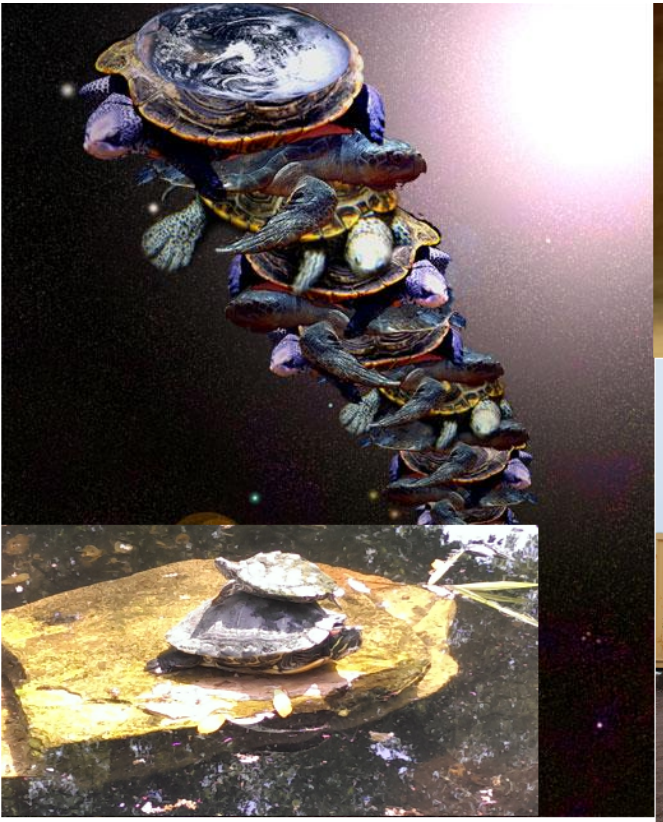
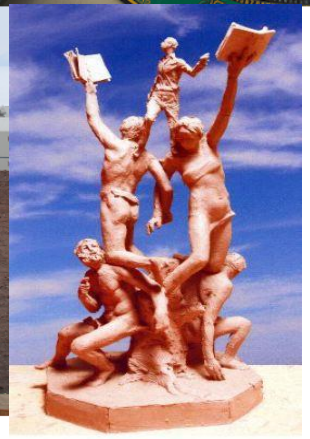
Roll your own AWS RDS, SQS, or other SaaS

Poor Man’s Service Orchestration with Chef

Alex Corley - Level 7 Cloud Wizard - AVAST! Antivirus

https://github.com/anthroprose/chefconf-saas-talk





Do not spin the wheel

Stand on shoulders of giants

Share the work with all

IaaS -> PaaS -> SaaS -> ?

Perspective

* Providing some form of SaaS using **Chef** & AWS for about 4 years.

* Started with hand built globally distributed workflow system using VPN and a centralized metadata store with workers for ecosystem state and business logic. Now leverage a suite of *aaS to provide faster, tightly integrated, holistic deployments.

* Budget/Headcount/Time/SLA constraints always push small teams UP stack to leverage other services for specialties.

* 100% AWS due to SDN and Model Defined Infrastructure requirements, along with secondary featureset

* RackSpace now has production OpenStack with a sufficient featureset.

Focus

* IT providing SaaS for for internal departments
* Small Startup providing SaaS for a niche tool/service
* This is not for thousands of nodes
* Sometimes existing SaaS is just too expensive (high volume, low margin)

This is from the trenches of small business and lean startups where sometimes everything runs on a single machine, or with limited in-house technical resources. The goal is to use standard **Chef** patterns and **Community Cookbooks** to do service orchestration in easy, repeatable ways, along with all the other things that need to be done like backups, restores, monitoring, reporting and auditing.

SOA Frontends

* Control your input flows with a simple quality gate
* Enforces formatting
* Enforces namespacing
* Enforces ACL
* Capacity Planning!

* Allows you to calculate per request charges
* Provide audit/security reports for auth failures
* Rate limit / Protect shared backend resources from misbehaving code without backoffs
* THIS MUST BE ASYNC! - You do not want to solve concurrency issues when they happen

* Dogfooding - Use your own API from the start, then CRM/Webforms can do Instant Signups

Architecture Design

* Really watch the actual scale & use-case requirements, I've designed multiple global fault tolerant failover systems, just to have them end up running all in a single region across availability zones.

* Over scale the networking before CPU/Storage, it is easy to add nodes/storage, but not with an over saturated pipe in times of duress

* Be vigilant of complex systems interactions in regards to security

    AWS + IAM Instance Profiles + OHAI EC2 Plugin == All your base are belong to us

    The IAM API Credentials for your Database machines will be accessible from the Web front-ends and as such a web attack vector could manipulate resource level permissions such as EBS unmount/remount for secure/encrypted services.

Responsive Scaling

* **chef_handler!**

    You should know when a bad provisioning happens, having 1/100th of the infrastructure erroring out behind a load balancer is not a fun thing to diagnose or even allow to happen.

* Periodically run integrations tests on nodes to determine health

    Use an individual test **recipe** and **chef-client -o** to adjust **run_lists** without changing the norm

* Clean up your nodes!

    Cleanup worker that watches node removals

    During intense scaling operations, you may end up with a conflicting clients/nodes

    ASG + SQS + **PyChef**

* Most other solutions use *YOU* as the oracle (i.e. **knife-ec2**, **chef-metal**, vagrant, fog/boto, etc..)

Chef Server & Node Attributes

* Use **Node Attributes** and **Chef Server** for a state machine

* **Cross-Node LWRP / Resource Idempotency** and **Multiple Convergences** to provide logic gates

* **Chef Server** / **Node Save** should have a Limited Post Size - Only store state!

* This is not meant to replace ZooKeeper, or etcD (No processing/pushing, but still coordinating)

* Around 350-400 lines of **Chef** to turn the **MySQL Community Cookbook** into a Basic MVP SaaS Offering

    * * Plus a little python (but this could be Ruby too)

Resources

* How many **Chef Resources** changed in each run?

* Put your business logic in **Resources/LWRPs**, this lets you control actions, rollbacks, idempotency and reporting

* You can always refactor the **Resource/LWRP** to use a different metadata source like etcd/zookeeper

* Write your own granular **chef_handler** report handler that takes into account proposed changes.

    You can whitelist planned changes with an admin panel and alert if deltas occur without intention

Tenancy

Different products have different types of Tenancy, what are the deployment and implementation challenges of managing a central cluster with namespacing and elasticity or a fleet of single stack snowflake installs?

* Multi Tenant - **Chef Environment** per Product Version
    - What does ‘Production’ really mean when you have multiple versions live at once?
    - Lets you easily identify & interact with Clusters for backups/migrations/upgrades

* Single Tenant - **Chef Environment** per Customer
    - Use **Roles** to categorize services if there are multiple machines per tenant (Master/Slave)

* Use internal **Chef Tags** for *EVERYTHING* in addition to any IaaS tagging
    - Reporting, Auditing, Resource Cleanup and Billing

Backup / Restore

Everything is considered ephemeral and “Chaos Monkey” compat from design of system, any machine may terminate for any reason at any time

* For Low I/O, group common services on the same mount point

* For High I/O, breakout mountpoints and leverage RAID0/PIOPS

* Use a Filesystem that supports freezing/thawing for snapshotting like XFS

* Snapshot according to your risk management and economics, you can do it every 10 minutes, once an hour or once a day depending on your needs. - A cron’d Custom **Chef Runlist** does this

* On machine boot, check for snapshots and attach at mountpoint instead of creating new volumes

Templates / Support Access

* Always put a header in your **Chef Templates**

    “* This file is managed by Chef and individual hand changes will not be saved ***”**

* Alert on Template changes via **chef_handler**

* Sometimes things *need* to be done by hand, instead of **Notifying** a **Service Resource** fire a change request to Operations/IT, this could also make API calls to IaaS/PaaS.

* Sometimes Support needs CLI access, give them an API Instead

* Use correct permission sets when templating service files (g+rw), support has a different user that can touch /etc/init.d but not have service level config/filesystem write ability

Workflow

* **Chefspec**

* **Serverspec** + **Busser**

* Have a cleanup **Recipe** for removing idempotency files of snowflake information, also stop or chkconfig services off depending on your order of operations of pre-chef and post-chef deployment actions. After you run this, burn your boot image to be used in responsive scaling actions.

* CloudFormation+ASG+S3+SQS+**Chef**+Gluecode will pretty much do anything you want

* Future Dev Chain - **Chef DK + Chef-Metal** + **Test-Kitchen** FTW!

Fancy Stuff

* ITIL / IPAM - (Information Technology Infrastructure Library / IP Address management)

* Change Management / Detection

* Security Audit Event Reporting

* PCI/HIPAA/FedRAMP Audit/Report Compliance

* You can do all these with a combination of **chef_handler** and **Chef Search**, go tell the CTO you can report all the things and they will love you forever

MySQL

* Single Tenant or Multi-Tenant
    - Service can handle namespacing and authentication
    - You still want an SOA/REST Frontend for reporting and extra auth/flow control layers

* innodb_file_per_table
    - Helps with billing calculations based on disk usage
    - Percona has tools that help with precise calculations that are not file based
    - Also useful for capacity planning

* Go ahead and offer some caching options (innodb plugin or memcache logic in SOA/REST)

* Super simple to do point in time restores with snapshots, you will need code for bin-log rollbacks

RabbitMQ

* **Chef Search** to get Cluster Member's Hostnames

    Template /etc/host for FQDN resolution

* **Chef Search** to issue HA commands on new queues.

    This has to be run on every new queue, make sure to alert on errors when adding customers

    You could namespace this like ha.*, but I like treating each customer individually, customer.*

* Use a combination of the SOA/REST frontend and rabbitmqctl to ensure resource balancing and quotas, otherwise a heavily used queue can affect others

ElasticSearch

* The ElasticSearch cookbook and AWS plugin are AWESOME! (although multicast is cool too)

* Most of your effort will be spent automating removing/expiring indexes and making sure that your customer pools are distributed among clusters depending on your deployment topology

* Put power users with higher SLAs/Performance plans on their own clusters

* Pool low end users on shared resources

* Upgrades are easy, just replace a node at a time until the entire cluster is upgraded

Notes

* This example used a plain text data bag, ***DO NOT STORE SECRETS IN NODE ATTRIBUTES***

* IAM Instance Profiles with customer specific S3 permissions download unique data bag encryption keys per customer or role so that no node can un-encrypt other node's attributes

* 100% of this code was greenfield for demo, but yes, I have done and do things like this every day in production, small hacks run the world

* All the gluecode for things like backups and auto-snapshot restore are using Python+Boto, but could easily be anything that can interface with an API (Ruby, Bash, C, JavaScript, etc…)

* As much as people dislike the state of community cookbooks, gluecode is much better than writing every product cookbook and orchestration layer from scratch to build a business from the ground up

Appendix

* This talk & demo

  https://github.com/anthroprose/chefconf-saas-talk

* Python+Boto+AWS EBS Snapshotting and Restore

  https://github.com/Jumpshot/aws-minions

* Python+Boto+SQS+PyChef Node/Client Cleanup (Also does CloudWatch Alerts too!)

  https://github.com/Jumpshot/aws-sqs-alert

* Single Stack Operations - Time Series Metrics, Log Aggregation, Alerting and Dashboarding

  https://github.com/Jumpshot/operations

  Use this to consume **chef-client** logs, have **chef_handler** post to elasticsearch directly

Future

* Chef-Metal - Multiple convergence handling

    https://github.com/opscode/chef-metal

* Enterprise Chef “Push Jobs” - Out of Band Actions

    http://docs.opscode.com/pushy.html

* Poise - Reusable HWRP framework

    https://github.com/poise/poise

* Citadel - SafeNode

    https://github.com/balanced-cookbooks/balanced-citadel