To implement solution to the problem, I have developed two batch script (GetVM and ReleaseVM) and one terraform script.

Batch script can be executed from any tool like Jenkins/rundeck. Authentication part will be taken care by that tool.

**Prerequisites**

* Install and configure the AWS CLI.
* Install and configure terraform.

To Get VM:

1. User logs in to Jenkins (any other system). Triggers “Get VM” job.
2. Get VM job calls get-VM batch script by passing logged in user as an argument.
3. Batch script accepts single argument as username:
   1. Checks if there is already any VM allotted to this user using AWS cli. If yes, returns IP address of that VM.
   2. If no VM is allotted, check if VM quota has exceeded using AWS cli.
   3. If not, deploy VM using terraform script. [AWS cli is also an option to launch VM instead of terraform script]. Return deployed VM IP address.

To Release VM:

1. User logs in to Jenkins (any other system). Triggers “Release VM” job.
2. Release VM job calls release-VM batch script by passing logged in user as an argument.
3. Batch script accepts single argument as username:
   1. Checks if there is already any VM allotted to this user using AWS cli. If no, returns message indicating no vm is allotted to this user.
   2. If VM is allotted, terminates VM using terraform script. [AWS cli is also an option to terminate VM instead of terraform script]. Clears folders.

To save the cost, I am launching VM only when needed and terminating it instead of just returning to the pool. I am using aws cli to get updated number of deployed VMs and their details instead of saving them locally to handle system failures.

AWS Cli to get existing VM description.

Terraform script to launch/ terminate VMs

Pass logged in username to bash script

User Interface

[Jenkins Job/Rundeck/ Node JS/ any other tool]

AWS

Release VM bash script

Get VM bash script

User logs in