OpenShift Container Platform 4.4 on Azure with Microsoft Windows Server 2019/1809 Nodes

Developer Preview

Pre-requisites

The document is tested against a Fedora release 31 Linux host for running all the Linux commands.

Please note that a command preceded by > is to be run in a PowerShell window on a Windows instance, and a command preceded by $ is to be run on a Linux console (localhost).

Installation of prerequisite packages

Python3 and pip are required to follow this guide. To check if python3 is installed on the system, run the following command:

```
$ python3 --version
```

To install python3, run the following command:

```
$ sudo dnf install python3
```

To check if pip is installed on the system, run the following command:

```
$ python3 -m pip --version
pip 19.0.3 from /usr/lib/python3.7/site-packages/pip (python 3.7)
```

To install pip refer pip installation

Install jq library to parse json files:

```
$ sudo dnf install jq
```

Install git to clone github repository:

```
$ sudo dnf install git
```

Bring up the OpenShift cluster with ovn-kubernetes

Download the dev-preview 4.4.0-rc.3 OpenShift Installer and Client and pull secret for AWS..
Note: Download both the openshift-install and openshift-client.

Unzip the files with the following commands: (Reference commands given below)

```
$ tar -xzvf openshift-install-linux-<version>.tar.gz
$ tar -xzvf openshift-client-linux-<version>.tar.gz
```

After extracting openshift client, move the oc and kubectl binaries to `/usr/local/bin`

```
$ sudo mv {kubectl,oc} /usr/local/bin
```

Configure Azure CLI

See Install the Azure CLI

Credentials setup

The installer (and the cluster credentials provider) needs a Service Principal for Azure API access.

You should only need to do this once unless you have some specific reason to regenerate your Service Principal.

Option 1: Use the wizard

First, create a service principal (replacing `--name` with your own in the format $user-$purpose or $team-$purpose, whichever is more appropriate):

```
$ az ad sp create-for-rbac --role Owner --name winc-installer
Changing "winc-installer" to a valid URI of "http://winc-installer", which is the
required format used for service principal names
Retrying role assignment creation: 1/36
Retrying role assignment creation: 2/36
Retrying role assignment creation: 3/36
{
  "appId": "e564ac9e-bd46-4c5a-1234-03724a82c3f2",
  "displayName": "winc-installer",
  "name": "http://winc-installer",
  "password": "111-11-1111",
  "tenant": "6047c7e9-b2ad-488d-a54e-dc3f6be6a7ee"
}
```

When you run the installer wizard, if the credentials file doesn't exist, you'll be prompted to enter an account and SP info and the credentials file will be generated.

Option 2: Generate the credentials file manually

First, create a service principal (replacing `--name` with your own):

Replace `--name` with your own.

```
$ az ad sp create-for-rbac --role Owner --name $service-principal-name \ | jq --arg
sub_id "$(az account show | jq -r '.id')" \
     '{subscriptionId:$sub_id,clientId:.appId,
clientSecret:.password,tenantId:.tenant}' \
     > ~/.azure/osServicePrincipal.json
```

Create the install-config

```
$ ./openshift-install create install-config --dir=<cluster_directory>
> SSH Public Key <path-to-your-rsa>/id_rsa.pub
> Platform <i.e. aws>
> Region  <region close by i.e. us-east-1>
> Base Domain <Your Domain>
> Cluster Name <cluster_name>
> Pull Secret <content of pull-secrets>
```

Official documentation should be consulted for credentials and other cloud provider related instructions.

The previous step will result in an `install-config.yaml` in your current directory. Edit the `install-config.yaml` to switch networkType from OpenShiftSDN to OVNKubernetes inside the cluster directory:

```
$ sed -i 's/OpenShiftSDN/OVNKubernetes/g' install-config.yaml
```

Create manifests

We want to enable hybrid networking on the cluster. For that we need to create the manifests:

```
$ ./openshift-install create manifests --dir=<cluster_directory>
```

Above step will create a `manifests` and `openshift` folder in your `<cluster_directory>`

Configuring OVNKubernetes on a Hybrid cluster

Inside the cluster_directory, create a copy of `manifests/cluster-network-02-config.yml` as `manifests/cluster-network-03-config.yml`.

```
$ cp manifests/cluster-network-02-config.yml manifests/cluster-network-03-config.yml
```

Edit `manifests/cluster-network-03-config.yml` as shown below:

1. Modify the `api version` to `operator.openshift.io/v1`

```
$ sed -i 's/config.openshift.io\/v1/operator.openshift.io\/v1/g'
manifests/cluster-network-03-config.yml
```

2. Add the following to the `spec:` section of `manifests/cluster-network-03-config.yml`

```
spec:
  defaultNetwork:
    type: OVNKubernetes
    ovnKubernetesConfig:
      hybridOverlayConfig:
        hybridClusterNetwork:
        - cidr: 10.132.0.0/14
          hostPrefix: 23
```

Here is an example `manifests/cluster-network-03-config.yml`:

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  creationTimestamp: null
  name: cluster
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  externalIP:
    policy: {}
  networkType: OVNKubernetes
  serviceNetwork:
  - 172.30.0.0/16
  defaultNetwork:
    type: OVNKubernetes
    ovnKubernetesConfig:
      hybridOverlayConfig:
        hybridClusterNetwork:
        - cidr: 10.132.0.0/14
          hostPrefix: 23
status: {}
```

The highlighted portion above is the required modification for the file.
NOTE: the hybridClusterNetwork CIDR cannot overlap clusterNetwork CIDR

Disable Azure Rate Limiting

On bringing up the cluster with network type OVNKubernetes, you might encounter an error related to Rate Limiting on Azure API calls.

```
Error updating load balancer with new hosts map[sumehta-winc8-2tk76-master-0:{}
sumehta-winc8-2tk76-master-1:{} sumehta-winc8-2tk76-master-2:{}
sumehta-winc8-2tk76-worker-centralus1-mmjkr:{}
sumehta-winc8-2tk76-worker-centralus2-qptsj:{}
sumehta-winc8-2tk76-worker-centralus3-pq5sk:{} winnode:{}]:
ensure(openshift-ingress/router-default):
backendPoolID(/subscriptions/52432811-04323-4342f-92309-ff3452333f0/resourceGroups/s
umehta-winc8-2tk76-rg/providers/Microsoft.Network/loadBalancers/sumehta-winc8-2tk76/
backendAddressPools/sumehta-winc8-2tk76) - failed to ensure host in pool: "azure -
cloud provider rate limited(read) for operation:NicGet"
```

As a temporary workaround, you can modify the cloud provider manifests to disable rate limiting on Azure API calls.

In `manifests/cloud-provider-config.yaml` change the following parameters to disable rate limiting.

```
"cloudProviderRateLimit": false,
"cloudProviderRateLimitQPS": 0,
"cloudProviderRateLimitBucket": 0,
"cloudProviderRateLimitQPSWrite": 0,
"cloudProviderRateLimitBucketWrite": 0
```

Here is an example of `manifests/cloud-provider-config.yaml` with the changes highlighted in red below:

```
apiVersion: v1
data:
  config: "{"cloud": "AzurePublicCloud","tenantId":
"63544359-b35345d-4353453d-353453e-d534435345534e",
  "aadClientId":
    "",
  "aadClientSecret": "",
  "aadClientCertPath": "",
  "aadClientCertPassword":
    "",
  "useManagedIdentityExtension": true,
  "userAssignedIdentityID":
    "",
  "subscriptionId": "5f343433-04fa-1234-4567-ffd8a354533f0",
  "resourceGroup":
    "pmahajan-az-44bwz-rg",
  "location": "centralus",
```

```
  "vnetName": "pmahajan-az-44bwz-vnet",
  "vnetResourceGroup":
    "pmahajan-az-44bwz-rg",
  "subnetName": "pmahajan-az-44bwz-worker-subnet",
  "securityGroupName":
    "pmahajan-az-44bwz-node-nsg",
  "routeTableName": "pmahajan-az-44bwz-node-routetable",
  "primaryAvailabilitySetName":
    "",
  "vmType": "",
  "primaryScaleSetName": "",
  "cloudProviderBackoff":
    true,
  "cloudProviderBackoffRetries": 0,
  "cloudProviderBackoffExponent":
    0,
  "cloudProviderBackoffDuration": 6,
  "cloudProviderBackoffJitter":
    0,
  "cloudProviderRateLimit": false,
  "cloudProviderRateLimitQPS": 0,
  "cloudProviderRateLimitBucket":
    0,
  "cloudProviderRateLimitQPSWrite": 0,
  "cloudProviderRateLimitBucketWrite":
    0,
  "useInstanceMetadata": true,
  "loadBalancerSku": "standard",
  "excludeMasterFromStandardLB":
    null,
  "disableOutboundSNAT": null,
  "maximumLoadBalancerRuleCount": 0
}
"
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: cloud-provider-config
  namespace: openshift-config
```

Create the cluster

Execute the following command to create the cluster and wait for it come up:

```
$ ./openshift-install create cluster --dir=<cluster_directory>
```

Export the `kubeconfig` so that oc can communicate with the cluster:

```
$ export KUBECONFIG=$(pwd)/<cluster_directory>/auth/kubeconfig
```

Make sure you can interact with the cluster:

```
$ oc get nodes
```

```
NAME                                        STATUS   ROLES    AGE     VERSION
pmahajan-az-44bwz-master-0                  Ready    master   14h     v1.17.1
pmahajan-az-44bwz-master-1                  Ready    master   14h     v1.17.1
pmahajan-az-44bwz-master-2                  Ready    master   14h     v1.17.1
pmahajan-az-44bwz-worker-centralus1-bc8mr   Ready    worker   13h     v1.17.1
pmahajan-az-44bwz-worker-centralus2-xwr2h   Ready    worker   13h     v1.17.1
pmahajan-az-44bwz-worker-centralus3-mprvk   Ready    worker   13h     v1.17.1
```

Verify Hybrid networking

The network.operator cluster CR spec should look like the example below:

```
$ oc get network.operator cluster -o yaml
```

```
...
spec:

  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  defaultNetwork:
    ovnKubernetesConfig:
      hybridOverlayConfig:
        hybridClusterNetwork:
        - cidr: 10.132.0.0/14
          hostPrefix: 23
    type: OVNKubernetes
  serviceNetwork:
  - 172.30.0.0/16
status: {}
…
```

Bring up the Windows node

Launch a *Windows 2019 Server Datacenter with Containers* instance and add it to the cluster.

Windows Node Installer

Windows Node Installer (WNI) is a tool that can automate the process of launching a Windows instance and prepare it for bootstrapping. For more details, please refer to the README.

Download WNI v4.4.3-alpha on your Linux host and ensure you can execute it:

```
$ wget
https://github.com/openshift/windows-machine-config-bootstrapper/releases/download/v
4.4.3-alpha/wni
$ chmod +x wni
```

Azure stores the credentials as a JSON file at `~/.azure/osServicePrincipal.json` by default. *`dir`* is an optional argument. The directory path provided should exist on the system. Default is the current directory.

The following command will create the instance in your Azure account and will store credentials for logging into the instance in the directory path:

```
$ ./wni azure create --kubeconfig <path to OpenShift cluster>/auth/kubeconfig
--credentials ~/.azure/osServicePrincipal.json --image-id
MicrosoftWindowsServer:WindowsServer:2019-Datacenter-with-Containers:latest
--instance-type Standard_D2s_v3 --dir <directory>
```

The output of this command will give the private and public IP required for creating the host file in the Setup Ansible connection step.

You can now setup Ansible before moving on to the next steps.

If the Windows node cannot be brought up due to some reason using the automated Windows Node Installer (WNI) tool, the Windows node can also be brought up manually using the steps outlined in the Appendix section.

Setup Ansible connection

Now we can use Ansible to configure the windows host. On the Linux host, install ansible and pywinrm, as well as selinux-python bindings:
Note: This step assumes that python3 is installed on the system. Ansible will not work without python.

```
$ sudo dnf install python3-libselinux
$ pip install ansible==2.9 pywinrm selinux --user
```

Create a hosts file with the following information:

```
[win]

<public_ip> ansible_password=<password> private_ip=<private_ip>

[win:vars]
ansible_user=<username>
cluster_address=<cluster_address>
ansible_connection=winrm
ansible_ssh_port=5986
ansible_winrm_server_cert_validation=ignore
```

<cluster_address> is the cluster endpoint. It is the combination of your cluster name and the base domain for your cluster.

```
$ oc cluster-info | head -n1 | sed 's/.*\/\/api.//g'| sed 's/:.*//g'
```

In Azure, <public_ip> is the public ip address of the Windows instance and can be found under <Windows node name>-ip (look at the screenshot below)
Please provide the IPv4 public ip here as <public_ip>

<username> and <password> are the login credentials for Windows instance. Note the username listed here must have administrative privileges. <private_ip> is the private IP of the node and can be found in the output of the Windows node creation using the automated WNI tool.

Here is an example hosts file:

```
[win]
40.69.185.26 ansible_password='mypassword' private_ip=10.0.32.7

[win:vars]
ansible_user=core
cluster_address=winc-cluster.winc.azure.devcluster.openshift.com
ansible_connection=winrm
ansible_ssh_port=5986
ansible_winrm_server_cert_validation=ignore
```

Test if Ansible is able to communicate with the Windows instance with the following command:

```
$ ansible win -i <name_of_the_hosts_file> -m win_ping -v
```

Note: If you do not want to provide the password in the hosts file, you can provide the same as an extra-variable to any ansible command. For example, the above command could be executed as:

```
$ ansible win -i <name_of_the_hosts_file> -m win_ping -v --extra-vars
"ansible_password=<password>"
```

Bootstrap the windows node

On a Linux host, run the Ansible Playbook that transfers the necessary files onto the Windows instance and bootstraps it so that it can join the cluster as a worker node.
Note: Playbook assumes you have jq installed. Your active RDP connection might be disrupted during the execution of the playbook.

Clone the github repository to download ansible playbook and all the required dependencies.

```
$ git clone https://github.com/openshift/windows-machine-config-bootstrapper.git
$ git fetch && git checkout release-4.4
```

Run the ansible playbook to bootstrap the windows worker node. Make sure you have at least 6GB of free space in `/tmp` directory.

```
$ ansible-playbook -i <path_to_hosts_file>
windows-machine-config-bootstrapper/tools/ansible/tasks/wsu/main.yaml -v
```

```
$ oc get nodes -l kubernetes.io/os=windows
```

You can now see the Windows instance has joined the cluster

```
NAME                                        STATUS   ROLES    AGE     VERSION
winworker-obm7a                             Ready    worker   2m11s   v1.17.1
```

API rate limit exceeded error when running WSU

WSU playbook uses GitHub API to fetch releases for WMCB. You might encounter an API rate limit exceeded error while running WSU playbook in `TASK [Get release]`. The issue occurs due to GitHub rate-limiting unauthenticated requests at 60 requests per hour. As a workaround, wait for the rate-limit to reset (at most 1 hour) before running the playbook again.

Test Windows workload

We can now create a pod that can be deployed on a Windows instance. We used an example WebServer deployment to create a pod:
Note: Given the size of Windows images, it is recommended to pull the Docker image mcr.microsoft.com/windows/servercore:ltsc2019 on the instance first, before creating the pods.

On Windows instance, run the following command in a PowerShell window:

```
> docker pull mcr.microsoft.com/windows/servercore:ltsc2019
```

Note: Refer RDP section to set up and RDP into your windows node

On the Linux host, deploy the pods:

```
$ oc create -f
https://gist.githubusercontent.com/suhanime/683ee7b5a2f55c11e3a26a4223170582/raw/d89
3db98944bf615fccfe73e6e4fb19549a362a5/WinWebServer.yaml -n default
```

Once the deployment has been created, we can check the status of the pods:

```
$ oc get pods -n default
NAME                             READY   STATUS    RESTARTS   AGE
win-webserver-6f5bdc5b95-x65tq   1/1     Running   0          14m
```

We have created a service of LoadBalancer type:

```
$ oc get service win-webserver -n default
NAME            TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)        AGE
win-webserver   LoadBalancer   172.30.0.31   20.185.74.192   80:31412/TCP   17m
```

```
$ curl 20.185.74.192:80
<html><body><H1>Windows Container Web Server</H1><p>IP 10.132.1.2 callerCount 4
</body></html>
```

Deploying in a namespace other than default

In order to deploy into a different namespace SCC must be disabled in that namespace. This should never be used in production, and any namespace that this has been done to should not be used to run Linux pods.

To skip SCC for a namespace, the label openshift.io/run-level = 1 should be applied to the namespace. This will apply to both Linux and windows pods, and thus Linux pods should not be deployed into this namespace.

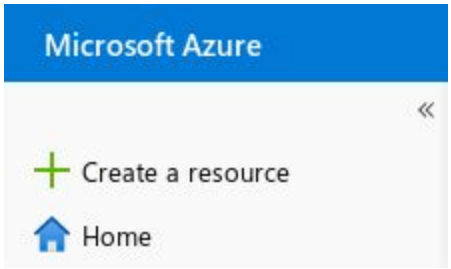
For example, to create a new project and apply the label:

```
$ oc new-project <project_name>
$ oc label namespace <project_name> "openshift.io/run-level=1"
```

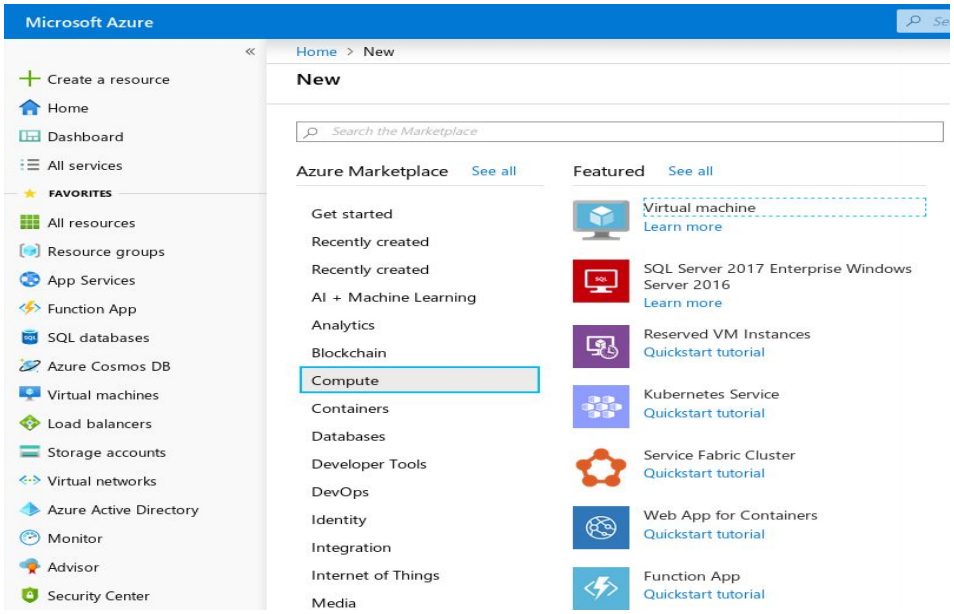
Appendix

Bring up the Windows node using the Azure portal

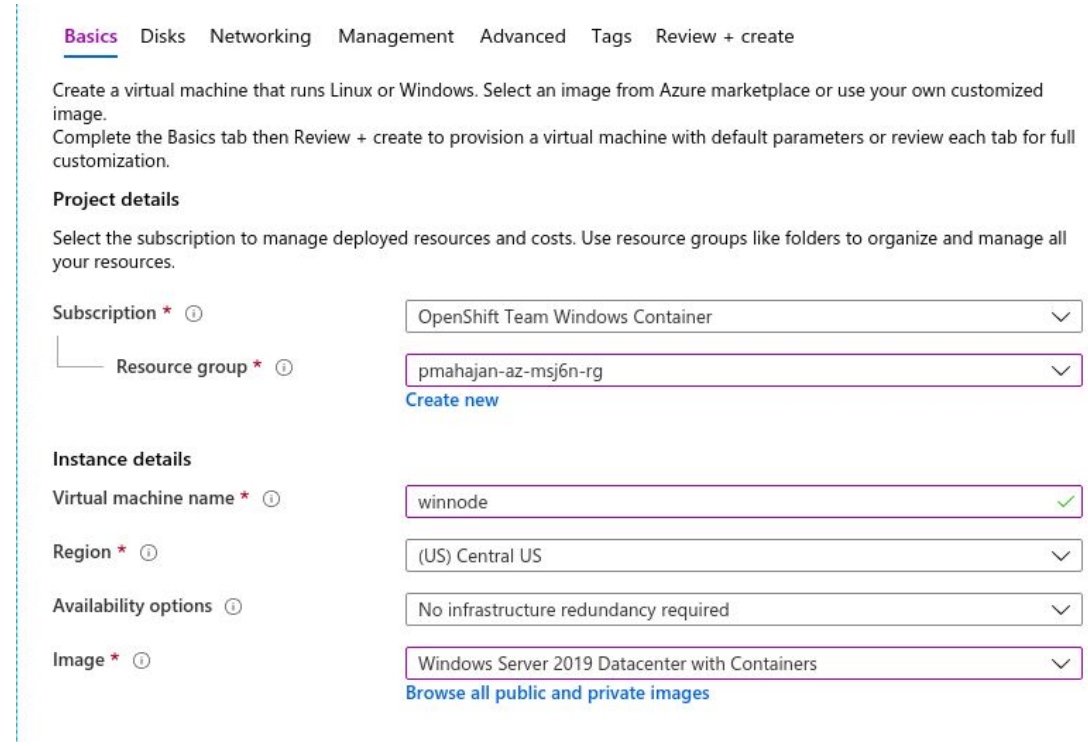
Login into the Azure portal. Navigate to the resource group related to your cluster, to see the masters and workers running. Select Create a resource on the top left:

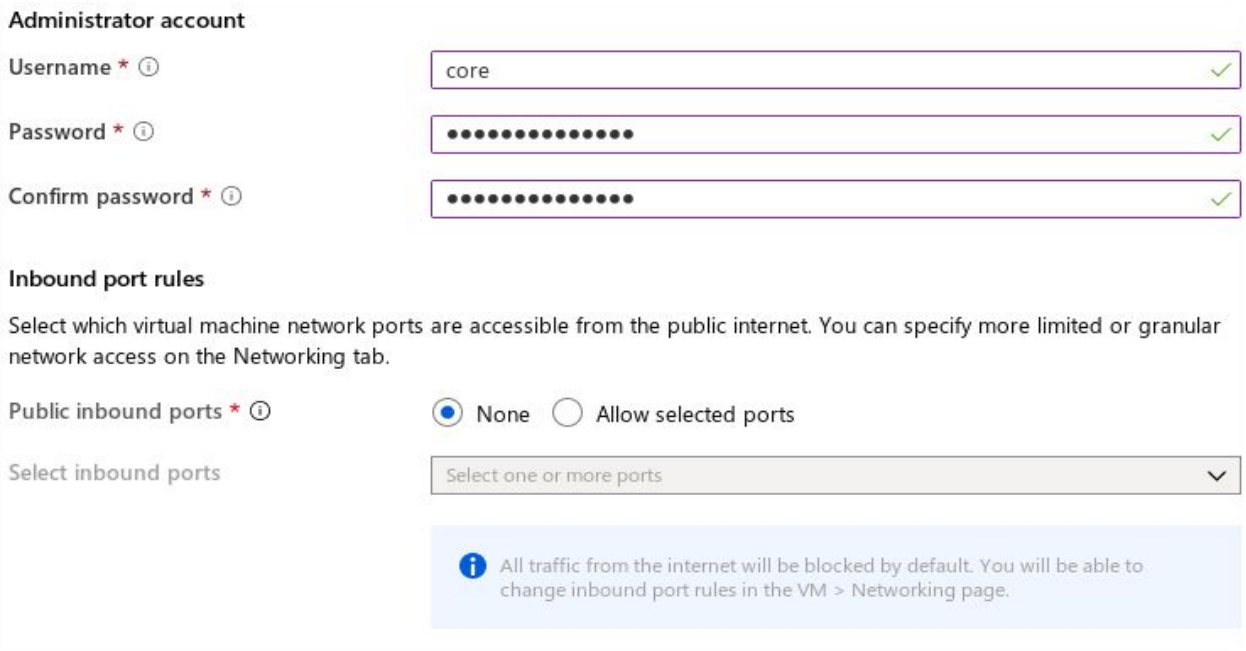


Select Virtual Machine under Compute:

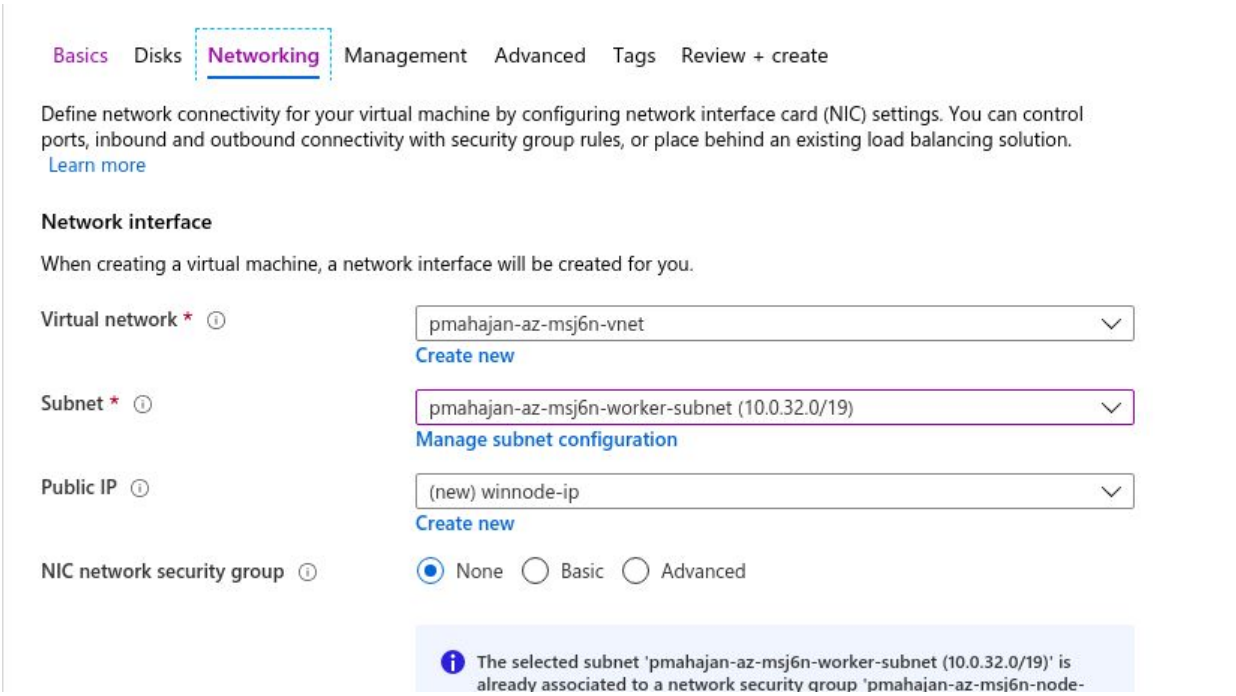


STEP 1: Provide your cluster's resource group and its region as the Resource Group and Region (We want to ensure the instance is created in the same region and is a part of the OpenShift cluster). The image should be Windows Server 2019 Datacenter with Containers and the size should be Standard_D2s_v3:
Note: The image might not be populated in the drop down automatically.

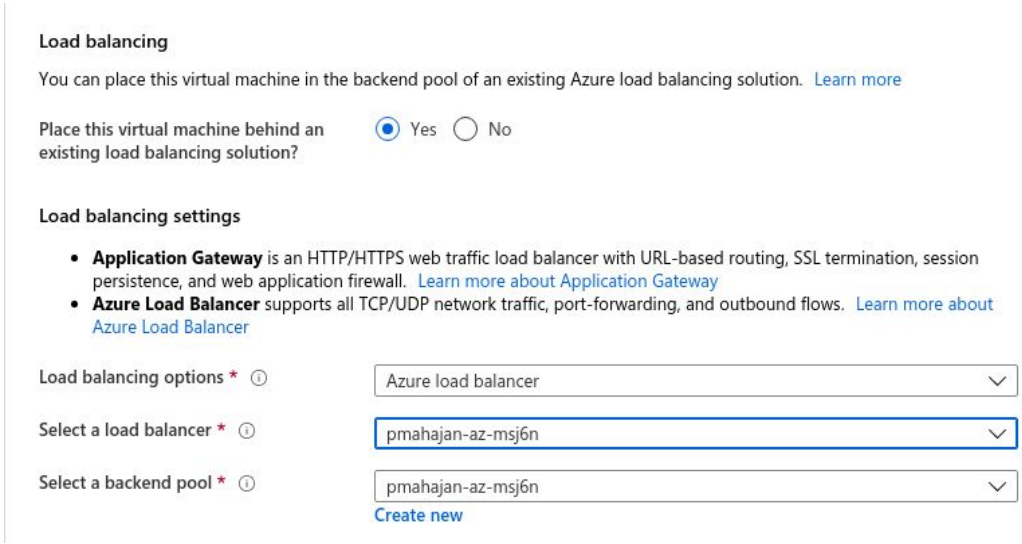




STEP 2: In the Networking tab, ensure the Virtual Network points to the same Virtual Network that your cluster is part of, and the subnet points to the worker subnet of the cluster. We also want a public ip to be created, so that we can RDP into the instance:

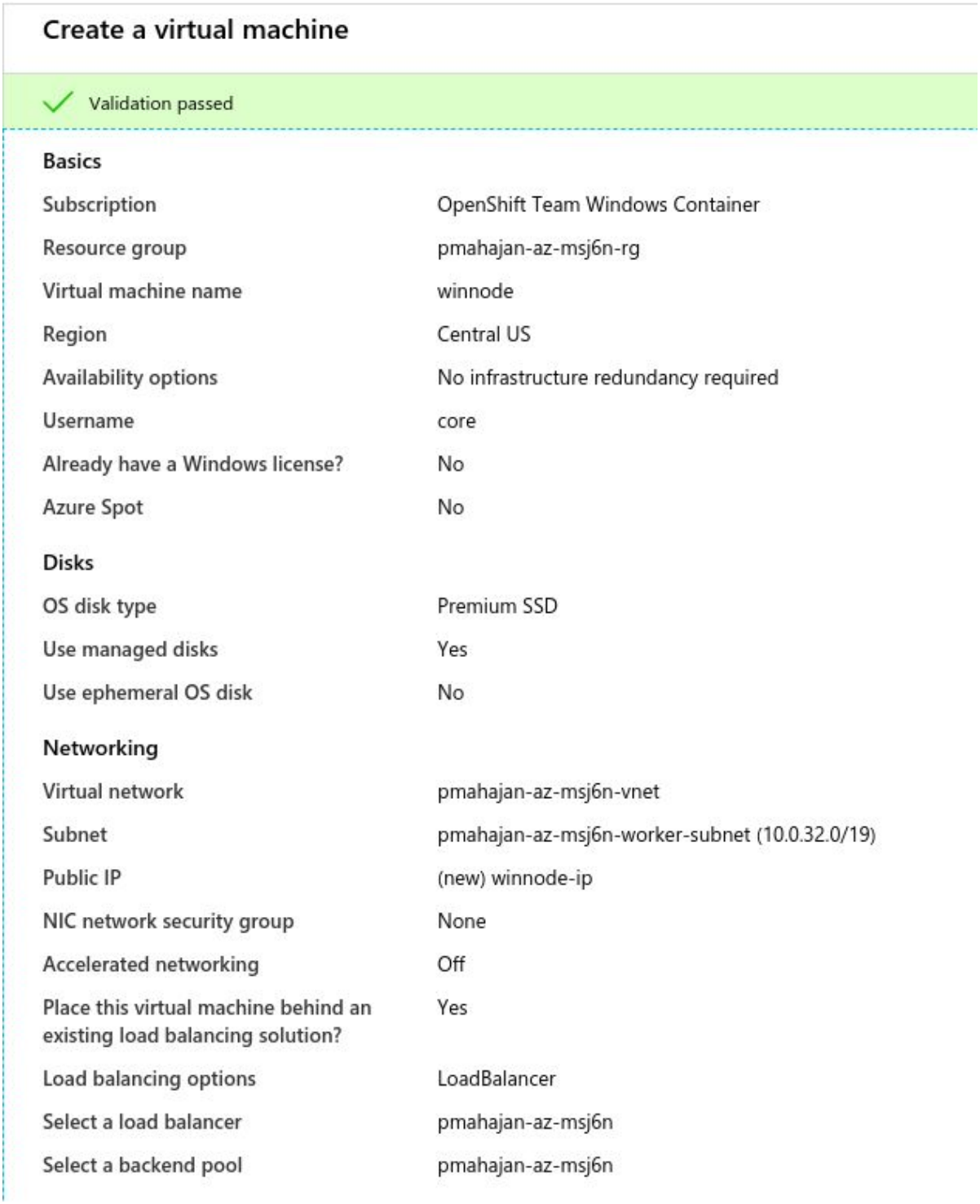


In the Load balancing section, enable the load balancer of type Azure load balancer and place the virtual machine behind the same load balancer as other worker nodes, i.e. a load balancer which is neither the `internal-lb` nor the `public-lb` as highlighted in the image below:
You might see 3 different load balancers similar to the ones shown in the image. Select the load balancer which corresponds to the name of your cluster. In the image below, pmahajan-az-msj6n is the load balancer that needs to be picked up for this example workflow.

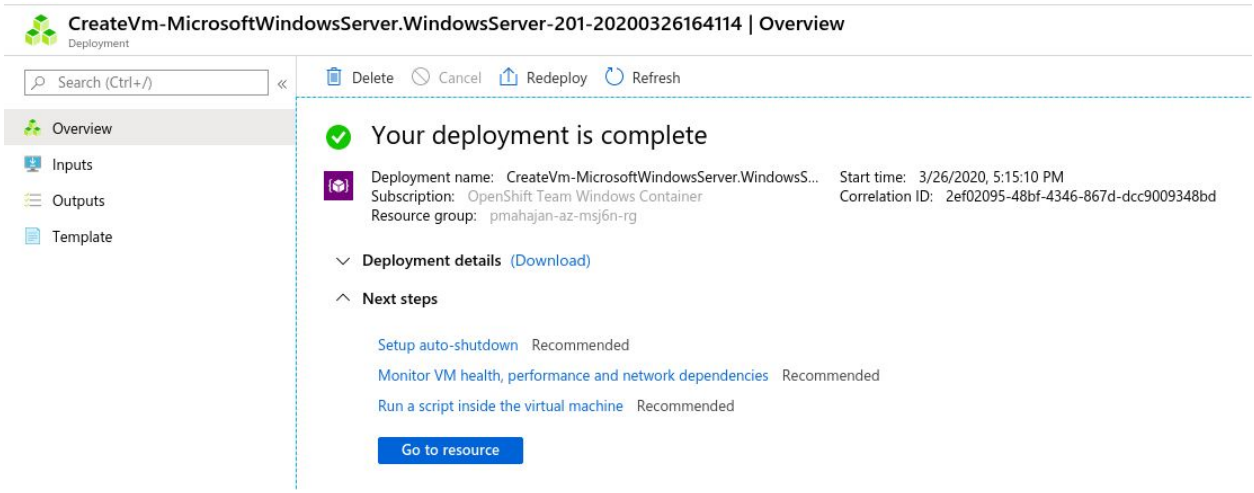


STEP 3: Under the Tags section, pick the tag of your cluster and set the value to owned:

STEP 4: Review and create the instance:



The portal will tell you once the instance is ready:



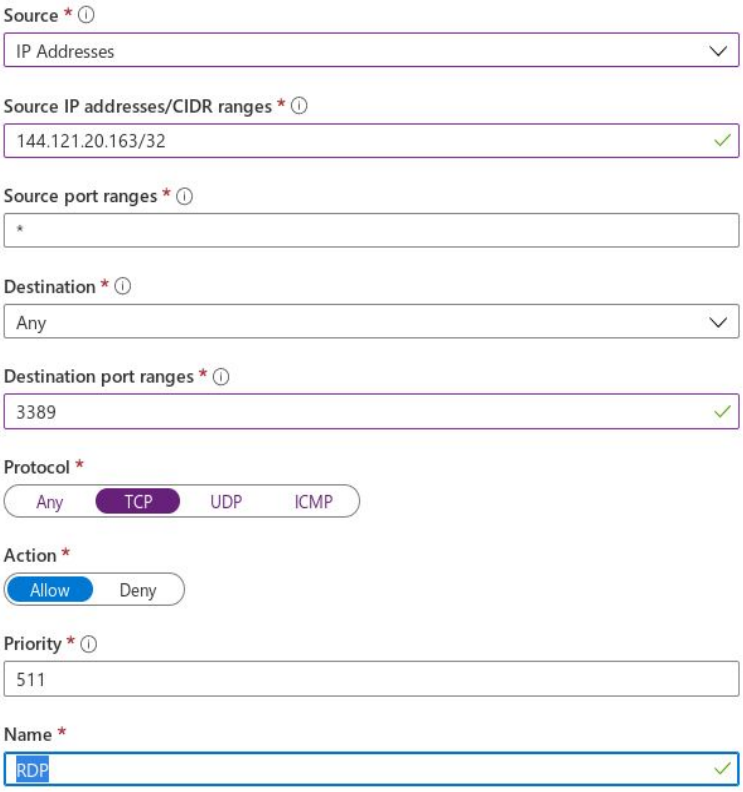
Now we need to set the security groups that will allow us to RDP to the node. Under your resource group, find the Network Security Group for the <cluster name>-node-nsg (This is the worker network security group)

STEP 5: Select the Inbound security rules on the left panel:

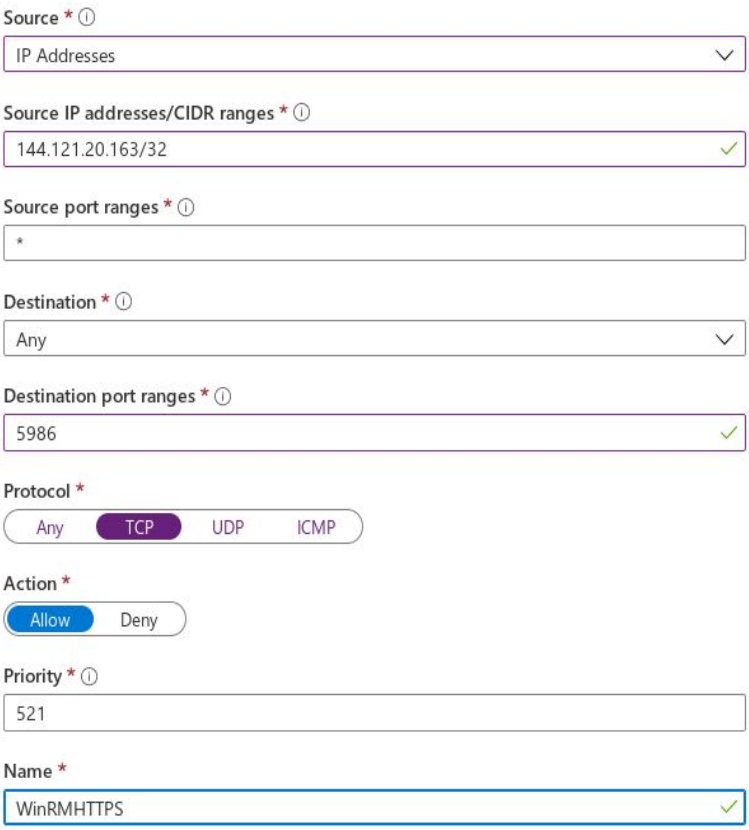
Add the following rules that will allow us to RDP into the instance, remotely manage with Ansible, and open up the communication from within the cluster. The Source IP for RDP and WinRM is the public ip address of your machine followed by /32.
You can get your public ip address by running the following command:

```
$ curl ifconfig.co
```

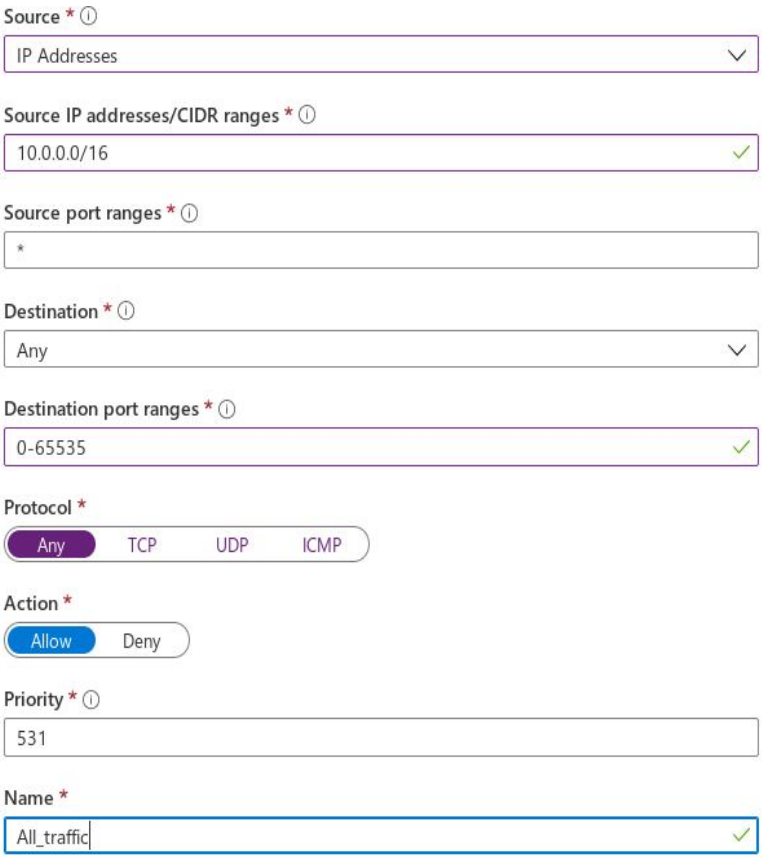
Security rule to allow access through RDP:



Security rule for WinRM:



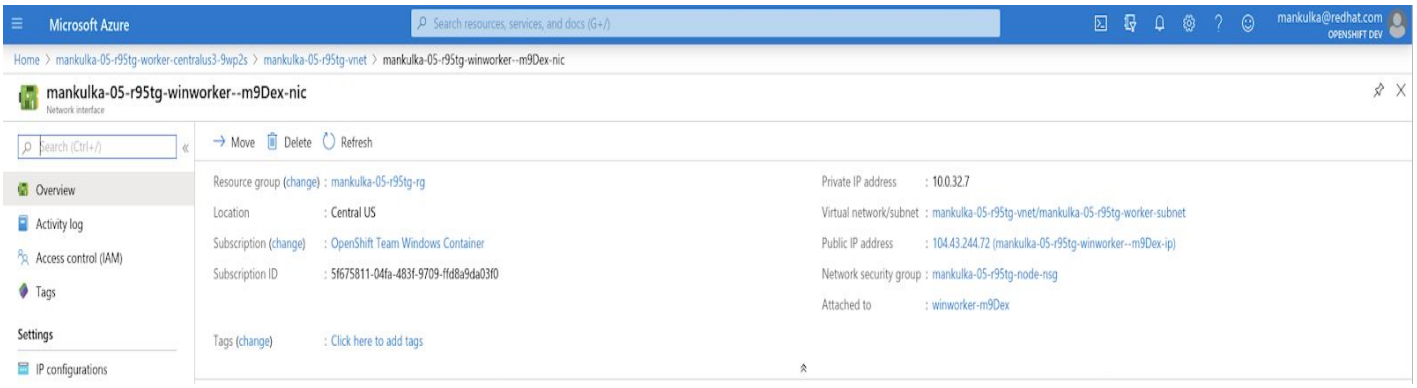
Security rule to open up communication within the cluster:



The final rules should look like the following:

Priority	Name	Port	Protocol	Source	Destination	Action
500	a9527ca8529d04412bd1f3ef...	80	TCP	Internet	52.141.216.100	Allow
501	a9527ca8529d04412bd1f3ef...	443	TCP	Internet	52.141.216.100	Allow
511	RDP	3389	TCP	144.121.20.163...	Any	Allow
521	WinRMHTTPS	5986	TCP	144.121.20.163...	Any	Allow
531	All_traffic	0-65535	Any	10.0.0.0/16	Any	Allow
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInBou...	Any	Any	AzureLoadBala...	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny

STEP 6: Note the private IP of your windows worker node. This is required for creating the host file in the Setup Ansible Connection step.



Now you can setup Ansible

RDP Setup

Install a RDP tool:

For Fedora, we used freerdp.

```
$ sudo dnf install freerdp
```

You can see the Public IP by looking at the <Name of Windows instance>-ip under your resource group
The user sets up the password while creating the virtual machine. There is no way to recover this once the machine is created. You can reset the password through Azure Portal > Virtual Machine > Settings > Reset Password

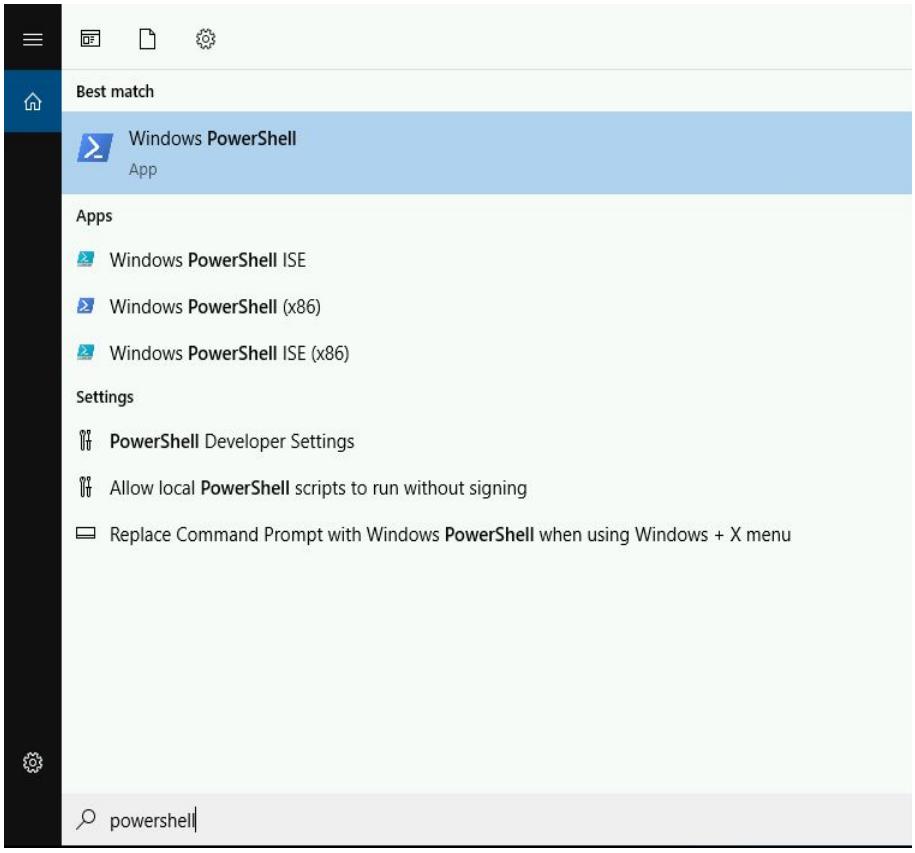

```
$ xfreerdp /u:<username of Windows instance> /v:<Public IP of Windows instance>
/p:'<password>' /f
```

You can now see a Windows screen come up.

Setup on Windows instance

Enable WinRM

Bring up PowerShell and execute the following commands:




```
> $url =
"https://raw.githubusercontent.com/ansible/ansible/devel/examples/scripts/ConfigureR
emotingForAnsible.ps1"
$file = "$env:temp\ConfigureRemotingForAnsible.ps1"
(New-Object -TypeName System.Net.WebClient).DownloadFile($url,  $file)
powershell.exe -ExecutionPolicy ByPass -File $file
```

Enable Console logs

Open TCP port 10250 in the Windows Firewall so that logs can be viewed in console and using *oc logs*. On your Windows instance, execute the following in a PowerShell window:

```
> New-NetFirewallRule -DisplayName "Enable console logs" -Direction Inbound -Action
Allow -Protocol TCP -LocalPort 10250 -EdgeTraversalPolicy Allow
```