Continuous code delivery and integration with SaltStack

SaltConf Seth House <shouse@saltstack.com>

2014-01-29

Continuous integration

- Continuously merge changes with a mainline branch.

Continuous integration

- Continuously merge changes with a mainline branch.
- Incoming changes can be tested against other incoming changes.

Continuous integration

- Continuously merge changes with a mainline branch.
- Incoming changes can be tested against other incoming changes.
- Catch errors fast and early.

Continuous integration

- Continuously merge changes with a mainline branch.
- Incoming changes can be tested against other incoming changes.
- Catch errors fast and early.

Requires: a build server; automating builds.

Continuous delivery

- Deliver tested code to an environment for users.

Continuous delivery

- Deliver tested code to an environment for users.
- Code review, QA, staging, perhaps production.

Continuous delivery

- Deliver tested code to an environment for users.
- Code review, QA, staging, perhaps production.

Requires: automating deploying to an environment.

Continuous deployment

- Deliver tested code to production as soon as it is ready.

Continuous deployment

- Deliver tested code to production as soon as it is ready.
- Automation is key; deployment should take minutes.

Continuous deployment

- Deliver tested code to production as soon as it is ready.
- Automation is key; deployment should take minutes.
- Logs and metrics determine effectiveness

Continuous deployment

- Deliver tested code to production as soon as it is ready.
- Automation is key; deployment should take minutes.
- Logs and metrics determine effectiveness
  - Error rate, conversion rate, user registration rate, response time, traffic, etc.

Continuous deployment

- Deliver tested code to production as soon as it is ready.
- Automation is key; deployment should take minutes.
- Logs and metrics determine effectiveness
  - Error rate, conversion rate, user registration rate, response time, traffic, etc.

Requires: continuous integration, continuous delivery.

Salt does not dictate your infrastructure

It depends.

> *"Don't get set into one form, adapt it and build your own, and let it grow, be like water. Empty your mind, be formless, shapeless — like water. Now you put water in a cup, it becomes the cup; You put water into a bottle it becomes the bottle; You put it in a teapot it becomes the teapot. Now water can flow or it can crash. Be water, my friend."*
>
> *—Bruce Lee*

Build tools, testing tools

- Travis-CI
- Jenkins-CI
- Buildbot

Web hooks

- Notify Salt
- Pass data
- Transfer files

Demo!

- Testrepo

Overview

- Send events from minions to the master.
- Trigger operations throughout your infrastructure.

Firing events

```
salt-call event.fire_master \
    '{"foo": "Foo!", "bar": "Bar!"}' \
    'myapp/myevent/somevalue'
```

Watching the event bus

```
python /path/to/salt/tests/eventlisten.py
```

Listening to events with the Salt Reactor

/etc/salt/master.d/react_myapp.conf:

```
reactor:
  - 'myapp/myevent/somevalue':
    - /srv/salt/react_myapp.sls
```

Listening to events with the Salt Reactor

/etc/salt/master.d/react_myapp.conf:

```
reactor:
  - 'myapp/myevent/somevalue':
    - /srv/salt/react_myapp.sls
```

/srv/salt/react_myapp.sls:

```
deploy_myapp:
  cmd.state.highstate:
    - tgt: 'web*'
    - arg:
      - 'pillar={{ data|yaml }}'
```

Debugging the Salt Reactor

```
salt-master -l debug
```

Step 1: post_recieve Git hook

```
#!/bin/bash
newrev=$(git rev-parse $2)

salt-call event.fire_master \
    "{\"newrev\": \"$newrev\"}" \
    "myapp/git/push"
```

Step 2: React to the push event

```
/etc/salt/master.d/react_git_push.conf:
    reactor:
      - 'myapp/git/push':
        - /srv/salt/react_git_push.sls
```

Step 2: React to the push event

```
/etc/salt/master.d/react_git_push.conf:
    reactor:
      - 'myapp/git/push':
        - /srv/salt/react_git_push.sls

/srv/salt/react_git_push.sls:
    test_myapp:
      cmd.state.sls:
        - tgt: 'buildserver'
        - arg:
          - run_tests
          - 'pillar={{ data|yaml }}'
```

Step 3: Run the test suite

/srv/salt/run_tests.sls:

```
run_tests:
  cmd:
    - run
    - name: python -m unittest tests
    - cwd: /path/to/testrepo

deploy_stage:
  module:
    - wait
    - name: event.fire_master
    - data:
        newrev: {{ salt['pillar.get']('data:newrev'
    - tag: myapp/tests/pass
    - watch:
      - cmd: run_tests
```

Step 4: React to the test event

```
/etc/salt/master.d/react_tests_pass.conf:
    reactor:
      - 'myapp/tests/pass':
        - /srv/salt/react_tests_pass.sls
```

Step 4: React to the test event

/etc/salt/master.d/react_tests_pass.conf:

```
reactor:
  - 'myapp/tests/pass':
    - /srv/salt/react_tests_pass.sls
```

/srv/salt/react_tests_pass.sls:

```
deploy_myapp:
  cmd.state.sls:
    - tgt: 'web*'
    - arg:
      - deploy_myapp
      - 'pillar={{ data|yaml }}'
```

Step 5: Deploy the new code to stage

/srv/salt/deploy_myapp.sls:

```
myapp:
  git:
    - latest
    - name: git@github.com/myorg/myapp
    - target: /var/www/myapp
    - rev: {{ salt['pillar.get']('data:newrev') }}
    - watch_in:
      - service: apache

apache:
  service:
    - running
    - name: httpd
```