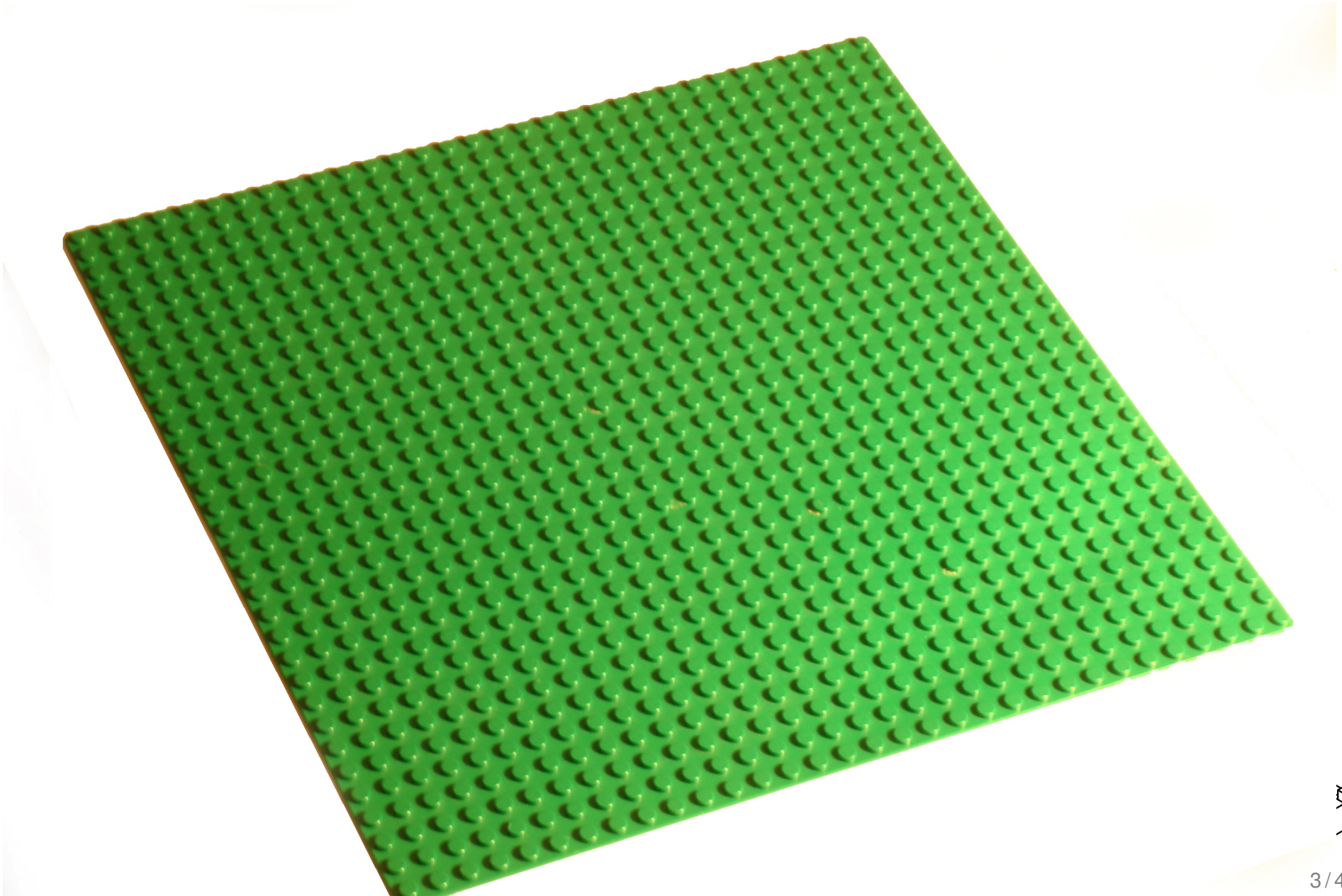
Real-time cloud management with Salt

Unlocked.io Seth House <shouse@saltstack.com>

2013-06-14

Building blocks



Building blocks

- Everything we have is built on earlier generations
- Building blocks of history
  - Agriculture
  - Cities
  - Governments
  - Mathematics
  - Manufacturing
  - Language

What are our building blocks?

- Computers Systems
  - Data centers are the modern platform of invention
  - Large scale computing drives virtually all high level human interactions
- Data centers themselves are becoming the building blocks

Building blocks and complexity

- As systems grow they become more complex
- Increased complexity makes using larger building blocks impossible
- Building blocks used today are simple
  - Easy to use
  - Easy to understand
  - Yet powerful

Success comes from simplicity

- Viable building blocks are fundamentally simple
- Things that are in themselves simple to understand and use
- Concepts that can cross barriers and cleanly apply to many higher level uses
- YAML is a great example
- Unix is a great example

Salt is aptly named

Salt is aptly named

- The name is the vision
- Salt is in nature, a building block
- Salt is in cooking, a building block
- Salt is in life, a building block
- Yes, this is where the logo came from
- Yes, this is why it is Salt STACK

Flow + state = EVERYTHING

- Core idea behind Salt
- Flow = Access and control
- State = The functional state of the system
- Between them All things are controlled

Building on state and flow

- Basics
  - Remote execution built on Flow
  - Config Management built on State
- Ramping Up
  - Monitoring build on Flow + State
  - Cloud management is Flow + State
  - Auto healing is Flow + Flow
  - Orchestration is Flow + State
  - Distributed Deployment is Flow + State

Building blocks for infra

- Salt **has** “Configuration Management”
- Salt **has** “Remote Execution”

Building remote ex

- Async Message Bus
- Micro publish commands
- Minion Side Logic
- Return to arbitrary location

Building states

- All about data
- Low level data structure calls state functions
- High level data structure defines state calls
- Languages are now arbitrary

Building cloud

- Remote Ex controls hypers
- Remote ex commands hypers
- All runs on Live data

Building monitoring

- Remote Ex build on system libs
- Data gathering is already there!
- States used to detect the state of a system
- States can declare that the state of a system is out of bounds
- Fires back to the event bus

Conclusion

Conclusion

- Salt is comprised of many building blocks
- The underlying components are available, but they do not need to be interfaced with
- Salt is made to morph into the needs of the infra
- Made to make small infrastructures easy, and large infrastructures powerful

Master

- `salt-master -d`
- Open two ports (pub/sub & reply channel)

Minions

- `salt-minion -d`
- Connect to the master
- No open ports required
- Listens for pubs from the master

`/etc/salt/minion`:

```
#master: salt
```

Execution modules

- Contain all the functionality

Execution example

```
salt 'web-*' network.interfaces
```

State modules

- Wrap execution modules
- Before-check
- `test=true`
- Call out to execution modules
- After-check

State module example

top.sls:

```
base:
  'web-*':
    - httpd
```

httpd.sls:

```
httpd:
  pkg:
    - installed
```

Communication

- ZeroMQ
- msgpack

pub/sub

- Asynchronous
- Minions determine targeting match
- Minions do all the work

- Live data; peer interface
- Recent data; Salt mine
- Historical data; returners

Peer

```
/etc/salt/master:
    peer:
      lb-.*:
        - network.interfaces
```

- Be mindful of data security
- Communication still goes through the master

Peer example

Configuring `haproxy.cfg`:

```
{% for server,ip in
    salt['publish.publish'](
        'web*',
        'network.interfaces',
        ['eth0']).items() %}
server {{ server }} {{ ip[0] }}:80 check
{% endfor %}
```

Salt Mine

```
/etc/salt/{master,minion}:
    mine_functions:
        network.interfaces: [eth0]

    mine_interval: 60
```

- New in Salt v0.15
- Either master or minion config
- Be mindful of data security

Salt Mine example

Configuring `haproxy.cfg`:

```
{% for server,ip in
    salt['mine.get'](
        'web-*',
        'network.interfaces',
        ['eth0']).items() %}
server {{ server }} {{ ip[0] }}:80 check
{% endfor %}
```

Returners

```
/etc/salt/{master,minion}:
    redis.db: 0
    redis.host: myredis
    redis.port: 6379
```

- Minions write directly
- Can be read into Pillar via `ext_pillar`

Returner full-circle example

Collect the data:

```
salt 'web-*' network.interfaces eth0 \
    --return redis_return
```

Fetch the data via a custom `ext_pillar` module.

Use the data:

```
{% for server,ip in
    salt['pillar.get']('web.ip_addrs', {}).items()
server {{ server }} {{ ip[0] }}:80 check
{% endfor %}
```

Fire events

```
salt 'lb-*' event.fire_master \
    refresh_pool loadbalancer
```

Watch for events (manually)

- Some assembly required
- `salt/tests/eventlisten.py`
- Coming soon to `salt-api`

Reactor (react to events)

/etc/salt/master:

```
reactor:
  - loadbalancer:
    - /src/reactor/refresh_pool.sls
```

/src/reactor/refresh_pool.sls:

```
{% if data['type'] == 'refresh_pool' %}
highstate_run:
  cmd.state.highstate:
    - tgt: lb-*
{% endif %}
```

Add events

`/etc/salt/{master,minion}` (or pillar):

```
schedule:
  highstate:
    function: state.highstate
    minutes: 60
```

Stats gathering

```
schedule:
  uptime:
    function: status.uptime
    seconds: 60
    returner: redis
  meminfo:
    function: status.meminfo
    minutes: 5
    returner: redis
```

Salt `v.0.next`

Monitoring states

- Configure inline with existing states
- Individual components are in place
- Needed: glue
- Needed: alerting

Data resolution

- Time-series data
- Thin and/or summarize older and older data
- Free with some returners