# NEMU CI

Design and implementation

# Goals

- Build testing
  - Keeping the main branch green
  - Does this PR compile? (GitHub integration)
  - Dependencies pinned and updated as part of the code
- Integration testing
  - Interact with the guest
  - Test on x86-64 and aarch64 - **need the ability to do KVM**
  - Runnable locally as well on central CI
- Fast turnaround time (sub 20 minutes for a branch build, 15 for PR)
- CI as code (keep configuration in source tree)
- Cost effective

# Implementation overview

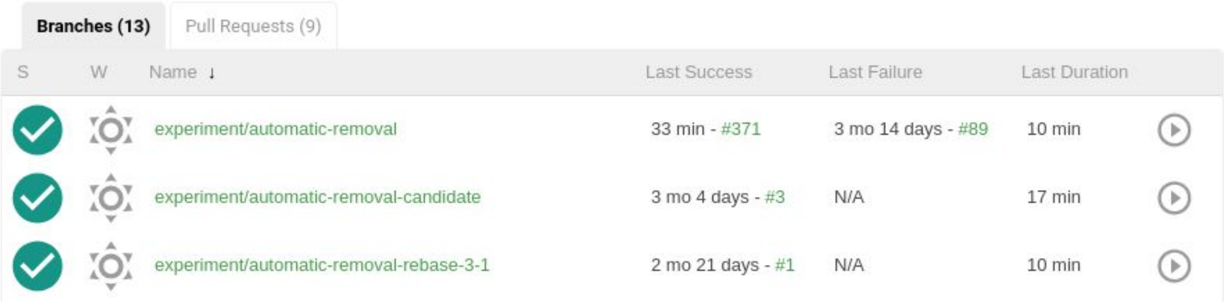
- Jenkins instance with master hosted on Azure
- x86-64 agents dynamically provisioned using Azure plugin - using machine class that supports nested KVM
- Dedicated server providing aarch64 instance
- GitHub integration for PRs/branch updates and authentication
- CI instructions stored in “Jenkinsfile” in source tree

# Jenkins Master

- Well established solution - complex but flexible
- Installed via “off-the-shelf” appliance
- Plugins in use:
    - Azure VM agents
    - GitHub Authentication
    - GitHub Branch Source
    - GitHub
    - SSH Slaves

# Jenkins GitHub integration

- Uses “multi-branch”
  - Pipeline added for each branch that contains a “Jenkinsfile” in the root
  - Pipeline created for each PR created



Branches (13) | Pull Requests (9)

| S | W | Name ↓ | Last Success | Last Failure | Last Duration | |
|---|---|---|---|---|---|---|
| ✓ | ☀ | experiment/automatic-removal | 33 min - #371 | 3 mo 14 days - #89 | 10 min | ▶ |
| ✓ | ☀ | experiment/automatic-removal-candidate | 3 mo 4 days - #3 | N/A | 17 min | ▶ |
| ✓ | ☀ | experiment/automatic-removal-rebase-3-1 | 2 mo 21 days - #1 | N/A | 10 min | ▶ |

- Repository hooks registered with webhooks for callback
- Use a “system account” as a bot to update status on builds
- Authentication via GitHub - no need for special credentials, authorization via GitHub usernames or teams

# Jenkins Agents

- For integration testing need to run in environment where KVM is available
  - For x86-64 use Azure machine class that supports nested KVM. VMs are created on demand and added as agents
  - For aarch64 use a rented dedicated server with persistent agent
- Fast build turnaround
  - Custom image used for VM agents with dependencies already preinstalled
  - Images used for testing cached in storage bucket in same region as VMs
  - Run with high number of VCPUs (16)

# CI as code

- Jenkinsfile stored in root of git repo
- Two forms - declarative (newer) or scripted
- Controls how builds are distributed across nodes (or types of nodes), what can be done in parallel and what the stages are.
- Stages are split into steps of which there are a large number of options available (e.g. git operations, integration with storage, notifications, etc)
- Most commonly used step is the shell one

# Jenkinsfile

```
stage ("Builds") {
    parallel ('xenial': {
        if (!env.BRANCH_NAME.contains("experiment/automatic-removal")) {
            node ('xenial') {
                stage ('Checkout: x86-64') {
                    checkout scm
                }
                stage ('Prepare: x86-64') {
                    sh "sudo apt-get update"
                    sh "sudo apt-get build-dep -y qemu"
                }
                stage ('Compile: x86-64') {
                    sh "SRCDIR=$WORKSPACE tools/build_x86_64.sh"
                }
                stage ('NATS: x86-64') {
                    sh "SRCDIR=$WORKSPACE tools/CI/run_nats.sh"
                }
            }
        }
    }
```

# NATS

- Test suite built in go for testing NEMU
- Control over hotplug of devices
- SSH into agent
- Runs under “go test”
- Highly parallel with each VM instance using dedicated files, etc to improve build turnaround

# Conclusion/Proposal

- Not a perfect solution but flexible
- We (Intel) are happy to setup and maintain a Jenkins CI PoC
- Will help create initial Jenkinsfiles for current repositories
- Can mix with other CI systems, e.g. Travis for a broad spectrum of testing