

Instant Operations

From 0 to Production in a Single Day

<https://github.com/anthroprose/devopsdays2014-instantops>

<https://github.com/Jumpshot/operations>

Alex Corley - @anthroprose - github: anthroprose

Who am I?

- * Cultural Anthropologist
- * Farmer (Chickens, Rabbits, Orchard, Greenhouse)
- * Hopeless Romantic (Inverse Conway's Law for Government via Automation and Technology)
- * Father
- * Technology Generalist
- * Problem Solver
- * Enjoy Learning
- * Excited about the Future



Do not spin the wheel

Stand on shoulders of giants

Share the work with all

Assumptions

- * Leveraging Public Cloud
- * Configuration Management
- * Model Defined Infrastructure
- * Business Logic Already Exists
- * Priorities of Maximizing Scale while Lowering Cost
- * Fully modern Operations Department

Requirements Gathering

- * Disk/Network Operation Frequency & Size
- * Size of Working Set in Memory and Disk
- * CPU, Memory or Disk (Application Bottleneck)
- * Concurrency Matters as much as Speed/#s
- * Compliance/Security Concerns
- * What metrics convey business logic?

Requirements Gathering

- * 1,000 Transactions per Second via WebApp
- * 1TB of Log Storage
- * 10Mb/s Log Throughput
- * 2,000 Individual Time Series Metrics
- * 1 Hour Business Recovery Time
- * SOA, Global, Scalable, Fault Tolerant

Stack

Public Cloud - AWS

VPC, ELB, ASG, SQS, S3, DDB, ElastiCache

Multi-AZ deploy

Self-Healing, Snapshot Restores, Service Clusters

Time Series: diamond+statsd+graphite

Logs: beaver+logstash+elasticsearch+kibana

Best Practices

- * IAM Users
- * IAM Instance Profiles
- * Cloudformation+ASG
- * Chef & Citadel
- * Chef + Public Keys instead of AWS Private Key
- * Tagging - AWS & Chef
- * Collect more data than you think you should

Networking

- * Cloudformation - Software Defined Networking
- * One template for network topology
- * VPC all the things!
- * NAT/Bastion is great, but what about network saturation?
- * EIPs are free with ASG, use them for AWS throughput
- * Route53 with a low TTL + ELBs allow you to swing traffic between clusters for testing or emergencies

Networking

```
"Production": {  
    "VpcCidr": "10.0.0.0/16",  
    "PublicSubnet1Cidr": "10.0.0.0/24", "PublicSubnet2Cidr": "10.0.1.0/24", "PublicSubnet3Cidr": "10.0.2.0/24",  
    "AWSSubnet1Cidr": "10.0.3.0/24", "AWSSubnet2Cidr": "10.0.4.0/24", "AWSSubnet3Cidr": "10.0.5.0/24",  
    "SemiPrivSub1Cidr": "10.0.6.0/24", "SemiPrivSub2Cidr": "10.0.7.0/24", "SemiPrivSub3Cidr": "10.0.8.0/24",  
    "PrivateSubnet1Cidr": "10.0.9.0/24", "PrivateSubnet2Cidr": "10.0.10.0/24", "PrivateSubnet3Cidr": "10.0.11.0/24"  
}
```

Public = VPN, Bastion - **AWS** = ElastiCache, ELB - **SemiPriv** = Things that need EIP, **Priv** = Things that can NAT

```
"Staging": { "VpcCidr": "10.1.0.0/16" }  
"us-east-1": { "VpcCidr": "10.2.0.0/16" }  
"us-west-1": { "VpcCidr": "10.3.0.0/16" }  
"us-west-2": { "VpcCidr": "10.4.0.0/16" }
```

AWS Infrastructure

- * ElastiCache (MemcacheD/Redis) is great, and can reside within a subnet on your VPC
- * ELBs reside within a subnet and you can offload all the SSL, 60s timeout - manual change through support
- * Cloudformation for all Services and Resources (Might have to request an increase, do not use describeAll API)

Web Infrastructure

- * Offload all static to S3, then distribute via CloudFront
- * Only 'API' should make it through ELB to EC2
- * uWSGI is great, but hard to do >300TPS without tricks
- * Just go ASYNC - Twisted/Tulip, Node, EventMachine, Elixir/Cowboy, Akka
- * Surprise, Boto does not scale!
- * Pre-bake your AMIs, boot to request speed matters here

Web Infrastructure

- * Insert custom metrics into ASG for whatever your bottleneck is, TPS (Concurrency), CPU, Linux Load Avg
- * You will have to hand tune the scale rules according to the individual webapp and user behavior, no magic settings, just analysis, prediction & adjustment
- * Consider the effect cluster-wide service restarts will have on scale metrics
- * Be extra verbose with the application layer logging metadata, session_id, job_id, widget_id, etc.. you want to trace job input, processing, and reporting

Web Infrastructure

```
"Properties": {  
  "AssociatePublicIpAddress": "true",  
  "UserData": {  
    "/opt/aws/bin/cfn-init -s ", { "Ref": "AWS::StackName" }, " -r WebLaunchConfig --region ",  
    { "Ref": "AWS::Region" }, "\n",  
    "curl -L https://www.opscode.com/chef/install.sh | bash -s -- -v ", { "Ref":  
    "ChefClientVersion" }, "\n",  
    "OPSCODE_USER=`hostname` chef-client -E ", { "Ref": "Environment" }, " -j /etc/chef/init.  
    json >> /var/log/chef-client.log\n"  
  }  
}
```

Processing Fleet

- * Scale based on queue size for maintaining response times, otherwise on app/machine load for simple scaling
- * Save an AMI after known GOLD releases, if the queue ever backs up you will want a large vertically scaled instance to clear the queue

Operations Machine

- * Disk I/O - PIOs, RAID0 or Object Store
- * You have to set Java memory and Redis queue limitations or else a single service will take down the box
- * The # of distinct time series matters more than log lines

Operations Machine

```
"run_list": [  
  "recipe[yum-epel]", "recipe[yum]", "recipe[user]", "recipe[cron]", "recipe[rsyslog::client]",  
  "recipe[git]", "recipe[python]", "recipe[python::pip]", "recipe[graphite]", "recipe[sudo]",  
  "recipe[redisio::install]", "recipe[redisio::enable]", "recipe[java]", "recipe[maven]",  
  "recipe[jenkins::master]", "recipe[postfix]", "recipe[mysql::server]", "recipe[nginx]",  
  "recipe[elasticsearch]", "recipe[statsd]", "recipe[kibana]", "recipe[logstash::server]",  
  "recipe[operations]", "recipe[operations::infrastructure]", "recipe[chatbot]"  
]
```

m3.xlarge - 4 Cores, 15Gb RAM, 1TB with 640 PIOPs (10Mb/s at 16K Blocks)

ElasticSearch gets 8Gb, Logstash 512Mb, Redis 1Gb - Most of the time redis is empty. statsD is the largest resource user with 60% of a CPU continuously while aggregating stats in ram.

Metrics

- * Business Logic - Watch the counter that makes \$\$ (New Users, Purchases)
- * Consider in-app rollup counters then send upstream and batch aggregates
- * Just use the time series agent for doing heartbeat process monitoring and exception reporting - `transformNull(stats_counts.production.*,0)`
- * Anything that looks funny on a single machine, just terminate instance
- * Count and time anything involving a third party (API Requests)

Log Aggregation

- * Pre-format (python-logstash-formatter, nginx log format)
- * Log straight to network from app (redis/logstash) instead of disk+shipping
- * Beaver as a lightweight logstash shipper
- * rsyslog works great by default

Alerting

- * Seyren has been working great
- * OpsGenie for Phone/App (hey, its free - root android & adjust PUSH settings)
- * HipChat / IRC or other Group Announce system
- * Inserting key metrics into Cloudwatch is worth the cost (\$0.5/month)
- * Always perform an action when you get an alert! Adjust levels or Escalate or Fix!

Chef

- * Just write a cookbook for your product, its OK
- * Report all the things - report_handlers
- * Resource cleanup - ASG -> SQS -> Python Worker -> Hosted Chef Cleanup
- * Run distributed tasks like backups `chef-client -o 'recipe[company::backup]'`
- * Run distributed integration tests to determine cluster health, don't integration/functional test a cluster through an ELB, you will miss nodes

How to get Secrets on the Box

- * Private encrypted S3 buckets using Chef+Citadel and IAM Resource level permissions to download files
- * Amazon has the certs, and the audits, *really* how good are you at key management?
- * Encrypted Databags are only good if you use a different key for every node

Continuous Integration

- * Use Jenkins to run Unit Tests & Code Builds
- * Use Jenkins to run Test-Kitchen/Cookbook Tests
- * Use Jenkins to run both Chef & Code in conjunction with additional Functional/Integration Tests via Busser
- * Build notifications in Hipchat within minutes of commit

Continuous Deployment

- * Packer to generate Vagrant boxes on Jenkins success
- * Roll updates across the cluster with knife ssh, fabric, ansible, paramiko
- * Blue/Green - You can adjust DNS Weighting, this allows simple rollbacks
- * Feature Branch to Master via Pull Request, Peer Review & Acceptance
- * Only difference between Development and Production is Vagrant/Cloudformation and a small amount of Metadata

Testing

- * Test-Kitchen (Foodcritic/Rubocop, ChefSpec + ServerSpec + Busser)
- * Leverage Unit Test Framework for Integration/Functional Tests
- * Check all types of IAM Access
- * Test Service Level Metadata Coherence (Am I Production and have a DDB Table Throughput Level of <100?)
- * How Many of this 'Thing' will happen in Production? Count Every Change!

Workflow

- * Get something stable first, the developers need to trust the process, just deploy an empty cookbook to multiple live instances then work from there
- * Needs to be easy and give immediate feedback
- * Packer + Chef-solo - Creates Development Boxes
- * Vagrant + Virtualbox - Developers type 'vagrant up'
- * Cloudformation + Chef - Creates Production Boxes

Workflow

```
{  
  "type": "chef-solo",  
  "cookbook_paths": ["/opt/code/chef-repo/cookbooks"],  
  "run_list": ["python::package", "python::pip", "company", "company::product"],  
  "json": {  
    "aws" : {  
      "key" : "{{user `aws_id`}}",  
      "secret" : "{{user `aws_key`}}",  
      "services" : {  
        "elasticache" : { "nodes" : "127.0.0.1" }  
      }  
    }  
  }  
}
```

Workflow

```
$script = <<SCRIPT
```

```
#!/bin/bash
```

```
echo "export AWS_ACCESS_KEY_ID=\"#{ENV['AWS_ACCESS_KEY_ID']}\" > /opt/company/product-env/bin/profile.  
d/AWS_ACCESS_KEY_ID.sh
```

```
echo "export AWS_SECRET_ACCESS_KEY=\"#{ENV['AWS_SECRET_ACCESS_KEY']}\" > /opt/company/product-  
env/bin/profile.d/AWS_SECRET_ACCESS_KEY.sh
```

```
SCRIPT
```

```
Vagrant.configure("2") do |config|
```

```
  config.vm.synced_folder ".", "/opt/company/product-env/product", create: true, owner: "company-product", group:  
  "company-product", mount_options: ["dmode=777,fmode=777"]
```

```
  config.vm.network "forwarded_port", guest: 80, host: 8000
```

```
  config.vm.provision "shell", inline: $script
```

```
end
```

Story

- * Actually took about a week
- * Constantly refactoring to take advantage of new features
- * Constantly lowering costs through architecture refactoring
- * Async is simple in concept, hard in practice, especially when refactoring a legacy codebase & support

Next Steps

- * PF_RING for userland networking
- * NginX + LUA for UDP network logging
- * Backing Time Series with Riak instead of Disk
- * C/C++ or Erlang statsD Server

References

- * Chef Single Stack Operations - <https://github.com/Jumpshot/operations>
- * Chef Citadel - <https://github.com/balanced-cookbooks/citadel>
- * AWS Minions - <https://github.com/Jumpshot/aws-minions>
- * AWS SQS Alerts - <https://github.com/Jumpshot/aws-sqs-alert>
- * Chef DK - <https://github.com/opscode/chef-dk>