openflightHPC

Next Generation HPC on Any Platform

An Example of Building OpenFlight Compute Clusters in Azure

Using open-source software to create a user friendly cloud cluster environment

Stu Franks - OpenFlightHPC
October 2019

Contents

Overview

OpenFlightHPC

OpenFlightHPC is an open-source community project developing a flexible, functional and stable HPC stack that can be launched on any platform.

The project establishes guides, best practices and helpful tools for both users and admins to create and run intuitive HPC environments on various platforms.

Microsoft Azure

Azure is a public cloud platform provided by Microsoft. As with most cloud providers, Azure has many datacentres across the world to handle the server load and deliver optimal performance to clients who use datacentres in their region. Azure provides many different applications that can be installed with a few clicks from the web interface - from databases to entire virtual machines.

Ansible

Ansible is an automation and configuration tool. It provides an abstraction to scripting that allows for consistent, reproducible, bulk deployment of OS and app configurations to systems.

Useful Links

OpenFlightHPC Documentation - https://openflighthpc.org/docs.html

Microsoft Azure Documentation - https://docs.microsoft.com/en-us/azure/

Ansible Documentation - https://docs.ansible.com

Summary: The Azure Template

Overview

Azure templates provide a solution to deploying many resources at once. The template used for creating an OpenFlight Compute cluster creates:

- A virtual network and security group
- A gateway node with a public IP address
- Between 2 and 8 compute nodes with public IP addresses

The entire template can be seen at https://github.com/openflighthpc/openflight-compute-cluster-builder/blob/master/templates/azure/cluster.json

Parameters

Parameters allow for deployment-specific resource configuration, this allows the template to be more flexible whilst retaining reproducibility. The template makes use of the following parameters:

- `sshPublicKey` - The value of this parameter will be set as the accepted public key in ~flight/.ssh/authorized_keys for ssh login to the gateway and compute nodes
- `sourceimage` - This value specifies the Azure Resource ID of the image to use for both gateway and compute nodes
- `clustername` - The desired name for the cluster, this will be displayed in the prompt within OpenFlight Env and be used as part of the FQDN for the resources
- `customdata` - Cloud-init custom data, encoded in base64, to be passed to all resources
- `computeNodesCount` - The number of compute nodes to deploy, between 2 and 8

Resources

In order for the compute resources to communicate with one another and securely with the wider world, the following resources are created:

- *Virtual Network* - The virtual network uses the address space 10.10.0.0/16 with the primary subnet for the cluster being 10.10.0.0/19
- *Security Group* - The security group is configured to only allow SSH access to the nodes from outside the cluster

The gateway and compute nodes are deployed with the same resources and dependencies. A node breaks down to:

- *Public IP Address* - An internet facing IP address that allows external access directly to the node
- *Primary Network Interface* - This network interface will get a DHCP IP address in the Virtual Network
- *Virtual Machine* - The machine itself, created from the source image provided as a parameter with a 16GB root filesystem

The deployed resources are the basis for the OpenFlight Compute cluster, however it is not complete until the Ansible playbook is run across the nodes.

Summary: The Ansible Playbook

Overview

An Ansible playbook consists of multiple roles which implement the OpenFlight Compute environment and additional configuration to setup NFS shares and SLURM.

Variables

There are a number of variables within the `group_vars/` directory of the Ansible playbook that provide default configuration for the Flight Environment. These variables define roles for the different nodes to configure them accordingly. The variables are:

- *Global Variables* - The cluster name, gateway node name and compute node range used in various parts of the ansible playbook
- *FlightEnv Settings* - Various settings to enable/disable bootstrapping the OpenFlight Environment with dependencies for software and desktop environments
- *SLURM Settings* - Defines the roles and munge authentication key for the SLURM queue system
- *NFS Settings* - Defines the roles and NFS mount information for sharing directories from the gateway to compute nodes

Roles

The roles in the playbook contain multiple tasks which provide the configuration for the given role. Each role is explained in more depth below.

Upstream

The upstream rule installs upstream package repositories for use with yum to install additional packages. This provides the EPEL repository as well as the OpenFlight Tools and OpenFlight SLURM repos.

SLURM

This role installs the dependencies, configures the database and SLURM for the cluster. Using the gateway node as the slurm controller and the compute nodes as available computation resources.

NFS

Installs various NFS utilities on the nodes with all exports being configured on the gateway and mounted by the compute nodes.

FlightEnv

Installs the various OpenFlight tools to support the compute environment. Additionally installs the pdsh tool and a genders file for improving node management and usage.

FlightEnv Bootstrap

This role provides additional configuration of the Flight Environment to get users going quickly. It installs requirements for most of the software and desktop environments to reduce initialisation time of new environments.

Find the playbook at https://github.com/openflighthpc/openflight-ansible-playbook

Summary: The Cluster Builder Scripts

Overview

In addition to the Azure template and ansible playbook there is a small project to tie the two together, openflight-compute-cluster-builder, that wraps around some of the essential commands for deploying azure resources and running the ansible playbook with instance-specific variables.

Config File

The config file, `config.sh`, contains variables for tweaking and customising deployment, namely:

- `PLATFORM` - This defaults to azure as this is the only supported platform in the builder as of writing although AWS support is in the pipeline
- `COMPUTENODES` - This is an integer between 2 and 8 for the number of compute nodes to deploy with the cluster
- `FLIGHTENVPREPARE` - This determines whether to run the additional bootstrapping of the Flight Environment, if set to true then the role in ansible will be toggled on
- `AZURE_SOURCEIMAGE` - This is empty by default and requires a value in order for the deployment scripts to work, the build documentation has more information on using OpenFlight-provided images in your Azure account
- `AZURE_LOCATION` - The Azure region to deploy resources to, this should be the same as the region that the source image is in

Azure Template

The Azure template described earlier is part of this repository and is used by the script to deploy resources.

Logs

The logs directory will contain a file called deploy.log that tracks the start and end deployment times along with other useful information for identifying and accessing deployed clusters.

Build Script

The build script itself takes a few arguments in order to build the cluster. Both a cluster name and public ssh key will need to be provided in order to deploy the cluster.

What the build script does is:

- Initiates variables variables from config.sh and additional variables:
  - `DIR` - The directory that the script is stored in, this is used for locating template files and the log directory
  - `CLUSTERNAMEARG` - The first argument to the script, this is the clean name of the cluster
  - `CLUSTERNAME` - To reduce the risk of issues occurring if multiple clusters with the same name are built, the clean cluster name is appended with a random seed string to ensure resource groups are unique
  - `SSH_PUB_KEY` - The second argument to the script, this is the public SSH access key to provide access to the cluster
- Creates an Azure resource group to contain the cluster

```
az group create --name "$CLUSTERNAME" --location "$AZURE_LOCATION"
```

- Deploys the cluster template resources into the resource group

```
az group deployment create --name "$CLUSTERNAME" --resource-group
"$CLUSTERNAME" \
--template-file $DIR/templates/azure/cluster.json \
--parameters sshPublicKey="$SSH_PUB_KEY" \
sourceimage="$AZURE_SOURCEIMAGE" \
clustername="$CLUSTERNAMEARG" \
computeNodesCount="$COMPUTENODES" \
customdata="$(cat $DIR/templates/cloudinit.txt |base64 -w 0)"
```

- Configure an Ansible hosts file for the cluster

```
cat << EOF > /opt/flight/clusters/$CLUSTERNAME
[gateway]
gateway1        ansible_host=$(az network public-ip show -g
$CLUSTERNAME    -n    flightcloudclustergateway1pubIP    --query
"{address: ipAddress}" --output yaml |awk '{print $2}')
[nodes]
$(i=1 ; while [ $i -le $COMPUTENODES ] ; do
echo "node0$i      ansible_host=$(az network public-ip show -g
$CLUSTERNAME    -n    flightcloudclusternode0$i\pubIP    --query
'{address: ipAddress}' --output yaml |awk '{print $2}')"
i=$((i + 1))
done)
EOF
```

- Waits for the cluster to be accessible
- Runs the ansible playbook on all the resources

```
ansible-playbook    -i    /opt/flight/clusters/$CLUSTERNAME    \
--extra-vars "cluster_name=$CLUSTERNAMEARG \
munge_key=$( (head /dev/urandom | tr -dc a-z0-9 | head -c 18 ;
echo '') | sha512sum | cut -d' ' -f1) \
compute_nodes=node[01-0$COMPUTENODES] \
flightenv_bootstrap=true" openflight.yml
```

- Logs the ending deployment time along with the public IP address for the gateway node

All the cluster builder scripts can be found at https://github.com/openflighthpc/openflight-compute-cluster-builder

In Depth: The Azure Template

This section provides a detailed look at the entire Azure template.

The template starts by initialising the schematic and Azure versioning.

```
{
  "$schema":
"https://schema.management.azure.com/schemas/2015-01-01/deploymen
tTemplate.json#",
  "contentVersion": "1.0.0.0",
```

The parameters key is next, this will contain definitions and constraints for the various parameters used in the template.

```
  "parameters": {
```

First, the SSH key parameter is defined as a string. The metadata description provides further clarification of the expected value.

```
      "sshPublicKey": {
           "type": "string",
           "metadata": {
               "description": "SSH public key for flight user"
           }
       },
```

Next, the source image parameter is defined as a string with the metadata clarifying the expected value.

```
      "sourceimage": {
           "type": "string",
           "metadata": {
               "description": "Source image to use for nodes"
           }
       },
```

The cluster name is the next variable to be defined as a string.

```
      "clustername": {
           "type": "string",
           "metadata": {
               "description": "Name of the cluster"
           }
       },
```

The custom data variable expects a single line, base64 encoded string containing cloud-init customisation directives.

```
      "customdata": {
           "type": "string",
           "metadata": {
               "description": "Cloud-init customdata for all
systems encoded in base64"
           }
       },
```

Lastly, the number of compute nodes to create is defined as an integer and constrained to be between the values of 2 and 8. Then the parameters key is closed.

```
      "computeNodesCount": {
           "type": "int",
           "defaultValue": 2,
           "minValue": 2,
           "maxValue": 8,
           "metadata": {
               "description": "Number of compute nodes to include
in cluster"
           }
       }
   },
```

A single variable is defined for the template to improve the convenience of accessing the subnet ID.

```
  "variables": {
       "subnet1Ref":
"[resourceId('Microsoft.Network/virtualNetworks/subnets',
'flightcloudclusternetwork', 'flightcloudclusternetwork1')]"
   },
```

A majority of the template contains the resource array

```
"resources": [
```

The first resource to be defined is the virtual network, it is initialised with a generic name (flightcloudclusternetwork). The address space is configured to be 10.10.0.0/16 with a single subnet being defined with a generic name (flightcloudclusternetwork1) which uses the 10.10.0.0/19 network range.

```
    {
       "type": "Microsoft.Network/virtualNetworks",
       "name": "flightcloudclusternetwork",
       "apiVersion": "2017-03-01",
       "location": "[resourceGroup().location]",
       "properties": {
         "addressSpace": {
           "addressPrefixes": [
             "10.10.0.0/16"
           ]
         },
         "subnets": [
           {
             "name": "flightcloudclusternetwork1",
             "properties": {
               "addressPrefix": "10.10.0.0/19"
             }
           }
         ]
       }
     },
```

The next resource is the security group containing firewall rules for cluster access. This allows SSH (TCP on port 22) and up to 10 VNC session ports (TCP on ports 5901-5010) through the cluster firewall.

```
    {
       "type": "Microsoft.Network/networkSecurityGroups",
       "name": "flightcloudclustersecuritygroup",
       "apiVersion": "2017-03-01",
       "location": "[resourceGroup().location]",
       "properties": {
         "securityRules": [{
            "name": "inbound-ssh",
              "properties": {
              "protocol": "TCP",
              "sourcePortRange": "*",
              "destinationPortRange": "22",
              "sourceAddressPrefix": "*",
              "destinationAddressPrefix": "*",
              "access": "Allow",
              "priority": 1000,
              "direction": "Inbound"
            }
          },
          {
          "name": "inbound-vnc",
            "properties": {
              "protocol": "TCP",
              "sourcePortRange": "*",
              "destinationPortRange": "5901-5910",
              "sourceAddressPrefix": "*",
              "destinationAddressPrefix": "*",
              "access": "Allow",
              "priority": 1010,
              "direction": "Inbound"
            }
          }]
        }
     },
```

The next resource is the public IP address for the gateway node, it is configured to have a static IP address so there's no risk of the IP changing while the cluster is up. The DNS name is set to "gateway" plus the cluster name.

```
    {
       "type": "Microsoft.Network/publicIPAddresses",
       "name": "flightcloudclustergateway1pubIP",
       "apiVersion": "2017-03-01",
       "location": "[resourceGroup().location]",
       "properties": {
         "publicIPAllocationMethod": "Static",
         "idleTimeoutInMinutes": 30,
         "dnsSettings": {
           "domainNameLabel": "[concat('gateway1-', 
parameters('clustername'))]"
         }
       }
     },
```

The network interface for the gateway is the next to be defined. It's configured to dynamically receive an IP address, use the public IP defined above for reaching the internet, use the previously defined subnet variable for the private network IP allocation, use the network security group created earlier for access control and is to delay creating this resource until the public IP has been created.

```
    {
       "type": "Microsoft.Network/networkInterfaces",
       "name": "flightcloudclustergateway1network1interface",
       "apiVersion": "2017-03-01",
       "location": "[resourceGroup().location]",
       "properties": {
         "ipConfigurations": [{
           "name": "flightcloudclustergateway1network1ip",
           "properties": {
             "privateIPAllocationMethod": "Dynamic",
             "publicIPAddress": {
               "id":
"[resourceId('Microsoft.Network/publicIpAddresses',
'flightcloudclustergateway1pubIP')]"
             },
             "subnet": {
               "id": "[variables('subnet1Ref')]"
             }
           }
         }],
         "networkSecurityGroup": {
           "id":
"[resourceId('Microsoft.Network/networkSecurityGroups',
'flightcloudclustersecuritygroup')]"
         }
       },
       "dependsOn": [
         "[resourceId('Microsoft.Network/publicIpAddresses',
'flightcloudclustergateway1pubIP')]"
       ]
     },
```

Next, the virtual machine definition for the gateway node is started.

```
   {
      "type": "Microsoft.Compute/virtualMachines",
      "name": "flightcloudclustergateway1",
      "apiVersion": "2016-04-30-preview",
      "location": "[resourceGroup().location]",
      "properties": {
```

Within the properties for the virtual machine, the hardware profile is set to use the DS1_v2 instance type.

```
       "hardwareProfile": {
              "vmSize": "Standard_DS1_v2"
            },
```

The storage profile defines the disks in the gateway node, the OS disk is configured to be 128GB in size, being created from an image (which is passed through in the source image parameter).

```
       "storageProfile": {
          "osDisk": {
            "createOption": "fromImage",
            "diskSizeGB": "128",
            "managedDisk": {
              "storageAccountType": "Premium_LRS"
            }
          },
          "imageReference": {
            "id": "[parameters('sourceimage')]"
          }
        },
```

The OS configuration is next, here the FQDN of the gateway and the admin username are set, the cloud-init custom data is passed through from the parameter and the SSH public key is put in place from the parameter.

```
        "osProfile": {
          "computerName": "[concat('gateway1.pri.',
parameters('clustername'), '.cluster.local')]",
          "adminUsername": "flight",
          "customdata": "[parameters('customdata')]",
          "linuxConfiguration": {
            "disablePasswordAuthentication": true,
            "ssh": {
            "publicKeys": [{
              "path": "[concat ('/home/flight',
'/.ssh/authorized_keys')]",
              "keyData": "[parameters('sshPublicKey')]"
              }]
            }
          }
        },
```

To ensure the gateway comes up in the correct order, a dependency on the network interface is put in place.

```
     "dependsOn": [
        "[resourceId('Microsoft.Network/networkInterfaces',
'flightcloudclustergateway1network1interface')]"
       ]
     },
```

The following resource is the public IP address for all the compute nodes. This is configured the same as the gateway node's public IP address except that the copy function of ARM templates is used to create a loop that will iteratively create multiple public IP addresses.

```
    {
       "type": "Microsoft.Network/publicIPAddresses",
       "name": "[concat('flightcloudclusternode0', copyindex(1),
'pubIP')]",
       "apiVersion": "2017-03-01",
       "location": "[resourceGroup().location]",
       "copy": {
           "name": "pubLoop",
           "count": "[parameters('computeNodesCount')]"
       },
       "properties": {
         "publicIPAllocationMethod": "Static",
         "idleTimeoutInMinutes": 30,
         "dnsSettings": {
           "domainNameLabel": "[concat('node0', copyindex(1), '-',
parameters('clustername'))]"
         }
       }
     },
```

The rest of the template files includes the definition of the network interfaces and virtual machines for the compute nodes. These, again, are configured in the exact same way as the gateway node except that the definitions use the copy ability to generate multiple resources.

In Depth: The Ansible Playbook

Playbook File

The file used for defining the roles of the playbook is described below. This file is called openflight.yml and contains the following.

```
- name: Configure HPC Environment for OpenFlightHPC Compute
  hosts: all
  remote_user: root
  roles:
    - upstream
    - slurm
    - nfs
    - flightenv
    - flightenv-bootstrap
```

This adds a description for the playbook, sets it to run on all hosts from a given hosts file (the roles and variables themselves handle exclusion of specific tasks on nodes) and ensures that the roles are run as the root user.

Variable Files

There are multiple files stored under group_vars that define variables globally (to all roles) and some specifically for different groups (whilst still being global to all roles).

The All File

This file contains variables that are applied to all nodes, regardless of group, and is broken down in further detail below.

The first entries in the variable file are considered global in the sense that multiple roles make use of this information. The cluster name, gateway node name and compute node range (in genders format) are specified here.

```
# Global
cluster_name: mycluster # Used by SLURM and Flight Env
gateway_node: gateway1 #Used by SLURM and Flight Env (genders)
compute_nodes: node[01-08]
```

Next settings are for FlightEnv, currently the only variable is a boolean to set whether to perform additional bootstrapping tasks.

```
# FlightEnv Settings

## If true will additionally install dependencies for desktop
environments.
## Only recommended for gateway nodes!
flightenv_bootstrap: false
```

Following the FlightEnv variables are SLURM variables. Here a global default slurm role is set (to an execution node) and a placeholder munge key is in place.

```
# SLURM Settings
slurm_role: exec # Default to SLURM compute node, override in
subgroups
munge_key:
ReplaceThisWithMungeKeyRandomisedStringOrSomethingElseReallySecur
e
```

Following the SLURM variables are the NFS variables, this starts off with a role definition.

```
# NFS Settings
nfs_role: client # Default to NFS client, override in subgroups
```

Each export/mount is contained within its own key. The first definition is an export of the directory /export/apps from the server gateway1 to be mounted at /opt/apps. Both the export and mount have mount options defined.

```
 /export/apps:
    server: gateway1
    mountpoint: /opt/apps
    export_opts: 10.10.0.0/255.255.0.0(rw,no_root_squash,sync)
    mount_opts: intr,rsize=32768,wsize=32768,vers=3,_netdev
```

There are 4 other NFS mounts which are defined in the same way by default.

```
 /export/data:
    server: gateway1
    mountpoint: /data
    export_opts: 10.10.0.0/255.255.0.0(rw,no_root_squash,sync)
    mount_opts: intr,rsize=32768,wsize=32768,vers=3,_netdev
  /export/service:
    server: gateway1
    mountpoint: /opt/service
    export_opts: 10.10.0.0/255.255.0.0(rw,no_root_squash,sync)
    mount_opts: intr,rsize=32768,wsize=32768,vers=3,_netdev
  /export/site:
    server: gateway1
    mountpoint: /opt/site
    export_opts: 10.10.0.0/255.255.0.0(rw,no_root_squash,sync)
    mount_opts: intr,rsize=32768,wsize=32768,vers=3,_netdev
  /home:
    server: gateway1
    mountpoint: /home
    export_opts: 10.10.0.0/255.255.0.0(rw,no_root_squash,sync)
    mount_opts: intr,rsize=32768,wsize=32768,vers=3,_netdev
```

The Compute File

Default variables for the compute nodes are stored in this file. Currently, this only ensures that the SLURM role of the nodes is an execution host.

```
slurm_role: exec
```

The Gateway File

Default variables for the gateway node are stored in this file. This ensures that the gateway node is a SLURM controller and NFS server.

```
# SLURM
slurm_role: controller

# NFS
nfs_role: server
```

Role: Upstream

The upstream role consists of a tasks file which is broken down below.

Firstly, this task will wait for all nodes to be reachable before proceeding with configuration.

```
- name: Waiting for nodes to be reachable
  wait_for_connection:
    sleep: 10
    timeout: 600
```

Next, the EPEL repository is installed.

```
- name: Install EPEL Repo
  yum: name=epel-release state=present
```

The OpenFlight SLURM repository is the next to be installed, this gives access to precompiled SLURM RPMs.

```
- name: Add OpenFlight SLURM Repo
  yum_repository:
    name: openflight-slurm
    description: OpenFlight SLURM Repo
    state: present
    baseurl: 
https://openflighthpc-compute.s3-eu-west-2.amazonaws.com/slurm/
```

Finally, the OpenFlight Release repository is installed, this gives access to the various Flight Environment packages

```
- name: Add OpenFlight Release Repo
  yum: 
name=https://openflighthpc.s3-eu-west-1.amazonaws.com/repos/openf
light/x86_64/openflighthpc-release-1-1.noarch.rpm state=present
```

Role: SLURM

The SLURM role consists of a list of tasks as well as a template for the SLURM config file.

Tasks

The first task installs munge and other dependencies for SLURM

```
- name: Install munge
  yum:
    name:
      - munge
      - munge-libs
      - perl-Switch
      - numactl
    state: present
```

Next, the SLURM packages are installed, these will be sourced from the OpenFlight SLURM repo.

```
- name: Install SLURM
  yum:
    name:
      - slurm
      - slurm-devel
      - slurm-perlapi
      - slurm-torque
      - slurm-slurmd
      - slurm-example-configs
      - slurm-libpmi
    state: present
    disable_gpg_check: yes
```

Next, additional SLURM and database packages are installed on the controller only.

```
- name: Install SLURM controller packages
  yum:
    name:
      - mariadb
      - mariadb-test
      - mariadb-libs
      - mariadb-embedded
      - mariadb-bench
      - slurm-slurmctld
    state: present
    disable_gpg_check: yes
  when: slurm_role == 'controller'
```

The database service is then started on the controller.

```
- name: Turn on MariaDB
  service: name=mariadb state=started enabled=yes
  when: slurm_role == 'controller'
```

Next, the spooling directories are created on the controller.

```
- name: Create spool directory
  file:
    path: /var/spool/slurm.state
    state: directory
    owner: nobody
    group: nobody
  when: slurm_role == 'controller'
```

The munge key is then put in place on all nodes, using the munge key variable set in the all file.

```
- name: Add munge key
  copy:
    content: "{{ munge_key }}"
    dest: /etc/munge/munge.key
    owner: munge
    mode: 0400
```

Munge is then started and enabled on all nodes.

```
- name: Turn on munge
  service: name=munge state=started enabled=yes
```

The log directories for SLURM files are created on all nodes.

```
- name: Create SLURM log dirs
  file:
    path: /var/log/slurm
    state: directory
    owner: nobody
```

Next, the SLURM config template is rendered and put in place on all nodes.

```
- name: Install slurm.conf
  template: src=slurm.conf dest=/etc/slurm/slurm.conf
```

For execution nodes, the slurmd service is started and enabled.

```
- name: Turn on SLURM exec daemon
  service: name=slurmd state=started enabled=yes
  when: slurm_role == 'exec'
```

For the controller, the slurmctld service is started and enabled.

```
- name: Turn on SLURM controller daemon
  service: name=slurmctld state=started enabled=yes
  when: slurm_role == 'controller'
```

SLURM Config File

A template for slurm.conf is setup to have some fields that ansible will replace using variables. Specifically, the cluster name, gateway node hostname and list of compute nodes all come from the global variables.

```
ClusterName={{ cluster_name }}
ControlMachine={{ gateway_node }}
SlurmUser=nobody
SlurmctldPort=6817
SlurmdPort=6818
AuthType=auth/munge
StateSaveLocation=/var/spool/slurm.state
SlurmdSpoolDir=/var/spool/slurmd.spool
SwitchType=switch/none
MpiDefault=none
SlurmctldPidFile=/var/run/slurmctld.pid
SlurmdPidFile=/var/run/slurmd.pid
ProctrackType=proctrack/pgid
ReturnToService=2
SlurmctldTimeout=300
SlurmdTimeout=300
InactiveLimit=0
MinJobAge=300
KillWait=30
```

```
Waittime=0
SchedulerType=sched/backfill
SelectType=select/linear
FastSchedule=1
SlurmctldDebug=3
SlurmctldLogFile=/var/log/slurm/slurmctld.log
SlurmdDebug=3
SlurmdLogFile=/var/log/slurm/slurmd.log
JobCompType=jobcomp/none
NodeName={{ compute_nodes }}
PartitionName=all Nodes=ALL Default=YES MaxTime=UNLIMITED
```

Role: NFS

The NFS role consists of a tasks file and a template file of /etc/exports. These are explained in further detail below.

Tasks

Firstly, the NFS utilities are installed on all nodes

```
- name: Install NFS utils
  yum: name=nfs-utils state=present
```

Next, the export directories are created on the server only by looping through the globally defined NFS shares.

```
- name: Create export directories
  when: nfs_role == "server"
  file:
    path: "{{ item.key }}"
    state: directory
    mode: 775
  with_dict: "{{ nfs_shares }}"
```

The NFS thread count is increased to 32 for the server.

```
- name: Increase NFS thread count
  when: nfs_role == "server"
  replace:
    path: /etc/sysconfig/nfs
    regexp: '^#\RPCNFSDCOUNT.*$'
    replace: 'RPCNFSDCOUNT=32'
```

The exports template file is then rendered and installed to /etc/exports on the server only.

```
- name: Create export file
  when: nfs_role == "server"
  template: src=exports dest=/etc/exports
```

The client fstab entries are created by checking for the existence of a matching mount line in /etc/fstab for all NFS shares.

```
- name: Create fstab file (client)
  lineinfile:
    path: /etc/fstab
    line: "{{ item.value.server }}:{{ item.key }}    {{
item.value.mountpoint }}    nfs    {{ item.value.mount_opts }}
0 0"
    state: present
  with_dict: "{{ nfs_shares }}"
  when: nfs_role == 'client'
```

For the server, the exports are bind-mounted to prevent any performance impact from accessing files which are local to the NFS server itself. A bind entry is ignored if the export path matches the mount directory.

```
- name: Create fstab file (server - bindmounts)
  lineinfile:
    path: /etc/fstab
    line: "{% if item.key != item.value.mountpoint %}{{ item.key
}}    {{ item.value.mountpoint }}    none    defaults,bind    0
0{% endif %}"
    state: present
  with_dict: "{{ nfs_shares }}"
  when: nfs_role == 'server'
```

Next, the mountpoint directories are created on all nodes.

```
- name: Create mount directories
  file:
    path: "{{ item.value.mountpoint }}"
    state: directory
    mode: 775
  with_dict: "{{ nfs_shares }}"
```

The NFS service is started and enabled on all nodes.

```
- name: Start NFS service
  service: name=nfs state=started enabled=yes
```

Export the shares on the server.

```
- name: Export mounts
  when: nfs_role == 'server'
  command: exportfs -va
```

Ensure that all the new mounts are setup on all nodes

```
- name: Mount NFS exports
  command: mount -a
  args:
    warn: no
```

Exports Template File

The file to template /etc/exports loops through all the NFS shares to grab the path and options for each export.

```
# /etc/exports generated by openflight-ansible-playbook
{% for (key,value) in nfs_shares.iteritems() %}
{{ key }} {{ value.export_opts }}
{% endfor %}
```

Role: FlightEnv

The FlightEnv role contains a task definition file and a couple for templates for additional content. All are broken down into more detail below.

Tasks

The first task gathers and installs all the Flight Environment packages.

```
- name: Install packages
  yum:
    name:
      - flight-runway
      - flight-starter
      - flight-env
      - flight-desktop
      - vte
      - vte-profile
      - pdsh
      - pdsh-mod-genders
    state: present
    disable_gpg_check: yes
```

Next, the cluster name is set to the global cluster name variable.

```
- name: Set cluster name
  command: "/opt/flight/bin/flight config set cluster.name {{
cluster_name }}"
```

The genders template file is then rendered and put into place on all nodes. This is setup so that pdsh can be used alongside a preconfigured file.

```
- name: Create genders file
  template: src=genders dest=/opt/flight/etc/genders
```

The next task tweaks the configuration of Flight Environment to set the global software environment installation directory to one of the NFS shares.

```
- name: Configure global software environment location
  template: src=config.yml
dest=/opt/flight/opt/flight-env/etc/config.yml
```

The final task in this role tweaks system limits to allow Singularity to work for non-root users.

```
- name: Configure Singularity environment to work for users
  sysctl:
    name: user.max_user_namespaces
    value: '100'
    sysctl_set: yes
    state: present
    reload: yes
```

Flight Env Config File

This file sets global build and cache paths to be on the /opt/apps/ NFS share.

```
# Path to global/system-wide depots
global_depot_path: /opt/apps/flight/env
# Path to global/system-wide build cache
global_build_cache_path: /opt/apps/flight/env/build
```

Genders file

The genders file uses the gateway node and compute node variables to populate a simple, ready-to-go genders file.

```
{{ gateway_node }}  gateway,all
{{ compute_nodes }} nodes,all
```

Role: FlightEnv Bootstrap

The FlightEnv Bootstrap role performs additional configuration to the environment. It consists of a single tasks file which is described in detail below.

Every task in this role is set to run if the bootstrap variable is set to true and only if the node is in the gateway group.

The first task runs the environment preparation to ensure that Gnome desktops are supported in the image.

```
- name: Prepare gnome desktop
  command: "/opt/flight/bin/flight desktop prepare gnome"
  async: 600
  poll: 15
  when:
    - flightenv_bootstrap
    - "'gateway' in group_names"
```

A similar task is then performed to prepare the xterm desktop environment.

```
- name: Prepare xterm desktop
  command: "/opt/flight/bin/flight desktop prepare xterm"
  when:
    - flightenv_bootstrap
    - "'gateway' in group_names"
```

The next task creates a directory which has the correct permissions to support gridware binary installation as any user.

```
- name: Add gridware binary support
  file:
    path: "/opt/apps/flight/env/u"
    state: directory
    mode: 1023
  when:
    - flightenv_bootstrap
    - "'gateway' in group_names"
```

Lastly, an upstream tarball is extracted to the system. This archive contains prerequisites for most of the software environments, allowing for quicker configuration of software environments.

```
- name: Prewarm software environments
  unarchive:
    src:
https://openflighthpc-compute.s3.eu-west-2.amazonaws.com/bootcamp
/openflight-env-prereqs.tar.gz
    dest: /
    remote_src: yes
  when:
    - flightenv_bootstrap
    - "'gateway' in group_names"
```

In Depth: The Cluster Builder Scripts

The cluster builder script will first source the config file and set additional variables (such as the cluster name and ssh key arguments) before attempting to build anything.

```
# Get directory of script for locating templates and config
DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" >/dev/null 2>&1 &&
pwd )"

# Source variables
source $DIR/config.sh

CLUSTERNAMEARG="$1"
SSH_PUB_KEY="$2"

LOG="$DIR/log/deploy.log"

SEED=$(head /dev/urandom | tr -dc a-z0-9 | head -c 6 ; echo '')
CLUSTERNAME="$CLUSTERNAMEARG-$SEED"

# The host IP which is sharing setup.sh script at
http://IP/deployment/setup.sh
CONTROLLERIP=$(dig +short myip.opendns.com
@resolver1.opendns.com)
```

Next, the script verifies that arguments are present and will error if any of the values are missing/invalid.

```
if [ -z "${CLUSTERNAME}" ] ; then
    echo "Provide cluster name"
    echo "  build-cluster.sh CLUSTERNAME SSH_PUB_KEY"
    exit 1
elif [ -z "${SSH_PUB_KEY}" ] ; then
    echo "Provide ssh public key"
    echo "  build-cluster.sh CLUSTERNAME SSH_PUB_KEY"
    exit 1
elif [ "$COMPUTENODES" -lt 2 -o "$COMPUTENODES" -gt 8 ] ; then
    echo "Number of nodes must be between 2 and 8"
    exit 1
fi
```

To prevent authentication issues, the script will check that the provided SSH key does not match the public key on the deployment server. This is because the public key on the server is used to authenticate ansible connections to the nodes.

```
# Don't allow SSH_PUB_KEY to be set to the controller's pub key
(as this is added via setup.sh on the deployed nodes)
if [[ *"$(cat /root/.ssh/id_rsa.pub)"* == *"$SSH_PUB_KEY"* ]] ;
then
    echo "Provide ssh public key that is *not* this controller's
public key."
    echo "This controller's key is automatically added to the
compute nodes at deployment"
    echo "to allow ansible setup to run on nodes"
    exit 1
fi
```

Once all the verification has completed, some general time and configuration information is logged to the terminal and the log file.

```
echo "$(date +'%Y-%m-%d %H-%M-%S') | $CLUSTERNAME | Start Deploy
| $PLATFORM | $SSH_PUB_KEY" |tee -a $LOG
```

A function is then defined for ensuring that the required variable information for Azure is present and that the Azure CLI has been authenticated to allow it to deploy resources.

```
function check_azure() {
    # Azure variables are non-empty
    if [ -z "${AZURE_SOURCEIMAGE}" ] ; then
        echo "AZURE_SOURCEIMAGE is not set in config.sh"
        echo "Set this before running script again"
        exit 1
    elif [ -z "${AZURE_LOCATION}" ] ; then
        echo "AZURE_LOCATION is not set in config.sh"
        echo "Set this before running script again"
        exit 1
    fi

    # Azure login configured
    if ! az account show > /dev/null 2>&1 ; then
        echo "Azure account not connected to CLI"
        echo "Run az login to connect your account"
        exit 1
    fi
}
```

Another function is defined for actually deploying to Azure. The first line in the function creates a resource group the cluster

```
function deploy_azure() {
    az group create --name "$CLUSTERNAME" --location
"$AZURE_LOCATION"
```

The next command in the function deploys the Azure template and provides all the necessary parameters.

```
   az group deployment create --name "$CLUSTERNAME"
--resource-group "$CLUSTERNAME" \
        --template-file $DIR/templates/azure/cluster.json \
        --parameters sshPublicKey="$SSH_PUB_KEY" \
        sourceimage="$AZURE_SOURCEIMAGE" \
        clustername="$CLUSTERNAMEARG" \
        computeNodesCount="$COMPUTENODES" \
        customdata="$(cat $DIR/templates/cloudinit.txt |base64 -w
0)"
```

Then the function creates an ansible hosts file for the cluster, containing the correct groupings and public IPs of the nodes.

```
   cat << EOF > /opt/flight/clusters/$CLUSTERNAME
[gateway]
gateway1    ansible_host=$(az network public-ip show -g
$CLUSTERNAME -n flightcloudclustergateway1pubIP --query
"{address: ipAddress}" --output yaml |awk '{print $2}')

[nodes]
$(i=1 ; while [ $i -le $COMPUTENODES ] ; do
echo "node0$i    ansible_host=$(az network public-ip show -g
$CLUSTERNAME -n flightcloudclusternode0$i\pubIP --query
'{address: ipAddress}' --output yaml |awk '{print $2}')"
i=$((i + 1))
done)
EOF
```

Finally, the run ansible function is called to configure the cluster.

```
    # Customise nodes
    run_ansible
}
```

The next function to be defined is the run ansible function. This function begins by determining whether the Flight Environment Bootstrap role should be run based on user configuration.

```
function run_ansible() {
    # Determine if extra flight env stuff to be run
    if [ "$FLIGHTENVPREPARE" = "true" ] ; then
        extra_flightenv_var="flightenv_bootstrap=true"
    fi
```

The function then navigates to the correct directory for the ansible playbook and disables ansible from failing if the host keys of the node are not yet known.

```
    # Run ansible playbook
    cd /root/openflight-ansible-playbook
    export ANSIBLE_HOST_KEY_CHECKING=false
```

Finally, the function runs the playbook using the previously created hosts file, passing through relevant variables and generating a random munge key.

```
    ansible-playbook -i /opt/flight/clusters/$CLUSTERNAME
--extra-vars "cluster_name=$CLUSTERNAMEARG munge_key=$( (head
/dev/urandom | tr -dc a-z0-9 | head -c 18 ; echo '') | sha512sum
| cut -d' ' -f1) compute_nodes=node[01-0$COMPUTENODES]
$extra_flightenv_var" openflight.yml
}
```

Next, a case statement determines what platform the deployment should be happening on and runs the appropriate functions.

```
case $PLATFORM in
    "azure")
        check_azure
        deploy_azure
    ;;
    "aws")
        echo "AWS is not a supported platform at this time"
    ;;
    *)
        echo "Unknown platform"
    ;;
esac
```

Finally, the builder script will log the end time, cluster name and IP address of the gateway node.

```
echo "$(date +'%Y-%m-%d %H-%M-%S') | $CLUSTERNAME | End Deploy |
Gateway1 IP: $(az network public-ip show -g $CLUSTERNAME -n
flightcloudclustergateway1pubIP --query "{address: ipAddress}"
--output yaml |awk '{print $2}')" |tee -a $LOG
```

Preparing Azure Account

This example uses 2 OpenFlightHPC provided images, one for the cloud controller (a system used for deploying cloud clusters) and one for compute nodes (a generic CentOS 7 installation that is ready for customisation).

These images can be found in the build documentation with links to helpful documentation for importing the images into an Azure account.

For this documentation, the following image versions were used in the UK South Azure region:

- Cloud Controller: `cloudcontroller-0310191057`
- Base CentOS Image: `CENTOS7BASE2808191247`

Cloud Controller

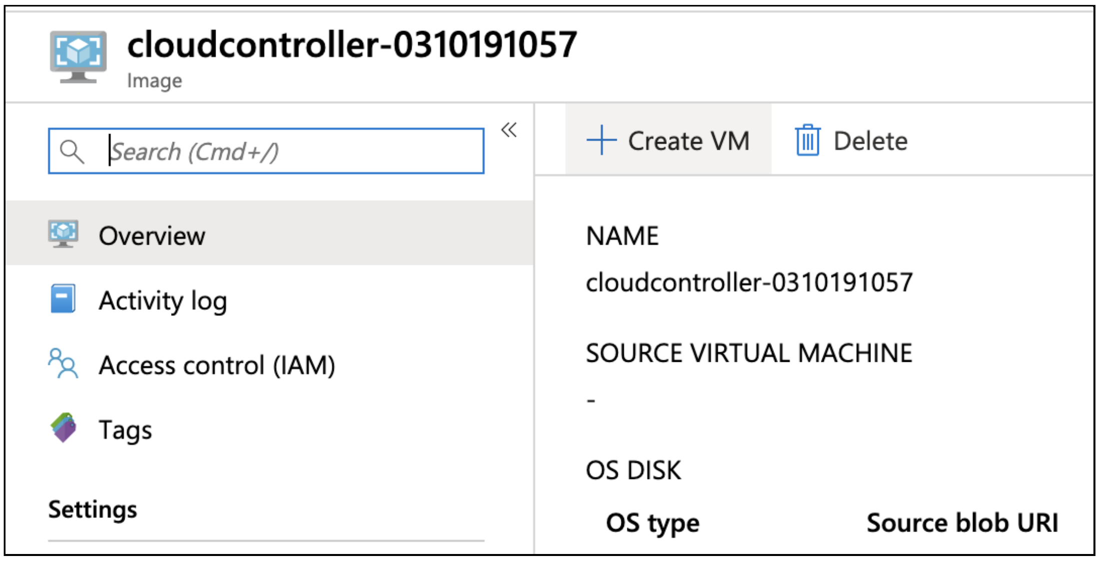
What is it?

The cloud controller is a single system that provides cloud tools and network shares for deploying and customising resources in the cloud. The benefit to having a centralised system is that it provides a single location for deploying many resources whilst tracking those deployments.

For more information - https://build.openflighthpc.org/en/latest/cloud/controller.html

Building Cloud Controller for Deploying Clusters

Locate the imported image for the cloud controller and select "Create VM" to create an instance from it



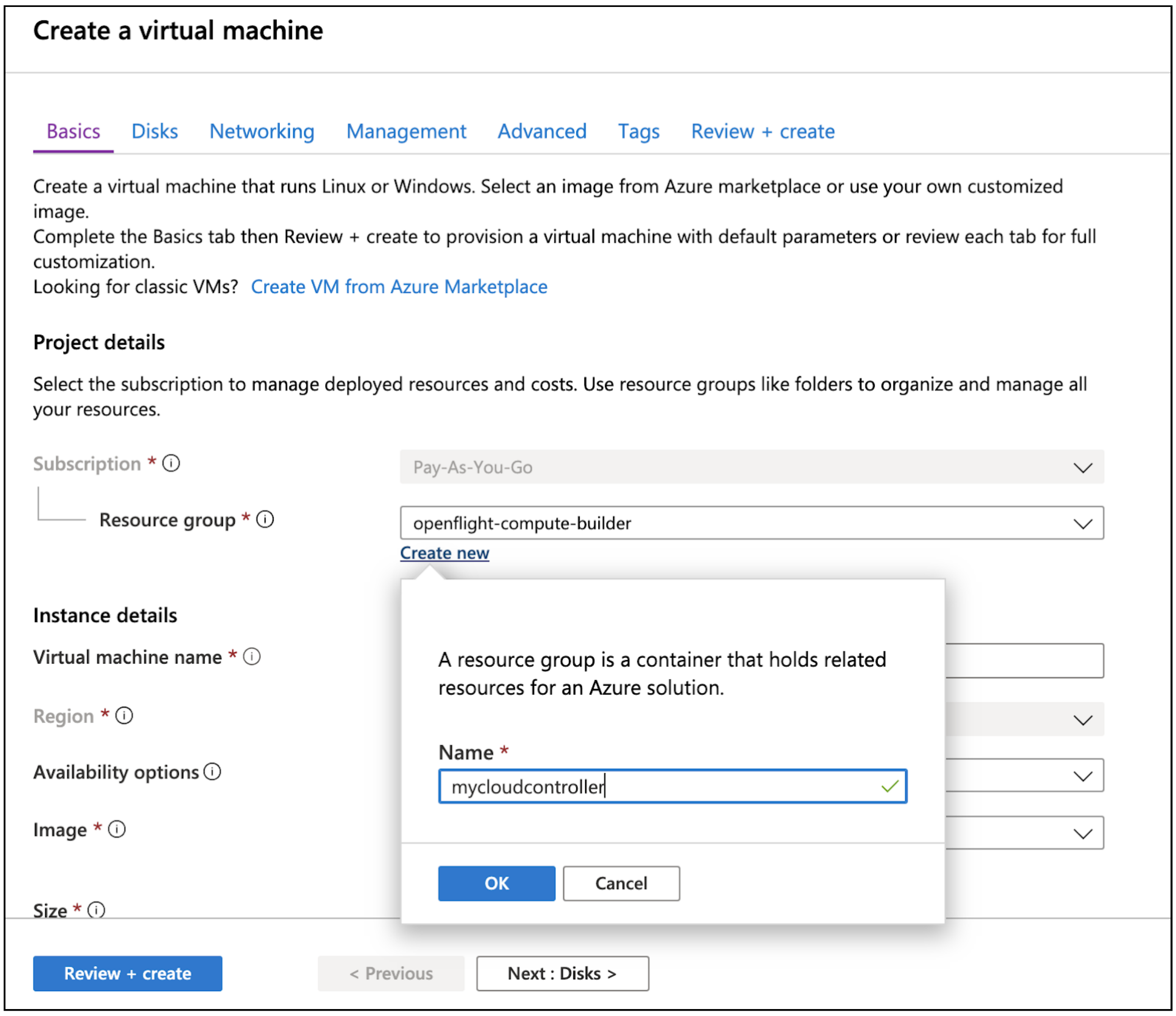
NAME
cloudcontroller-0310191057

SOURCE VIRTUAL MACHINE
-

OS DISK

OS type	Source blob URI

On the configuration screen, set the resource group name (in this example, a new resource group is created called `mycloudcontroller`)



Create a virtual machine

Create a virtual machine that runs Linux or Windows. Select an image from Azure marketplace or use your own customized image.
Complete the Basics tab then Review + create to provision a virtual machine with default parameters or review each tab for full customization.
Looking for classic VMs? Create VM from Azure Marketplace

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

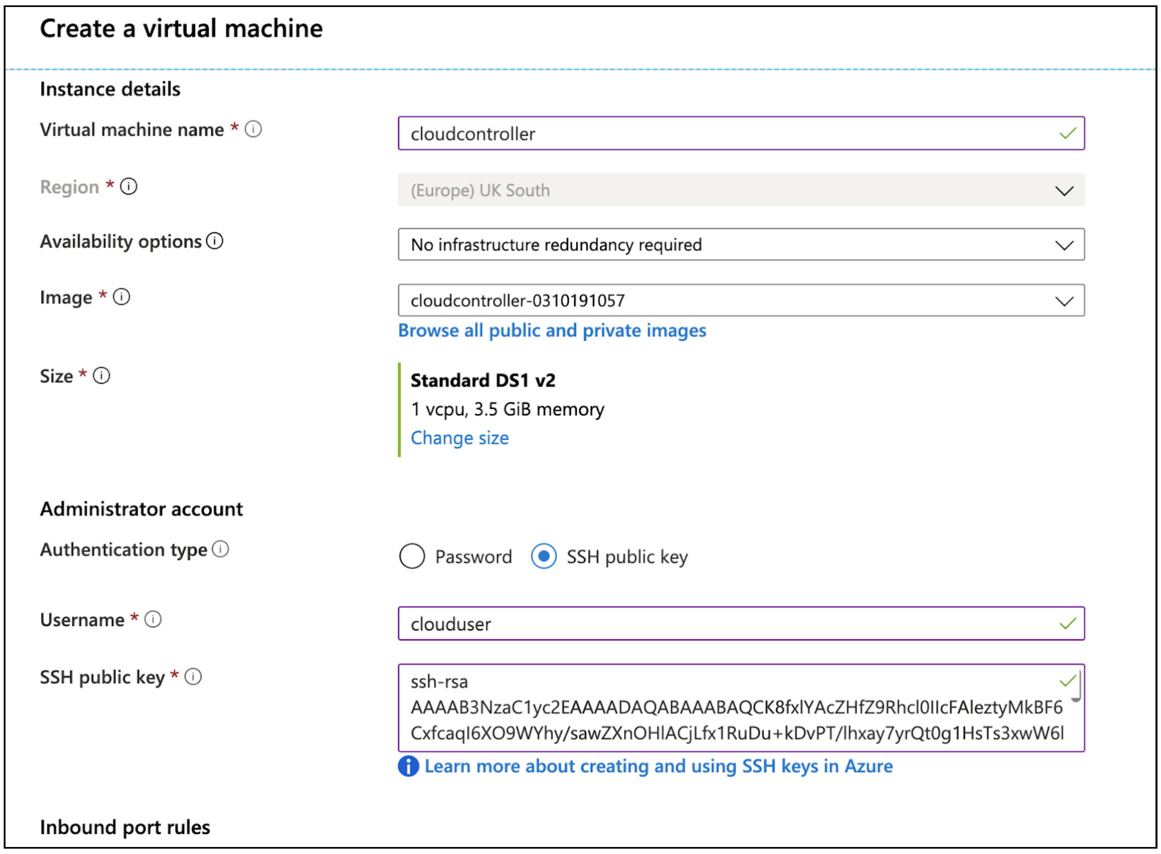
Subscription * : Pay-As-You-Go
Resource group * : openflight-compute-builder
Create new

Instance details
Virtual machine name *
Region *
Availability options
Image *
Size *

A resource group is a container that holds related resources for an Azure solution.

Name * : mycloudcontroller

Set the virtual machine name, select the instance size, set the username and SSH key for logging in



Create a virtual machine

Instance details

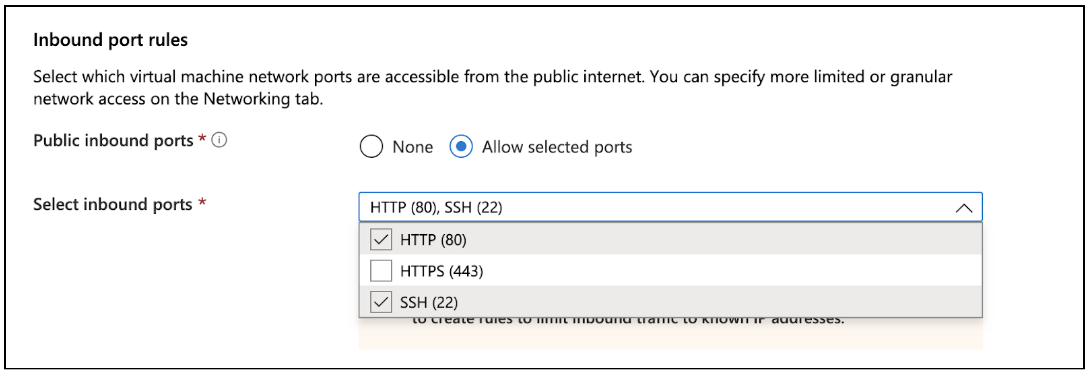
Virtual machine name *	cloudcontroller
Region *	(Europe) UK South
Availability options	No infrastructure redundancy required
Image *	cloudcontroller-0310191057
	Browse all public and private images
Size *	**Standard DS1 v2** 1 vcpu, 3.5 GiB memory Change size

Administrator account

Authentication type	Password [✓ SSH public key]
Username *	clouduser
SSH public key *	ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCK8fxlYAcZHfZ9Rhcl0IIcFAleztyMkBF6 CxfcaqI6XO9WYhy/sawZXnOHlACjLfx1RuDu+kDvPT/lhxay7yrQt0g1HsTs3xwW6l
	Learn more about creating and using SSH keys in Azure

Inbound port rules

Allow SSH and HTTP connections to the system



Inbound port rules

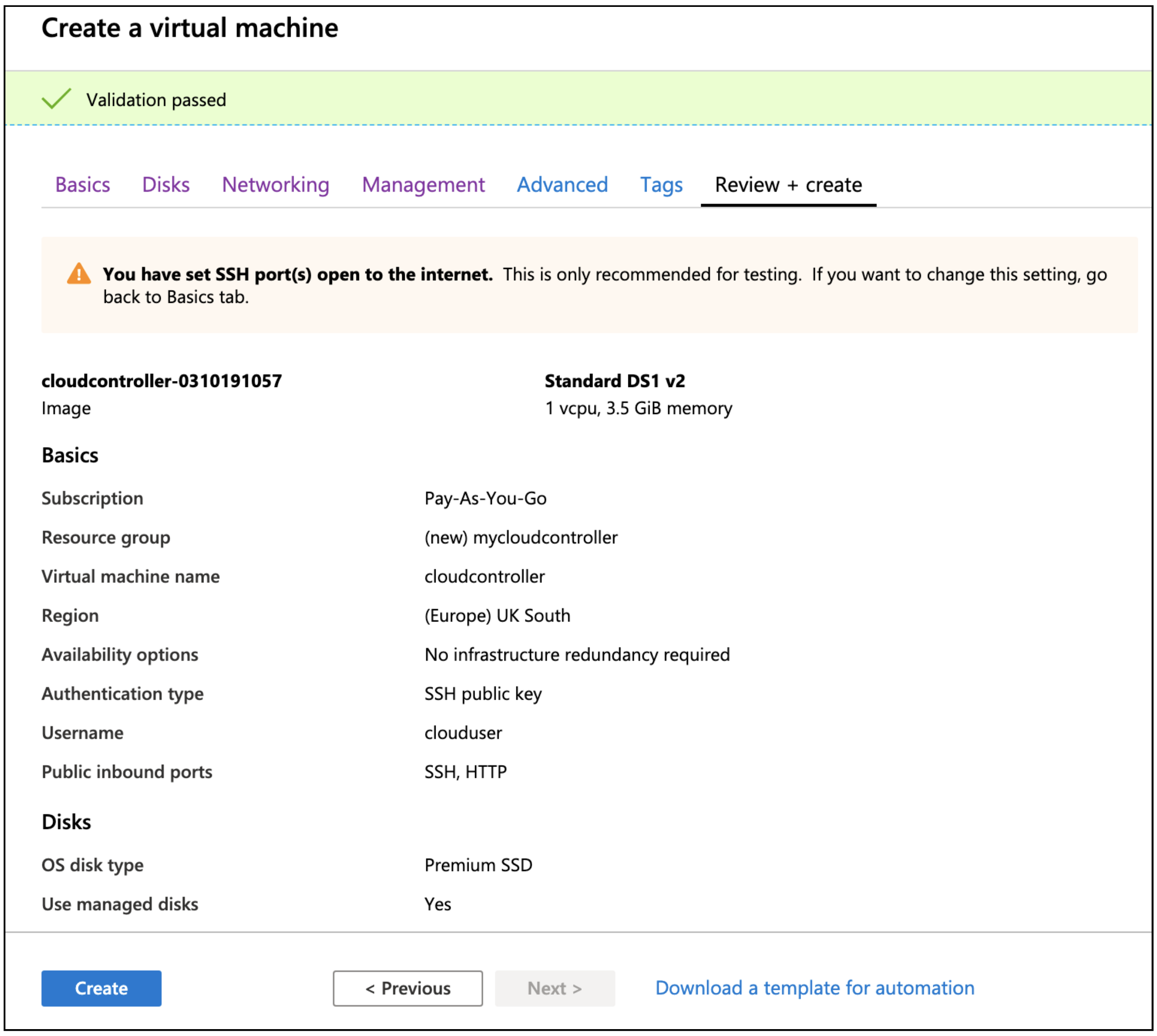
Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

Public inbound ports *	None [✓ Allow selected ports]
Select inbound ports *	HTTP (80), SSH (22)

- ✓ HTTP (80)
- ☐ HTTPS (443)
- ✓ SSH (22)

to create rules to limit inbound traffic to known IP addresses.

Defaults can be used for all the other tabs so "Review and Create" is selected to validate the configuration and then "Create" to deploy the resources



You have set SSH port(s) open to the internet. This is only recommended for testing. If you want to change this setting, go back to Basics tab.

cloudcontroller-0310191057	**Standard DS1 v2**
Image	1 vcpu, 3.5 GiB memory

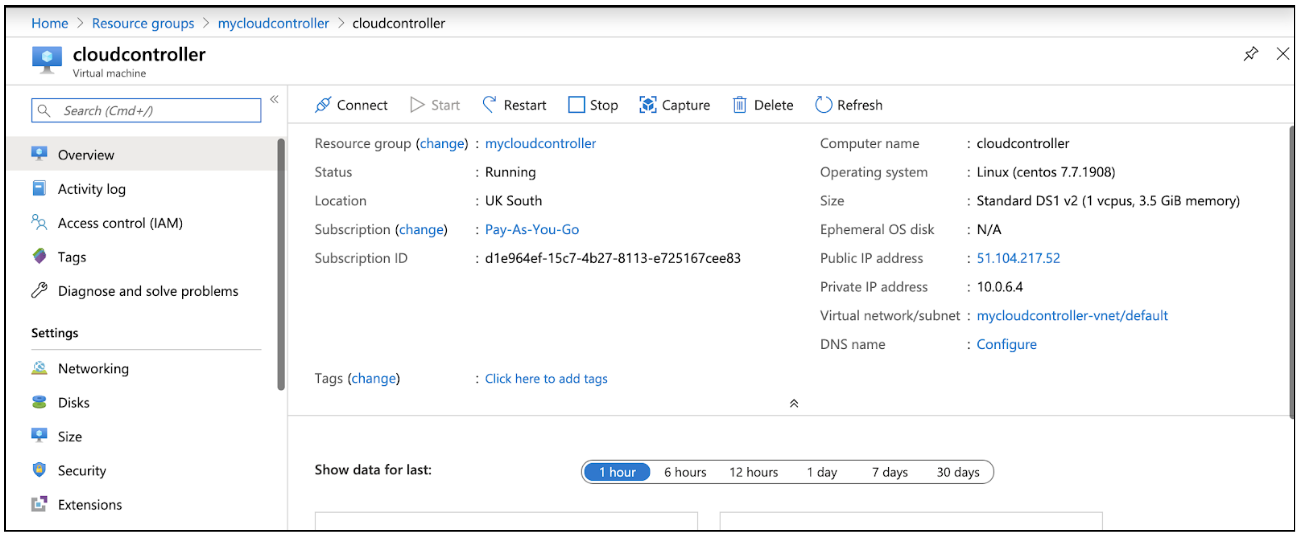
Basics

Subscription	Pay-As-You-Go
Resource group	(new) mycloudcontroller
Virtual machine name	cloudcontroller
Region	(Europe) UK South
Availability options	No infrastructure redundancy required
Authentication type	SSH public key
Username	clouduser
Public inbound ports	SSH, HTTP

Disks

OS disk type	Premium SSD
Use managed disks	Yes

Locate the public IP address to remotely connect to the instance



Resource group (change) : mycloudcontroller
Status : Running
Location : UK South
Subscription (change) : Pay-As-You-Go
Subscription ID : d1e964ef-15c7-4b27-8113-e725167cee83
Tags (change) : Click here to add tags
Computer name : cloudcontroller
Operating system : Linux (centos 7.7.1908)
Size : Standard DS1 v2 (1 vcpus, 3.5 GiB memory)
Ephemeral OS disk : N/A
Public IP address : 51.104.217.52
Private IP address : 10.0.6.4
Virtual network/subnet : mycloudcontroller-vnet/default
DNS name : Configure

Login and switch to the root user

```
$ ssh clouduser@51.104.217.52
[clouduser@cloudcontroller ~]$ sudo su -
[root@cloudcontroller ~]#
```

Configure Azure CLI

```
[root@cloudcontroller ~]# az login
To sign in, use a web browser to open the page
https://microsoft.com/devicelogin and enter the code ABCDEFGHI to
authenticate.
```

Clone the cluster build repository

```
[root@cloudcontroller ~]# git clone https://github.com/openflight
hpc/openflight-compute-cluster-builder
```

Run the setup script to configure access keys and general share configuration for the controller

```
[root@cloudcontroller ~]# cd openflight-compute-cluster-builder/
[root@cloudcontroller  openflight-compute-cluster-builder]#  bash
setup.sh
Generating public/private rsa key pair.
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
<snip>
Cloning into 'openflight-ansible-playbook'...
remote: Enumerating objects: 90, done.
remote: Counting objects: 100% (90/90), done.
remote: Compressing objects: 100% (49/49), done.
remote: Total 90 (delta 21), reused 85 (delta 17), pack-reused 0
Unpacking objects: 100% (90/90), done.
```

Now all the configuration scripts and ansible roles are on the controller it's ready to build clusters.

Deploying Cluster

The controller has a script that will deploy the resources and customise them to contain the Flight Environment. This script can be run multiple times to create many clusters.

Before deploying the cluster, a few environment variables are required to set the source image and deployment location of the cluster, set these in the session that will be running the script

```
[root@cloudcontroller ~]# vim openflight-compute-cluster-builder/
config.sh
```

To deploy a simple cluster (with a gateway node and 2 compute nodes) run the following (replacing `ssh pub key here` with the desired access key for the cluster)

```
[root@cloudcontroller ~]# cd openflight-compute-cluster-builder/
[root@cloudcontroller  openflight-compute-cluster-builder]#  bash
build-cluster.sh clustername 'ssh pub key here'
2019-10-24 13-43-53 | clustername-uveoei | Start Deploy | azure |
ssh pub key here
<snip>
PLAY                                                          RECAP
*******************************************************************
*************************************************************
gateway1                      : ok=27    changed=24    unreachable=0
failed=0
node01                        : ok=20    changed=17    unreachable=0
failed=0
node02                        : ok=20    changed=17    unreachable=0
failed=0

2019-10-24 13-53-14 | clustername-uveoei | End Deploy | Gateway1
IP: 51.104.228.29
```

The IP address for the gateway node is printed at the end of the cluster build script, use this to connect to the gateway node (using the private key for the specified public key)

```
$ ssh -i ~/.ssh/my_priv_key flight@51.104.228.29
 -[ OpenFlightHPC ]-
Welcome to mycluster, based on CentOS Linux 7.6.1810.

This cluster provides an OpenFlight HPC environment.

'flight start' - activate OpenFlight now
'flight info'  - get some brief help about OpenFlight
'flight set'    - change login defaults (see 'flight info' for
details)

[flight@gateway1 ~]$
```

The cluster is now build and is ready to be used for running HPC workflows. Flight Environment provides many different software management tools and environment tools to easily create desktop environments and more.

Find out how to use the Flight Environment at https://use.openflighthpc.org/en/latest/