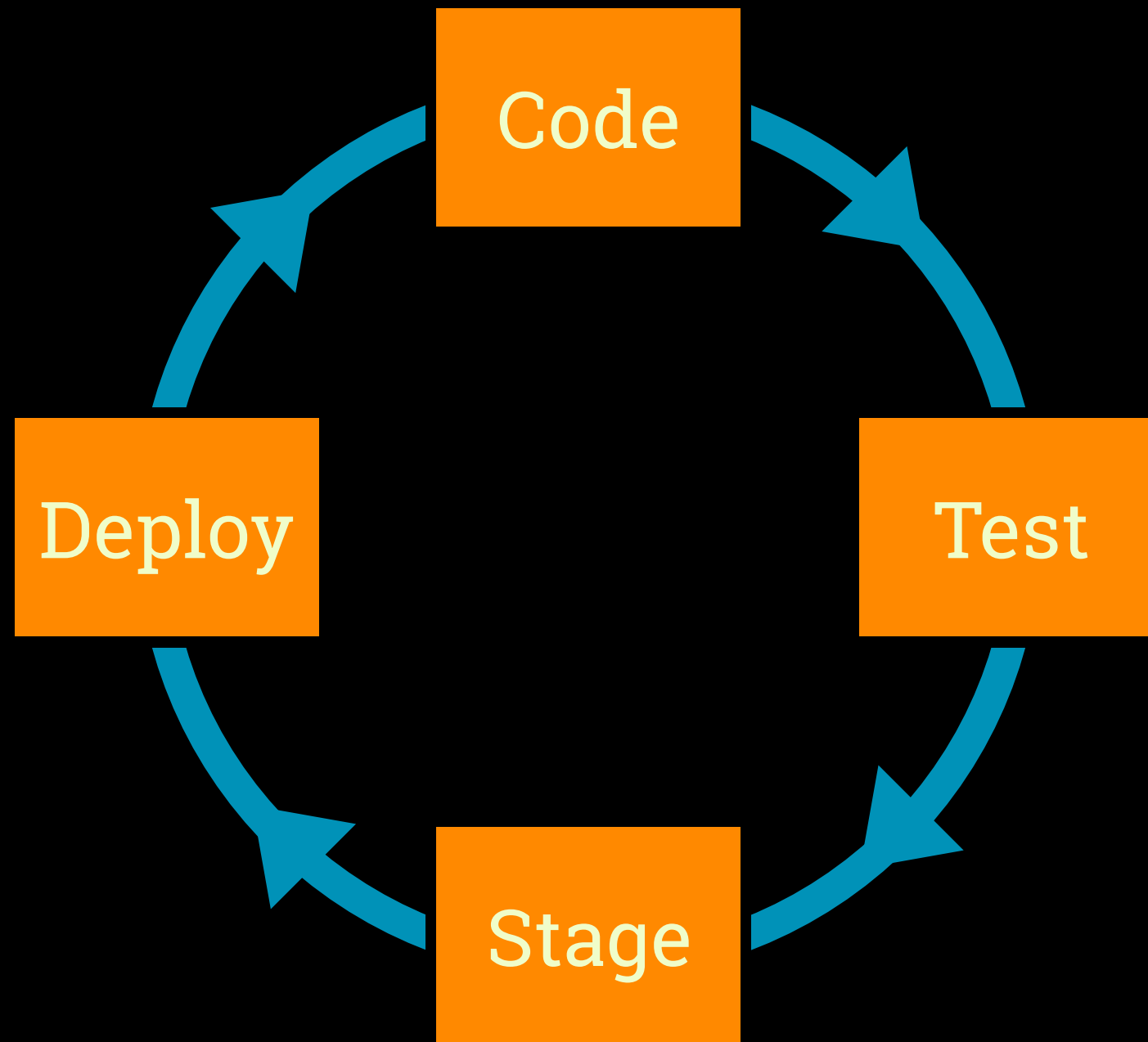


Using Policyfiles

YoloVer as a Workflow

tl;dr

Workflow



New Words

- chef Command – Part of ChefDK, like knife
- Policyfile – Source code for a policy
- Policy Name – Replaces role, web/db/etc
- Policy Group – Replaces env, SN1/2/etc
- Compiled Policy – Snapshot of a policy

Smile for the Camera

```
{
  "revision_id": "288ed244f8db8bfff3caf58147e840bbe079f76e0",
  "name": "demo_policy",
  "run_list": ["recipe[demo::default]"],
  "cookbook_locks": {
    "demo": {
      "version": "1.0.0",
      "identifier": "f04cc40faf628253fe7d9566d66a1733fb1afbe9",
      "dotted_decimal_identifier": "67630690.23226298.2550585",
      "source": "cookbooks/demo",
      "cache_key": null,
      "scm_info": null,
      "source_options": {"path": "cookbooks/demo"}
    }
  }
}
```

... And Push It Over There

- `chef install`
- Compile and download
- `chef push`
- Upload to Chef Server



Policyfile.rb

Policyfile.rb

```
name "kafka"
```

```
default_source :community
```

```
run_list "base", "kafka::server"
```


Run Lola Run

```
run_list "foo"
```

```
run_list ["foo", "bar"]
```

```
# Same as above
```

```
run_list "foo", "bar"
```

Marathon Man

```
named_run_list :deploy, "app::deploy"
```

```
$ chef-client -n deploy
```

```
# Doesn't work anymore
```

```
$ chef-client -o "recipe[app::deploy]"
```

Alice's Restaurant

```
cookbook "monit"
```

```
cookbook "monit", "1.0.0"
```

```
cookbook "monit", "~> 1.0"
```

```
cookbook "monit", path: "../chef-monit"
```

```
cookbook "monit", github: "poise/monit"
```

```
cookbook "monit", git: "https://..."
```

The Usual Suspects

```
default_source :community
```

```
default_source :supermarket, "http://..."
```

```
default_source :chef_server, "http://..."
```

```
default_source :chef_repo, "../"
```

Lone Star

```
default_source :... do |s|  
  s.preferred_source_for "monit", "..."  
end
```

M*A*S*H

```
default["myapp"]["root"] = "/app"
```

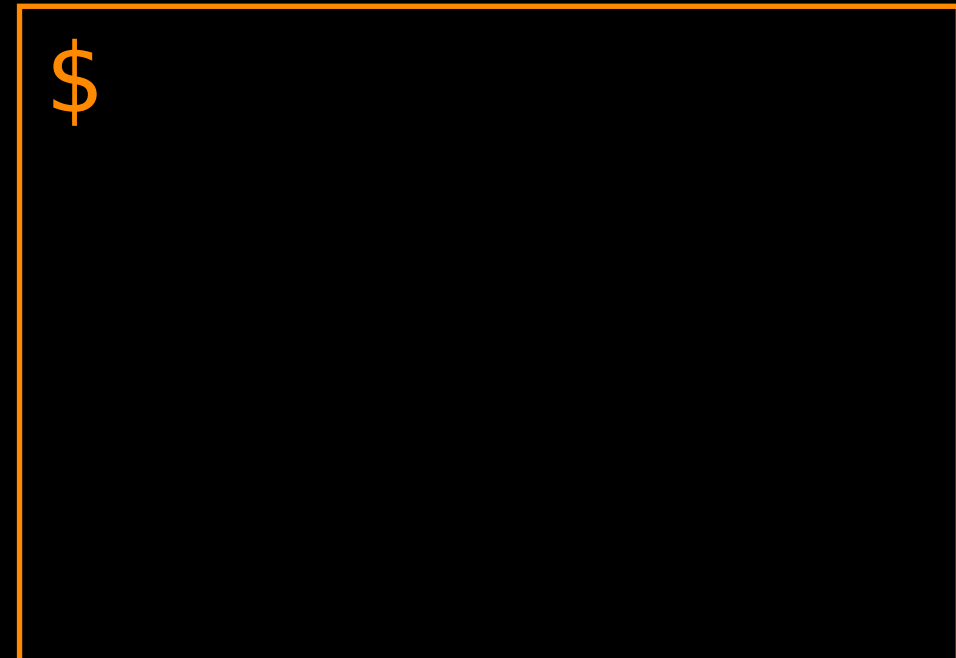
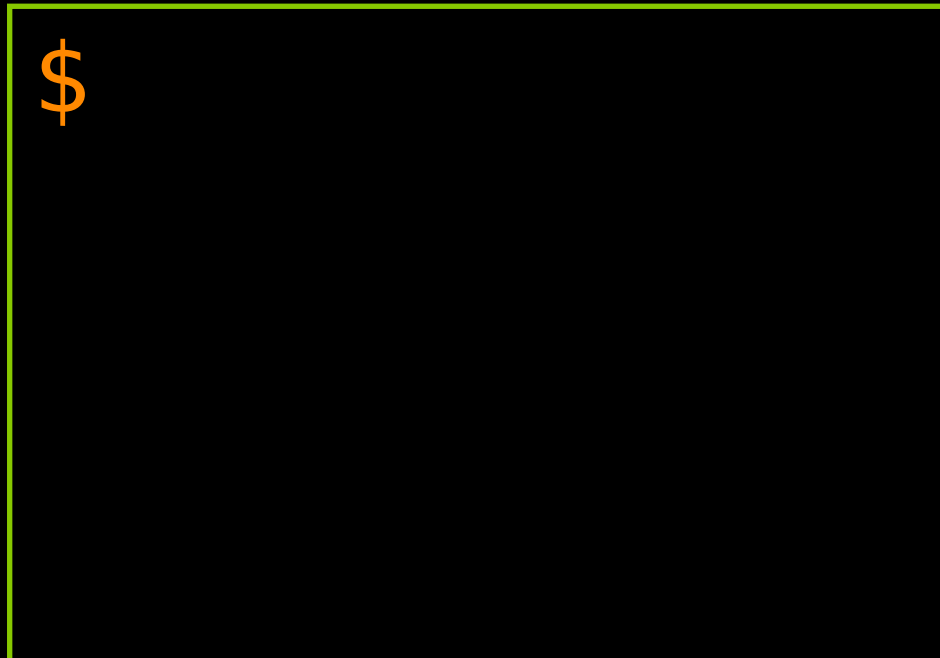
```
override["monit"]["port"] = 8080
```

Bleeding Edge

The Talking Stick

- There is only one pipeline
- Multiple releases must be mutex-d

The Talking Stick



S1

S2

S3

The Talking Stick

```
$ chef install  
Policy compiled  
$ chef push s1
```

```
$
```

S1

S2

S3

The Talking Stick

```
$ chef install  
Policy compiled  
$ chef push s1  
$ chef push s2
```

```
$
```

S1

S2

S3

The Talking Stick

```
$ chef install  
Policy compiled  
$ chef push s1  
$ chef push s2
```

```
$ chef install  
Policy compiled  
$ chef push s1  
$ chef push s2  
$ chef push s3
```

S1

S2

S3

The Talking Stick

```
$ chef install  
Policy compiled  
$ chef push s1  
$ chef push s2  
$ chef push s3
```

```
$ chef install  
Policy compiled  
$ chef push s1  
$ chef push s2  
$ chef push s3
```

S1

S2

S3

The Talking Stick

- For now: make sure no one else is deploying
- Future: Deploy UI may help lock clusters
- Situational awareness is required

Environment Attributes

- default/override in the policy act like role attires
- No specific support for group-level values

Nesting

```
# Policyfile.rb
```

```
default["SN1"]["app"]["dbhost"] = "..."
```

```
default["SN2"]["app"]["dbhost"] = "..."
```

```
# recipes/default.rb
```

```
node[node.policy_group]["app"]["dbhost"]
```


Hoisting

```
# Policyfile.rb
```

```
default["SN1"]["app"]["dbhost"] = "..."
```

```
default["SN2"]["app"]["dbhost"] = "..."
```

```
# attributes/default.rb
```

```
default.update(default[node.policy_group])
```

Data Bag

```
# attributes/default.rb
item = DataBagItem.load("env",
                        node.policy_group)
default.update(item.raw_data)
```

Base Role

```
# base.rb
```

```
default_source :community
```

```
run_list "base"
```

```
default["key"] = "value"
```

```
# Policyfile.rb
```

```
instance_eval(IO.read("base.rb"))
```

```
name "web"
```

```
run_list << "web"
```

Partial Updates

- `chef update` can only regenerate a policy
- Planned for the future
- Use `chef diff` for safety

Danger Zone

- LANA, LANAAAAAAAAAAAAA
- New, fresh, well-tested
- Growing quickly

Trouble Spots

- Single pipeline
- Group-level attributes
- Shared base configuration
- Partial updates
- Young tooling

Order's Up

Release Process

- Update cookbook version
- Make a git tag
- Maybe push to (internal?) Supermarket
- Push to Chef Server organization
- Update Chef environments in order
- Repeat for each changed cookbook

SemVer FTW!

- Allows looser environment restrictions ($\sim > 1.0$)
- Better control for other teams
- More semantic info for UI/tools
- Warm and fuzzy feelings

More Like LameVer

- Mental overhead to establish "compatible"
- Ensure all dependencies are released in order
- Must have linearized x.y.z versions
- No concurrent git branches or pre-releases

I Don't Wanna

YoloVer

- Policyfile(s) linked directly to git
- Use a cookbooks/ folder if desired
- `chef install/update` to take a new snapshot
- `chef push` to deploy to stages

Example Repo

```
$ ls .  
cookbooks/ policies/
```

```
$ ls cookbooks  
bb-kafka/ bb-graphite/ bb-collected/
```

```
$ ls policies/  
db.rb frontend.rb
```

policies/db.rb

```
name "db"
```

```
default_source :community
```

```
default_source :chef_repo, ".."
```

```
cookbook "clojure", github: ".../clojure"
```

```
run_list "clojure", "git", "bb-kafka"
```

More?

Local development with the policyfile-zero provisioner and Test Kitchen.

replace env cookbook pattern

Mise en Place

Chef Push

Discuss the chef push command and how to use it.

What baseline are we starting from?

A: Name-level familiarity but teaching from scratch.

Household Staff

Jenkins, CI, maybe ChefSpec and InSpec?

Testing Tools

- ChefSpec – Unit testing
- Test Kitchen – Integration testing
- InSpec – Integration/acceptance testing

Unit Testing

- Test a single unit of logic (recipe, resource)
- Mock/stub at unit boundaries
- Ensure isolation between tests
- Move fast and break things

Unit Testing

- Edge cases
- Complex inputs
- Regression checks

Integration Testing

- Test the integration of multiple units
- Check side effects
- Slower, but closer to real life

Integration Testing

- Real world use cases
- Performance tests (sometimes)

ChefSpec

- RSpec 4lyfe
- Really runs Chef
- Provider stubs, `step_into` specific providers
- Stub helpers for `data_bags`, `search`, `commands`

Test Kitchen

- Create a fresh virtual machine
- Install and run Chef
- Run verification tests via InSpec
- Uses Policyfile via policyfile_zero plugin

RSpec

RSpec

```
describe "a thing" do
  it "is a thing" do
    expect(1).to eq 1
  end
end
```

Describe

```
describe MyClass do  
  # ...  
end
```

```
describe "label" do  
  # ...  
end
```

Context

```
describe "a thing" do
  context "with A" do
    # ...
  end
  context "with B" do
    # ...
  end
end
```

It

```
it "works" do  
  expect(val).to ...  
end
```

```
it { expect(val).to ... }
```

All Together

Point out that I have no "it" labels here

```
describe "addition" do
  context "with 1" do
    it { expect(1+1).to eq 2 }
  end
  context "with 2" do
    it { expect(2+2).to eq 4 }
  end
end
```

Running

- Put that in `spec/thing_spec.rb`
- `$ chef exec rspec`

Expectations

`expect(value).to matcher`

eq Matcher

Point out function-y nature just this once.

```
expect(value).to eq(other)
```

```
expect(1).to eq 1
```

```
expect("a").to eq "a"
```

to_not Mode

Quick diversion, not a matcher but a
matcher mode

```
expect(1).to_not eq 2
```

```
expect("a").to_not eq "b"
```

be Matcher

`expect(1).to be > 0`

`expect(-1).to be < 0`

`expect(0).to be == 0`

Same as eq matcher

Boolean Matchers

`expect(nil).to be_nil`

`expect(true).to be true`

`expect(false).to be false`

`expect(1).to be_truthy`

`expect(nil).to be_falsey`

String Matchers

`expect("abc").to include "a"`

`expect("abc").to match /a.c/`

`expect("abc").to start_with "a"`

`expect("abc").to end_with "c"`

Class Matchers

`expect("a").to be_a String`

`expect(1).to be_an Integer`

Error Matchers

Point out the block here

```
expect { myfunc() }.to raise_error
```

```
...to raise_error ArgumentError
```

```
...to raise_error /message/
```


Subject

```
describe "a thing" do
  subject { 1 }
  it do
    expect(subject).to eq 1
  end
end
```

Is Expected

```
describe "a thing" do
  subject { 1 }
  it do
    is_expected.to eq 1
  end
end
```

Should (Okay)

```
describe "a thing" do
  subject { 1 }
  it do
    should eq 1
  end
end
```

Should (Not Okay)

```
describe "a thing" do
  subject { 1 }
  it do
    subject.should eq 1
  end
end
```

Let

```
describe "a thing" do
  let(:myval) { 1 }
  it do
    expect(1+myval).to eq 2
  end
end
```

Complex Let

```
describe "a thing" do
  subject { val + 1 }
  context "with 1" do
    let(:val) { 1 }
    it { is_expected.to eq 2 }
  end
  context "with 2" do
    let(:val) { 2 }
    it { is_expected.to eq 3 }
  end
end
```

Before

```
describe "a thing" do
  before do
    puts "BEFORE!"
  end
  it { ... }
end
```

Before Timing

```
before(:each) { ... }
```

```
before(:all) { ... }
```


Other Hooks

before { ... }

Mention timing works on all of them

after { ... }

around { |ex| ...; ex.run; ... }

Spec Helper

```
# spec/spec_helper.rb  
require "..."
```

```
RSpec.configure do |config|  
  # ...  
end
```

Spec Helper

```
# spec/thing_spec.rb  
require "spec_helper"  
  
describe ...
```

Lab?

Mocks

- Helpers for faking out methods
- Avoid "dangerous" call (IO.write, shell_out)
- Call without depending on internals (unit isolation)

Mocks

```
allow(I0).to receive(:read)
```

```
expect(I0).to receive(:read)
```

Argument Matchers

```
... receive(:read).with("/foo")
```

```
... receive(:read).with(match /foo.*/)
```

Return Value

```
... receive(:read).and_return("lorem")
```

```
... receive(:read) { |path| "lorem" }
```


Doubles

```
double()
```

```
double(method: "1")
```

```
double("label", method: "1")
```

Doubles

```
double(x: 1).x == 1
```

```
fake = double  
expect(fake).to receive(:x) { 1 }
```

Default mode is allow

Example

```
describe MyLib do
  let(:node) do
    double(name: "test.example",
            chef_environment: "prod")
  end
  subject { MyLib.myfunc(node) }
  it { is_expected.to eq ... }
end
```

Example

```
describe "myfunc" do
  subject { myfunc("foo", "abc") }
  it do
    expect(I0).to receive(:write) \
      .with("/foo", "abc")
  subject
end
end
```

Explain expect mock before subject

Example

```
describe "myotherfunc" do
  subject { myotherfunc("bar") }
  before do
    allow(IO).to receive(:read) \
      .with("/bar").and_return("abc")
  end
  it { is_expected.to eq "abc" }
end
```

ChefSpec

ChefSpec

```
# spec/spec_helper.rb
require "chefspec"
require "chefspec/policyfile"

# Policyfile.rb
name "cookbookname"
run_list name
default_source :community
cookbook name, path: "."
```

Runner

```
subject do
  ChefSpec::SoloRunner.converge('name')
end
```


Basics

```
describe "myrecipe" do
  subject do
    ChefSpec::SoloRunner.converge("myrecipe")
  end

  it { is_expected.to ... }
end
```

Matchers

```
...to ACTION_RESOURCE(NAME)
```

```
...to install_package("nginx")
```

```
...to create_user("myapp")
```

With

■ `with(prop: val, prop: val)`

```
install_package("nginx").with(version: "1.2")  
create_user("myapp").with(group: "nogroup")
```

Team Players

How this workflow operates with a team.