Los Alamos NATIONAL LABORATORY

Continuous Delivery of HPC Compute Infrastructure

Ethan Clark
Nick Jones (Advisor)
Devon Bautista (Advisor)

August 2025

LA-UR-25-28157

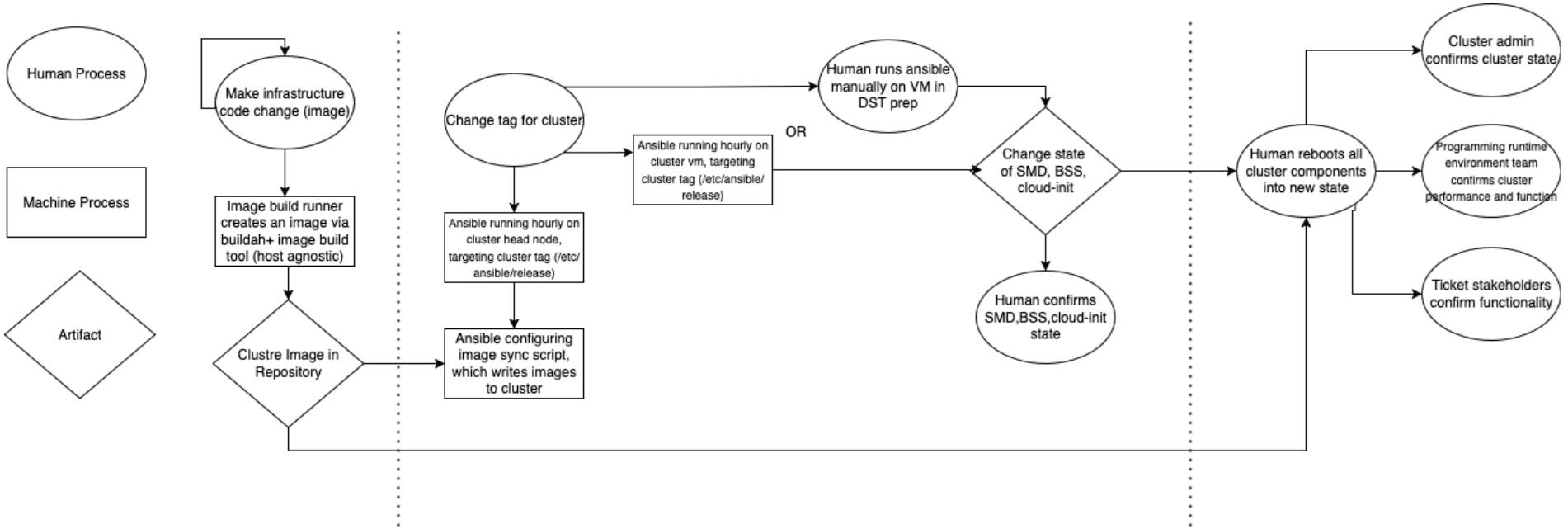
Minimizing dedicated system time via automation

- Automation is a time-tested method for speeding processes up
- DSTs often have a lot of waiting, which can lead to missed cues and delayed progress
- Automation can also help improve reliability by integrating more error checking and causing less typos

What is dedicated system time?

- Dedicated system time (DST) is when a system is reserved for non-user purposes, such as bugfixes, updates, reconfiguration, testing, etc
- A DST can usually last anywhere from an hour to two weeks
- Minimizing the time DSTs take is important to ensuring users get the maximum amount of run time possible

A normal DST

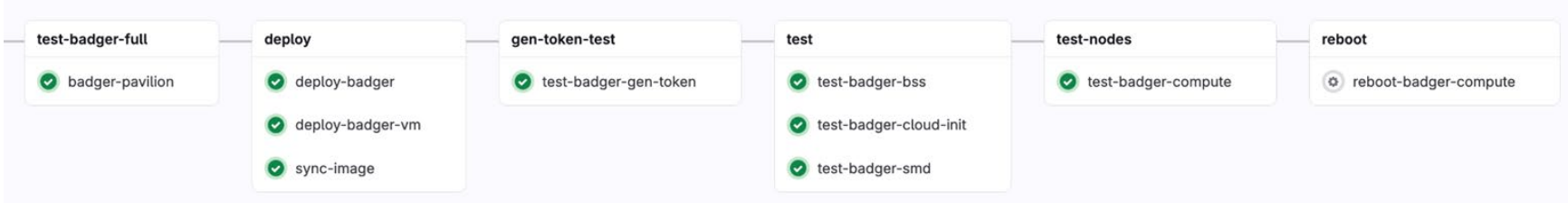

Human Process
Machine Process
Artifact
Make infrastructure code change (image)
Image build runner creates an image via buildah+ image build tool (host agnostic)
Clustre Image in Repository
Change tag for cluster
Ansible running hourly on cluster head node, targeting cluster tag (/etc/ansible/release)
Ansible configuring image sync script, which writes images to cluster
Human runs ansible manually on VM in DST prep
OR
Ansible running hourly on cluster vm, targeting cluster tag (/etc/ansible/release)
Change state of SMD, BSS, cloud-init
Human confirms SMD,BSS,cloud-init state
Human reboots all cluster components into new state
Cluster admin confirms cluster state
Programming runtime environment team confirms cluster performance and function
Ticket stakeholders confirm functionality

Git

- Git provides several extremely useful features
  - Version control
  - Centralized repository
  - Branches
  - Blame (tracking who wrote what)
- Various Git providers (GitLab, Github, etc) also provide extra features such as pull/merge requests and deployment pipelines.

Pipelines

- Series of stages that execute when a condition is met
- Attached to the event (usually a commit) that spawned them
- Can be run either manually or automatically, and can be stopped partway
- Can be separated into stages with or without additional conditions
  - Multiple pipelines can be run at once with no relation



Modelling deployment as software

- Scripts set cluster state
- Git stores the scripts
- Git stores the state

- Pipelines run scripts
- Pipelines set state
- Tagging a commit triggers a pipeline
- Tagging a commit sets state

Cluster Specs and Configuration - Badger

- 600 Nodes
- OpenCHAMI
- Ansible - Baremetal and VM
- Slurm Job Scheduler
- Stateless nodes
  - booted via PXE (TFTP and HTTP)
  - uses read-only NFS for the root filesystem
  - Some directories (/home, etc) are mounted read-write

Testing

Smoke test / Hard Regressions

- OpenCHAMI
- Reboot a single node
- Failed Services

Performance / Soft Regressions

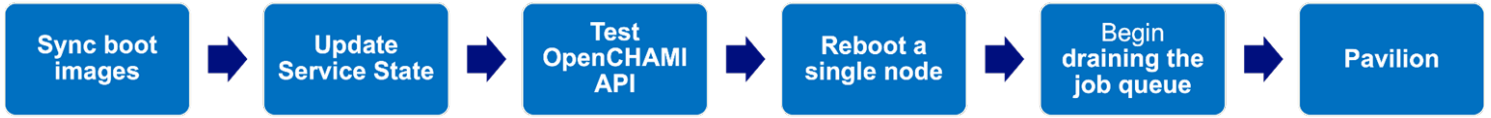
- Benchmarks
  - CPU
  - Memory
  - Filesystems
- Pavilion

```
while $SSH_CMD systemctl is-system-running | grep starting; do
  sleep 1
done

test 0 -eq "$($SSH_CMD systemctl list-units --failed --no-pager --no-legend | grep -vP 'openibd' | tee >(cat >&2) | wc -l)"
```

```
[root@ba005 ~]# systemctl list-units --failed
  UNIT                    LOAD   ACTIVE SUB    DESCRIPTION
● dnf-makecache.service   loaded failed failed dnf makecache
● openibd.service         loaded failed failed openibd - configure Mellanox devices

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

2 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
[root@ba005 ~]#
```

```
echo Starting Pavilion Test
set -ve

export MODULEPATH=$(/usr/share/lmod/lmod/libexec/addto --append MODULEPATH /usr/share/lmod/lmod/modulefiles/Core)
export MODULEPATH=$(/usr/share/lmod/lmod/libexec/addto --append MODULEPATH /home/smehta/pe/rhel8-x86_64/modulefiles/linux-rhel8-x86_64/Core/)
OLD_OWNER="$(stat -c '%u:%g' .git)"
chown -R root:root .
source ./activate.sh

pav run supermagic.badger kickstart.badger hello_mpi.badger -c schedule.nodes=2
```

Pipeline



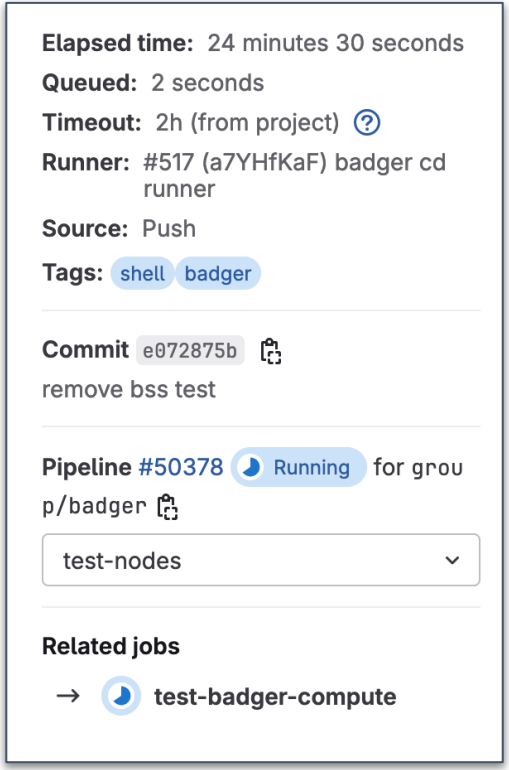
- **Sync boot images** – Pull down the boot images from the container image repository so nodes can reboot into them
- **Update Service State** – Reconfigure the head node by executing the Ansible playbook to enable the compute nodes to reboot into the new state
- **Test various OpenCHAMI API endpoints** – Smoke test the ensure that the Ansible playbook applied the correct values for the new state
- **Reboot a single node into the new state** – Another smoke test to ensure no hard regressions were introduced by the new state
- **Begin draining the job queue on all nodes** – Stops new jobs from being added in order to reboot each node individually as their queues empty
- **Pavilion** - Once the entire compute plane is in new state, run tests to detect any soft regressions that may have been introduced

Results

- Faster
  - Runs in 15-30 minutes
- More reliable
  - Less Typos
  - Better error checking
- Easy reversion
  - Rolling back is as easy as going forward
- Better access controls
  - Pull requests require review and approval
- Zero* down time

Future Work

- A/B testing during reboots
- Green/blue job queues
  - Allow for more changes to be made faster
- Better timeout handling
- Better pavilion integration
  - JUnit output plugin



Questions

Ethan Clark
ethan.clark@trojans.dsu.edu

Nick Jones
njones@lanl.gov

Devon Bautista
devonb@lanl.gov

Implementation

- The implementation can be split into three primary parts:

- These components are represented in the Git repository, as scripts, playbooks, and pipelines
- The Git repository is hosted on a local version of GitLab, with runners for linting, image building, and cluster deployment
- The test cluster this was developed with has about 600 nodes and uses OpenCHAMI as management software.

Configuration

- In this implementation, almost all configuration is done via Ansible, with a minority using legacy Bash scripts
- These playbooks and scripts act as a representation of the system state they create when executed
- The Git repository storing the scripts contains the version history not only of the scripts, but of the cluster state itself
- This is known as Infrastructure as Code, and it has several benefits:

Faster repairs
- Configuration is represented completely in a text file

Time savings
- Deployment can be easily automated

Faster action
- Autonomous deployment reduces wait times

Deployment

- GitLab runners manage Continuous Delivery
- Modularization of deployment recipes as pipeline stages
- When a commit is given a tag matching a specific pattern, GitLab will run the deployment pipeline, matching the cluster state to the state represented in the commit
- If the commit causes a regression, the state can be reverted by retagging an old commit with known good state

Smoke Test

- In addition to Pavilion, there are a few tests baked into the pipeline for testing the state of OpenCHAMI:

- These components are responsible for booting, managing, and configuring compute nodes
- Each service runs in a Podman container via SystemD
- Using the OpenCHAMI API, we can get the current state of the services, then compare that to the state in the Ansible playbook

Testing

- Both hard and soft regressions can cause major issues
- Pavilion integrates with Slurm, which allows for tests to be run in parallel both amongst themselves and with existing user jobs
- It is already used internally at LANL (with a wealth of preexisting tests), so the only work involved is integrating it into the pipeline.

Results

- The pipeline takes 15-30 minutes to fully execute on the test cluster
- It can be started, stopped, and paused at any point during the process.
- Forcing all changes through Git can also allow for more flexible and precise access controls on changes, such as requiring review.
- The time to repair after a failure is significantly lower due, as reverting is the same procedure as rolling out new state

Compute Node Reboot Test

```
FE_CMD="sudo ssh -o LogLevel=error ba-fe1 --"

RESERVATION="$($FE_CMD scontrol create reservation user=root starttime=now duration=15 flags=maint nodecnt=1 | tee >(cat >&2) | grep -oP '(?<=Reservation created: ).+')"
test -n "$RESERVATION"
sleep 2
while [ -z "$NODE" ]; do
  NODE="$($FE_CMD sinfo "$RESERVATION" -t MAINT --json | jq -r '.sinfo[] | select(.node.state | contains(["MAINTENANCE"])) | .nodes.nodes | .[]' | head -n1)"
done
echo "$RESERVATION $NODE"

SSH_CMD="sudo ssh -o StrictHostKeychecking=no -o LogLevel=error $NODE --"

# $SSH_CMD kexec-update.sh || echo "Failed to load kexec"
# $SSH_CMD systemctl kexec || STATUS=$?
# if [ $STATUS -ne 0 ] && [ $STATUS -ne 255 ]; then
  $SSH_CMD systemctl reboot || STATUS=$?
  if [ $STATUS -ne 0 ] && [ $STATUS -ne 255 ]; then
    ipmipower -h "${NODE}-bmc" --cycle
    echo "Rebooted via IPMI successfully"
  else
    echo "Rebooted via SSH successfully"
  fi
# else
#   echo "Kexec'ed successfully"
# fi

until $SSH_CMD id; do
  sleep 1
done

while $SSH_CMD systemctl is-system-running | grep starting; do
  sleep 1
done

test 0 -eq "$($SSH_CMD systemctl list-units --failed --no-pager --no-legend | grep -vP 'openibd' | tee >(cat >&2) | wc -l)"

#$SSH_CMD tee /etc/resolv.conf </etc/resolv.conf

$SSH_CMD stress-ng --cpu 4 --vm 2 --hdd 1 --fork 8 --timeout 10s --metrics --yaml /dev/stdout | yq -r '.metrics[] | "# TYPE " + .stressor + " histogram\n" + .stressor + " " + .bogo-ops-per-second-real-time + " " + .wall-clock-time' > metrics.txt

$FE_CMD scontrol delete ReservationName="$RESERVATION"
```

Pavilion

```
sudo ssh -o LogLevel=error -t ba-fe1 -- sbatch --wait --verbose -N2 <<"EOF" || STATUS=$?
#!/bin/bash
cd /home/smehta/pav2-lanl/

exec 1>last_pav.out
exec 2>last_pav.err
echo '' >last_pav.json

echo Starting Pavilion Test
set -ve

export MODULEPATH=$(/usr/share/lmod/lmod/libexec/addto --append MODULEPATH /usr/share/lmod/lmod/modulefiles/Core)
export MODULEPATH=$(/usr/share/lmod/lmod/libexec/addto --append MODULEPATH /home/smehta/pe/rhel8-x86_64/modulefiles/linux-rhel8-x86_64/Core/)
OLD_OWNER="$(stat -c '%u:%g' .git)"
chown -R root:root .
source ./activate.sh

pav run supermagic.badger kickstart.badger hello_mpi.badger -c schedule.nodes=2

pav wait

pav results --json > last_pav.json
```

```
# Generate JUnit
(
    echo '<?xml version="1.0" encoding="UTF-8"?><testsuites>'
    jq -r '.[] |
"<testsuite><testcase time=\""
  + (.duration|tostring)
+ "\" name=\""
  + (.name|@html)
+ "\" timestamp=\""
  + (.started|todateiso8601)
+ "\">\n  <properties>\n    "
+ (
    .sched | to_entries | map(
        "<property name=\""
      + (.key|@html)
      + "\" value=\""
      + (.value|@html)
      + "\" />"
    ) | join("\n")
) + "\n"
+ "  </properties>\n"
+ (if .result == "FAIL" then "<failure>\n" + .results_log + "\n</failure>\n" else "<system-out>\n" + .results_log + "\n</system-out>\n" end)
+ "</testcase></testsuite>"' last_pav.json
    echo '</testsuites>'
) | xargs -0a <(for i in $(jq -r '[.[].results_log] | unique | .[]' last_pav.json); do printf -- '-e\0s|^%s$|jq -Rrs @html <%s|eg\0' "$i" "$i"; done) -- sed | xmllint --format - > last_pav.xml

chown -R "$OLD_OWNER" .
EOF

sudo sh -c 'mv -t . -- /home/smehta/pav2-lanl/last_pav.* && chown gitlab-runner last_pav.*'
jq -r '.[] | if .result == "FAIL" then ("Test \"" + .name + "\" failed\n" | halt_error) else empty end' last_pav.json
exit $STATUS
```

Los Alamos NATIONAL LABORATORY

OpenCHAMI Smoke Tests

```
test-badger-cloud-init:
  stage: test
  dependencies:
    - test-badger-gen-token
  tags:
    - badger
    - ssh
  rules:
    - if: $CI_COMMIT_TAG == "group/badger"
  script:
    - export BADGER_ACCESS_TOKEN="$(cat ~/.token.env)"
    - test -n "$BADGER_ACCESS_TOKEN"
    - export TARGET="$(yq '.nodes[].xname' <inventory/group_vars/ochami_cluster_badger/nodes.yaml | shuf -n1)"
    - echo "$TARGET"
    - 'curl -H "Authorization: Bearer $BADGER_ACCESS_TOKEN" "127.0.0.1:8081/cloud-init/admin/impersonation/${TARGET}/meta-data" | yq .instance_data.v1.public_keys.[] | grep "$(yq .cloud_init_node_authorized_key <inventory/group_vars/ochami_cluster_badger/cloud_init.yaml)"'

test-badger-smd:
  stage: test
  dependencies:
    - test-badger-gen-token
  tags:
    - badger
    - ssh
  rules:
    - if: $CI_COMMIT_TAG == "group/badger"
  script:
    - export BADGER_ACCESS_TOKEN="$(cat ~/.token.env)"
    - test -n "$BADGER_ACCESS_TOKEN"
    - |
      diff \
        <(yq -o=json '[.nodes[] | {"type": .type, "xname": .xname}] as $nodes | [$nodes.[] | .type] | unique | [.[] | . as $type | [$nodes.[] | select(.type == $type) | .xname] | sort] | sort_by(length)' <inventory/group_vars/ochami_cluster_badger/nodes.yaml) \
        <(for GROUP in $(yq '[.nodes[] | {"xname": .xname, "type": .type}] as $nodes | [$nodes.[] | .type] | unique | .[]' <inventory/group_vars/ochami_cluster_badger/nodes.yaml); do ochami -k smd group get --name $GROUP | jq '.[].members.ids | sort'; done | jq -s 'sort_by(length)')

test-badger-bss:
  stage: test
  dependencies:
    - test-badger-gen-token
  tags:
    - badger
    - ssh
  rules:
    - if: $CI_COMMIT_TAG == "group/badger"
  script:
#    - test -n "$BADGER_ACCESS_TOKEN"
    - |
      diff \
        <(ochami -k bss dumpstate | jq '[.Params[] | [.macs[] | ascii_upcase] | sort] | sort_by(length)') \
        <(yq -o=json '[.nodes[] | {"type": .type, "mac": .interfaces[0].mac_addr}] as $nodes | [$nodes.[] | .type] | unique | [.[] | . as $type | [$nodes.[] | select(.type == $type) | .mac] | sort] | sort_by(length)' <inventory/group_vars/ochami_cluster_badger/nodes.yaml | tr [:lower:] [:upper:])
```