Automated Application Management

with SaltStack

Arnold Bechtoldt
Cologne, 08.06.2016
Arnold Bechtoldt
Systems Engineer / Consultant @ inovex

› Software-Defined Datacenters
› Continuous Integration/Delivery
› DevOps

WE’RE HIRING!
Assets & Links

github.com/bechtoldt/talk-salt-orchestration

inovex.de

arnoldbechtoldt.com
Orchestration in General
Highway To Heaven

Cloud

1. Asset-Management, Accounting
2. Bare-Metal Deployment
3. Configuration Management/Containerization
4. Orchestration
5. Auto-Scaling + Elasticity
Orchestration...

- ... uses *remote execution* to distribute system commands
- ... should be more than just executing commands (most of the tools end here)
- ... should be able to respect internal & external dependencies/relations
- ... should be deterministic and *fast*
- ... must [RFC2119] be *easy* to understand/learn (YMMV)
A not-so-cool Example

#!/bin/bash -e
for node in $(mco find --np -C roles::node)
do
    echo "restarting ssh on $node"
    mco service --np ssdhd restart -l $node >/dev/null
    sleep 2
    mco rpc service status service=sshd -l $node -j | \
    jgrep data.status=running -s data.status >/dev/null
    echo "ssh is up on $node"
done
For instance, if you use a single-purpose deployment tool [...] to trigger some legacy configuration management, that might be called “orchestrating X with Y”. In reality, it’s usually just replacing a non-scaling [...] server solution X with a stand-alone implementation of Y.

http://www.ansible.com/blog/orchestration-you-keep-using-that-word
SaltStack Orchestration Basics
SaltStack Orchestration

SaltStack implements the same techniques that other tools do:

- send commands to servers
- send commands to servers that have different operating systems
- send commands only to a subset of servers
- run command A, then B, then C

Often this isn’t enough for us!
SaltStack Orchestration

What we really want to have:

› scale across thousands of servers

› easy configuration (less software programming)

› an interface to implement internal/external relationships (pre/post tasks)

› fully automated workflows/procedures (no manual interaction)
SaltStack Orchestration

What we really want to have:

› scale across thousands of servers
  
  SOA – Message Bus Architecture

› easy configuration (less software programming)
  
  YAML – JSON – Python DSL

› an interface to implement internal/external relationships (pre/post tasks)
  
  State – Remote Execution – Pillar – Returner – Beacon – Cloud (~ 600 modules)

› fully automated workflows/procedures (no manual interaction)
  
  Event System – Reactor – Runner – APIs
Salt Peer-Publish: Realtime Execution
Salt Peer-Publish: Realtime Execution

- Peer-2-Peer like
- Privilege management on master-side
- Synchronous & blocking

Use Cases:
- Gathering realtime metrics of other systems (minions)
- Automatically-triggered remote execution
Salt Mine: Dynamic Runtime Data Cache
Salt Mine: Dynamic Runtime Data Cache

- Caching arbitrary data/metrics
- Replaces slow/expensive peer public calls

Use Cases:
- Gathering backend addresses for LB config
- Gathering SSH public keys (host/user) of other other hosts
- Asynchronous collecting of metrics for monitoring
Salt Reactor: Event-Driven System Automation

<table>
<thead>
<tr>
<th>salt/key</th>
<th>salt/job/new</th>
<th>node/stonith/new</th>
<th>update/application</th>
</tr>
</thead>
<tbody>
<tr>
<td>salt/auth</td>
<td>salt/job/return</td>
<td>scale/out</td>
<td>loadbalancer/config/update</td>
</tr>
<tr>
<td>salt/minion/start</td>
<td>salt/presence/present</td>
<td>monitoring/hosts/new</td>
<td>firewall/config/update</td>
</tr>
<tr>
<td>salt/minion/stop</td>
<td>salt/presence/change</td>
<td>monitoring/hosts/remove</td>
<td>coffee/new</td>
</tr>
</tbody>
</table>
Event-Driven System Automation
Salt Reactor: Event-Driven System Automation

› Automated cfg mgmt and remote execution across several hosts
› „Real Orchestration“
› Asynchronous
› Synchronous/blocking alternative: Orchestration Runner

Use Cases:
› Automated loadbalancer configuration updates
› Automated host bootstrapping and software provisioning
Event Structure

salt/job/20151104191820394966/new{
  "_stamp": "2015-11-04T18:18:20.512126",
  "arg": [
    "orchestration.bootstrap",
    {
      "__kwarg__": true,
      "test": false
    }
  ],
  "fun": "state.sls",
  "jid": "20151104191820394966",
  "minions": [
    "mw42"
  ],
  "tgt": "mw42",
  "tgt_type": "glob",
  "user": "root"
}

custom/minion/haste_server_started{
  "_stamp": "2015-11-04T18:33:54.650568",
  "cmd": "_minion_event",
  "data": {
    "custom": {
      "onchanges": [],
      "foo": "bar",
      "num": 42,
    },
    "sfun": "wait"
  },
  "id": "mw2",
  "pretag": null,
  "tag": "custom/minion/haste_server_started"
}
# SaltStack Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>A resource should look like {…}</td>
</tr>
<tr>
<td>Pillar</td>
<td>Database/CMDB (Files, RDBMS, NoSQL)</td>
</tr>
<tr>
<td>Execution</td>
<td>Execute command X on a server. Used by other Salt components.</td>
</tr>
<tr>
<td>Returner</td>
<td>Log store of job results (DB, Elasticsearch, Syslog, Monitoring)</td>
</tr>
<tr>
<td>Beacon</td>
<td>Special event triggers (inotify, load, procs, network/user activity)</td>
</tr>
<tr>
<td>Reactor</td>
<td>Reacts to events by triggering subsequent procedures (execution modules)</td>
</tr>
<tr>
<td>Cloud</td>
<td>IaaS provisioning (EC2, OpenStack, Digital Ocean, Linode, GCE, VMware)</td>
</tr>
</tbody>
</table>

…
Hands on: Demo
Demo Architecture

fe (HAProxy) → mw (NodeJS) → db (Redis)

fe (HAProxy) → mw (NodeJS) → db (Redis)

fe (HAProxy) → mw (NodeJS) → db (Redis)
Demo Concept

Salt Master/Reactor

1. Tells MW hosts to install MW

MW hosts

2. Install Node.JS (MW)

3. Report back to Master

4. Tells FE hosts to install FE

5. Install Haproxy (FE)

6. Report back to Master

6. Tells MW hosts to deploy App

7. Deploy App

8. Send Notification Mail

9. Report back to Master

10. Tells FE to reconfigure FE

11. Add/Remove HAProxy backends

12. Report back to Master

FE hosts
Demo Concept

Salt Master/Reactor

1. Tells MW hosts to install MW
4. Tells FE hosts to install FE
6. Tells MW hosts to deploy App
10. Tells FE to reconfigure FE

MW hosts

2. Install Node.JS (MW)
3. Report back to Master
7. Deploy App
8. Send Notification Mail
9. Report back to Master

FE hosts

5. Install Haproxy (FE)
6. Report back to Master
11. Add/Remove HAProxy backends
12. Report back to Master
Assets & Links

github.com/bechtoldt/talk-salt-orchestration  
sh.arbe.io/cloud-provision  
youtu.be/9MzeK4u4pkM (demo)
Arnold Bechtoldt
inovex GmbH

abechtoldt@inovex.de

CC BY-NC-ND inovex.de +ArnoldBechtoldtGER
github.com/bechtoldt arbe.io youtube.com/inovexGmbH