UberCloud on VMware vSphere:

Introduction:

Enterprise HPC Infrastructure teams supporting engineers have a challenging task keeping up with the latest advances in Hardware and capabilities. Until recently, engineers relied on their workstations and bare metal compute clusters for their HPC simulations. The downside, a workstation is not suitable for running very large, compute intensive simulation jobs and bare metal clusters are not the best solution for an organization. Legacy HPC are usually independent HPC Silos per organization resulting in underutilization of resources. In the article VMWare Solutions Enhance HPC, we showed the tremendous benefits of consolidation and virtualizing HPC infrastructure compared to bare metal environments

Cloud like Capabilities for HPC environments:

HPC Cloud solution providers like UberCloud can combine powerful virtualized HPC hardware with superior management capabilities to help, IT dynamically provision and manage HPC clusters, deploy simulation software applications, monitor their spend, and give them complete control. In this solution we combine the capabilities of the VMware platform, with the solution provided by HPC Cloud provider UberCloud. The UberCloud solution leverages the automation capabilities of Terraform with the unique packaging and distributing capabilities of docker containers to dynamically deploy HPC applications on the vSphere platform. Containers complement VMs in application management and distribution. By sandboxing applications in containers, applications become portable, and the same application can be deployed to different VMs without requiring creating multiple variations of VM images for every application. It speeds up the development and testing, yielding significant acceleration to the change management process. Containers that fail can be removed and replaced without noticeable impact on service.

The core capabilities that are enabled by the solution include:

- Provisioning and managing HPC clusters that are easily setup for users in minutes.
- Rapidly deploying ready-to-run instances with pre-installed HPC software eliminating the need for complex software installation and configuration.
- Optimized use of resources shared across multiple groups of HPC users with scale up and scale down capabilities.

UberCloud on vSphere Solution:

In this solution we deployed two different use cases

- Single node deployment with all components included
- Distributed deployment with a head node, a GUI node and three worker nodes for a distributed workload

Single Node Deployment:

This is a simple deployment of UberCloud on vSphere. A Single VM is deployed and all components are downloaded and installed dynamically. The single node includes the head node, GUI node and the worker node components.

The vCenter access and the components of the deployment are defined in the Terraform file. The single virtual machine with the requisite cores and memory is automatically created in the vCenter at the specified location and the template UberCloud VM image is deployed. Followed by deployment the requisite docker images are downloaded and installed automatically.

The Terraform script is launched and is use to create the VM, download and install docker, configure the application and verify its readiness for use as shown below

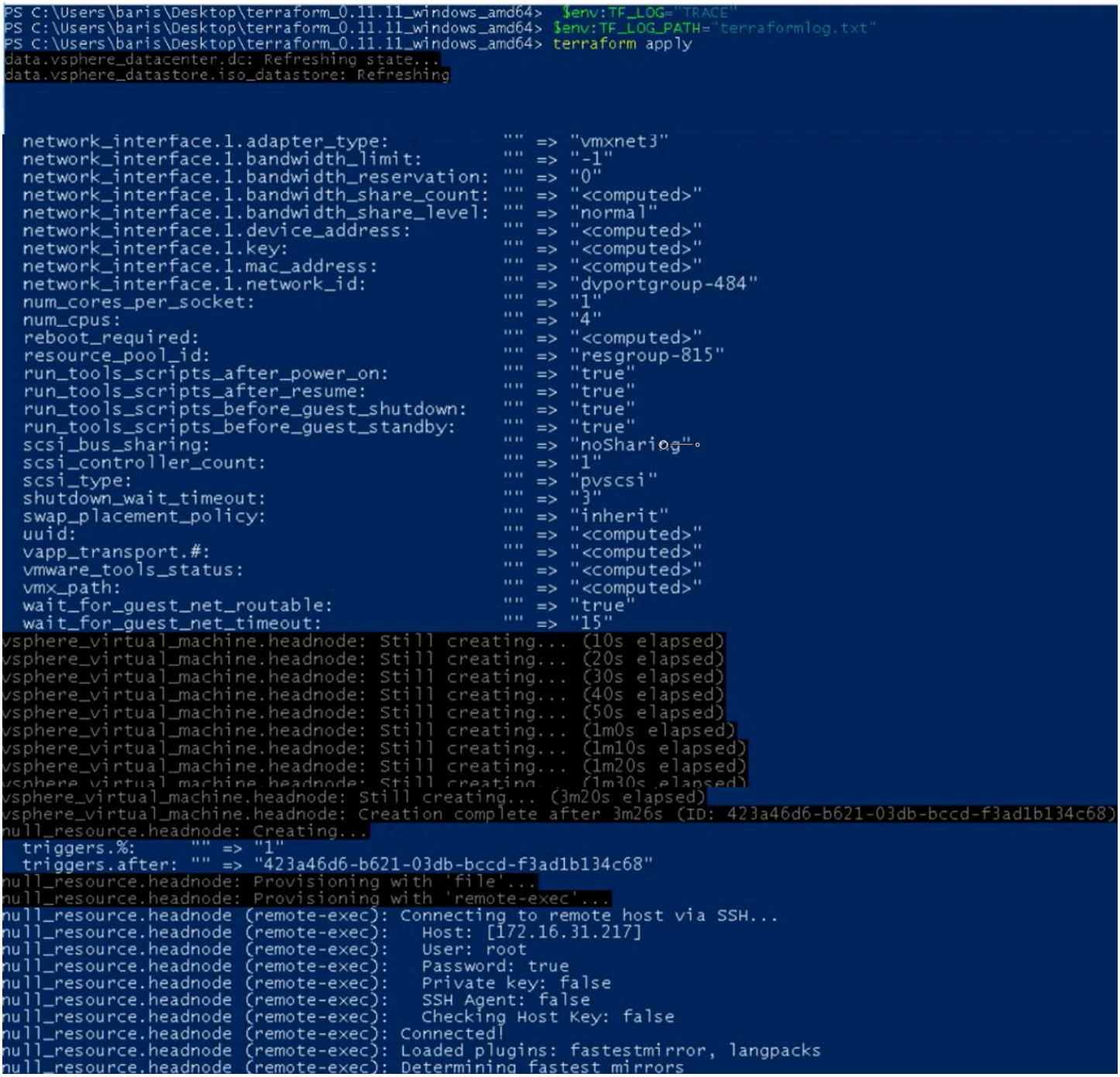

Figure 1: Virtual machine deployment and configuration

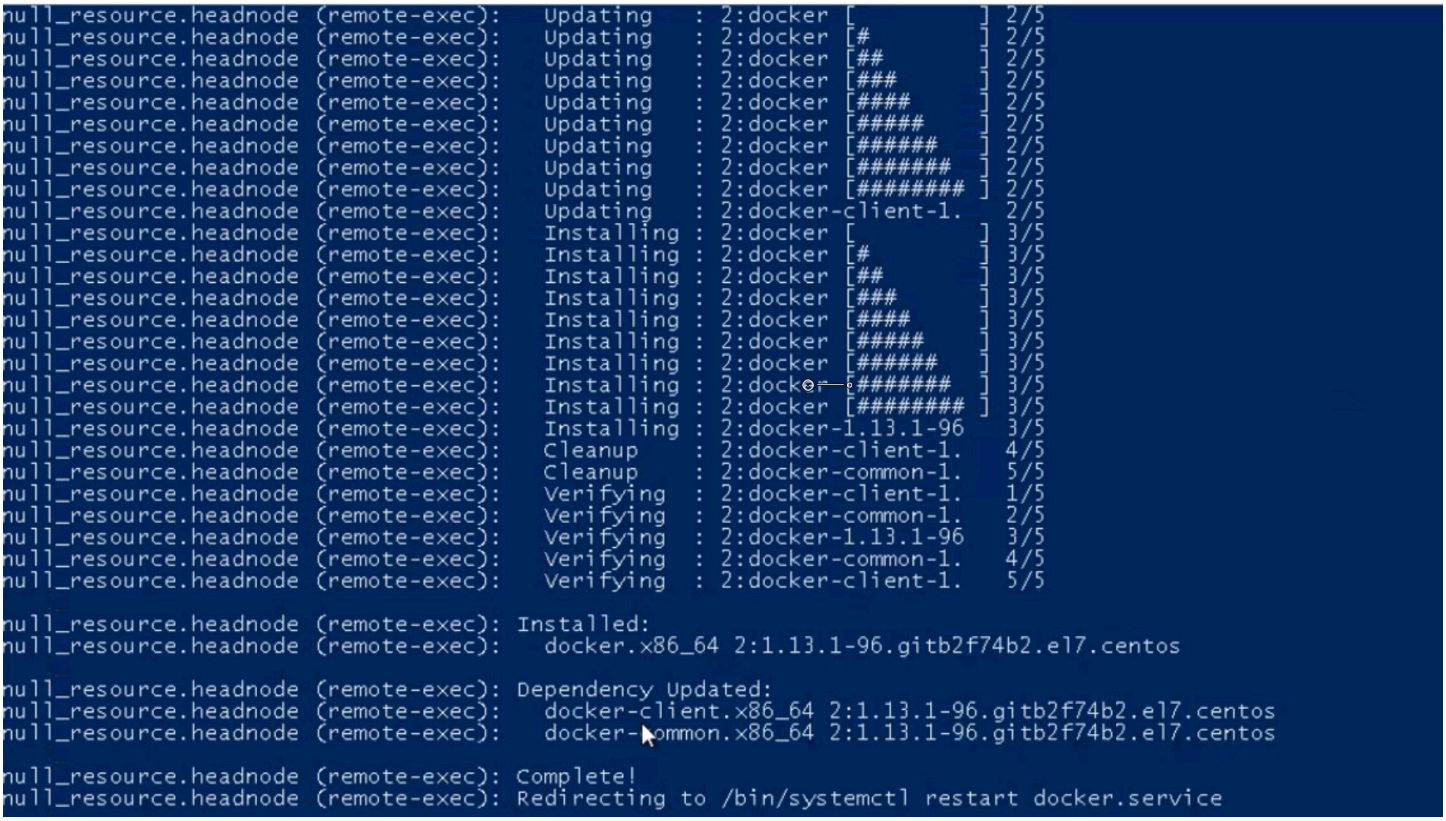

Figure 2: Docker components and containers with HPC applications are installed

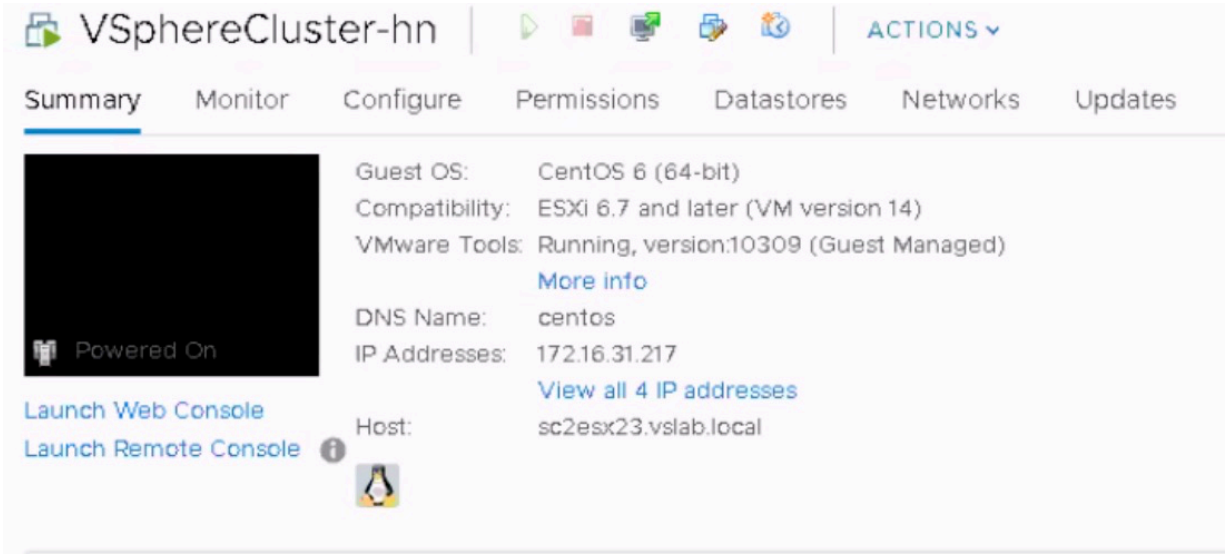

Figure 3: Virtual machine representing the single node deployment

Once all the components are deployed successfully the end user is sent an email with a link to login to the environment along with a uniquely generated password as shown below.

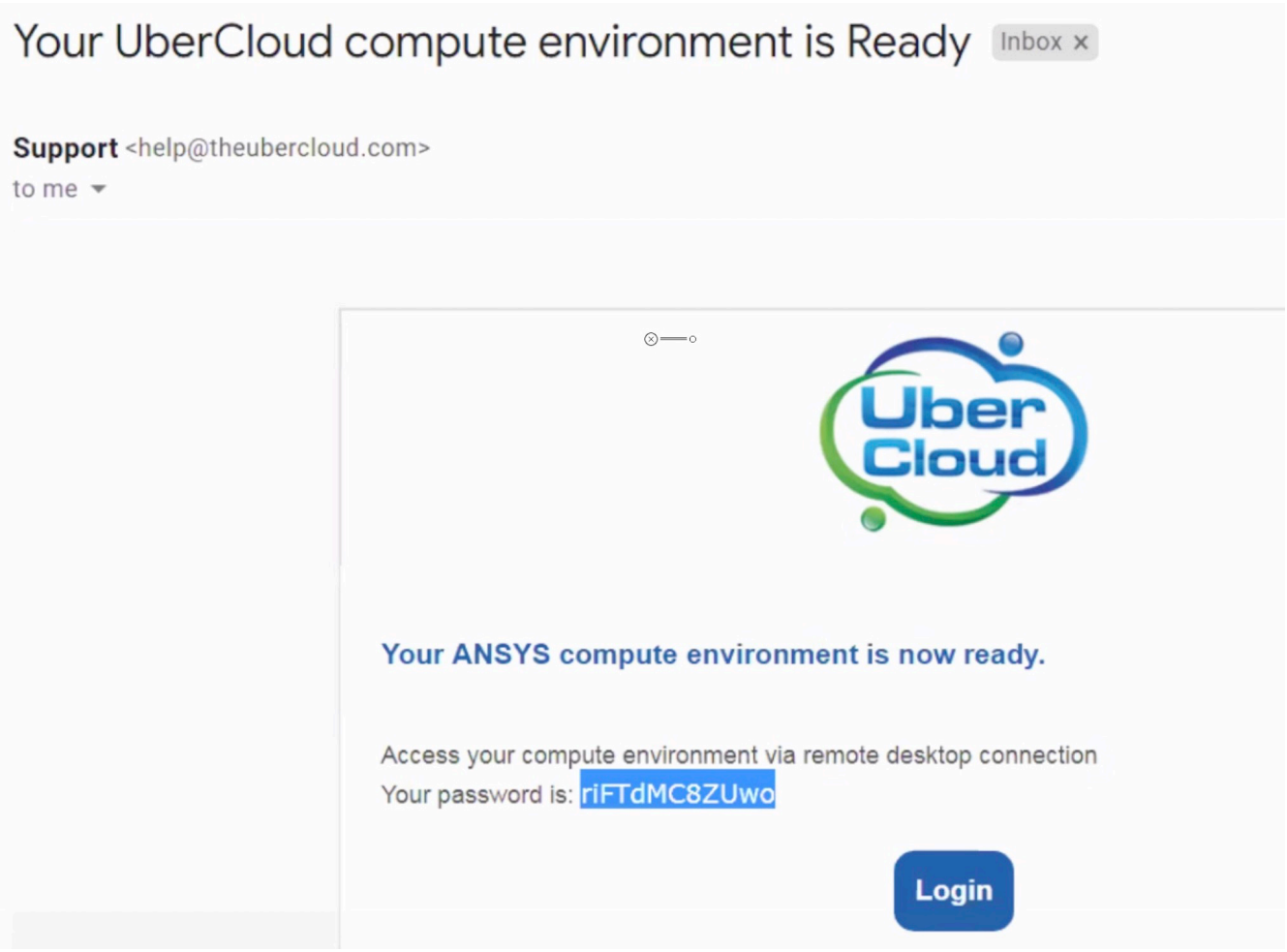

Figure 4: UberCloud email with remote access details

The link and the password can be used to login to the environment with a browser and the user interface on login is as shown below.

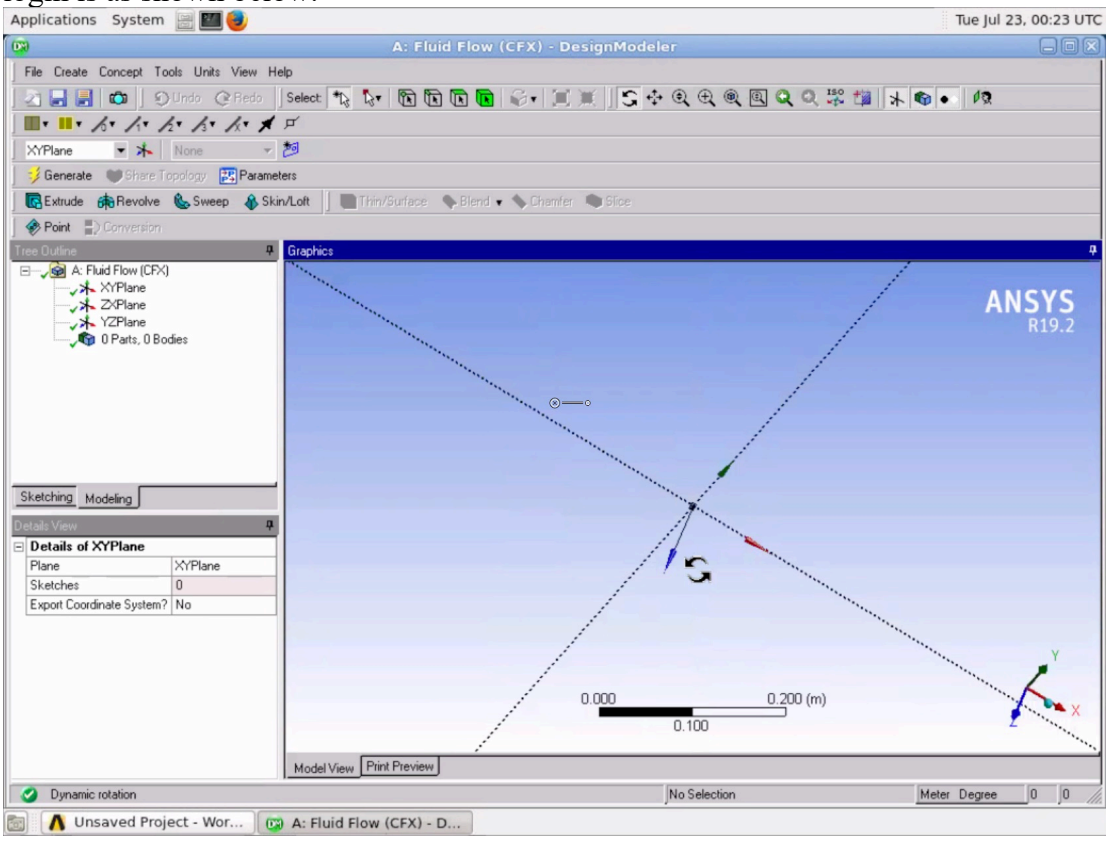

Figure 5: ANSYS application running on the vSphere based private cloud

Distributed Use cases (ANSYS)

HPC uses distributed computed pervasively to solve many of the most difficult problems. Ansys provides the Distributed Solve Option (DSO) as a productivity enhancement tool that accelerates sweeps of design variations by distributing the design parameters across a network of processors. A distributed solution was deployed with UberCloud and ANSYS leveraging Terraform automation as a second phase of the solution.

Distributed Computing solves many high-level business challenges:

- Enterprises can more effectively and robustly utilize their compute resources to optimize their designs.
- Design iterations are faster, delivering new products to market in a fraction of the time.
- Design evaluation is approaching true scalability.
- Businesses can use this flexible heterogeneous computing infrastructure in private and public cloud environments to solve engineering challenges locally or tie into any compute resources worldwide.

Distributed Deployment:

Using similar Terraform based automation a distributed deployment of ANSYS HPC components was accomplished. The virtual machines deployed for the distributed solution as shown below. This deployment uses a GUI node, a head node and three compute nodes. The GUI node was allocated a NVIDIA vGPU for good graphical performance leverage vSphere capabilities with acclerators.

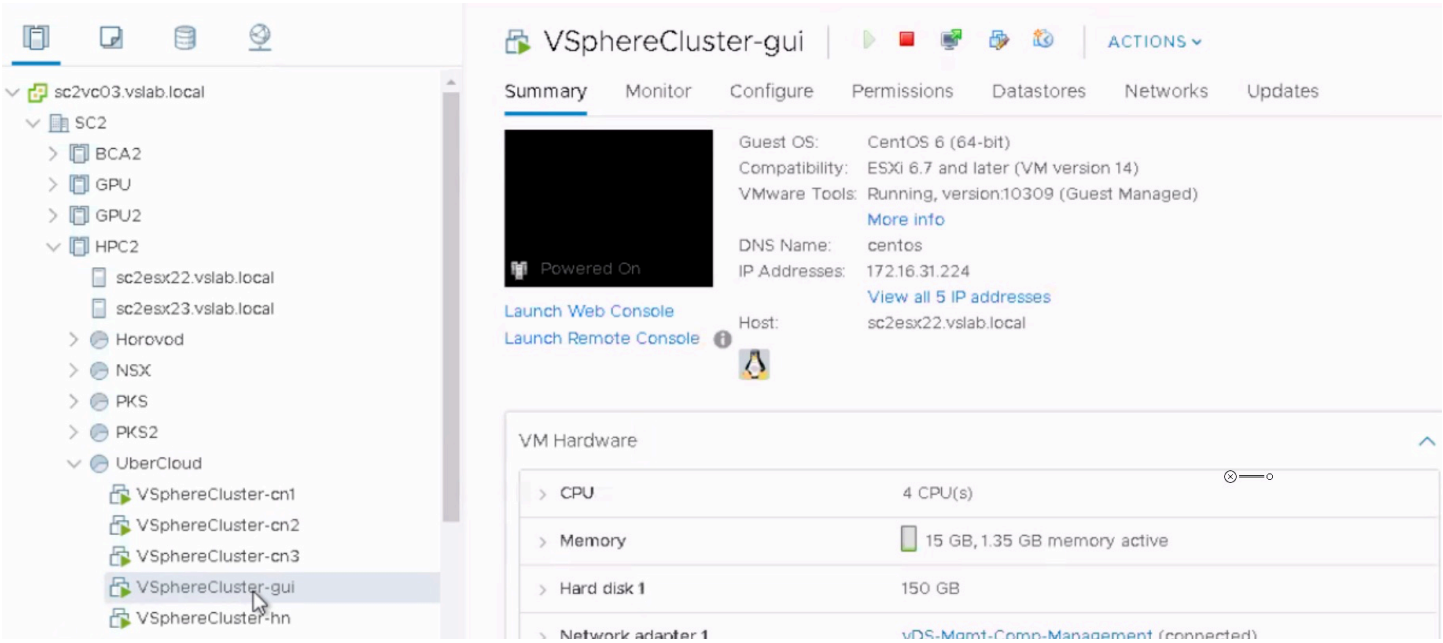

Figure 6: Virtual components of distributed HPC workload

The solution was deployed and a distributed test script was executed on the HPC cluster leveraging ANSYS FLUENT. The script shown below leverages MPI over TCP and uses 12 processors.

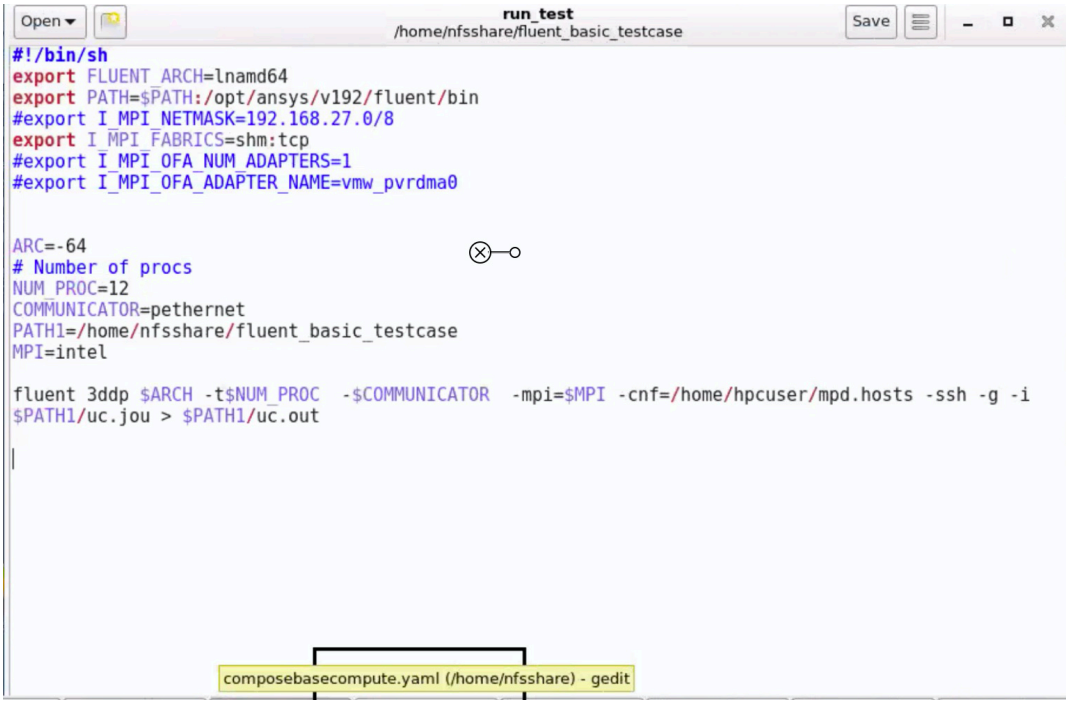

Figure 7: Script for test use case

The image below shows execution of the script with the three worker nodes.

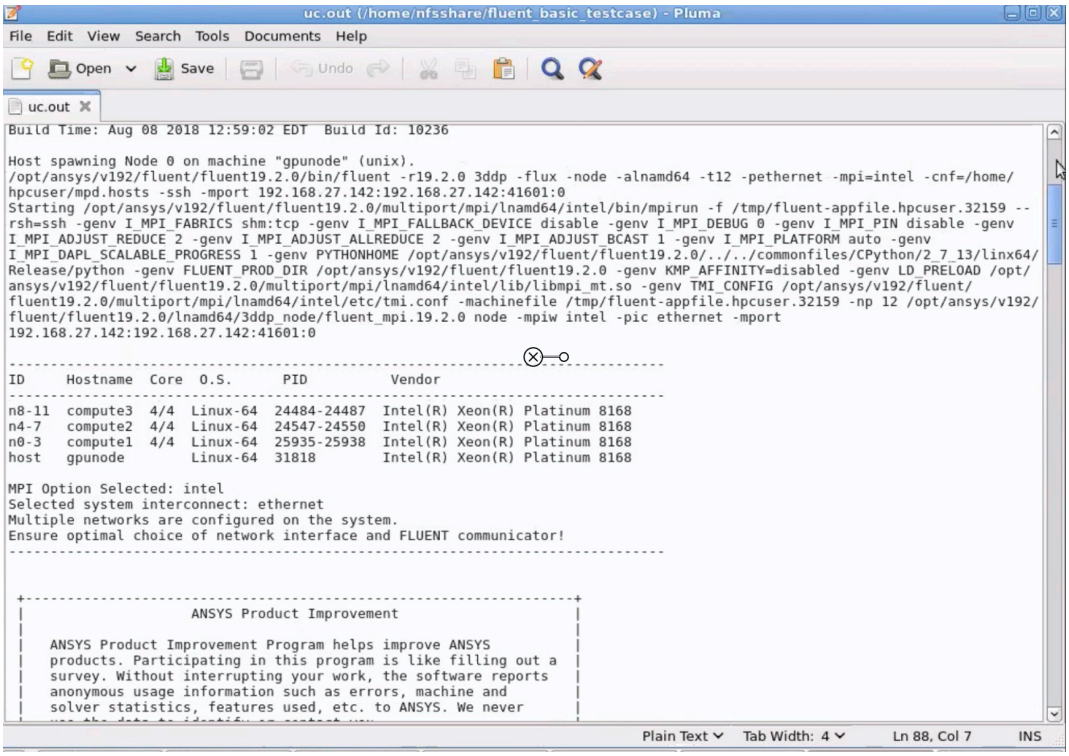

Figure 8: Execution of distributed workload

The computation proceeds to convergence over multiple steps as show below.

```
Updating solution at time level N...
 done.

  iter  continuity  x-velocity  y-velocity  z-velocity     time/iter
!  167 solution is converged
   167  4.0056e-08  4.2138e-06  4.3938e-09  3.3581e-09  0:00:12  100
!  168 solution is converged
   168  2.9120e-08  3.9440e-06  4.4914e-09  3.6130e-09  0:00:10   99
Flow time = 1.06s, time step = 106
4 more time steps

Updating solution at time level N...
 done.

  iter  continuity  x-velocity  y-velocity  z-velocity     time/iter
!  168 solution is converged
   168  2.9120e-08  3.9440e-06  4.4914e-09  3.6130e-09  0:00:10  100
!  169 solution is converged
   169  2.1400e-08  3.6921e-06  4.5005e-09  3.7538e-09  0:00:08   99
Flow time = 1.07s, time step = 107
3 more time steps

Updating solution at time level N...
 done.

  iter  continuity  x-velocity  y-velocity  z-velocity     time/iter
!  169 solution is converged
   169  2.1400e-08  3.6921e-06  4.5005e-09  3.7538e-09  0:00:08  100
!  170 solution is converged
   170  1.5988e-08  3.4567e-06  4.4424e-09  3.7956e-09  0:00:07   99
Flow time = 1.08s, time step = 108
2 more time steps
```

Figure 9: Convergence of the distributed computation

Conclusion:

vSphere is an excellent platform for High Performance computing. This UberCloud based HPC solution on vSphere was deployed with Terraform and successfully demonstrated. All the applications were containerized and hosted within vSphere. The solution show cased the ability to fully package a HPC application with automated deployment and tear down in a matter of minutes. HPC users can be highly productive and get the environment ready on demand rather than having to wait for many weeks or months in the case of bare metal environments. The solution can scale to tens of nodes and also be leveraged for distributed computing.

Appendix A: Terraform Script used for environment creation

```
variable "vsphere_user" {
  default = "xxxx@xxxx.local"
}

variable "vsphere_password" {
  default = "XXXXXX"
}

variable "vsphere_server" {
  default = "sc2vc03.xxxx.local"
}

variable "datacenter_name" {
  default = "SC2"
}

variable "datastore_name" {
  default = "UberCloud01"
}

variable "iso_datastore_name" {
  default = "UberCloud01"
}

variable "resource_pool_name" {
  default = "UberCloud"
}

variable "network_name" {
  default = "vDS-Mgmt-Comp-Management"
```

```
}

variable "network1_name" {
  default = "vDS-Comp-vMotion"
}
variable "network2_name" {
  default = "vDS-1610"
}
variable "vm_template_name" {
  default = "newbscentos"
}

variable "resource_name" {
  default = "centos7-VM"
}

variable "vm_admin_password" {
  default = "******"
}




variable "vm_admin_user" {
  default = "root"
}

variable "headnode_number_of_cpucore" {
  default = "4"
}

variable "headnode_memorysize" {
  default = "15360"
```

```
}

variable "number_of_computenode" {
  default = 3
}

variable "computenode_number_of_cpucore" {
  default = "4"
}

variable "computenode_memorysize" {
  default = "15360"
}

variable "guinode_number_of_cpucore" {
  default = "4"
}

variable "guinode_memorysize" {
  default = "15360"
}

/*****************************************/

variable "cluster_name" {
    default = "VSphereCluster"
}
```

```
locals {

head_node_name = "${var.cluster_name}-hn"
gui_node_name = "${var.cluster_name}-gui"

compute_node_prefix="${var.cluster_name}-cn"
}




variable "customer_email_address" {
  default = "baris.inaloz@theubercloud.com"
}




variable "isv_license_server" {
  default = "2325:1055@ansys-ls118.theubercloud.net"
}

variable "dcv_license_server" {
  default = "5053@ls-001.theubercloud.net"
}

variable "container_ssh_port" {
  default = "22"
```

```
}

variable "container_gui_port" {
  default = "443"
}

/*****************************************/

variable "docker_registry_login_uri" {
  default = "registry.theubercloud.com"
}

variable "docker_registry_username" {
  default = "baris"

}

variable "docker_registry_password" {
  default = "nnobankimun1001"
}

variable "container_image_uri" {
  default = "registry.theubercloud.com/library_staging/ansys_19.2_centos_7"
}

/*********************************************/
```

```
provider "vsphere" {
  user           = "${var.vsphere_user}"
  password       = "${var.vsphere_password}"
  vsphere_server = "${var.vsphere_server}"

  # If you have a self-signed cert
  allow_unverified_ssl = true
}

data "vsphere_datacenter" "dc" {
  name = "${var.datacenter_name}"
}




data "vsphere_datastore" "datastore" {
  name          = "${var.datastore_name}"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

data "vsphere_datastore" "iso_datastore" {
  name          =  "${var.iso_datastore_name}"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}




data "vsphere_resource_pool" "pool" {
  name          = "${var.resource_pool_name}"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}
```

```
data "vsphere_network" "network" {
  name          = "${var.network_name}"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

data "vsphere_network" "network1" {
  name          = "${var.network1_name}"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}
data "vsphere_network" "network2" {
  name          = "${var.network2_name}"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}

data "vsphere_network" "emptynetwork" {
  name          = "VM Network"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}



data "vsphere_virtual_machine" "template" {
  name          = "${var.vm_template_name}"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}
```

```
resource "vsphere_virtual_machine" "headnode" {
  name             =  "${local.head_node_name}"
  resource_pool_id = "${data.vsphere_resource_pool.pool.id}"
  datastore_id     = "${data.vsphere_datastore.datastore.id}"

  num_cpus = "${var.headnode_number_of_cpucore}"
  memory   = "${var.headnode_memorysize}"
  guest_id = "${data.vsphere_virtual_machine.template.guest_id}"
  scsi_type = "${data.vsphere_virtual_machine.template.scsi_type}"



  network_interface {
    network_id   = "${data.vsphere_network.network.id}"
    adapter_type = "${data.vsphere_virtual_machine.template.network_interface_types[0]}"
  }
  network_interface {
    network_id   = "${data.vsphere_network.network2.id}"
    adapter_type = "${data.vsphere_virtual_machine.template.network_interface_types[0]}"
  }



  disk {
    label            = "disk0"
    size             = "${data.vsphere_virtual_machine.template.disks.0.size}"
    size             = 160
    eagerly_scrub    = "${data.vsphere_virtual_machine.template.disks.0.eagerly_scrub}"
    thin_provisioned = "${data.vsphere_virtual_machine.template.disks.0.thin_provisioned}"
  }

wait_for_guest_net_timeout=15
```

```
 clone {
    template_uuid = "${data.vsphere_virtual_machine.template.id}"

  }

}

resource "null_resource" "headnode" {

connection {
      host="${vsphere_virtual_machine.headnode.*.default_ip_address}"
      user     = "${var.vm_admin_user}"
      password = "${var.vm_admin_password}"
  }

   provisioner "file" {
    source      = "compose.yaml"
    destination = "/home/centos/compose.yaml"
  }

  provisioner "remote-exec" {

    inline = [
      "${local.headnode_init_script}",
    ]

  }

  triggers = {
    "after" = "${vsphere_virtual_machine.headnode.id}"
```

```
}

resource "vsphere_virtual_machine" "guinode" {
  name             =  "${local.gui_node_name}"
  resource_pool_id = "${data.vsphere_resource_pool.pool.id}"
  datastore_id     = "${data.vsphere_datastore.datastore.id}"

  num_cpus = "${var.guinode_number_of_cpucore}"
  memory   = "${var.guinode_memorysize}"
  guest_id = "${data.vsphere_virtual_machine.template.guest_id}"
  scsi_type = "${data.vsphere_virtual_machine.template.scsi_type}"



  network_interface {
    network_id   = "${data.vsphere_network.network.id}"
    adapter_type = "${data.vsphere_virtual_machine.template.network_interface_types[0]}"
  }
  network_interface {
    network_id   = "${data.vsphere_network.network2.id}"
    adapter_type = "${data.vsphere_virtual_machine.template.network_interface_types[0]}"
  }



  disk {
    label            = "disk0"
    size             = "${data.vsphere_virtual_machine.template.disks.0.size}"
    eagerly_scrub    = "${data.vsphere_virtual_machine.template.disks.0.eagerly_scrub}"
    thin_provisioned = "${data.vsphere_virtual_machine.template.disks.0.thin_provisioned}"
  }
```

```
wait_for_guest_net_timeout=15




 clone {

    template_uuid = "${data.vsphere_virtual_machine.template.id}"


  }


}


resource "null_resource" "guinode" {


connection {

      host="${vsphere_virtual_machine.guinode.*.default_ip_address}"

      user     = "${var.vm_admin_user}"

      password = "${var.vm_admin_password}"

  }


   provisioner "file" {

    source      = "compose.yaml"

    destination = "/home/centos/compose.yaml"

  }


  provisioner "remote-exec" {


    inline = [

      "${local.gui_init_script}",

    ]


  }
```

```
  triggers = {
    "after" = "${vsphere_virtual_machine.guinode.id}"
  }
}




resource "vsphere_virtual_machine" "computenode" {

    depends_on = ["vsphere_virtual_machine.headnode","vsphere_virtual_machine.guinode"]

  count ="${var.number_of_computenode}"
  name             = "${local.compute_node_prefix}${count.index+1}"
  resource_pool_id = "${data.vsphere_resource_pool.pool.id}"
  datastore_id     = "${data.vsphere_datastore.datastore.id}"

  num_cpus = "${var.computenode_number_of_cpucore}"
  memory   = "${var.computenode_memorysize}"
  guest_id = "${data.vsphere_virtual_machine.template.guest_id}"
  scsi_type = "${data.vsphere_virtual_machine.template.scsi_type}"

  network_interface {
    network_id   = "${data.vsphere_network.network.id}"
    adapter_type = "${data.vsphere_virtual_machine.template.network_interface_types[0]}"
  }
  network_interface {
    network_id   = "${data.vsphere_network.network2.id}"
    adapter_type = "${data.vsphere_virtual_machine.template.network_interface_types[0]}"
  }
```

```
  disk {
    label            = "disk0"
    size             = "${data.vsphere_virtual_machine.template.disks.0.size}"
    eagerly_scrub    = "${data.vsphere_virtual_machine.template.disks.0.eagerly_scrub}"
    thin_provisioned = "${data.vsphere_virtual_machine.template.disks.0.thin_provisioned}"
  }
wait_for_guest_net_timeout=15
 clone {
    template_uuid = "${data.vsphere_virtual_machine.template.id}"


  }
}
resource "null_resource" "computenode" {
  count = "${var.number_of_computenode}"
  triggers {
    cluster_instance_ids = "${vsphere_virtual_machine.headnode.id}"
  }
connection {
      host="${element(vsphere_virtual_machine.computenode.*.default_ip_address, count.index)}"
      user     = "${var.vm_admin_user}"
      password = "${var.vm_admin_password}"
  }
   provisioner "file" {
    source      = "compose.yaml"
    destination = "/home/centos/compose.yaml"
  }
  provisioner "remote-exec" {
    inline = [
      "${local.compute_init_script}",
    ]
  }
```

ſ