Virtualization and deployment management for the KAT-7 / MeerKAT control and monitoring system

Neilen Marais
Senior Software Engineer
nmarais@ska.ac.za
Introduction

- Radio Telescopes in the Karoo
  - KAT-7
  - MeerKAT
- Deployment of CAM subsystem
  - virtualization
  - automated deployment
  - share some experiences
- Work to improve started end 2011
  - Deployment fraught
  - Hardware failure -> extended downtimes
  - Limited development environments
    - quite different from deployments
Requirements

- Deterministic+repeatable system configuration
- Versioned configuration history
  - quick revision roll-back/forward
- Minimize manual steps in deployment
- Minimize downtime
  - CAM software deployment
  - CAM system hardware failure
- Isolate resource usage on a shared server
- Easily deploy development environments
  - similar to site deployment environments
  - limited development hardware resource
Shape of KAT-7 CAM

- Instrument is distributed
  - Karoo Array Telescope Control Protocol (KATCP)
  - Ethernet as fieldbus
- Telescope is Remote
  - Avoid human generated RFI
  - Control via high speed SANReN fibre (ring) network
- Operational Centre in Cape Town
  - Control room 700 km from site
  - Backup and long-term archiving
- Development
  - Mostly coded in the Python language
Future MeerKAT Array
KAT-7 Receptor Receiver

- Receiver Horn Antenna
- RF Low Noise Amplifier (LNA)
- Stirling Cryo cooler with Ion Pump
- RF Noise diode coupler
- RF Amplifier
KAT-7 Receptor Pedestal

- Antenna positioner control unit
- RF amplifier/attenuator
- RF to optical transducer
- Pedestal chiller
- Building Management
- Weather station
Inside KAT-7 Compute Container

- Antenna positioner control unit
- RF amplifier/attenuator
- RF to optical transducer
- Pedestal chiller
- Optical to RF transducers
- RF Down-conversion and conditioning
- FPGA based Digital Back End
- Data capture server
- Time/Frequency reference
- CAM Servers
- BMS
- Chiller
Control Room
Pieces of the Puzzle

- Server Virtualization
- Automated Deployment
- Deployment Configuration Database
- Combined: Hassle-free, deterministic, reliable deployment
Virtualization

◆ Many Technologies, Many Makers
  ❖ Full virtualization more flexible
  ❖ Containerization more efficient

◆ Blurring of lines

◆ Other considerations
  ❖ Familiarity
  ❖ Licensing
  ❖ Supported Host environments
Specialized Hypervisor distribution based on Debian GNU/Linux

- FOSS licensing: no cost, no hassle

Supports both:

- Containers: OpenVZ
- Full virtualization: KVM

Simple and quick host install

Easy to use web-based management tools

Pre-configured base OS containers
Performance

- CAM uses soft-realtime design
  - Only needs enough aggregate CPU throughput
- Similar aggregate CPU utilization on host before and after virtualization
- IO-bound tests using 10 GbE interface
- Using different virtualization options

Test Machine

SUN FIRE X4150
2x Intel(R) Quad-core Xeon(R) E5450 CPUs
16 GB RAM
Gen 1 Myricom Myri10GE 10GbE
### Performance results

<table>
<thead>
<tr>
<th>Config</th>
<th>Rate (Gb/s)</th>
<th>CPU use (%)</th>
<th>Relative rate (%)</th>
<th>CPU / Gb/s (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>5.49</td>
<td>65.8</td>
<td>100.0</td>
<td>12.0</td>
</tr>
<tr>
<td>Host</td>
<td>4.80</td>
<td>59.2</td>
<td>87.5</td>
<td>12.3</td>
</tr>
<tr>
<td>OVZ exclusive</td>
<td>4.65</td>
<td>61.2</td>
<td>84.7</td>
<td>13.2</td>
</tr>
<tr>
<td>OVZ veth</td>
<td>3.72</td>
<td>21.1</td>
<td>67.8</td>
<td>5.8</td>
</tr>
<tr>
<td>OVZ venet</td>
<td>3.86</td>
<td>20.3</td>
<td>70.4</td>
<td>5.3</td>
</tr>
<tr>
<td>KVM virtio</td>
<td>2.39</td>
<td>60.5</td>
<td>43.6</td>
<td>25.3</td>
</tr>
</tbody>
</table>

- **Baseline: Ubuntu 10.04**
- **CAM uses veth**
  - Most flexible
- **Only 2x1Gb interfaces in production**
Desktop virtualization: Virtualbox

- Simulated system on developer, commissioner workstations
  - Toy-KAT VM
- Variety of workstation OSes
  - Can't take over whole machine for hypervisor
- Virtualbox virtualization host runs on them all
- Also FOSS licensing
- Not production use
  - positive experience
Software Configuration Management

- Automated Deployment Scripts
- Deployment Configuration Database
- Configuration Manager
Preferred a Python based solution

Considered existing 'full stack' systems (e.g. Puppet, Chef, Saltstack)

- Central management server
- Upfront time investment
Our Sw Conf Management System

- Experimented with Fabric
  - Python based SSH automation library
- Script logic is defined in Python
  - Only requires SSH server on nodes to be managed
- Started capturing node configuration details
  - simple text file in INI-format
- Started implementing a Configuration Manager
  - Parsed the configuration database file
- Fabric library functions to deploy tasks to nodes
- Soon got team buy-in
Configuration Database

- Node network configuration
- Node hosting type
- Assigned node resources
  - number of CPUs
  - RAM
  - Unique container / VM ID number
  - Diskpace, etc.
- Other meta information
Disposability and Persistence

- Node containers are treated as disposable
  - Production containers rebuilt at each major release

- Development environments are routinely rebuilt

- Persistent data has to be managed separately

- NAS server, exported via NFS
  - Node NFS mounts configured as part of deployment
  - Potential issues with changing dB schemas
  - Central point for backups
Deployment steps

◆ Configuring Proxmox hypervisor hosts
◆ Provisioning
◆ Configuration
Configuring Proxmox host

- Rarely done
- Install base Proxmox from CD
- Takes about 15 minutes including fab below

```
fab -H root@kat-cam-r410-1,root@kat-cam-r410-2\proxmox.configure_host
```
fab proxmox.create_containers_by_group:karoo_system_nodes,700
Virtual Node Containers Provisioned

**kat-cam-r410-1**
- **proxy**
  - 8 X vCPUs
  - 8 GB RAM
  - 10 GB Local disk
- **monctrl**
  - 2 X vCPUs
  - 8 GB RAM
  - 20 GB Local disk
- **veth0**, **veth1**
- **vmbr0**, **vmbr1**

**kat-cam-r410-2**
- **worker**
  - 8 X vCPUs
  - 8 GB RAM
  - 10 GB Local disk
- **observe**
  - 2 X vCPUs
  - 4 GB RAM
  - 50 GB Local disk
- **veth0**, **veth1**
- **vmbr0**, **vmbr1**

**Telescope Network**

Network connections:
- **eth0**
- **eth1**
Configure Nodes

```
fab kat_deploy.install_nodes_by_group:
karoo_system_nodes,karoocamv7-requirements.txt
```

- Install system packages (apt-get install ...)
- Install python packages (pip install ...)
- Configure NFS mounts
- Check out, build, install and configure packages from internal SVN
- Configure web servers, cron jobs, other OS level services as required
- Set up databases (schemas if needed)
Nodes Configured

**kat-cam-r410-1**
- **proxy**
  - 8 X vCPUs
  - 8 GB RAM
  - 10 GB Local disk
- **monct1**
  - 2 X vCPUs
  - 8 GB RAM
  - 20 GB Local disk

**kat-cam-r410-2**
- **worker**
  - 8 X vCPUs
  - 8 GB RAM
  - 10 GB Local disk
- **observe**
  - 2 X vCPUs
  - 4 GB RAM
  - 50 GB Local disk

**Network Diagram**
- **veth0**, **veth0**, **veth1**, **vmbr0**, **vmbr1**, **eth0**, **eth1**
- **Telescope Network**

The diagram illustrates the network configuration with various nodes and their specifications.
Some Experiences

- Importance of transitioning from a mostly manual to automated deployment step by step.
- Remaining deployment problems are entered into our issue tracker
  - Deployment issues are prioritised for fixing
- Important to make deployment processes idempotent
- Important to make each step reliable
  - Local copies of internet based resources (PyPI and Ubuntu repositories)
  - Unexpected race conditions when things are not done at 'human' speed
- Usefulness of virtualization to allow testing and experiment with the deployment process -- you can just throw away and re-build a virtual node to test from-scratch deployment.
  - Also means we can test deployment, and not just our software
Most frequently experienced advantages are:

- Easy deployment of realistic development/testing environments
  - including virtual networking mirroring actual configurations)
- Ability to quickly switch between software versions by switching containers
- Have largely met our goals

Future work:

- Deployment to fresh containers in the Continuous Integration process
  - Running full integration test suite on these containers
- Automatic daily building of Toy-KAT VMs
- Converting all legacy configuration scripts to the Fabric framework
- Future MeerKAT deployment should be more of the same
  - more complex network configuration
Thank you!

http://www.ska.ac.za

nmarais@ska.ac.za