Managing Complexity

Config As Code, Custom Ansible Modules, and You

Zach Marine, Squarespace

About me

- Software engineer
- 3.5 years at Squarespace
- Focus on data tooling and infrastructure

SQUARESPACE

- Platform for creating websites, domains, commerce stores, etc.
- “Squarespace makes beautiful products to help people with creative ideas succeed.”
- Millions of active subscriptions
- 650+ employees across New York, Portland, and Dublin

Agenda

1. Config As Code
2. Ansible
3. Testing

Agenda

Postgres Permissions

1. Config As Code
2. Ansible
3. Testing

Our Example: Postgres Permissions

Behavior	Effect

Our Example: Postgres Permissions

Behavior	Effect
Postgres permissions are highly flexible	Requires high level of knowledge Similar “boilerplate” required for additional entities

Our Example: Postgres Permissions

Behavior	Effect
Postgres permissions are highly flexible	Requires high level of knowledge Similar “boilerplate” required for additional entities
Info is distributed throughout the database	Hard to fully understand the setup

Our Example: Postgres Permissions

Behavior	Effect
Postgres permissions are highly flexible	Requires high level of knowledge Similar “boilerplate” required for additional entities
Info is distributed throughout the database	Hard to fully understand the setup
Changes can be made live	No vetting from others Testing environments may not accurately represent production

How can we make this better?

Our Example: Postgres Permissions

Behavior	Effect
Postgres permissions are highly flexible	Requires high level of knowledge Similar “boilerplate” required for additional entities
Info is distributed throughout the database	Hard to fully understand the setup
Changes can be made live	No vetting from others Testing environments may not accurately represent production

Our Example: Postgres Permissions

Behavior	Effect
Postgres permissions are highly flexible	Requires high level of knowledge Similar "boilerplate" required for additional entities **Simplify end user API**
Info is distributed throughout the database	Hard to fully understand the setup
Changes can be made live	No vetting from others Testing environments may not accurately represent production

Our Example: Postgres Permissions

Behavior	Effect
Postgres permissions are highly flexible	Requires high level of knowledge Similar "boilerplate" required for additional entities **Simplify end user API**
Info is distributed throughout the database	Hard to [illegible] **Put all config together**
Changes can be made live	No vetting from others Testing environments may not accurately represent production

Our Example: Postgres Permissions

Behavior	Effect
Postgres permissions are highly flexible	Requires high level of knowledge Similar "boilerplate" required for additional entities **Simplify end user API**
Info is distributed throughout the database	Hard to [illegible] **Put all config together**
Changes can be made live	No [illegible] **Get config into version control** Testing environments may not accurately represent production

Our Example: Postgres Permissions

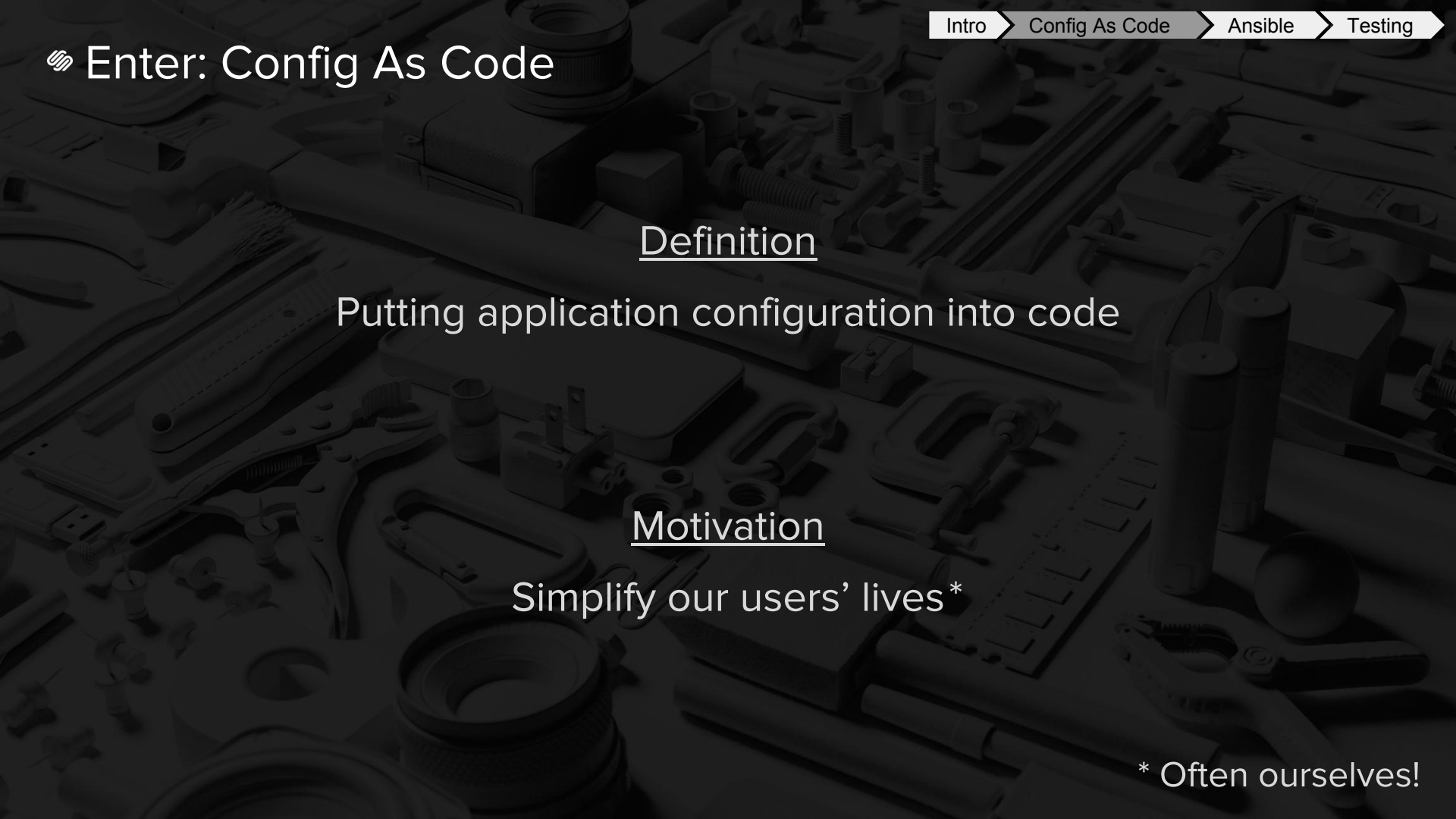
Behavior	Effect
Postgres permissions are highly flexible	Requires high level of knowledge Similar "boilerplate" required for additional entities **Simplify end user API**
Info is distributed throughout the database	Hard to [illegible] **Put all config together**
Changes can be made live	No [illegible] **Get config into version control** Testing environments may not accurately represent production **Make config deployable**

Our Example: Postgres Permissions

Behavior	Effect
Postgres permissions are highly flexible	Requires high level of knowledge Similar "boilerplate" required for additional entities **Simplify end user API**
Info is distributed throughout the database	Hard to [illegible] **Put all config together**
Changes can be made live	No [illegible] **Get config into version control** Testing environments may not accurately represent production **Make config deployable**

What is config as code?

Enter: Config As Code



Definition

Putting application configuration into code

Motivation

Simplify our users' lives*

* Often ourselves!

Config As Code: Best Practices

Before

```
CREATE ROLE "svc-mlcompute" LOGIN;

GRANT is_app TO "svc-mlcompute";

GRANT USAGE ON SCHEMA dmart TO "svc-mlcompute";
GRANT USAGE ON SCHEMA ods TO "svc-mlcompute";
GRANT USAGE ON SCHEMA sqsp TO "svc-mlcompute";
GRANT USAGE ON SCHEMA staging TO "svc-mlcompute";

GRANT SELECT ON ALL TABLES IN SCHEMA dmart TO "svc-mlcompute";
GRANT SELECT ON ALL TABLES IN SCHEMA ods TO "svc-mlcompute";
GRANT SELECT ON ALL TABLES IN SCHEMA sqsp TO "svc-mlcompute";
GRANT SELECT ON ALL TABLES IN SCHEMA staging TO "svc-mlcompute";

ALTER DEFAULT PRIVILEGES IN SCHEMA dmart GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA ods GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA sqsp GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA staging GRANT SELECT ON TABLES TO "svc-mlcompute";
```

After

```
- name: svc-mlcompute
  can_login: yes
  membership:
      - is_app
  privileges:
      read:
          schema:
              - dmart
              - ods
              - sqsp
              - staging
          table:
              - dmart.*
              - ods.*
              - sqsp.*
              - staging.*
```

Config As Code: Best Practices

1. Simplify our users' lives by *using terminology they understand*.

Before

```
CREATE ROLE "svc-mlcompute" LOGIN;

GRANT is_app TO "svc-mlcompute";

GRANT USAGE ON SCHEMA dmart TO "svc-mlcompute";
GRANT USAGE ON SCHEMA ods TO "svc-mlcompute";
GRANT USAGE ON SCHEMA sqsp TO "svc-mlcompute";
GRANT USAGE ON SCHEMA staging TO "svc-mlcompute";

GRANT SELECT ON ALL TABLES IN SCHEMA dmart TO "svc-mlcompute";
GRANT SELECT ON ALL TABLES IN SCHEMA ods TO "svc-mlcompute";
GRANT SELECT ON ALL TABLES IN SCHEMA sqsp TO "svc-mlcompute";
GRANT SELECT ON ALL TABLES IN SCHEMA staging TO "svc-mlcompute";

ALTER DEFAULT PRIVILEGES IN SCHEMA dmart GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA ods GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA sqsp GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA staging GRANT SELECT ON TABLES TO "svc-mlcompute";
```

After

```
- name: svc-mlcompute
  can_login: yes
  membership:
      - is_app
  privileges:
      read:
          schema:
              - dmart
              - ods
              - sqsp
              - staging
          table:
              - dmart.*
              - ods.*
              - sqsp.*
              - staging.*
```

Config As Code: Best Practices

1. Simplify our users' lives by *using terminology they understand*.

Before

```
CREATE ROLE "svc-mlcompute" LOGIN;

GRANT is_app TO "svc-mlcompute";

GRANT  USAGE  ON SCHEMA dm
GRANT  USAGE  ON SCHE
GRANT  USAGE  ON
GRANT  US

GRA
GRA
GRAN
GRANT  SELECT  ON ALL TABLES IN SCHEMA staging TO "svc-mlcompute";

ALTER DEFAULT PRIVILEGES IN SCHEMA dmart GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA ods GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA sqsp GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA staging GRANT SELECT ON TABLES TO "svc-mlcompute";
```

Database -> CONNECT

After

```
- name: svc-mlcompute
  can_login: yes
  membership:
      - is_app
  privileges:
      read:
          schema:
              - dmart
              - ods
              - sqsp
              - staging
          table:
              - dmart.*
              - ods.*
              - sqsp.*
              - staging.*
```

Config As Code: Best Practices

1. Simplify our users' lives by *using terminology they understand*.

Before

```
CREATE ROLE "svc-mlcompute" LOGIN;

GRANT is_app TO "svc-mlcompute";

GRANT  USAGE  ON SCHEMA dm
GRANT  USAGE  ON SCHE
GRANT  USAGE
GRANT  US

GRA
GRA
GRAN
GRANT  SELECT  ON ALL TABLES IN SCHEMA staging TO "svc-mlcompute";

ALTER DEFAULT PRIVILEGES IN SCHEMA dmart GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA ods GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA sqsp GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA staging GRANT SELECT ON TABLES TO "svc-mlcompute";
```

Database -> CONNECT

Sequence -> SELECT

After

```
name: svc-mlcompute
login: yes

privi
    read:
        schema:
            - dmart
            - ods
            - sqsp
            - staging
        table:
            - dmart.*
            - ods.*
            - sqsp.*
            - staging.*
```

Config As Code: Best Practices

Foreign Server -> USAGE

1. Simplify our users' lives by *using terminology they understand*.

Before

```
CREATE ROLE "svc-mlcompute" LOGIN;

GRANT is_app TO "svc-mlcompute";

GRANT  USAGE  ON SCHEMA dm
GRANT  USAGE  ON SCHE
GRANT  USAGE
GRANT  US

GRA
GRA
GRAN
GRANT SELECT ON ALL TABLES IN SCHEMA staging TO "svc-mlcompute";

ALTER DEFAULT PRIVILEGES IN SCHEMA dmart GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA ods GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA sqsp GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA staging GRANT SELECT ON TABLES TO "svc-mlcompute";
```

Database -> CONNECT

Sequence -> SELECT

After

```
name: svc-mlcompute
login: yes

privi
    read:
        schema:
            - dmart
            - ods
            - sqsp
            - staging
        table:
            - dmart.*
            - ods.*
            - sqsp.*
            - staging.*
```

Config As Code: Best Practices

1. Simplify our users' lives by *using terminology they understand*.

Foreign Server -> USAGE

Database -> CONNECT

Sequence -> SELECT

Large Object -> SELECT

Before

```
CREATE ROLE "svc-mlcompute" LOGIN;

GRANT is_app TO "svc-mlcompute";

GRANT  USAGE  ON SCHEMA dm
GRANT  USAGE  ON SCHE
GRANT  USAGE
GRANT  US

GRA
GRA
GRAN
GRANT SELECT ON ALL TABLES IN SCHEMA staging TO "svc-mlcompute";

ALTER DEFAULT PRIVILEGES IN SCHEMA dmart GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN S
ALTER DEFAULT PRIVILEGES IN S
ALTER DEFAULT PRIVILEGES IN S
```

After

```
name: svc-mlcompute
login: yes

priv
    read:
        schema:
            - dmart
            - ods
            - sqsp
            - staging
        table:
            - dmart.*
            - ods.*
            - sqsp.*
            - staging.*
```

Config As Code: Best Practices

1. Simplify our users' lives by *using terminology they understand*.

Foreign Server -> USAGE

Sequence -> SELECT

Database -> CONNECT

Language -> USAGE

Large Object -> SELECT

Before

```
CREATE ROLE "svc-mlcompute" LOGIN;

GRANT is_app TO "svc-mlcompute";

GRANT  USAGE  ON SCHEMA dm
GRANT  USAGE  ON SCHE
GRANT  USAGE  ON
GRANT  US

GRA
GRA
GRAN
GRANT  SELECT  ON ALL TABLES IN SCHEMA staging TO "svc-mlcompute";

ALTER DEFAULT PRIVILEGES IN SCHEMA dmart GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN S
ALTER DEFAULT PRIVILEGES IN S
ALTER DEFAULT PRIVILEGES IN S
```

After

```
name: svc-mlcompute
login: yes

priv
  read:
    schema:
      - dmart
      - ods
      - sqsp
      - staging
    tab
      - dmart.*
      - ods.*
      - sqsp.*
      - staging.*
```

Config As Code: Best Practices

1. Simplify our users' lives by *using terminology they understand*.

Foreign Server -> READ

Sequence -> READ

Database -> READ

Language -> READ

Large Object -> READ

Before

```
CREATE ROLE "svc-mlcompute" LOGIN;

GRANT is_app TO "svc-mlcompute";

GRANT USAGE ON SCHEMA dm
GRANT USAGE ON SCHE
GRANT USAGE ON
GRANT US

GRA
GRA
GRAN
GRANT SELECT ON ALL TABLES IN SCHEMA staging TO "svc-mlcompute";

ALTER DEFAULT PRIVILEGES IN SCHEMA dmart GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN S
ALTER DEFAULT PRIVILEGES IN S
ALTER DEFAULT PRIVILEGES IN S
```

After

```
name: svc-mlcompute
login: yes

privi
  read:
    schema:
      - dmart
      - ods
      sp
      taging
    tabl
      - dmart.*
      - ods.*
      - sqsp.*
      - staging.*
```

Config As Code: Best Practices

2. Simplify our users' lives by *reusing techniques they're familiar with*.

Before

```
CREATE ROLE "svc-mlcompute" LOGIN;

GRANT is_app TO "svc-mlcompute";

GRANT USAGE ON SCHEMA dmart TO "svc-mlcompute";
GRANT USAGE ON SCHEMA ods TO "svc-mlcompute";
GRANT USAGE ON SCHEMA sqsp TO "svc-mlcompute";
GRANT USAGE ON SCHEMA staging TO "svc-mlcompute";

GRANT SELECT ON ALL TABLES IN SCHEMA dmart TO "svc-mlcompute";
GRANT SELECT ON ALL TABLES IN SCHEMA ods TO "svc-mlcompute";
GRANT SELECT ON ALL TABLES IN SCHEMA sqsp TO "svc-mlcompute";
GRANT SELECT ON ALL TABLES IN SCHEMA staging TO "svc-mlcompute";

ALTER DEFAULT PRIVILEGES IN SCHEMA dmart GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA ods GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA sqsp GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA staging GRANT SELECT ON TABLES TO "svc-mlcompute";
```

After

```
- name: svc-mlcompute
  can_login: yes
  membership:
      - is_app
  privileges:
      read:
          schema:
              - dmart
              - ods
              - sqsp
              - staging
          table:
              - dmart.*
              - ods.*
              - sqsp.*
              - staging.*
```

Config As Code: Best Practices

3. Simplify our users' lives by *abstracting away implementation details*.

Before

```
CREATE ROLE "svc-mlcompute" LOGIN;

GRANT is_app TO "svc-mlcompute";

GRANT USAGE ON SCHEMA dmart TO "svc-mlcompute";
GRANT USAGE ON SCHEMA ods TO "svc-mlcompute";
GRANT USAGE ON SCHEMA sqsp TO "svc-mlcompute";
GRANT USAGE ON SCHEMA staging TO "svc-mlcompute";

GRANT SELECT ON ALL TABLES IN SCHEMA dmart TO "svc-mlcompute";
GRANT SELECT ON ALL TABLES IN SCHEMA ods TO "svc-mlcompute";
GRANT SELECT ON ALL TABLES IN SCHEMA sqsp TO "svc-mlcompute";
GRANT SELECT ON ALL TABLES IN SCHEMA staging TO "svc-mlcompute";

ALTER DEFAULT PRIVILEGES IN SCHEMA dmart GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA ods GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA sqsp GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA staging GRANT SELECT ON TABLES TO "svc-mlcompute";
```

After

```
- name: svc-mlcompute
  can_login: yes
  membership:
      - is_app
  privileges:
      read:
          schema:
              - dmart
              - ods
              - sqsp
              - staging
          table:
              - dmart.*
              - ods.*
              - sqsp.*
              - staging.*
```

Config As Code: Best Practices

4. Simplify our users' lives by *removing boilerplate.*

Before

```
CREATE ROLE "svc-mlcompute" LOGIN;

GRANT is_app TO "svc-mlcompute";

GRANT USAGE ON SCHEMA dmart TO "svc-mlcompute";
GRANT USAGE ON SCHEMA ods TO "svc-mlcompute";
GRANT USAGE ON SCHEMA sqsp TO "svc-mlcompute";
GRANT USAGE ON SCHEMA staging TO "svc-mlcompute";

GRANT SELECT ON ALL TABLES IN SCHEMA dmart TO "svc-mlcompute";
GRANT SELECT ON ALL TABLES IN SCHEMA ods TO "svc-mlcompute";
GRANT SELECT ON ALL TABLES IN SCHEMA sqsp TO "svc-mlcompute";
GRANT SELECT ON ALL TABLES IN SCHEMA staging TO "svc-mlcompute";

ALTER DEFAULT PRIVILEGES IN SCHEMA dmart GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA ods GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA sqsp GRANT SELECT ON TABLES TO "svc-mlcompute";
ALTER DEFAULT PRIVILEGES IN SCHEMA staging GRANT SELECT ON TABLES TO "svc-mlcompute";
```

After

```
- name: svc-mlcompute
  can_login: yes
  membership:
      - is_app
  privileges:
      read:
          schema:
              - dmart
              - ods
              - sqsp
              - staging
          table:
              - dmart.*
              - ods.*
              - sqsp.*
              - staging.*
```

YAML (“Yet Another Markup Language”)

We chose to put our config into YAML

Benefits for our users:

- Familiar to many users
- Not a lot of boilerplate
  - Easy to read
  - Easy to write

Benefits for ourselves:

- Easy to parse in Python

```
- name: service_account1
  can_login: yes
  membership:
      - is_app
  privileges:
      read:
          schema:
              - metadata
          table:
              - metadata.oplog_tail_status

- name: person1
  can_login: yes
  has_personal_schema: yes
  membership:
      - analyst

- name: zmarine
  can_login: yes
  has_personal_schema: yes
  membership:
      - data-engineer
```

YAML ("Yet Another Markup Language")

```
(e) code In [1]  import yaml

                 with open('spec.yml', 'r') as f:
                     spec = yaml.load(f)

(e) code In [2]  from pprint import pprint

                 pprint(spec)
```

```
[{'can_login': True,
  'membership': ['is_app'],
  'name': 'service_account1',
  'privileges': {'read': {'schema': ['metadata'],
                          'table': ['metadata.oplog_tail_status']}}},
 {'can_login': True,
  'has_personal_schema': True,
  'membership': ['analyst'],
  'name': 'person1'},
 {'can_login': True,
  'has_personal_schema': True,
  'membership': ['data-engineer'],
  'name': 'zmarine'}]
```

```
- name: service_account1
  can_login: yes
  membership:
      - is_app
  privileges:
      read:
          schema:
              - metadata
          table:
              - metadata.oplog_tail_status

- name: person1
  can_login: yes
  has_personal_schema: yes
  membership:
      - analyst

- name: zmarine
  can_login: yes
  has_personal_schema: yes
  membership:
      - data-engineer
```

Great! We've got our config into code.

...Now what?

Our Example: Postgres Permissions

Behavior	Effect
Postgres permissions are highly flexible	Requires high level of knowledge Similar "boilerplate" required for additional entities **Simplify end user API**
Info is distributed throughout the database	Hard to [illegible] **Put all config together**
Changes can be made live	No [illegible] **Get config into version control** Testing environments may not accurately represent production **Make config deployable**

Our Example: Postgres Permissions

Behavior	Effect
Postgres permissions are highly flexible	Requires high level of knowledge Similar "boilerplate" required for additional entities **Simplify end user API**
Info is distributed throughout the database	Hard to [illegible] **Put all config together**
Changes can be made live	No [illegible] **Get config into version control** Testing environments may not accurately represent production **Make config deployable**

What is ansible?

What is Ansible?

Ansible described itself as:

> “A radically simple IT automation engine”

Often described as “infrastructure as code”

Similar tools:

- Chef
- Puppet
- SaltStack
- Terraform

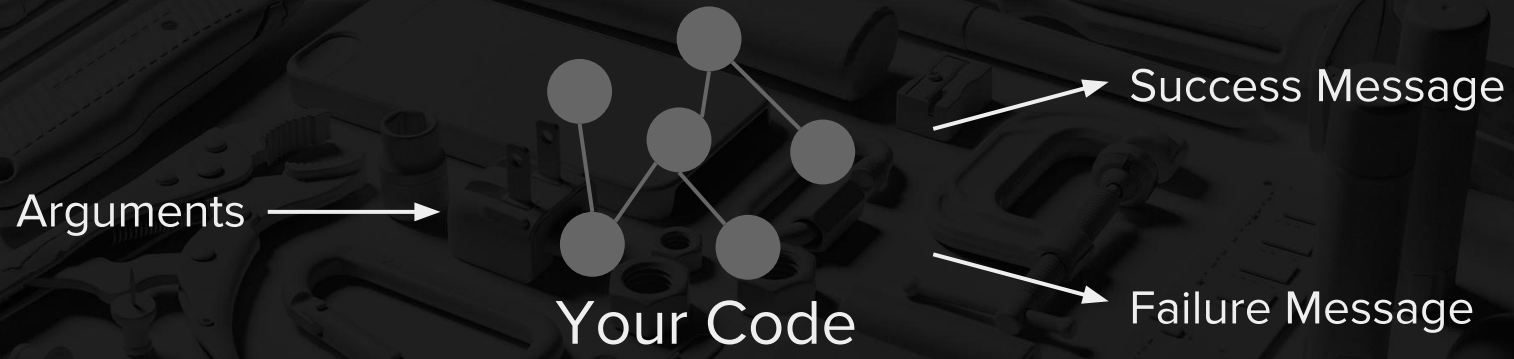
Why Ansible?

- Currently the automation tool of choice at Squarespace
- Easy to write your own module
- Supports secure secret storage
- Modules typically support “check” mode
  (see what will change without actually doing it)

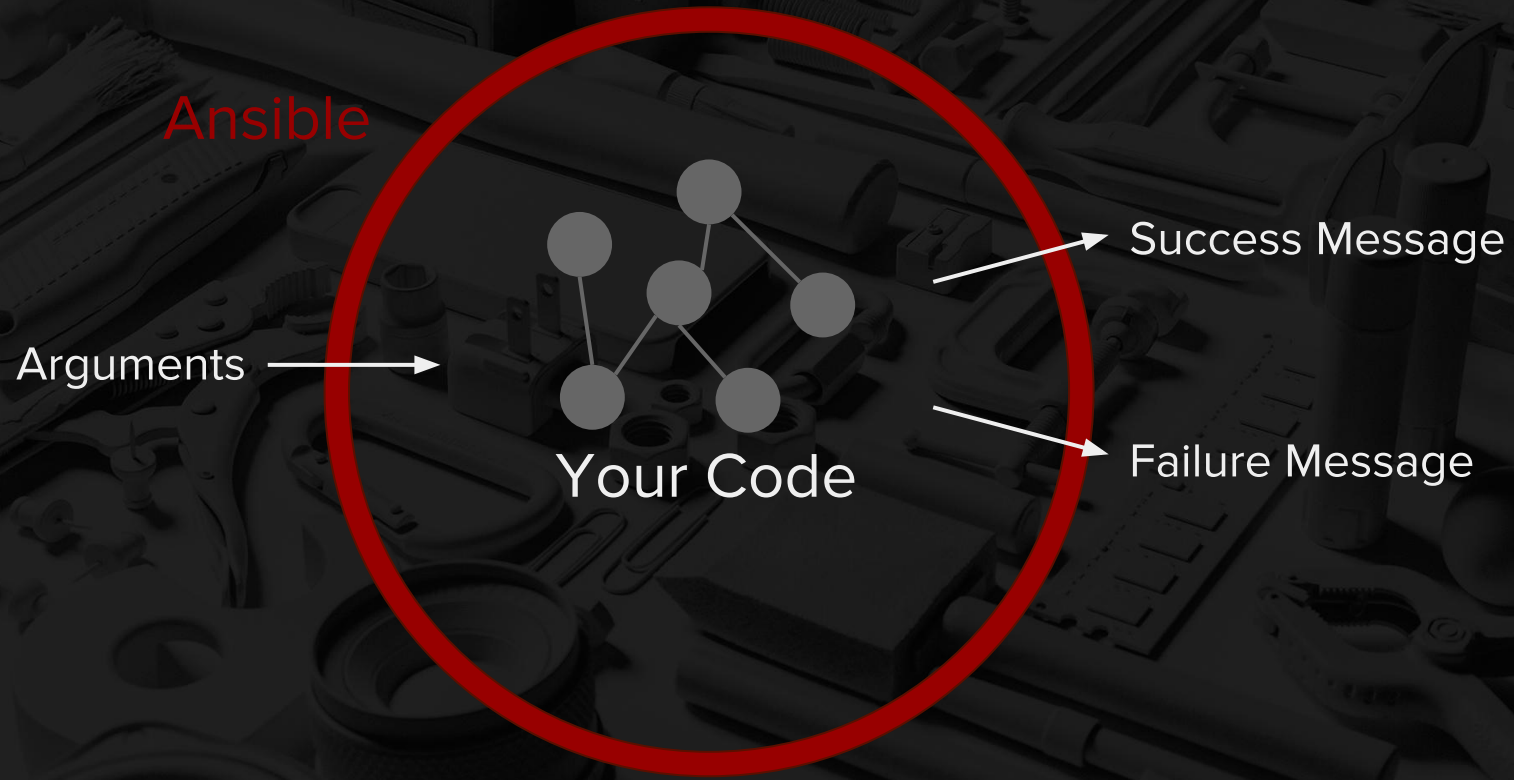
Why Ansible?

- Currently the automation tool of choice at Squarespace
- Easy to write your own module
- Supports secure secret storage
- Modules typically support “check” mode
  (see what will change without actually doing it)

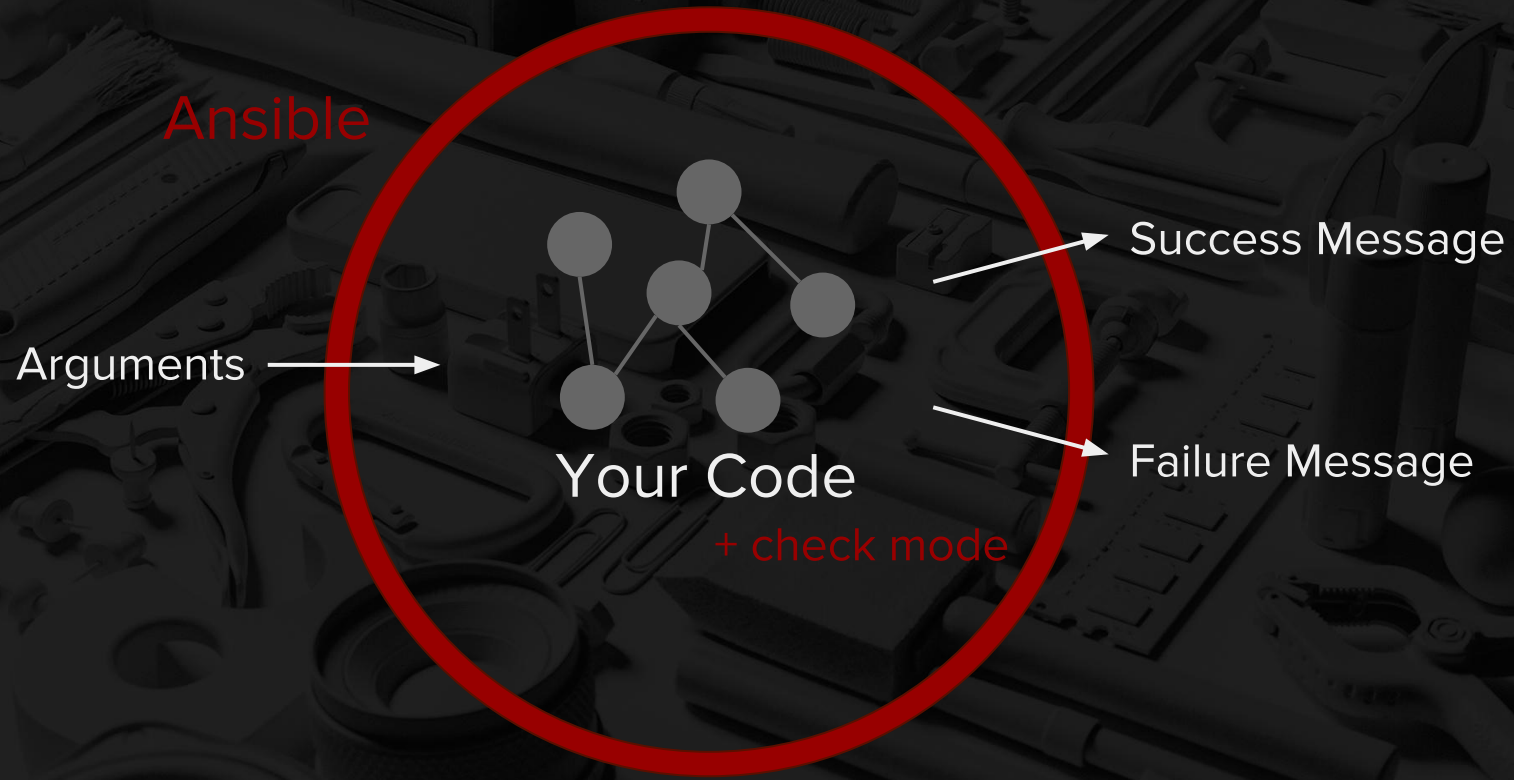
Ansible And Custom Modules



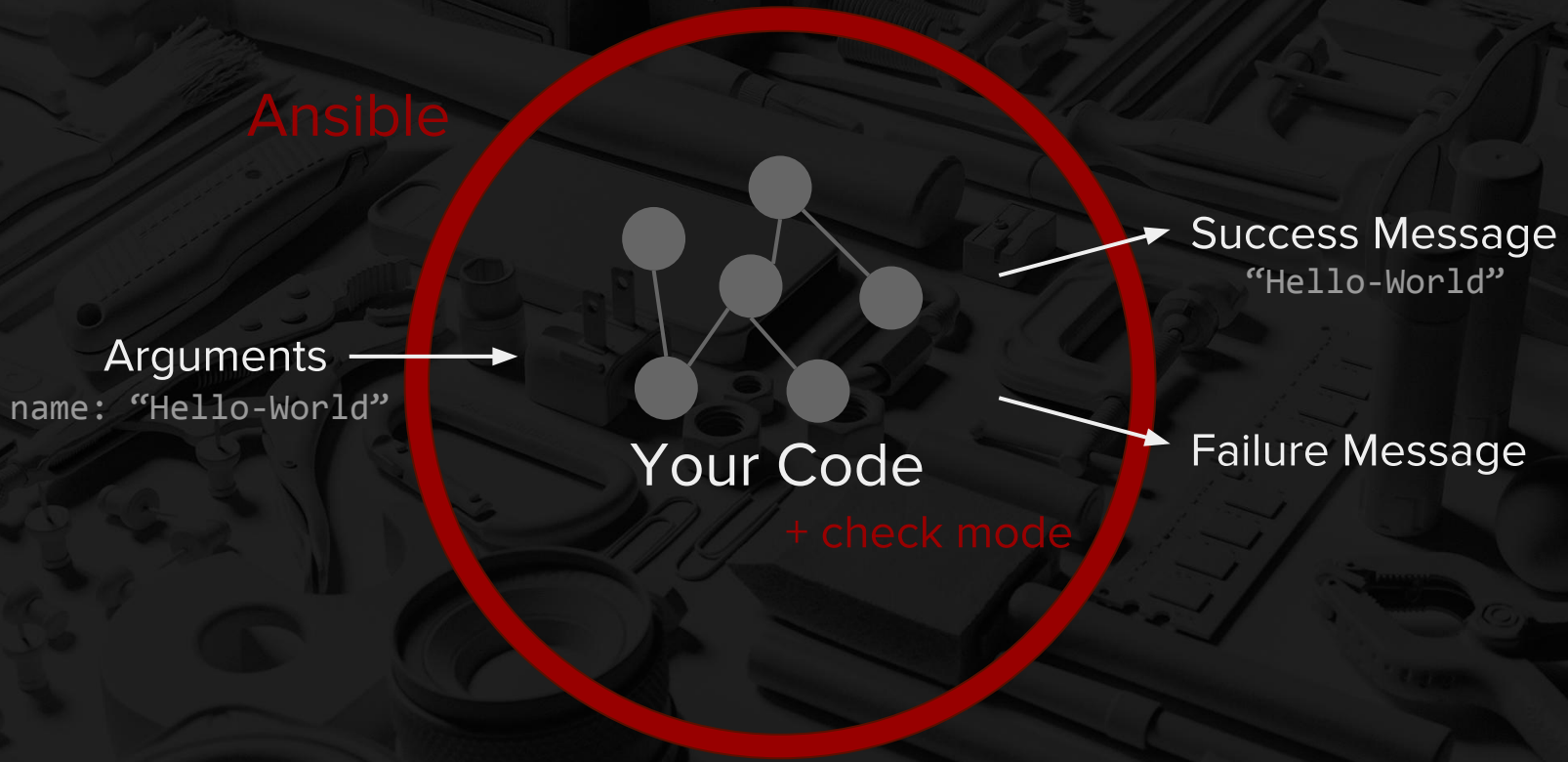
Ansible And Custom Modules



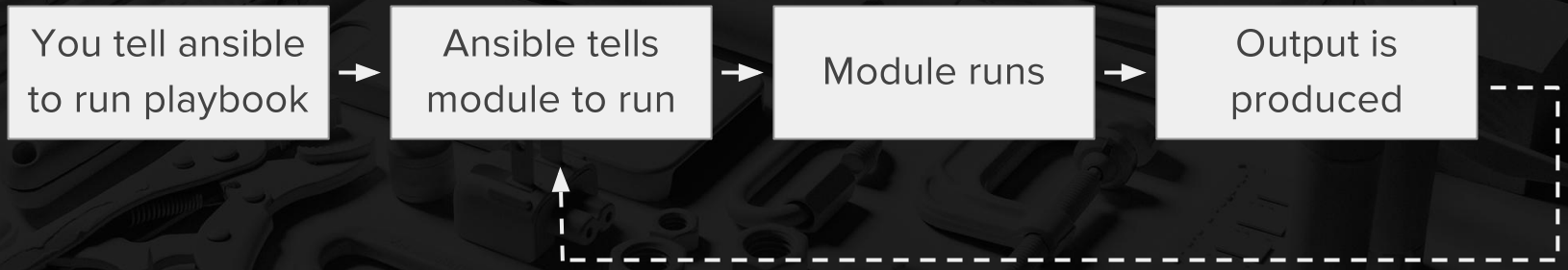
Ansible And Custom Modules



Ansible And Custom Modules



Ansible Execution



Ansible Execution


```
>>> pip install ansible
>>> export ANSIBLE_LIBRARY='.'
>>> ansible-playbook -v basic_playbook.yml
```

Ansible Execution

You tell ansible to run playbook → Ansible tells module to run → Module runs → Output is produced

./basic_playbook.yml

```
1 ---
2 - name: A basic playbook
3   hosts: localhost
4   tasks:
5     - name: Test that my module works
6       basic_module:
7         name: "Hello-World"
```

Servers to run against → line 3

Description of task → line 5

Module name → line 6

Module arguments → line 7

Ansible Execution

You tell ansible to run playbook → Ansible tells module to run → Module runs → Output is produced

./basic_module.py

```python
#!/usr/bin/python

from ansible.module_utils.basic import AnsibleModule


def setup_module():
    return AnsibleModule(
        argument_spec=dict(
            name=dict(required=True, type='str'),
        ),
        supports_check_mode=True
    )


def main():
    module = setup_module()
    module.exit_json(changed=True, msg=module.params['name'])


if __name__ == '__main__':
    main()

```

Ansible Execution

You tell ansible to run playbook → Ansible tells module to run → Module runs → Output is produced

Modules don't have to be Python!

./basic_module.py

```
#!/usr/bin/python

from ansible.module_utils.basic import AnsibleModule


def setup_module():
    return AnsibleModule(
        argument_spec=dict(
            name=dict(required=True, type='str'),
        ),
        supports_check_mode=True
    )


def main():
    module = setup_module()
    module.exit_json(changed=True, msg=module.params['name'])


if __name__ == '__main__':
    main()

```

Ansible Execution

You tell ansible to run playbook → Ansible tells module to run → **Module runs** → Output is produced

Define expected arguments

./basic_module.py

```python
#!/usr/bin/python

from ansible.module_utils.basic import AnsibleModule


def setup_module():
    return AnsibleModule(
        argument_spec=dict(
            name=dict(required=True, type='str'),
        ),
        supports_check_mode=True
    )


def main():
    module = setup_module()
    module.exit_json(changed=True, msg=module.params['name'])


if __name__ == '__main__':
    main()

```

Ansible Execution

You tell ansible to run playbook → Ansible tells module to run → Module runs → Output is produced

./basic_module.py

```python
#!/usr/bin/python

from ansible.module_utils.basic import AnsibleModule


def setup_module():
    return AnsibleModule(
        argument_spec=dict(
            name=dict(required=True, type='str'),
        ),
        supports_check_mode=True
    )


def main():
    module = setup_module()
    module.exit_json(changed=True, msg=module.params['name'])


if __name__ == '__main__':
    main()

```

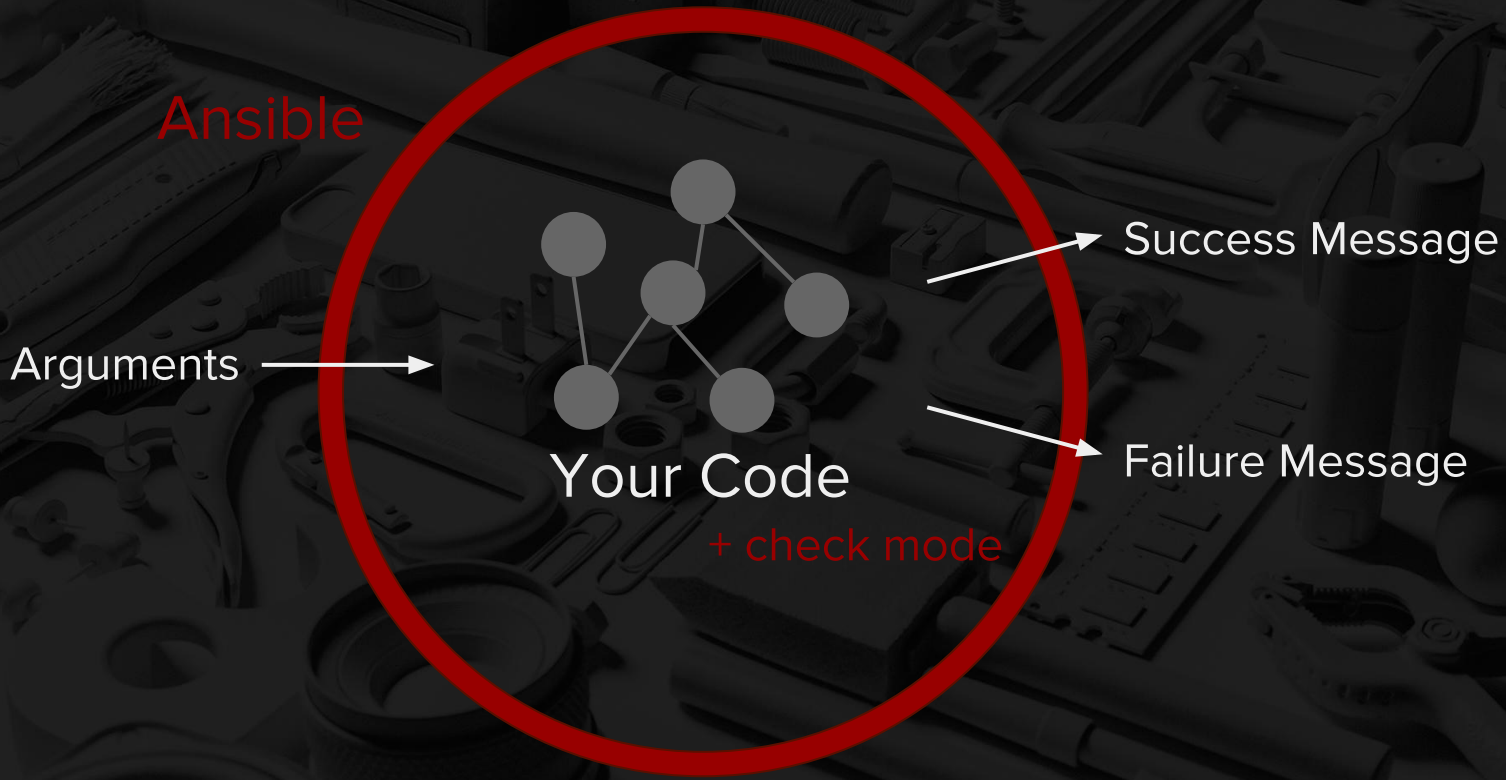
Provide info upon exit

Ansible Execution


```
PLAY [localhost] ***************************************************************

TASK [Gathering Facts] *********************************************************
ok: [localhost]

TASK [Test that my module works] ***********************************************
changed: [localhost] => {"changed": true, "failed": false, "msg": "Hello-World"}

PLAY RECAP *********************************************************************
localhost                  : ok=2    changed=1    unreachable=0    failed=0
```

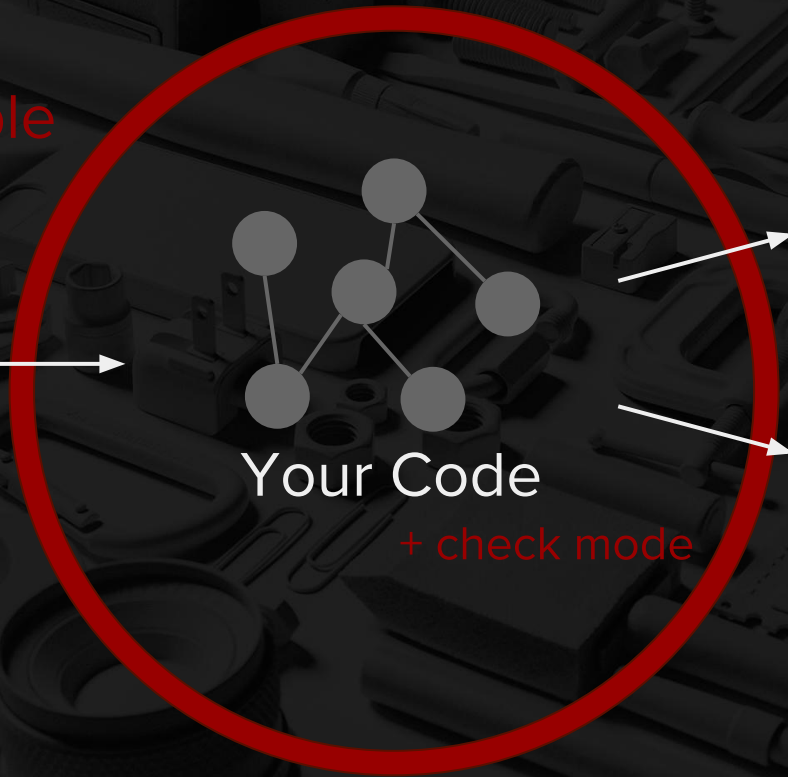
Ansible Execution

You tell ansible to run playbook → Ansible tells module to run → Module runs → Output is produced



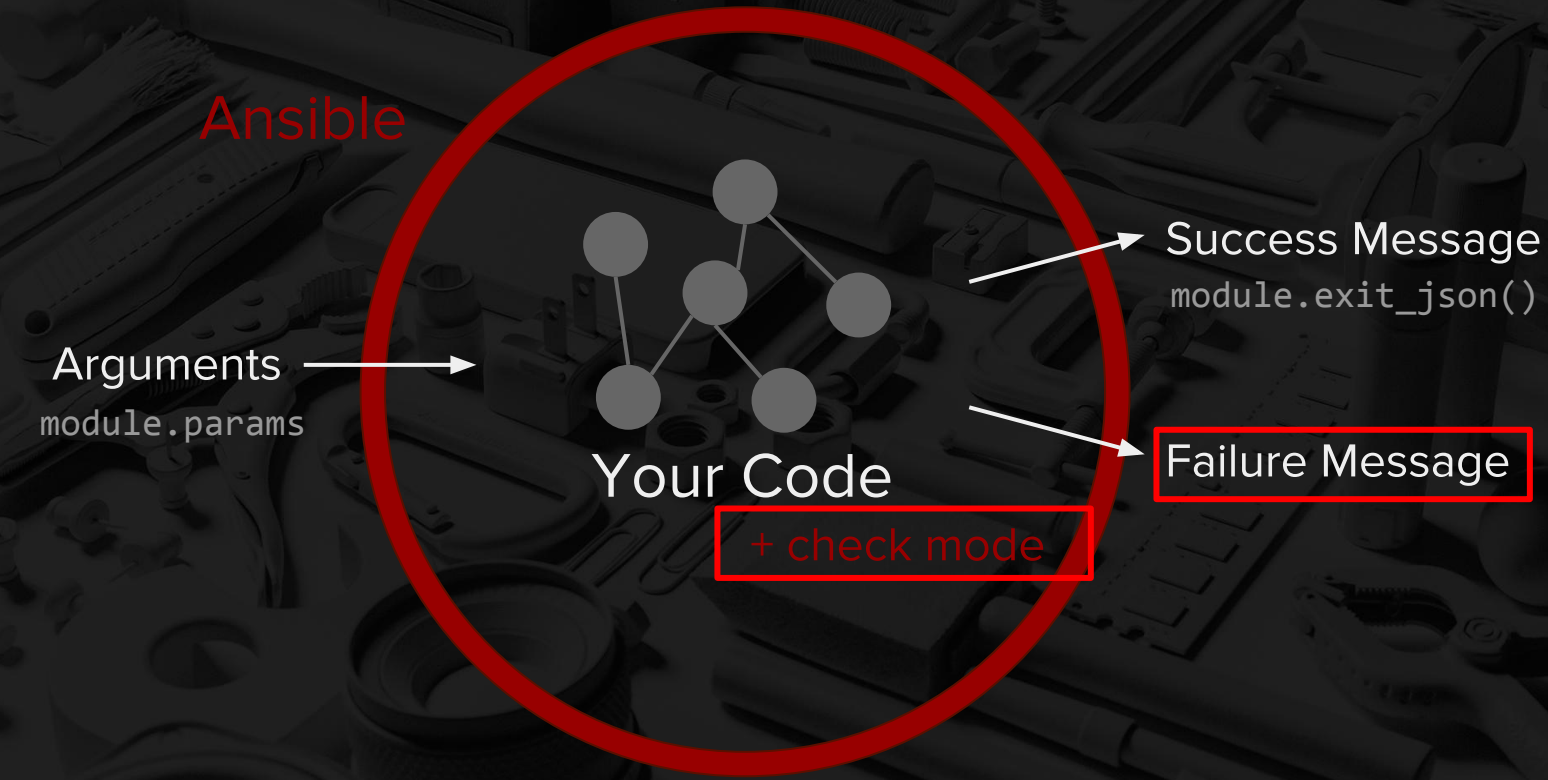
Ansible Execution

You tell ansible to run playbook → Ansible tells module to run → Module runs → Output is produced



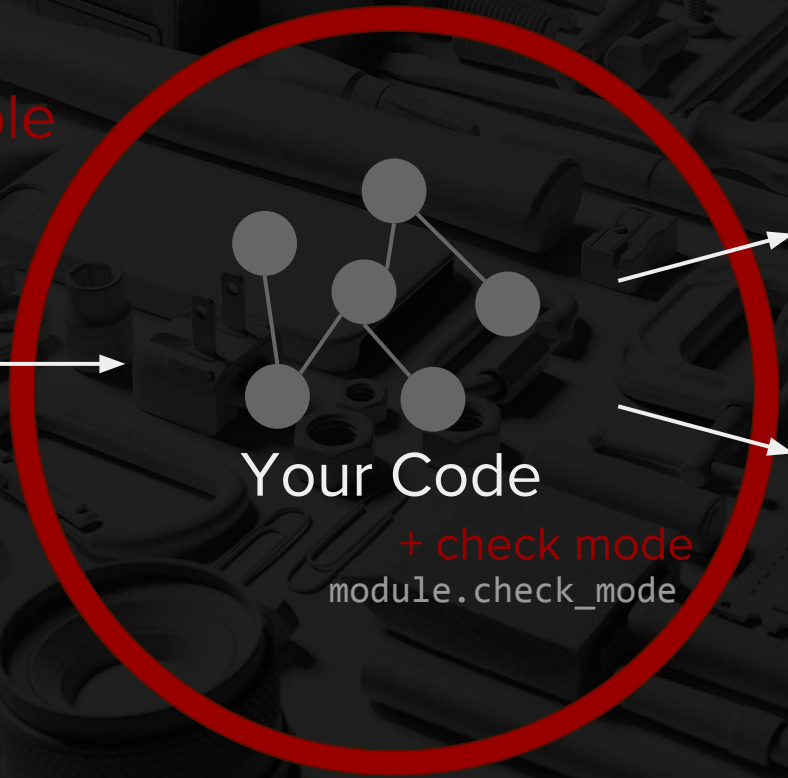
Ansible Execution

You tell ansible to run playbook → Ansible tells module to run → Module runs → Output is produced



Ansible Execution

You tell ansible to run playbook → Ansible tells module to run → Module runs → Output is produced



Ansible Execution: fail_json + check mode

```
14
15 def main():
16     module = setup_module()
17
18     if module.check_mode:
19         module.exit_json(changed=False, msg='This is check mode')
20
21     if module.params['name'] == 'Oscar':
22         module.fail_json(msg="Don't use the name 'Oscar'")
23
24     module.exit_json(changed=False, msg=module.params['name'])
25
```

```
>>> ansible-playbook -v basic_playbook.yml --check

PLAY [localhost] ***************************************************************

TASK [Gathering Facts] *********************************************************
ok: [localhost]

TASK [Test that my module works] ***********************************************
ok: [localhost] => {"changed": false, "failed": false, "msg": "This is check mode"}

PLAY RECAP *********************************************************************
localhost                  : ok=2    changed=0    unreachable=0    failed=0
```

Ansible Execution: fail_json + check mode

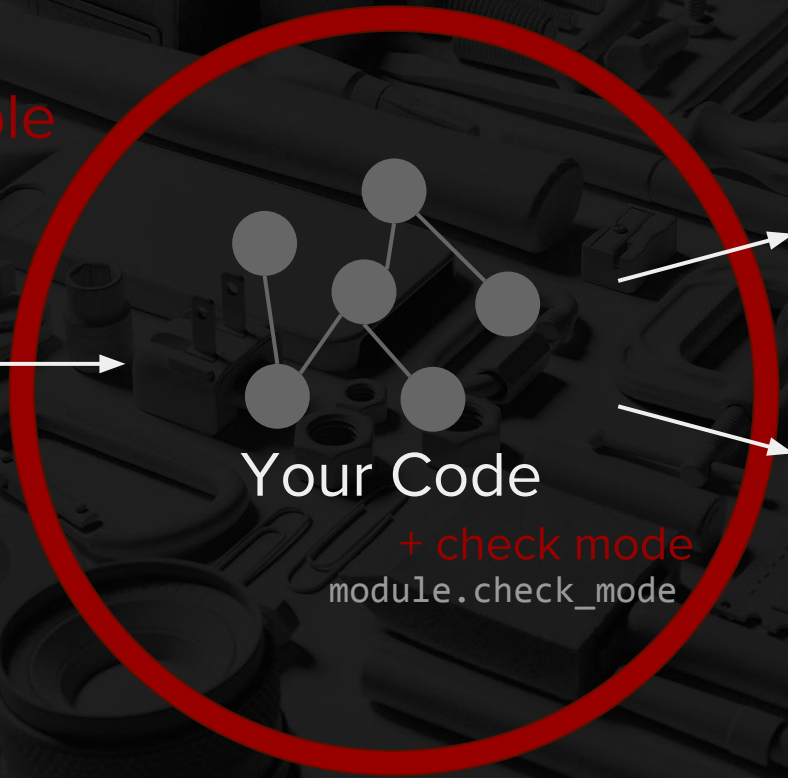
```
14
15 def main():
16     module = setup_module()
17
18     if module.check_mode:
19         module.exit_json(changed=False, msg='This is check mode')
20
21     if module.params['name'] == 'Oscar':
22         module.fail_json(msg="Don't use the name 'Oscar'")
23
24     module.exit_json(changed=False, msg=module.params['name'])
25
```

```
>>> ansible-playbook -v basic_playbook.yml
```

```
PLAY [localhost] ***************************************************************

TASK [Gathering Facts] *********************************************************
ok: [localhost]

TASK [Test that my module works] ***********************************************
fatal: [localhost]: FAILED! => {"changed": false, "failed": true, "msg": "Don't use the name 'Oscar'"}

PLAY RECAP *********************************************************************
localhost                  : ok=1    changed=0    unreachable=0    failed=1
```

Ansible Execution

You tell ansible to run playbook → Ansible tells module to run → Module runs → Output is produced



Testing

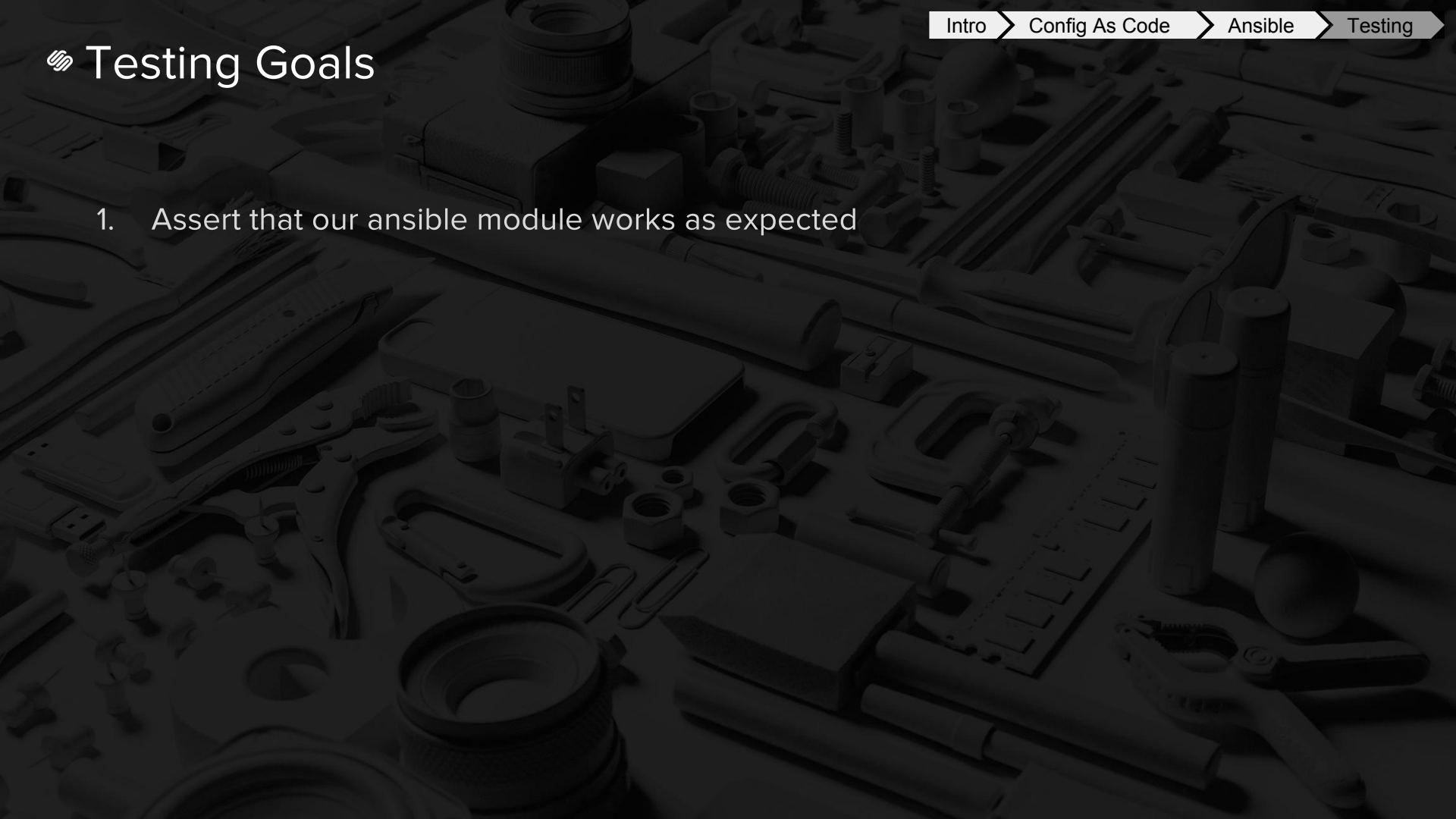
[x] Config as code

[x] Ansible

[?] Testing

Testing Goals

1. Assert that our ansible module works as expected



Testing Goals

1. Assert that our ansible module works as expected
2. Keep tests isolated from real systems

Testing Goals

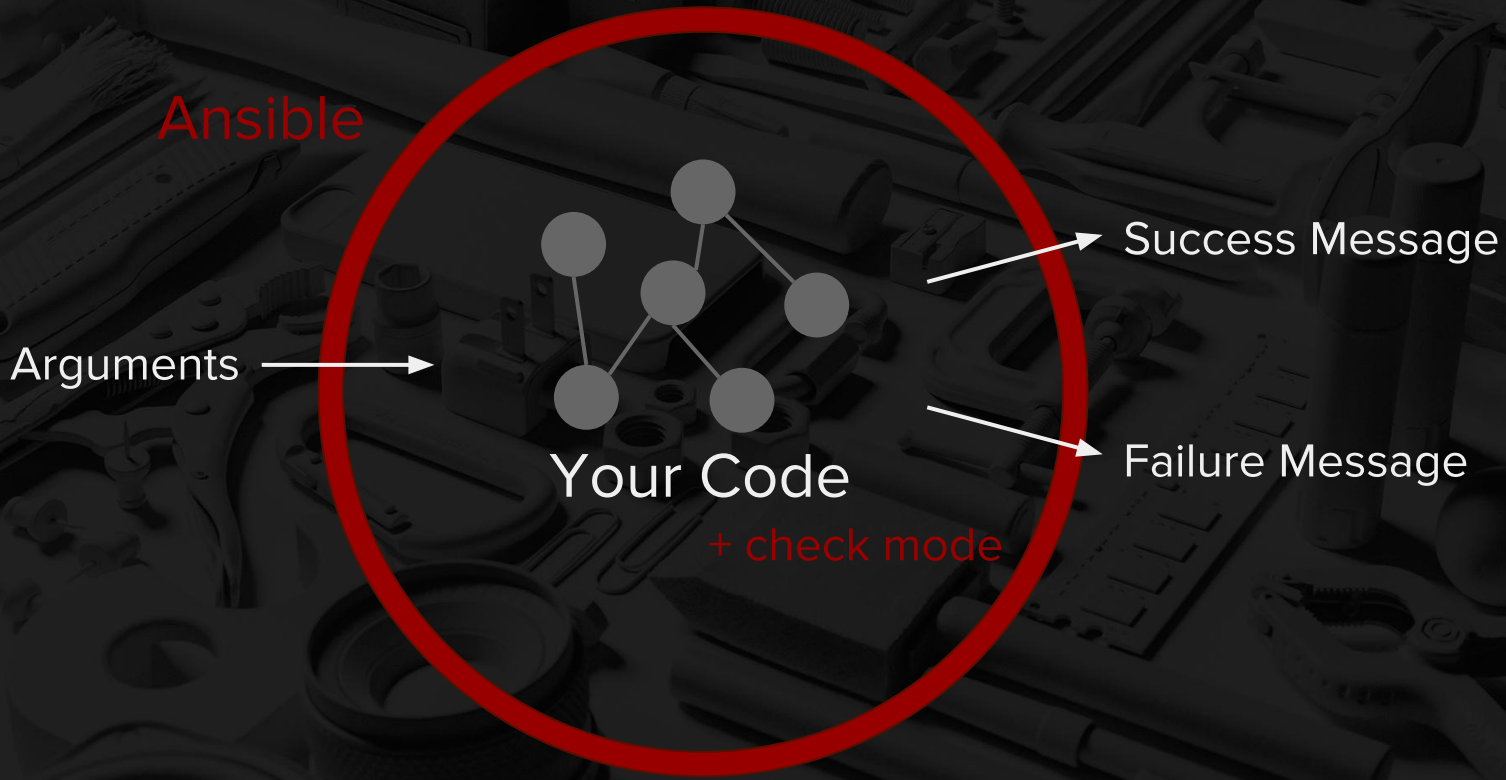
1. Assert that our ansible module works as expected
2. Keep tests isolated from real systems
3. Make testing both Python 2 and Python 3 possible

Testing Goals

1. Assert that our ansible module works as expected
2. Keep tests isolated from real systems
3. Make testing both Python 2 and Python 3 possible

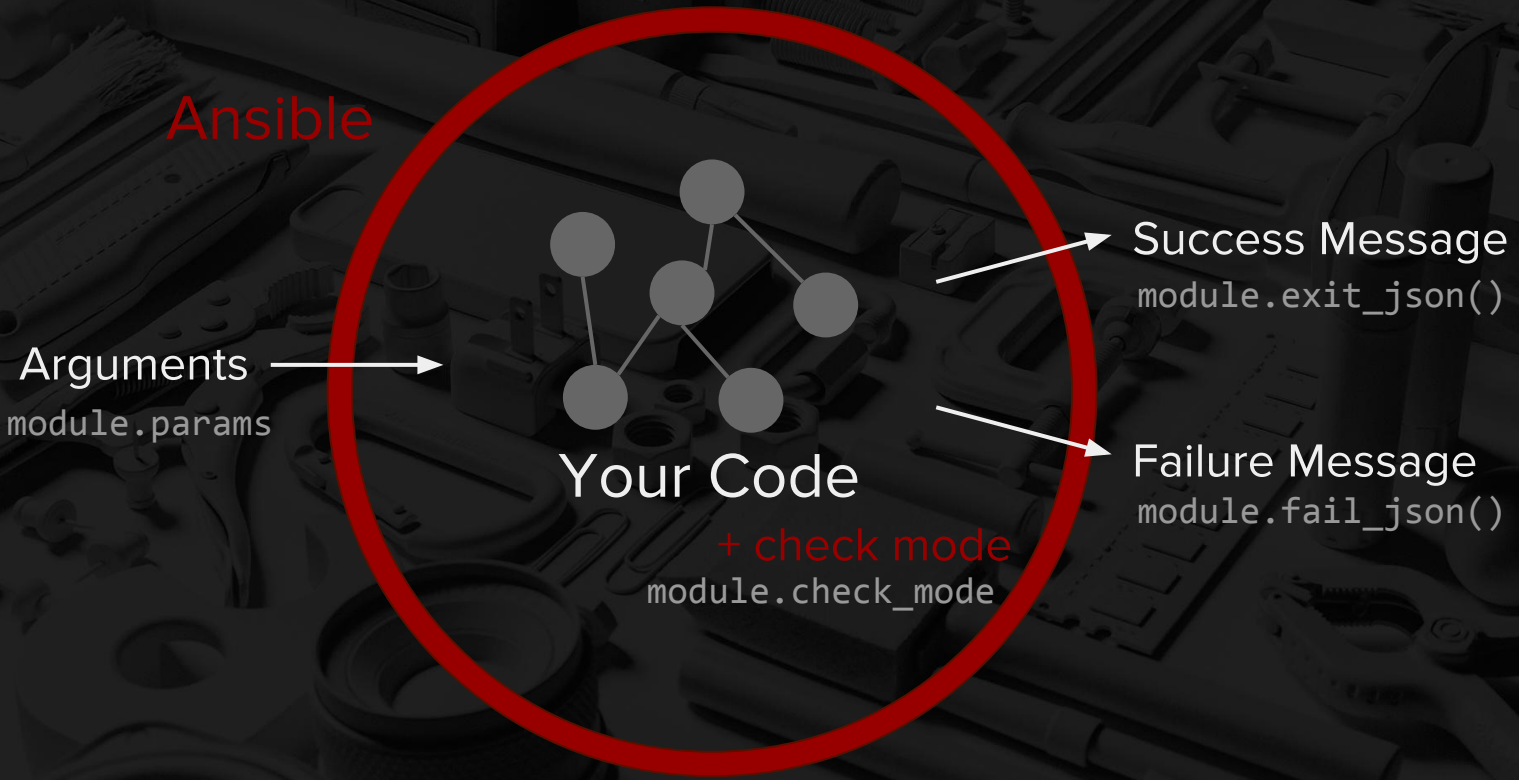
Testing Implementation

1. Assert that our ansible module works as expected



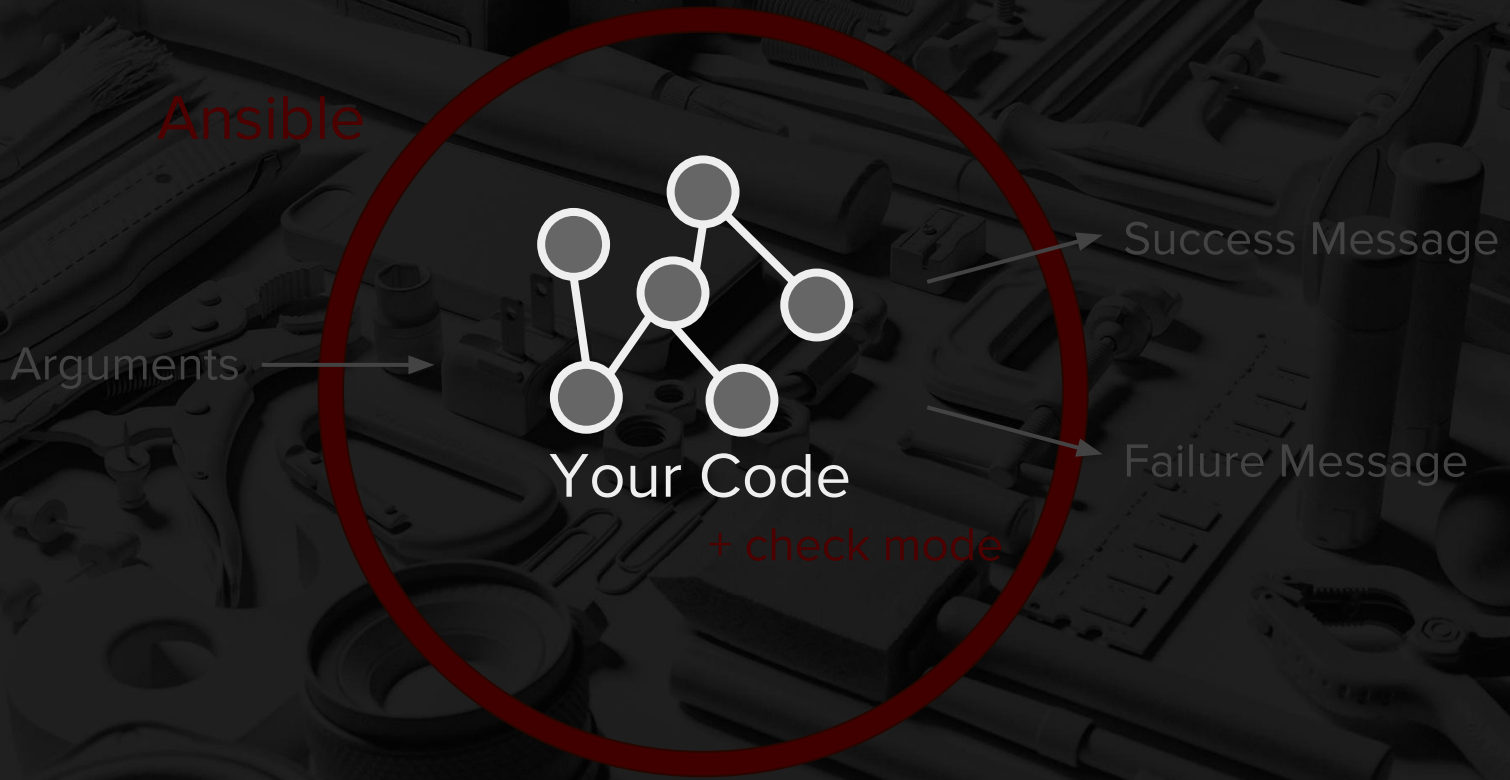
Testing Implementation

1. Assert that our ansible module works as expected



Testing Implementation

1. Assert that our ansible module works as expected



Testing Implementation

Mock fail_json to log its output: *

```
44 @pytest.fixture(scope='session')
45 def mockmodule():
46
47     class MockAnsibleModule(object):
48         def fail_json(self, msg):
49             logging.basicConfig()
50             logging.error(msg)
51
52     return MockAnsibleModule()
```

Use the ‘caplog’ fixture from the pytest-catchlog plugin:

```
16 def test_init_errors_on_single_quotes_in_name(caplog, mockmodule, cursor):
17     role = {'name': "bad'_name"}
18     _ = pgb._BaseConfigurer(mockmodule, cursor, role)
19     assert caplog.records[0].msg == 'Role "bad\'_name" contains an unsupported character: \' or "'
```

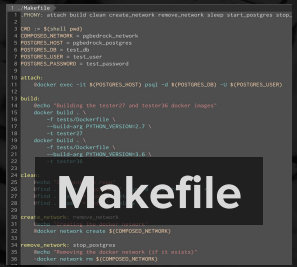
* Other mocked methods not shown

Testing Implementation

1. Assert that our ansible module works as expected
2. Keep tests isolated from real systems
3. Make testing both Python 2 and Python 3 possible

Testing Implementation

1. Assert that our ansible module works as expected
2. Keep tests isolated from real systems
3. Make testing both Python 2 and Python 3 possible



Testing Implementation

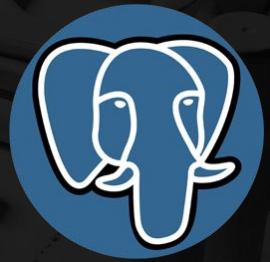
1. Assert that our ansible module works as expected
2. Keep tests isolated from real systems
3. Make testing both Python 2 and Python 3 possible

Why don’t you just use tox??

Testing Implementation

1. Assert that our ansible module works as expected
2. Keep tests isolated from real systems
3. Make testing both Python 2 and Python 3 possible

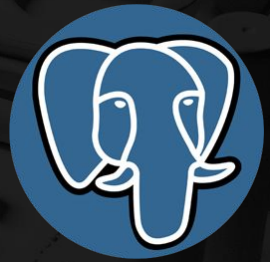
#1) Spin up a new postgres database



Testing Implementation

1. Assert that our ansible module works as expected
2. Keep tests isolated from real systems
3. Make testing both Python 2 and Python 3 possible

#1) Spin up a new postgres database



#2) Speed

Basic setup

```
>>> make build
>>> make test
```

Basic setup - make build

```
13 build:
14     @echo "Building the tester27 and tester36 docker images"
15     docker build . \
16         -f tests/Dockerfile \
17         --build-arg PYTHON_VERSION=2.7 \
18         -t tester27
19     docker build . \
20         -f tests/Dockerfile \
21         --build-arg PYTHON_VERSION=3.6 \
22         -t tester36
```

Basic setup - make build

```
1  ARG PYTHON_VERSION
2  FROM python:$PYTHON_VERSION
3
4  LABEL maintainer=zmarine@squarespace.com
5  VOLUME /opt
6  WORKDIR /opt
7
8  COPY requirements.txt /tmp
9  COPY tests/test-requirements.txt /tmp
10 RUN pip install --no-cache-dir -r /tmp/test-requirements.txt -r /tmp/requirements.txt
11
12 CMD ["python", "-m", "pytest", "/opt/tests"]
```

Basic setup - make build

```
1 ARG PYTHON_VERSION
2 FROM python:$PYTHON_VERSION
3
4 LABEL maintainer=zmarine@squarespace.com
5 VOLUME /opt
6 WORKDIR /opt
7
8 COPY requirements.txt /tmp
9 COPY tests/test-requirements.txt /tmp
10 RUN pip install --no-cache-dir -r /tmp/test-requirements.txt -r /tmp/requirements.txt
11
12 CMD ["python", "-m", "pytest", "/opt/tests"]
```

Using Python 2 (or 3)

Basic setup - make build

```
1  ARG PYTHON_VERSION
2  FROM python:$PYTHON_VERSION
3
4  LABEL maintainer=zmarine@squarespace.com
5  VOLUME /opt
6  WORKDIR /opt
7
8  COPY requirements.txt /tmp
9  COPY tests/test-requirements.txt /tmp
10 RUN pip install --no-cache-dir -r /tmp/test-requirements.txt -r /tmp/requirements.txt
11
12 CMD ["python", "-m", "pytest", "/opt/tests"]
```

Using Python 2 (or 3)

Install our requirements

Basic setup - make build

```
1  ARG PYTHON_VERSION
2  FROM python:$PYTHON_VERSION
3
4  LABEL maintainer=zmarine@squarespace.com
5  VOLUME /opt
6  WORKDIR /opt
7
8  COPY requirements.txt /tmp
9  COPY tests/test-requirements.txt /tmp
10 RUN pip install --no-cache-dir -r /tmp/test-requirements.txt -r /tmp/requirements.txt
11
12 CMD ["python", "-m", "pytest", "/opt/tests"]
```

Using Python 2 (or 3)

Install our requirements

And run pytest

Basic setup - make test

```
74 test: clean create_network start_postgres sleep test27 test36 stop_postgres
```

Additional Testing Nuances

- Want each test to be independent
- Tests require setup

Additional Testing Nuances

- Want each test to be independent → Roll back changes after each test
- Tests require setup → Use pytest fixtures for setup

Additional Testing Nuances

Yield fixtures separates test setup and test teardown:

```
29 @pytest.fixture(scope='function')
30 def cursor():
31     host = 'pgbedrock_postgres' if os.environ.get('WITHIN_DOCKER_FLAG') else 'localhost'
32     db_config = {'host': host,
33                  'port': 5432,
34                  'user': 'test_user',
35                  'password': 'test_password',
36                  'dbname': 'test_db'}
37     db_connection = psycopg2.connect(**db_config)
38     cursor = db_connection.cursor(cursor_factory=psycopg2.extras.DictCursor)
39     yield cursor
40     db_connection.rollback()
41     db_connection.close()
```

Additional Testing Nuances

Yield fixtures separates test setup and test teardown:

```
29 @pytest.fixture(scope='function')
30 def cursor():
31     host = 'pgbedrock_postgres' if os.environ.get('WITHIN_DOCKER_FLAG') else 'localhost'
32     db_config = {'host': host,
33                  'port': 5432,
34                  'user': 'test_user',
35                  'password': 'test_password',
36                  'dbname': 'test_db'}
37     db_connection = psycopg2.connect(**db_config)
38     cursor = db_connection.cursor(cursor_factory=psycopg2.extras.DictCursor)
39     yield cursor
40     db_connection.rollback()
41     db_connection.close()
```

Setup

Teardown

Additional Testing Nuances

Yield fixtures separates test setup and test teardown:

```
29 @pytest.fixture(scope='function')
30 def cursor():
31     host = 'pgbedrock_postgres' if os.environ.get('WITHIN_DOCKER_FLAG') else 'localhost'
32     db_config = {'host': host,
33                  'port': 5432,
34                  'user': 'test_user',
35                  'password': 'test_password',
36                  'dbname': 'test_db'}
37     db_connection = psycopg2.connect(**db_config)
38     cursor = db_connection.cursor(cursor_factory=psycopg2.extras.DictCursor)
39     yield cursor
40     db_connection.rollback()
41     db_connection.close()
```

Setup

Teardown

Additional Testing Nuances

Yield fixtures separates test setup and test teardown:

```
29 @pytest.fixture(scope='function')
30 def cursor():
31     host = 'pgbedrock_postgres' if os.environ.get('WITHIN_DOCKER_FLAG') else 'localhost'
32     db_config = {'host': host,
33                  'port': 5432,
34                  'user': 'test_user',
35                  'password': 'test_password',
36                  'dbname': 'test_db'}
37     db_connection = psycopg2.connect(**db_config)
38     cursor = db_connection.cursor(cursor_factory=psycopg2.extras.DictCursor)
39     yield cursor
40     db_connection.rollback()
41     db_connection.close()
```

Setup

Teardown

Agenda

1. Config As Code
2. Ansible
3. Testing

Questions?

Zach Marine, Squarespace


zmarine@squarespace.com
zcmarine
@zcmarine