IRONIC

A CLOUDY APPROACH TO HARDWARE

DEVANANDA VAN DER VEEN

GITHUB.COM/DEVANANDA/TALKS
• Cloud Architect @ IBM / SoftLayer
• HP Cloud
• OpenStack since 2012
• Yoichi single malt
• @devananda
Virtual Machines are abstractions of computers.
VIRTUALIZATION & CLOUD COMPUTING
If sufficiently abstracted, a physical machine behaves like a virtual machine.
Consumption API + hardware abstraction?
nova boot
  --flavor baremetal.high_mem
  --image Ubuntu_Trusty
  --key-name my_keypair
  my_high_mem_server
Every company* had built framework for PXE boot & install OS

* every company that owns hardware

However, binary image copy improves repeatability, reduces entropy
Install vs. Deploy

User

Provisioning Service

DHCP service

TFTP service

Physical Host

installer-based

upload ISO image

pre-build image w/ needed drivers, apps

upload QCCW image

deploy it!

set DHCP opts

power on & boot from network

DHCP BOOT request

IP + next-ref + kernel params

fetch kernel & ramdisk

installer-based
SERVICE COMPONENTS
STANDARD PROTOCOLS

Power

- **IPMI**: intelligent platform management interface, for remote control of machine power state, boot device, serial console, etc.
- **SNMP**: simple network management protocol, often used with Power Distribution Units for remote control of power status.
STANDARD PROTOCOLS

**Boot**

- **DHCP**: dynamic host configuration protocol, used to locate the NBP on the network, and provide the host OS with IP address during init
- **TFTP**: trivial file transfer protocol, copies the NBP over the network
- **PXE**: pre-boot execution environment, allows host to boot from network
- **[g,i]PXE**: recent enhancements make PXE more flexible, supported on most hardware
IPMI has not changed in the last 10 years.

DMTF just published "Redfish" spec ... but ...
Vendor value is derived from quality of hardware, services, support, and integration not from proprietary solutions to common problems.
DRIVER API

**Power Interface**
- get_power_state()
- set_power_state()
- reboot()

**Management Interface**
- get_supported_boot_devices()
- get_boot_device()
- set_boot_device()
- get_sensors_data()

**Deploy Interface**
- prepare()
- clean_up()
- deploy()
- tear_down()
- take_over()

**Console Interface**
- start_console()
- stop_console()
- get_console()

**Boot Interface**
- prepare_ramdisk()
- clean_up_ramdisk()
- prepare_instance()
- clean_up_instance()

**Vendor Interface**
- validate()
- driver_validate()

decorators:
- @passthru
- @driver_passthru
simplicity $\rightarrow$ flexibility

consistency $\rightarrow$ repeatability
GET /v1/nodes/

{
    "nodes" : [
        {
            "name" : "nuc",
            "maintenance" : false,
            "instance_uuid" : null,
            "power_state" : "power off",
            "uuid" : "ba031dea-e7a8-4917-89f1-0f3ad3134ee",
            "provision_state" : "available",
            "links" : [
                .... snip ....
            ],
        },
    ]
}
GET /v1/nodes/ba031dea-e7a8-4917-89f1-0f3ad31344ee

```json
{
    "name": "nuc",
    "uuid": "ba031dea-e7a8-4917-89f1-0f3ad31344ee",
    "driver": "pxe_amt",
    "properties": {
        "ram": 8096,
        "cpu_arch": "x86_64",
        "cpus": 2,
        "disk_size": 500
    },
    "driver_info": {
        "amt_password": "*****",
        "amt_address": "192.168.2.3",
        "amt_username": "admin"
    },
    "power_state": "power off"
}
```
Every driver is different and requires different driver_info
OPERATIONS
"Thousands" of servers managed today per region.
However...

- Parallel deploys can saturate network
- Periodic health checking can be delayed by bad BMCs
- Nova scheduler does resource selection in Python
- Neutron DHCP config reload is slow
"But OpenStack is too complex! Is there a simple alternative?"

Yes

Run Ironic straight from Ansible.
First
input environment vars

$ cat bifrost/playbooks/inventory
---
node_default_network_interface: eth0
network_interface: eth2
ipv4_subnet_mask: 255.255.255.0
ipv4_gateway: 192.168.1.1
ipv4_nameserver: 8.8.8.8
dhcp_pool_start: 192.168.2.200
dhcp_pool_end: 192.168.2.250
deploy_kernel: "{{http_boot_folder}}/coreos_production_pxe.vmlinux"
deploy_ramdisk: "{{http_boot_folder}}/coreos_production_pxe_image-oem"
deploy_image_filename: "deployment_image.qcow2"
deploy_image: "{{http_boot_folder}}/{{deploy_image_filename}}"
Then **Install**

external dependencies & environment prep

```bash
$ bash ./scripts/env-setup.sh
$ source /opt/stack/ansible/hacking/env-setup
$ cd playbooks

$ ansible-playbook -K -vvvv -i inventory/localhost install.yaml
```
Then **Enroll**

with an inventory file

```
$ ansible-playbook -vvvv -i inventory/localhost enroll.yml \
  -e baremetal_csv_file=baremetal.csv
```

or by using CLI to create Nodes, Ports

```
$ ironic node-create -d agent_amttool -n nuc \
  -i amt_password='Pa$$w0rd' -i amt_address='192.168.2.3' -i amt_use \
  -p cpu_arch=x86_64 -p local_gb=64 -p memory_mb=8192 -p cpus=2 \
  -i deploy_ramdisk='http://192.168.2.2:8080/coreos_production_pxe_\n  -i deploy_kernel='http://192.168.2.2:8080/coreos_production_pxe.\n
$ ironic port-create -n $UUID -a ec:a8:6b:fe:e1:b0
```
Finally, **Deploy** from the inventory file or hit all inventory in Ironic

```
$ ansible-playbook -vvvv -i inventory/bifrost_inventory.py deploy-dynamic.yaml
```
DEMO?

Q & A?
THANKS!

@devananda
devananda.github.io/talks
github.com/openstack/ironic
github.com/openstack/bifrost