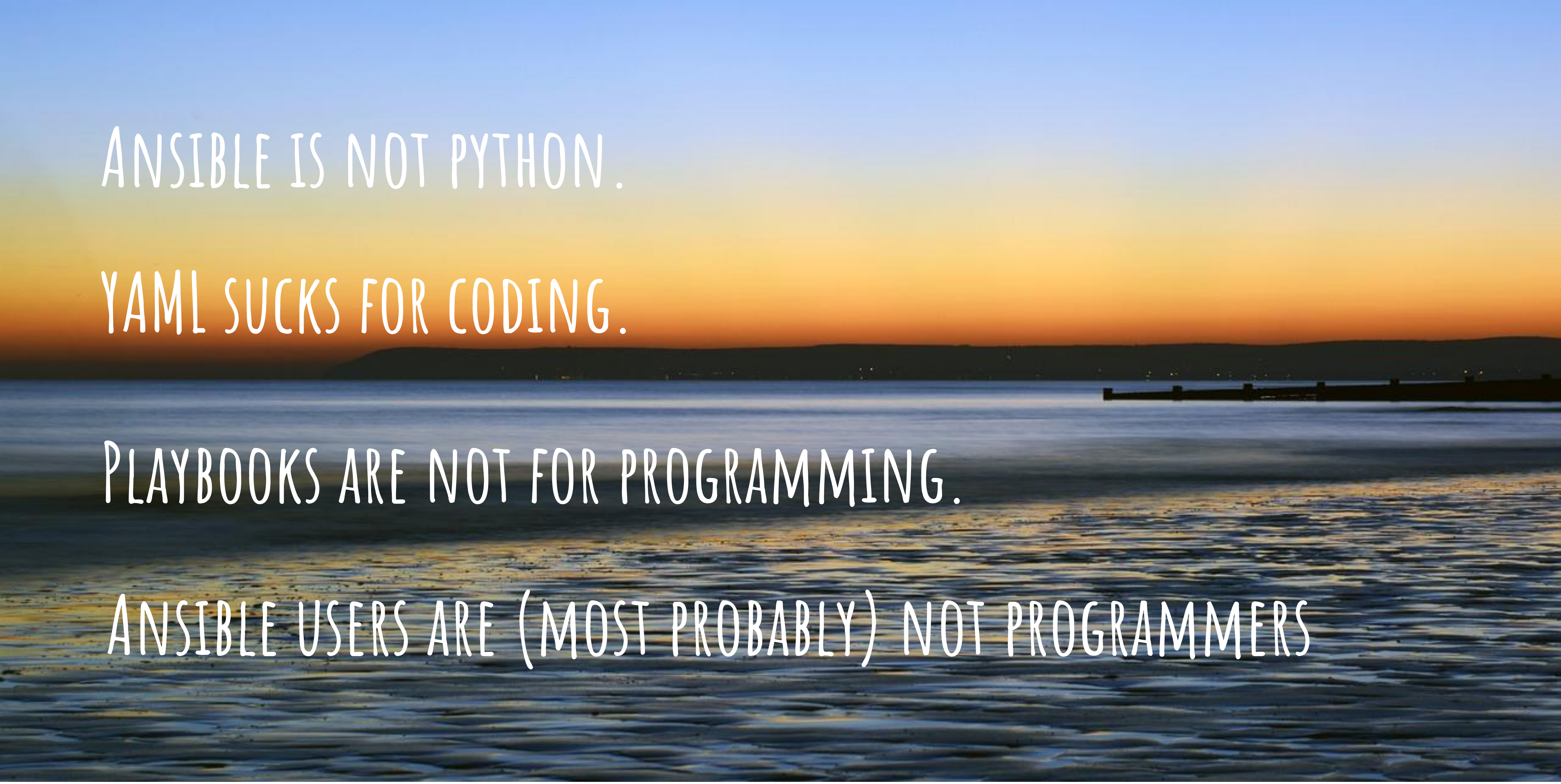
**Agentless**

Easily integrate with hybrid environments

# About this talk

The Zen of Ansible applies to playbooks, roles and the interfaces and functional design of modules and plugins.

Apply The Zen of Python to how you write and develop your code.



Ansible is not python.

YAML sucks for coding.

Playbooks are not for programming.

Ansible users are (most probably) not programmers

## Example: Overuse of command modules forcing programming

```
- hosts: all
  vars:
    cert_store: /etc/mycerts
    cert_name: my cert
  tasks:
  - name: check cert
    ansible.builtin.shell: certify --list --name={{ cert_name }} --cert_store={{ cert_store }} | grep "{{ cert_name }}"
    register: output

  - name: create cert
    ansible.builtin.command: certify --create --user=chris --name={{ cert_name }} --cert_store={{ cert_store }}
    when: output.stdout.find(cert_name)" != -1
    register: output

  - name: sign cert
    ansible.builtin.command: certify --sign  --name={{ cert_name }} --cert_store={{ cert_store }}
    when: output.stdout.find("created")" != -1
```

## Example: Custom module to abstract user from programming  in playbook

```yaml
- hosts: all
  vars:
    cert_store: /etc/mycerts
    cert_name: my cert
  tasks:
    - name: create and sign cert
      umbrellacorp.nest.certify:
        state: present
        sign: yes
        user: chris
        name: "{{ cert_name }}"
        cert_store: "{{ cert_store }}"
```

## Example: External script and programming to pass parameters

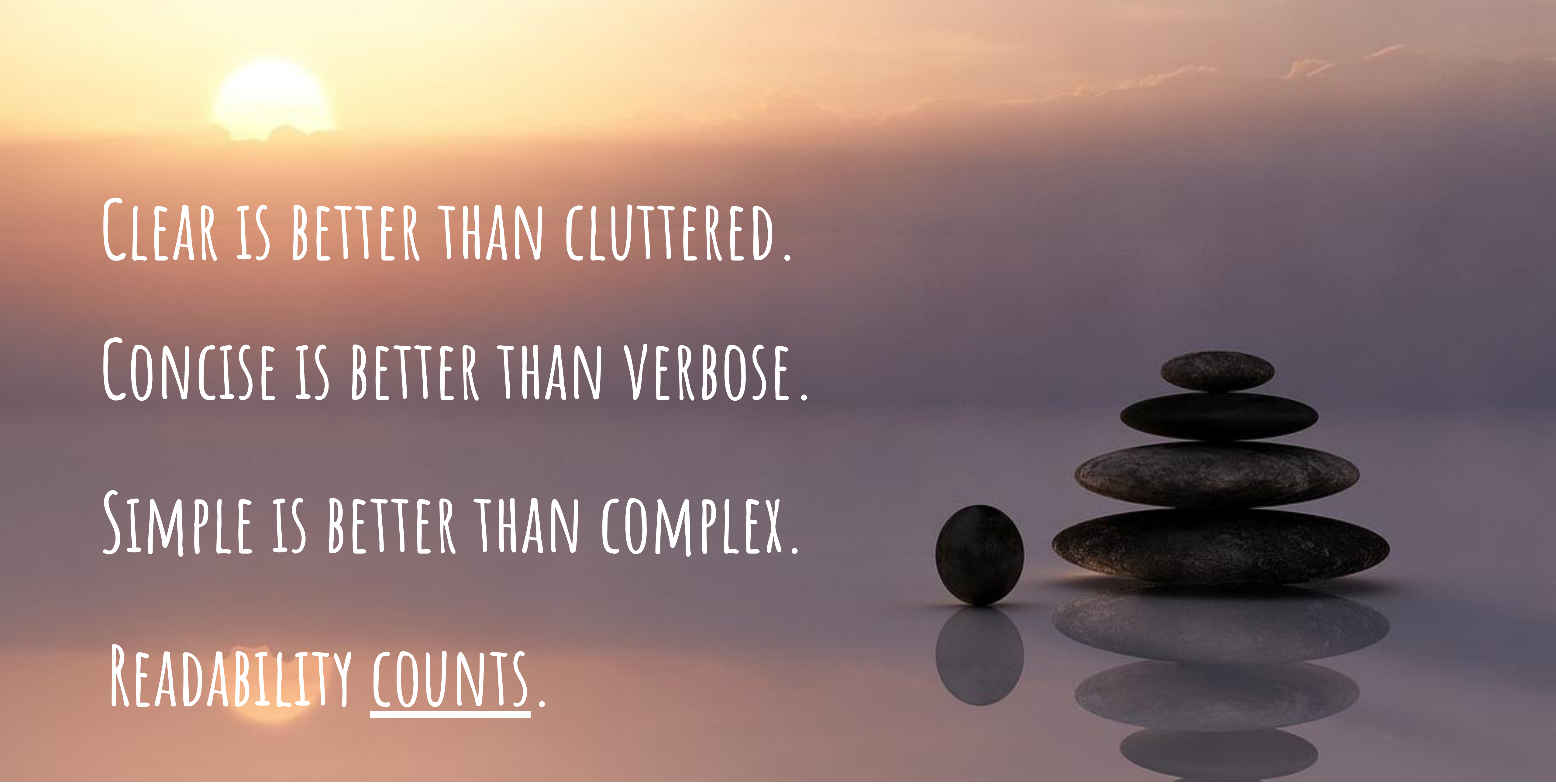
```
- ansible.builtin.set_fact:
   splitter_cmd: >
     python3 /tmp/list_changed_targets.py
     --branch {{ zuul.branch }}
     {% if ansible_test_splitter__releases_to_test is defined %}
         --ansible-releases {{ ansible_test_splitter__releases_to_test | join(' ') }}{% endif %}
     {% if ansible_test_splitter__total_job is defined %}--total-job {{
   ansible_test_splitter__total_job }}{% endif %}
     {% if ansible_test_splitter__test_changed|bool %}--test-changed{% else %}--test-all-the-targets{%
   endif %}
     {{ ansible_test_splitter__check_for_changes_in | join(' ') }}
```

## Example: External script and building command line

```
- name: Evaluate security group rules
  ansible.builtin.command: >
    python {{ ansible_role_dir }}/files/validate_security_group_rules.py
    --dest_subnet_cidrs "{{ rds_subnets_cidrs }}"
    --dest_security_groups "{{ rds_security_groups.security_groups }}"
    --dest_port "{{ rds_instance_endpoint_port }}"
    --src_security_groups "{{ ec2_security_groups.security_groups }}"
    --src_private_ip "{{ ec2_private_ip_addrs | first }}"
```

## Example: Purpose built module in a collection that avoids programming in the playbook.

```yaml
- name: Evaluate Security Group Rules
  cloud.aws_troubleshooting.validate_security_group_rules:
    dest_subnet_cidrs: "{{ rds_subnets_cidrs }}"
    dest_security_groups: "{{ rds_security_groups.security_groups }}"
    dest_port: "{{ rds_instance_endpoint_port }}"
    src_security_groups: "{{ ec2_security_groups.security_groups }}"
    src_private_ip: "{{ ec2_private_ip_addrs | first }}"
```



Clear is better than cluttered.

Concise is better than verbose.

Simple is better than complex.

Readability counts.

## Example: Functional  automation tasks using shorthand that is hard to read

```
- name: install telegraf
  ansible.builtin.yum: name=telegraf-{{ telegraf_version }} state=present update_cache=yes disable_gpg_check=yes enab
  notify: restart telegraf

- name: configure telegraf
  ansible.builtin.template: src=telegraf.conf.j2 dest=/etc/telegraf/telegraf.conf

- name: start telegraf
  ansible.builtin.service: name=telegraf state=started enabled=yes
```

## Example: Functional and more readable Ansible automation

```yaml
- name: install telegraf
  ansible.builtin.yum:
    name: telegraf-{{ telegraf_version }}
    state: present
    update_cache: yes
    disable_gpg_check: yes
    enablerepo: telegraf
  notify: restart telegraf

- name: configure telegraf
  ansible.builtin.template:
    src: telegraf.conf.j2
    dest: /etc/telegraf/telegraf.conf
  notify: restart telegraf

- name: start telegraf
  ansible.builtin.service:
    name: telegraf
    state: started
    enabled: yes
```

Helping users get things done matters most.

User experience > ideological purity.

## Example: User experience beating ideological purity

From “Ansible Best Practices: Roles & Modules” (circa 2018):

- [Modules] abstract users from having to know the details to get things done
- [Modules] are **not** one-to-one mapping of an API or command line tool interface
  - This is why you should not auto-generate your modules
- Keep parameters focused and narrowly defined — refrain from parameters that take complex data structures

```
- name: Create Service object with inline definition
  kubernetes.core.k8s:
    state: present
    definition:
      apiVersion: v1
      kind: Service
      metadata:
        name: web
        namespace: testing
        labels:
          app: galaxy
          service: web
      spec:
        selector:
          app: galaxy
          service: web
        ports:
        - protocol: TCP
          targetPort: 8000
          name: port-8000-tcp
          port: 8000
```

## "Magic" conquers the manual.

When giving users options, always use convention over configuration.

## Example: Providing common values and reasonable defaults for users

EXHIBIT A

```
# defaults_no_playbook.yml
---
- hosts: webservers
  roles:
    - role: apache_simple
      apache_http_port: 80
      apache_doc_root: /var/www/html
      apache_user: apache
      apache_group: apache
    - role: apache_simple
      apache_http_port: 8080
      apache_doc_root: /www/example.com
      apache_user: apache
      apache_group: apache
```

EXHIBIT B

```
# defaults_yes_playbook.yml
---
- hosts: webservers
  roles:
    - role: apache_simple
    - role: apache_simple
      apache_http_port: 8080
      apache_doc_root: /www/example.com


# default/main.yml
---
apache_http_port: 80
apache_doc_root: /var/www/html
apache_user: apache
apache_group: apache
```

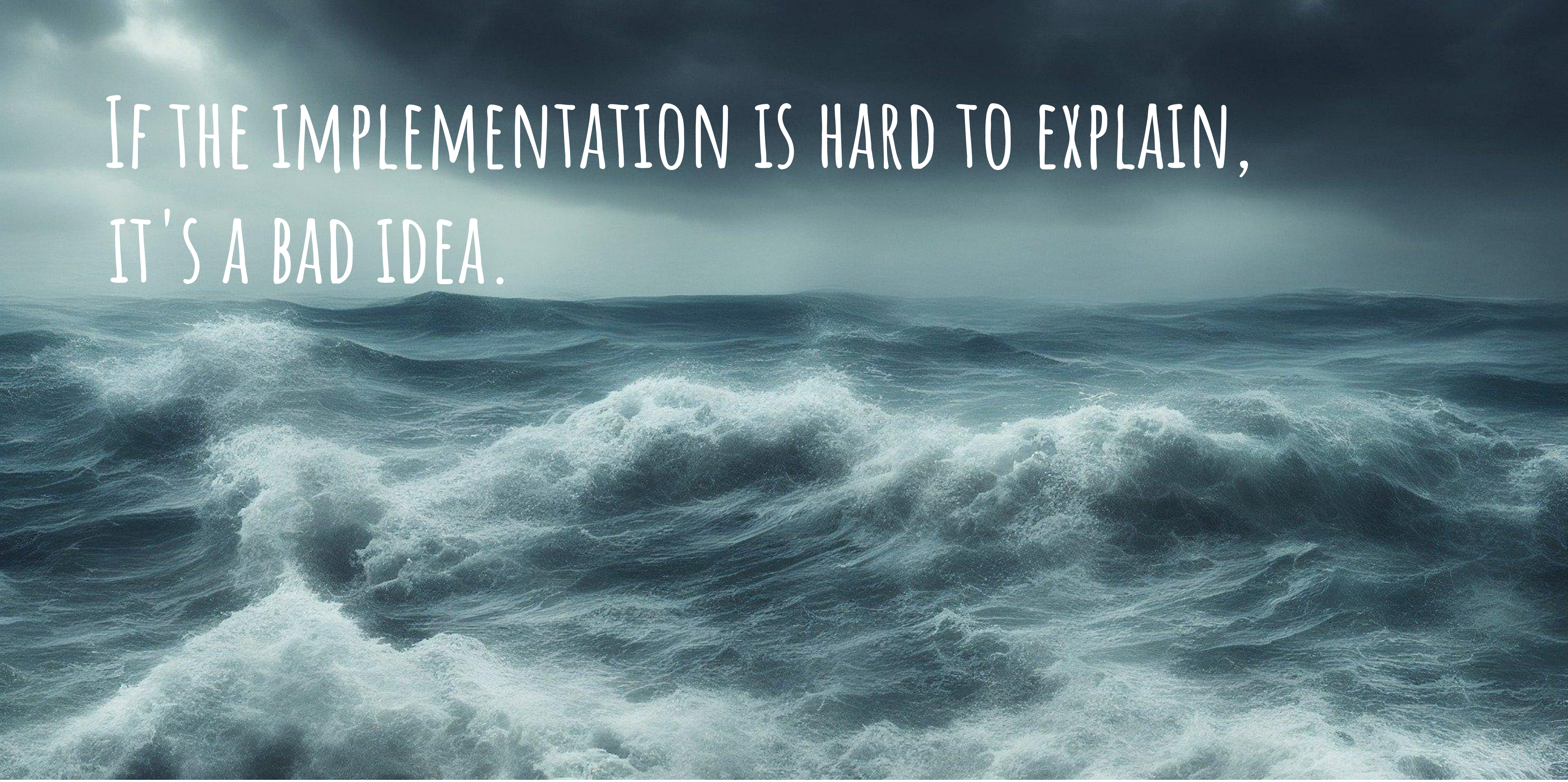
Declarative is <u>always</u> better than imperative.

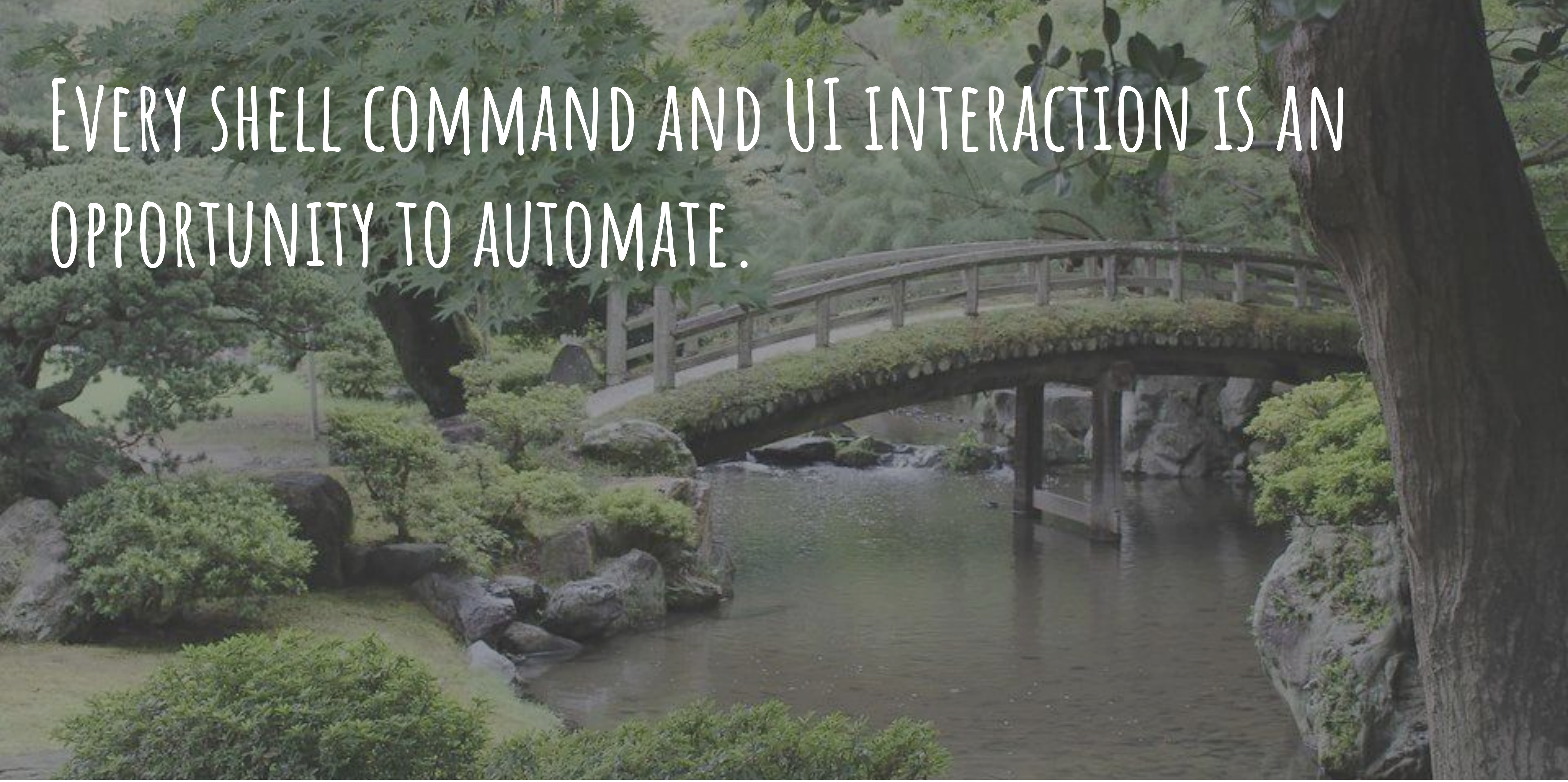
Most of the time.

Focus avoids complexity

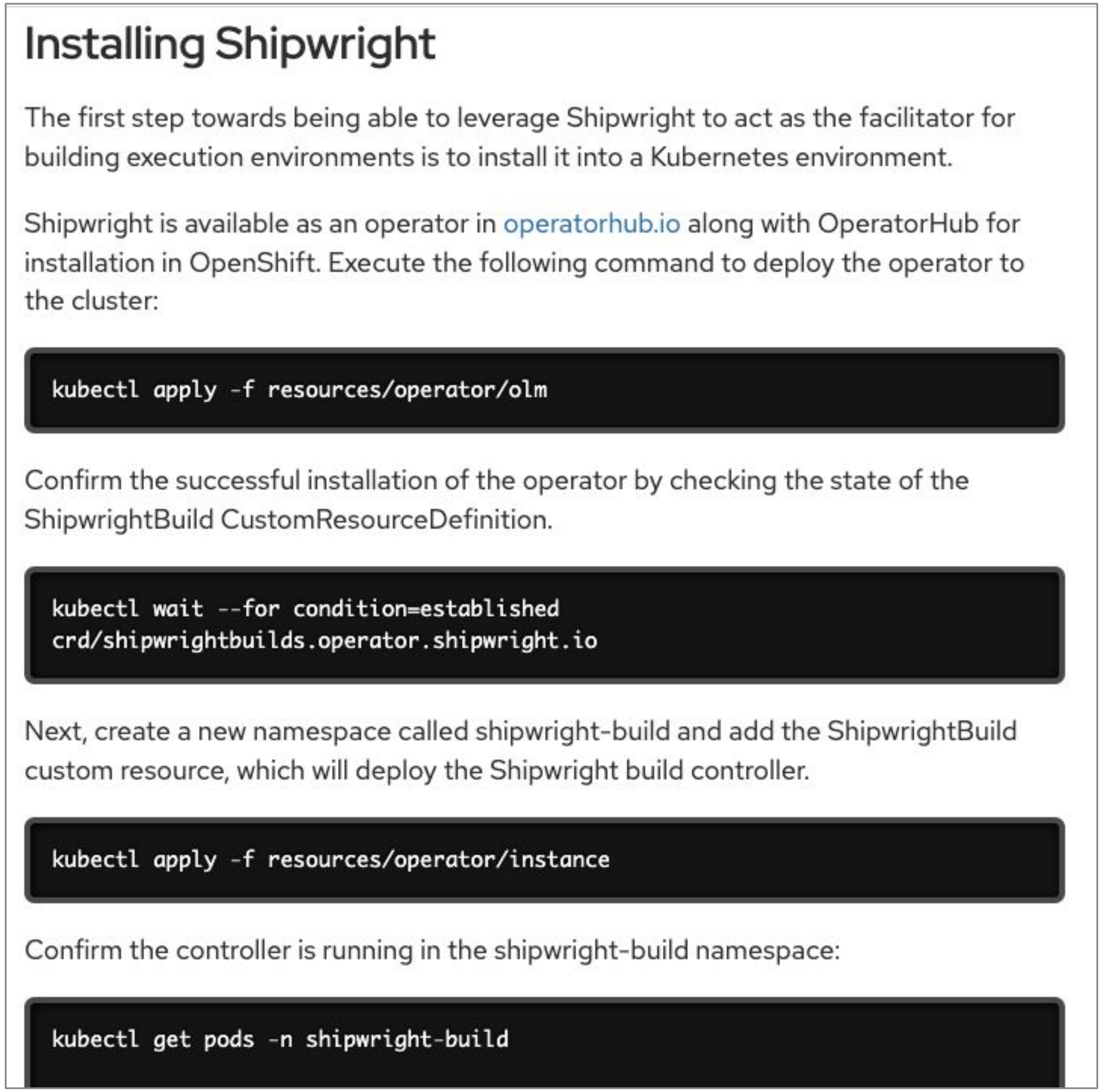
Complexity kills productivity

If the implementation is hard to explain, it's a bad idea.





## Every shell command and UI interaction is an opportunity to automate.



## Installing Shipwright

The first step towards being able to leverage Shipwright to act as the facilitator for building execution environments is to install it into a Kubernetes environment.

Shipwright is available as an operator in operatorhub.io along with OperatorHub for installation in OpenShift. Execute the following command to deploy the operator to the cluster:

```
kubectl apply -f resources/operator/olm
```

Confirm the successful installation of the operator by checking the state of the ShipwrightBuild CustomResourceDefinition.

```
kubectl wait --for condition=established
crd/shipwrightbuilds.operator.shipwright.io
```

Next, create a new namespace called shipwright-build and add the ShipwrightBuild custom resource, which will deploy the Shipwright build controller.

```
kubectl apply -f resources/operator/instance
```

Confirm the controller is running in the shipwright-build namespace:

```
kubectl get pods -n shipwright-build
```

Just because something works, doesn't mean it can't be improved.

Friction should be eliminated whenever possible.

## Example: Continuously improving by removing friction and clutter for users

```yaml
# Create a Deployment reading a definition template from the Ansible controller local file
- name: My testing Deployment exists
  kubernetes.core.k8s:
    state: present
    definition: "{{ lookup('template', '/testing/deployment.j2') | from_yaml }}"


# Read definition template file from the Ansible controller file system
- name: My testing Deployment exists
  kubernetes.core.k8s:
    state: present
    template: '/testing/deployment.j2'
```

Automation is a continuous journey that never ends.

## More Reading & Resources

- Good Practices for Ansible
  - ↳ https://redhat-cop.github.io/automation-good-practices/
- Ansible Lint Documentation
  - ↳ https://ansible-lint.readthedocs.io/
- PEP 20 – The Zen of Python
  - ↳ https://peps.python.org/pep-0020/
- The Zen of Python, Explained
  - ↳ https://inventwithpython.com/blog/2018/08/17/the-zen-of-python-explained/

## Thanks!

GitHub: https://github.com/ansible/community
Matrix: #community:ansible.com or #social:ansible.com

ANSIBLE