

# Automated OpenCHAMI Integration Testing and Cluster Deployment

Marcos Johnson-Noya, Harvard University; Alana Kihn, Oregon State University; Madison Mejia, New Mexico Institute of Technology | HPC-DO, Supercomputer Institute | Mentors: Travis Cotton, Sakul Koirala, Jim Williams

## Background

### OpenCHAMI -

The **open-source system manager OpenCHAMI** aims to integrate cloud design principles with HPC system management. It was designed to be a simple, lightweight alternative to Cray System Management (CSM). Using a modular approach, OpenCHAMI utilizes microservices that are intended to be standalone and replaceable, providing users freedom in configuration.

### GitHub Actions -

GitHub Actions is a **continuous integration and continuous delivery (CI/CD)** platform that is built into GitHub. It allows developers to build, test, and deploy code all from within a repository.

### Hurl -

Hurl is a **HTTP request command line tool**, useful for asserting if certain HTTP requests are returning the expected response.

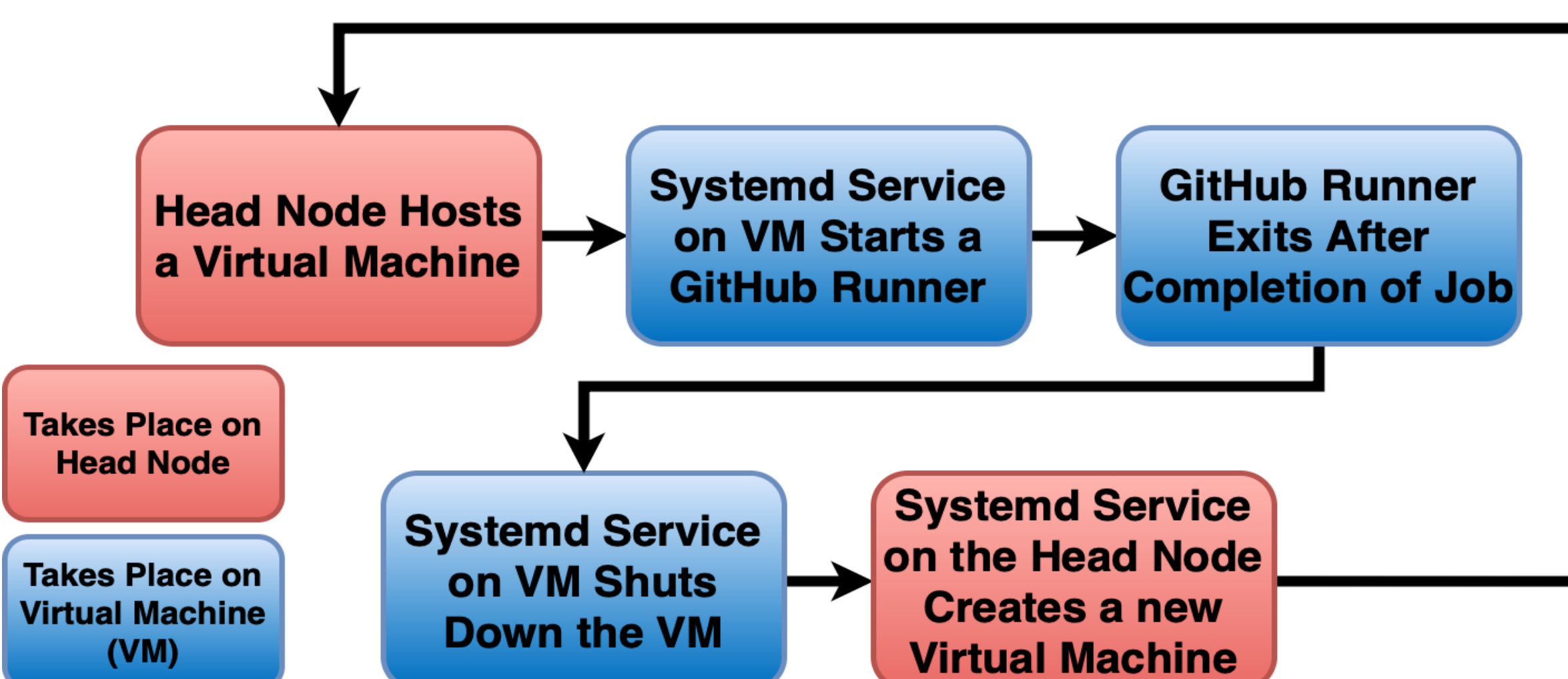


Figure 1 - Life Cycle of our Self-Hosted, Ephemeral GitHub Runner

## Challenges

### Virtual Machine Reboots After Each Job

Since the virtual machine reboots after each job completes, all steps of the runner have to occur in the same workflow file.

### OpenCHAMI Lacks Cloud-init capabilities -

The post-boot configuration manager Cloud-init is not supported yet, so we were unable to add users to our compute nodes *within the pipeline*. As a result of running as an unprivileged user, it was impossible to run SLURM jobs on the compute nodes.

### Finding a Fitting Testing Framework -

We were unsuccessful in finding a testing framework that automatically managed the order of test execution.

## The Pipeline

### Pipeline Workflow -

As depicted in **Figure 2**, the pipeline is triggered when OpenCHAMI developers push to the remote **deployment-recipes** repository. The pipeline then begins by following the **OpenCHAMI quickstart guide**. The steps within the quickstart guide configure the secrets file, system name, access token and certificates needed to start the OpenCHAMI services. Once the OpenCHAMI services are brought up, our pipeline continues by running **endpoint checks** to ensure the services are running or have exited successfully. When the services are confirmed to be up and running, all files within the **Hurl** test directories are executed. The Hurl test directories are designed to make it easy for developers to add or remove tests as needed.



OpenCHAMI  
Quickstart GitHub

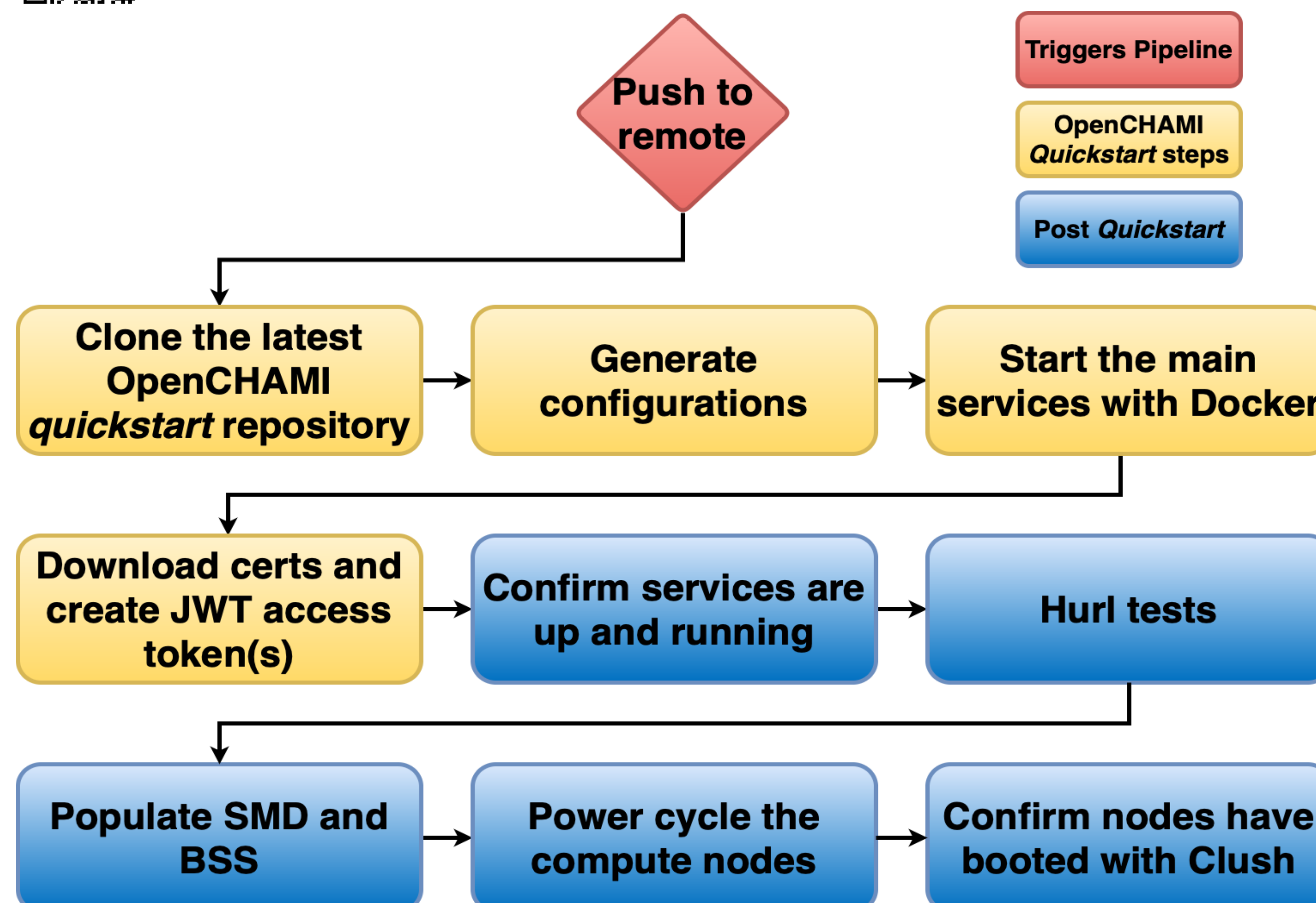


Figure 2 - Workflow of GitHub Pipeline

The pipeline then populates the **State Management Database (SMD)** and the **Boot Script Service (BSS)**. Both the SMD and the BSS are populated using YAML configuration files and the **OpenCHAMI Command Line Interface**. After it is confirmed that the BSS and SMD have been populated successfully, the compute nodes are power cycled using **Powerman**. The pipeline waits until at least 70% of the nodes have powered on successfully before continuing. The last step of the pipeline is confirming that the nodes have successfully booted. This is tested with the use of the **Clush** `uptime` command. Similar to the power cycle step, the pipeline checks that at least 70% of the compute nodes have successfully booted before completing successfully.

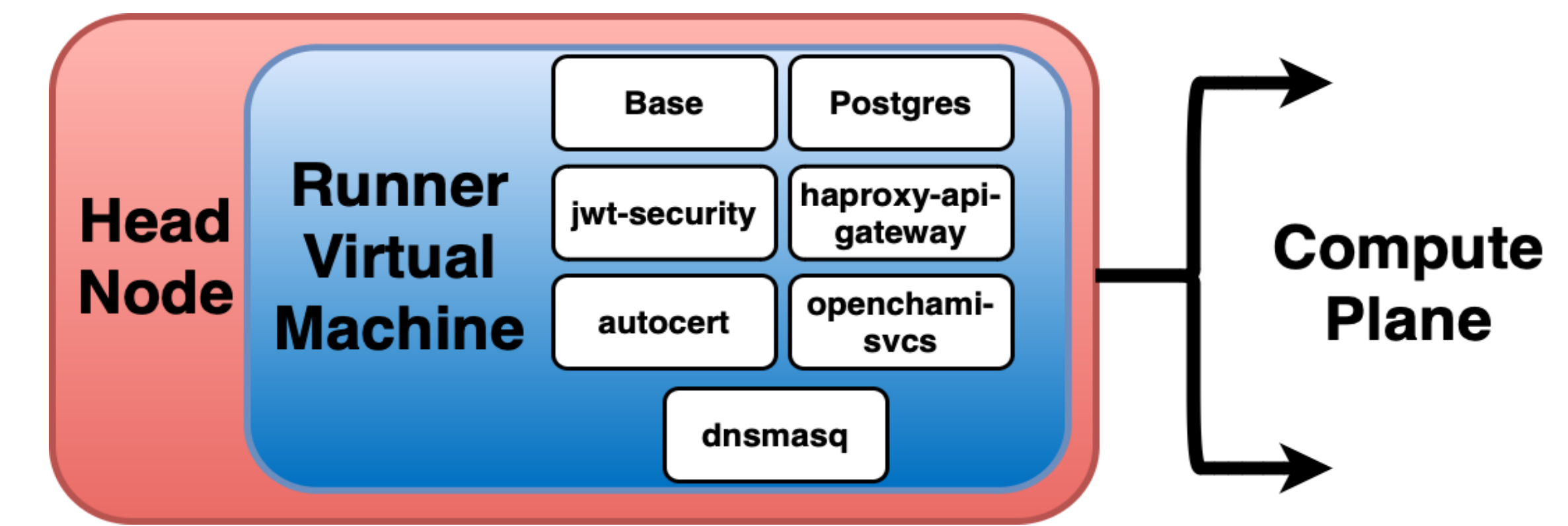


Figure 3 - Structure of Cluster

The cluster consists of a head node which is responsible for S3 as well as hosting the VM. This VM is responsible for booting and managing our 9 compute nodes with the use of OpenCHAMI.

## Methods

### Ephemeral Runner Environment -

The key to implementing our *self-hosted, ephemeral runner* was creating **systemd services**. We developed a systemd service on our head node that continuously checks for a booted virtual machine (VM) and then re-creates the VM once it is no longer running. This provides a clean virtual environment for each GitHub Actions runner that is deployed.

### Establishing systemd on the VM -

Our second **systemd service** is located on our VM. This service is in charge of deploying our *self-hosted, GitHub runner*. It utilizes a **GitHub app token** to register a runner on our GitHub repository. This token is received from a bash script, and it's subsequently passed into the **configuration arguments** of the GitHub actions. This systemd service is in charge of passing all needed command line arguments to the runner, ensuring it's started with the `--ephemeral` tag and assigned a name as well. The runner itself will remain in an **idle** state until a job has been sent to it from the remote GitHub repository. After the completion of the job, the runner automatically unregisters itself.

### Completing the Life Cycle -

The same **systemd service** on the VM is then in charge of self powering off once the GitHub runner has terminated. It is at this point that the head node creates a new virtual machine and the process restarts. This ephemeral life cycle is depicted in **Figure 1**.

## Conclusion

### Time Saved -

**Our full pipeline executes on a 10-node cluster in about 10 minutes!** This automation greatly accelerates the time it takes to develop, test, and deploy OpenCHAMI code.