# OS deployment
# at Chalmers e-Commons

Physical setup, management, PXE boot OS deployment

CHALMERS UNIVERSITY OF TECHNOLOGY | E-COMMONS

# Design spec and prepwork

- Before any parts arrive we
  - Write down host list with all IPs
  - Plan for where to rack them up and how to place them
  - Send to integrator for preparing management and labeling.
  - Create dsh groups
  - Create conman lists of hosts
  - Icinga nodes
  - Generate passwords

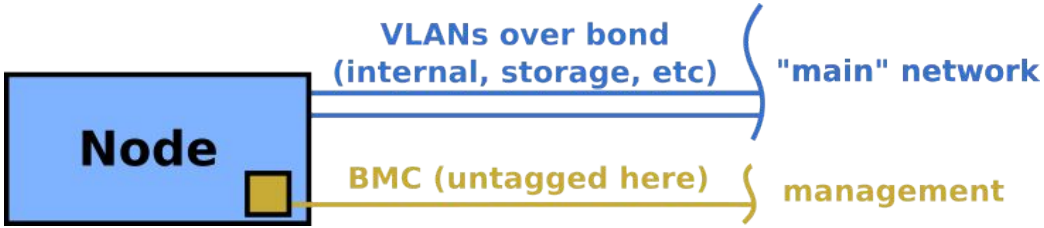
# Hardware installation

- Follow prepared design documents with all IPs and physical placement
  - Prepare PDUs and power on sockets. Add to UPS iff desired.
- Switches
  - Connect via serial console and configure IP and SSH access over management port
  - Connect management ports to central management network
  - Configure (C-LAG) uplinks and connect all other cabling
- Servers
  - Power should be spread evenly across all phases. Power on socket groups
  - Connect management and cabling
  - Configure static management IPs
  - Adjust cooling and seal up any gaps for optimal airflow
- Labeling

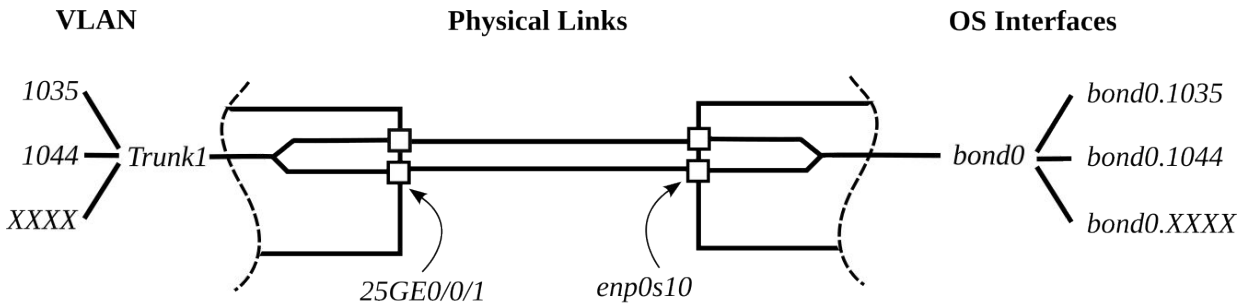
# BMC (baseboard management controller)

- Essential for managing machines remotely.
  - Very vendor specific
  - Disable risky features like sharelink (dangerous to expose to the host OS)
  - Set up static IPs
  - Enable SNMP
  - Enable IPMI
  - Enable redfish
  - Enable health checking protocols
  - Hostname (visible over LLDP and web interface)
- More possibilities
  - KVM
  - rsyslog
  - Boot from virtual media

# Network configuration




- Management
  - Preferably physically separate 1GbitE network (SFP ports bridging up to core switches)
  - Very simple configuration, sometimes unmanaged, access (untagged)
- Bring out all the necessary VLANs
  - M/C-LAG bonds where appropriate
  - Tagging
- PXE deployment
  - Typically (always) require untagged VLAN
  - Not too happy about bonds either. Use of “lacp force-up” modes may circumvent it.
  - Some models can’t PXE boot on mellanox cards (BIOS doesn’t understand them).




## Switch config

```
#
interface 25GE1/0/1
 eth-trunk 1
 device transceiver 25GBASE-COPPER
#
interface 25GE1/0/2
 eth-trunk 1
 device transceiver 25GBASE-COPPER
#
interface Eth-Trunk1
 port link-type hybrid
 port hybrid pvid vlan 1035
 port hybrid tagged vlan 1033 1043 to 1044
 port hybrid untagged vlan 1035
 mode lacp-dynamic
```

## Ansible (after we have the OS)

```
- name: "bond0: setup bond"
  nmcli:
    type: bond
    conn_name: bond0
    autoconnect: yes
    state: present

- name: "bond0: add bond-slaves"
  nmcli:
    type: bond-slave
    conn_name: '{{ item }}.bond0'
    ifname: '{{ item }}'
    master: bond0
    state: present
  with_items:
    - 'ens801f0np0'
    - 'ens801f0np1'

- name: "bond0: add bond-vlans"
......
```

# Passwords

- Before even using them we store it all in keepassx database
  - Host passwords
  - Management network passwords
  - Various services (adding the links as well)
  - Keys / access tokens
- Currently just a shared keepassx database. Easiest to bulk update and upload than to trickle in passwords.
  - Download the latest
  - Add all new things
  - Upload new (move date stamped old database to backup directory)
  - Inform everyone in the chat to grab the new one

# Management

- Needed before any deployment
  - Serial console (managed via conman, which logs them as well)
  - IPMI (via handy “ipmi” script to simplify rebooting of nodes and such using hostlists)
    - Using hostlist, sets cipher, password
  - Vendor tools like onecli, kmscli, sum for setting UEFI parameters, boot order etc.
    - Save the settings of nodes. Handy for restoring after motherboard replacements
    - Also save license keys for BMCs
    - Diff and make sure all the desired settings are present, SMT, Numa, power, security (no management accessible via OS!) and more
  - Need to be able to change boot order of machines (at least “PXE next boot”)
- Also set up any /etc/dsh/groups/ for accessing the machines conveniently

# PXE deployment

- Currently using cobbler to manage distros profiles and systems
  - Cobbler is half dead, looking for replacement
  - Manages PXE boot images and DHCP via dnsmasq and kickstart files
  - Almost only using anaconda kickstarts (RHEL based method)
  - Intentionally minimal deployment strategy
    - OS drive partitioning
    - OFED driver installation (else network cards change names which is annoying)
    - Minimal packages
    - root ssh key from Janne
  - Using simple scripts that sets PXE boot on + icinga downtime + reboot

# Make handy scripts to work many nodes

```
$ ipmi dev-cirrus-c[01-12] power status | pshbak -c
----------------
dev-cirrus-c[01-12]
----------------
Chassis Power is on
```

```
$ schedule_downtime.py dev-cirrus-c12
$ reinstall_system.sh dev-cirrus-c12
$ manycli cirrus-c[01-04] config restore --file=cirrus.cfg
$ pdsh -g cirrus_dev uptime
```

# PXE - DHCP

- Machines set to boot with PXE first
- UEFI/BIOS will request DHCP for every interface
  - Slows down booting: disable everything we can in UEFI keeping only IPv4 PXE boot on the correct interface if possible
- Monitor DHCP requests on Janne
  - systemctl status dnsmasq -n 99
  - `May 08 08:13:23 janne dnsmasq-dhcp[3043581]: DHCPREQUEST(eth11) 10.44.255.2 bc:24:11:57:c5:11`
- No requests coming through? Debugging during PXE boot is a pain in the ass
  - Just manually deploy a node, log in via conman and bring up network manually to test things out. Usually missed something some switch configuration, like VLAN or lacp-force-up

# Anaconda kickstart file (slightly trimmed)

```
keyboard us
lang en_US.UTF-8
timezone --utc Europe/Stockholm
text
skipx
firstboot --enable
reboot

#raw
rootpw --iscrypted $6$....
#end raw

firewall --disabled
selinux --disabled

zerombr
clearpart --all --initlabel
bootloader --location=mbr --append="edd=off"
part /boot/efi --fstype=vfat --size=128
part /boot --fstype ext4 --size=1024
part / --fstype ext4 --size=18432
#if $use_weka == 'True'
part /opt/weka --size=30720
#end if
part /tmp --fstype ext4 --size=1024
part /local --size=100 --grow
```

```
repo --name=c3se-base --baseurl=http://$server/repo/rocky9/base/
repo --name=c3se-appstream --baseurl=http://$server/repo/rocky9/appstream/
repo --name=c3se-mlnx --baseurl=http://$server/repo/rocky9/mlnx/
%packages --ignoremissing
@core --nodefaults

rdma-core
rdma-core-devel
mlnx-ofa_kernel
mlnx-ofa_kernel-devel
mlnx-ofa_kernel-modules
-firewalld
-iwl*firmware
-ModemManager-glib
%end

%post
dracut -f # maybe needed when using mlnx-ofa_kernel-modules
hostnamectl set-hostname $hostname
$SNIPPET('post_install_kernel_options')
$SNIPPET('mgmt_authorized_pubkey')
$SNIPPET('autoinstall_done')
%end
```

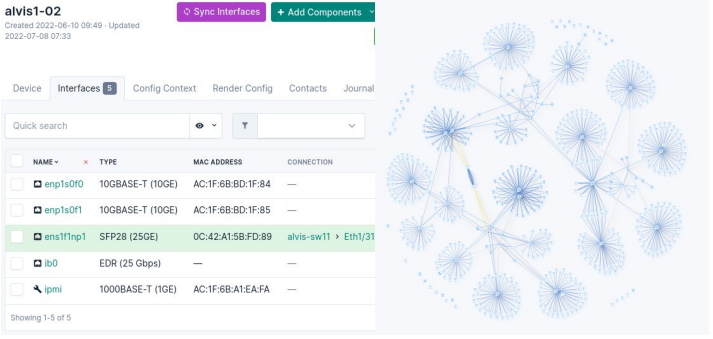
# Ansible - generally applicable configurations:

- Networking, static IPs, bond, VLANs
- Hostname, timezone
- Network tuning + tuned - Larger buffer sizes and such
- arp-cache - OS defaults are quite limited
- LLDP - essential for network discovery and debugging
- Icinga health checks, mcelog
- Firewall
- Rsyslog
- (TSM backups)

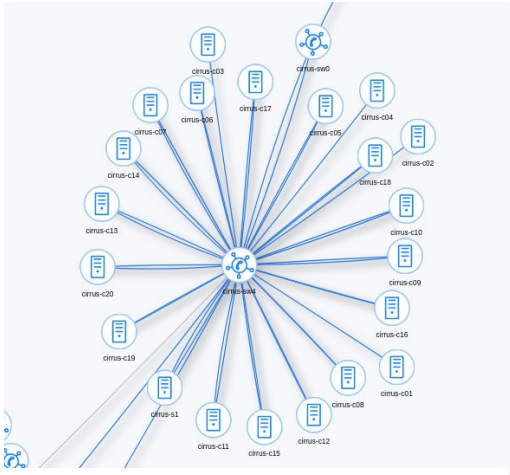
```yaml
- name: Enable firewalld
  service:
    name: "firewalld"
    enabled: yes
    state: started

- name: Look up public IP from /etc/hosts
  command: gethostip -d {{inventory_hostname}}-c3se
  delegate_to: localhost
  register: public_ip4
  check_mode: no
  changed_when: False

- name: Setup external interface
  nmcli:
    ifname: 'external'
    conn_name: 'external'
    state: present
    type: vlan
    ip4: '{{ public_ip4.stdout }}/16'
    gw4: '{{ gateway }}'
    ip6: '{{ ipv6_address }}/64'
    gw6: '2001:6b0:2:2010::1'
    vlandev: "{{ interface_public.split('.')[0] }}"
    vlanid: "{{ interface_public.split('.')[1] }}"
    zone: public
    dns4:
     - 129.16.1.53
     - 129.16.2.53
    conn_reload: True
```

# Inventory (Netbox)

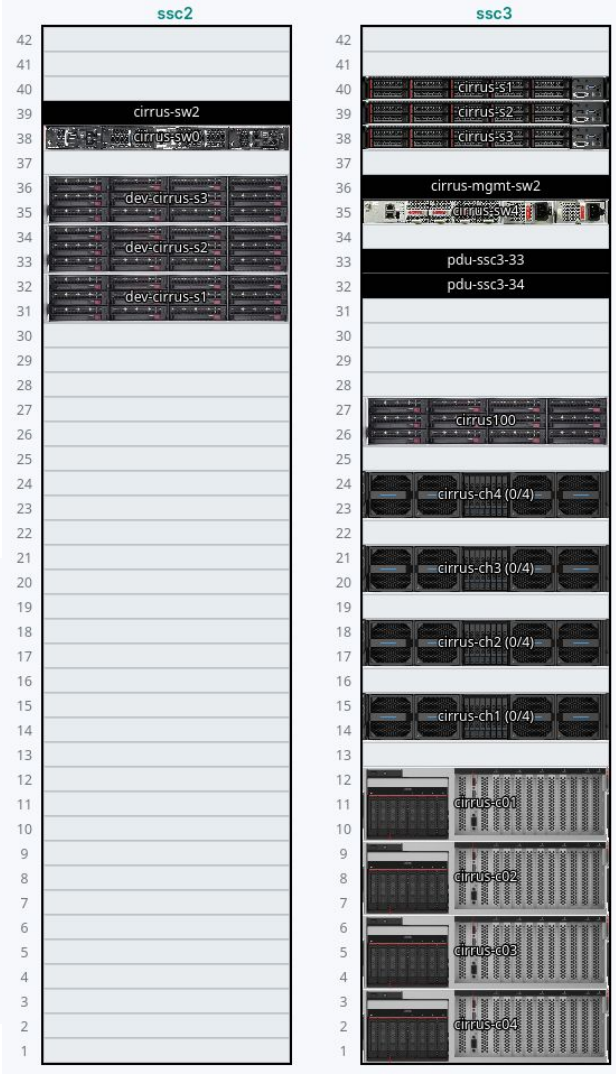
- Rack-arrangement (add to Netbox by hand)
- MAC addresses and link information (ansible)
- Switch configs backup (ansible)
- BMC and BIOS configurations (scripts, still vendor-specific)





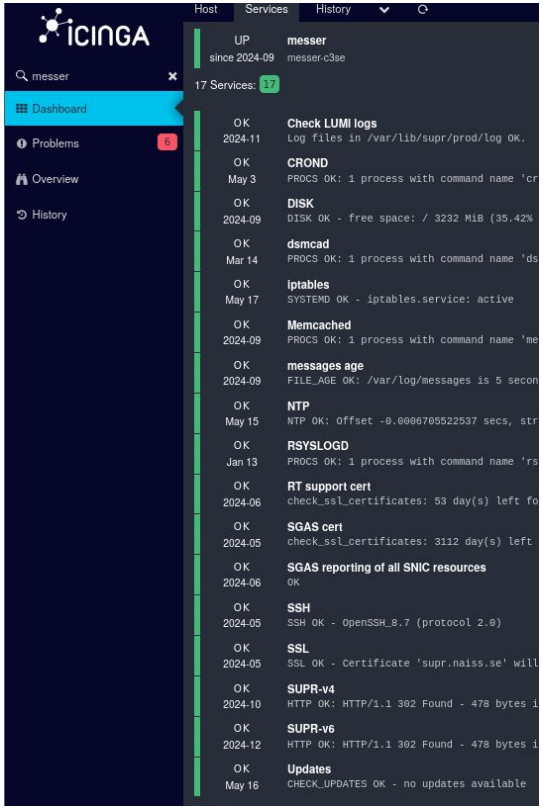




CHALMERS UNIVERSITY OF TECHNOLOGY | E-COMMONS



# Health checks

- Icinga for health checks
  - Ping
  - SSH
  - Probe management for general hardware health events, fans, cooling, temperatures
  - Disk checks
  - systemd unit failures
  - Firewall
  - Updates
  - Certificates
  - NTP
  - Application specific scripts for health checks, e.g. ceph status
- Set alerts in chat if deemed urgent




CHALMERS UNIVERSITY OF TECHNOLOGY | E-COMMONS

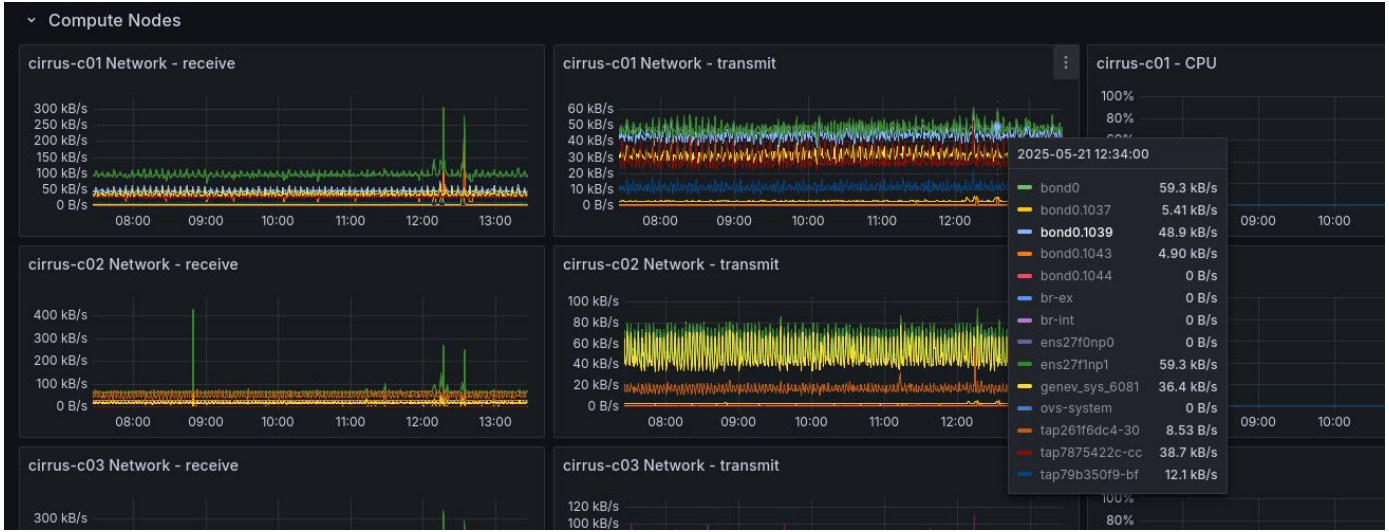
# Icinga example

```
object Endpoint "cirrus-s1" { host = "cirrus-s1" }
object Zone "cirrus-s1" {
    endpoints = ["cirrus-s1"],
    parent = "mgmt"
}
object Host "cirrus-s1" {
  import "lenovo-cirrus-host"
  address = "cirrus-s1"
  vars.mgmt_ip = "cirrus-s1-mgmt"
  vars.docker_containers = [
      "mariadb",
      …
  ]
}
```

```
apply Service "firewalld" {
  import "generic-service-5min"
  groups += ["Daemons"]
  check_command = "systemd_service"
  vars.unit = "firewalld"
  vars.slack_notifications = "enabled"
  assign where host.name in ["cirrus-s1", "cirrus-s2", "cirrus-s3"]
  command_endpoint = host.name
}

apply Service "Failed services" {
  import "generic-service-5min"
  groups += ["Health"]
  check_command = "failed_services"
  assign where regex("^cirrus-s\\d\\d", host.name)
  assign where regex("^cirrus-c\\d\\d", host.name)
  assign where "Foo" in host.groups
  command_endpoint = host.name
}
```

# Metrics

1. Collect metrics
2. ???
3. Profit!





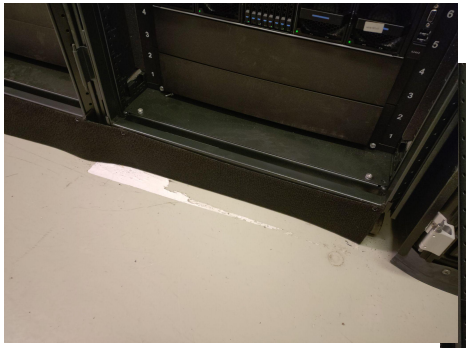
CHALMERS UNIVERSITY OF TECHNOLOGY | E-COMMONS

# Maintenance

- Basic service doesn’t stop after deploy
  - OS security updates
    - reboot with kexec helper script speed things up tremendously
  - Firmware updates (BMC, UEFI, drives, network devices)
    - More custom helper scripts to run in parallel (tmux windows) across dozens of machines
  - OS security updates
    - Should at least have semi-automated procedures for updates
- Procedures for downtime and service
- Prolonging service contracts

# Backup slides after this one

# Pictures



CHALMERS UNIVERSITY OF TECHNOLOGY | E-COMMONS

# Alternatives provisioning services

- The overall procedure would be similar:
  - Setup management network
  - Setup server for PXE-boot
  - Configure machines with ansible or alike

- Differences
  - Image with cloudinit & configdrive (as opposed to kickstart files), e.g. ironic, matchbox;
  - Expose deployment to other components of k8s, e.g. metal$^3$, MAAS.