Reference Architectures 2017
Deploying Red Hat OpenShift Container Platform 3.5 on Microsoft Azure

---

Glenn West Ryan Cook Eduardo Minguez

Reference Architectures 2017 Deploying Red Hat OpenShift Container Platform 3.5 on Microsoft Azure

Glenn West

Ryan Cook

Eduardo Minguez
refarch-feedback@redhat.com

Legal Notice

Abstract

The purpose of this document is to provide guidelines on deploying OpenShift Container Platform 3.5 on Microsoft Azure.

Table of Contents

COMMENTS AND FEEDBACK

In the spirit of open source, we invite anyone to provide feedback and comments on any reference architecture. Although we review our papers internally, sometimes issues or typographical errors are encountered. Feedback allows us to not only improve the quality of the papers we produce, but allows the reader to provide their thoughts on potential improvements and topic expansion to the papers. Feedback on the papers can be provided by emailing refarch-feedback@redhat.com. Please refer to the title within the email.

CHAPTER 1. EXECUTIVE SUMMARY

OpenShift Container Platform is built around a core of application containers powered by **Docker**, with orchestration and management provided by **Kubernetes**, on a foundation of **RHEL Atomic Host** and **Red Hat Enterprise Linux**. **OpenShift Origin** is the upstream community project that brings it all together along with extensions, to accelerate application development and deployment.

This reference document provides a comprehensive example demonstrating how **OpenShift Container Platform** can be set up to take advantage of the native high availability capabilities of **Kubernetes** and **Microsoft Azure** in order to create a highly available **OpenShift Container Platform** environment. The configuration consists of three **OpenShift Container Platform** masters, three **OpenShift Container Platform** infrastructure nodes, three to thirty **OpenShift Container Platform** application nodes, and native **Microsoft Azure** integration. In addition to the configuration, operational management tasks are shown to demonstrate functionality.

Please note the reference architecture deploys **OpenShift Container Platform** in the **Microsoft Azure public cloud**, not the **Microsoft Azure Stack** version nor the **Microsoft National Clouds** regional cloud version.

Note

The number of application nodes deployed by this reference architecture has been tested from 3 to 30. The sizing and limits supported by **OpenShift Container Platform** is explained in Planning document

CHAPTER 2. COMPONENTS AND CONFIGURATION

This chapter describes the reference architecture environment that is deployed providing a highly available **OpenShift Container Platform** environment on **Microsoft Azure**.

The image below provides a high-level representation of the components within this reference architecture. By using **Microsoft Azure**, resources are highly available using a combination of VM placement using **Azure Availability Sets**, **Azure Load Balancer (ALB)**, and **Azure VHD** persistent volumes. Instances deployed are given specific roles to support **OpenShift Container Platform**:

- The bastion host limits the external access to internal servers by ensuring that all **SSH** traffic passes through the bastion host.
- The master instances host the **OpenShift Container Platform** master components such as **etcd** and the **OpenShift Container Platform** API.
- The application node instances are for users to deploy their containers.
- Infrastructure node instances are used for the **OpenShift Container Platform** infrastructure elements like the **OpenShift Container Platform router** and **OpenShift Container Platform integrated registry**.

The authentication is managed by the **htpasswd** identity provider but **OpenShift Container Platform** can be configured to use any of the supported identity providers (including **GitHub**, **Google** or **LDAP**). **OpenShift Container Platform** on **Microsoft Azure** uses a combination of premium and standard storage, which is used for the filesystem of the instances and for persistent storage in containers.

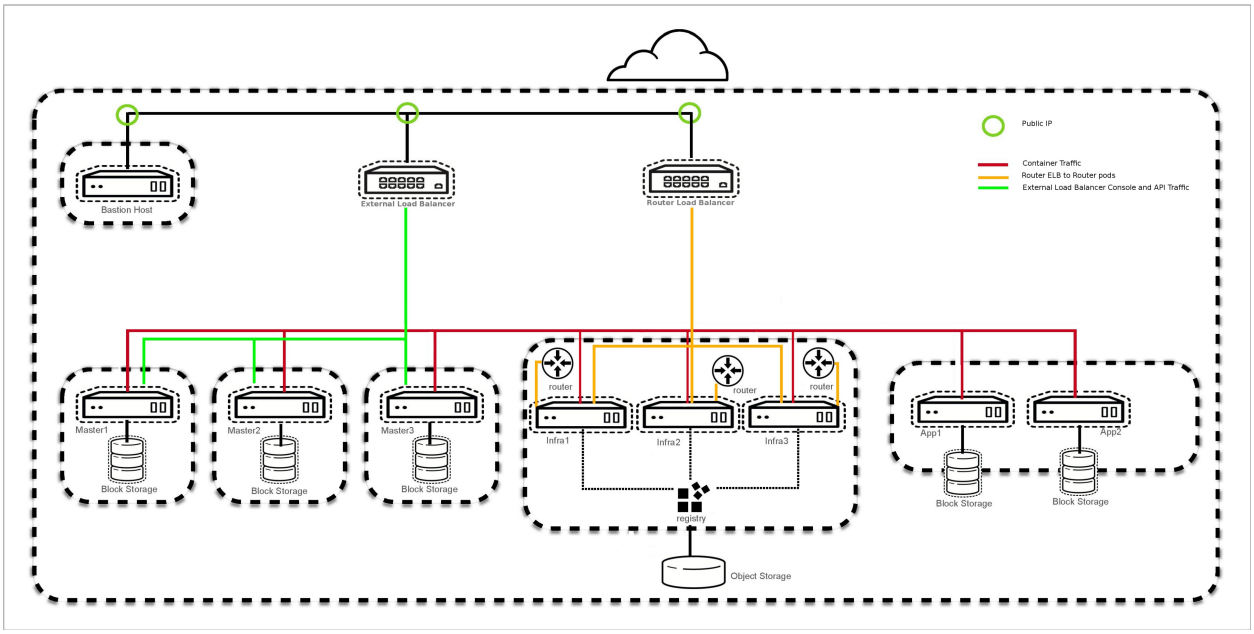
The network is configured to leverage two **Azure Load Balancers**:

- **External load balancer** gives access to the **OpenShift Container Platform** web console and API from outside the cluster
- **Router load balancer** for application access from outside the cluster

The **OpenShift Container Platform** web console and API can be accessed directly via the automatically created **DNS** entry while the application access is accessed using the **nip.io** service that provides a wildcard **DNS** A record to forward traffic to the **Router load balancer**.

Note

See Microsoft Azure DNS section for more information about the DNS configuration



This reference architecture breaks down the deployment into three separate phases.

- Phase 1: Provision the Virtual Machines on **Microsoft Azure**
- Phase 2: Install **OpenShift Container Platform** on **Microsoft Azure**
- Phase 3: Post deployment activities

For Phase 1, the provisioning of the environment is done using a series of **`Azure Resource Manager`** templates (**`ARM`**) provided in the openshift-ansible-contrib **`git`** repository. Once the infrastructure is deployed by **`ARM`**, as the last action, the **`ARM`** templates will start the next phase by running a bash script that starts phase 2.

Phase 2 is the provisioning of **OpenShift Container Platform** using the ansible playbooks installed by the **`openshift-ansible-playbooks`** RPM package. This is driven by a set of bash scripts that setup the inventory, setup parameters, and make sure all the needed playbooks are coordinated. As the last part of phase 2, the router and registry are deployed.

The last phase, Phase 3, concludes the deployment, which is done manually. This consists of optionally configure a custom DNS entry to point to the application load balancers (to avoid the default **nip.io** domain) and by manually verifying the configuration. This is done by running tools like **`oadm diagnostics`** and the systems engineering teams validation ansible playbook.

Note

The scripts provided in the **GitHub** repository are not supported by **Red Hat**. They merely provide a mechanism that can be used to build out an **OpenShift Container Platform** environment.

2.1. MICROSOFT AZURE CLOUD INSTANCE DETAILS

Within this reference environment, the instances are deployed in a single **`Azure Region`** which can be selected when running the **`ARM`** template. Although the default region can be changed, the reference architecture deployment should only be used in regions with premium storage for performance reasons.

All VMs are created using the **On-Demand Red Hat Enterprise Linux** (**RHEL**) image and the size used by default is **Standard_DS4_v2** for masters and nodes and **Standard_DS1_v2** for the bastion host. Instance sizing can be changed when the **ARM** template is run which is covered in later chapters.

Note

For higher availability, multiple clusters should be created, and federation should be used. This architecture is emerging and will be described in future reference architecture.

2.1.1. Microsoft Azure Cloud Instance Storage Details

Linux VMs in **Microsoft Azure** are created by default with two virtual disks attached, where the first one is the **operating system disk** and the second one is a **temporary disk** where the data persistence is not guaranteed and it is used by default to store a swap file created by the **Azure Linux Agent**.

As a best practice, instances deployed to run containers in **OpenShift Container Platform** include a dedicated disk (**datadisk**) configured to store the container images as well as a dedicated disk configured to store the **emptyDir** volumes. The disk setup is provided in the **ARM** template of each virtual machine type like **master.json** in the git repository for **OpenShift Container Platform** master instances.

Data disks can be created up to 1023 GB where **operating system disk** and **temporary disk** size depend on the size of the virtual machine, where the default **Standard_DS4_v2** used in this reference architecture for masters and nodes is:

Table 2.1. Instance Storage details for masters and nodes by default

Type	Name	Mountpoint	Size	Purpose
operating system disk	sda	**/boot** & **/**	32GB	Root filesystem
temporary disk	sdb	**/mnt/resource**	128 GB	Temporary storage
data disk	sdc	**/var/lib/origin/openshift.local.volumes**	128 GB	**OpenShift Container Platform emptyDir** volumes
data disk	sdd	none	128 GB	Docker images storage

The following is a sample output in a **OpenShift Container Platform** master virtual machine deployed using this reference architecture where the mountpoints as well as the disks can be seen as described:

```
$ lsblk
NAME                                 MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
fd0                                    2:0    1    4K  0 disk
sda                                    8:0    0   32G  0 disk
├─sda1                                 8:1    0  500M  0 part /boot
└─sda2                                 8:2    0 31,5G  0 part /
sdb                                    8:16   0   28G  0 disk
└─sdb1                                 8:17   0   28G  0 part
/mnt/resource
sdc                                    8:32   0  128G  0 disk
└─sdc1                                 8:33   0  128G  0 part
/var/lib/origin/openshift.local.volumes
sdd                                    8:48   0  128G  0 disk
└─sdd1                                 8:49   0  128G  0 part
  ├─docker--vg-docker--pool_tmeta 253:0    0  132M  0 lvm
  │ └─docker--vg-docker--pool     253:2    0   51G  0 lvm
  └─docker--vg-docker--pool_tdata 253:1    0   51G  0 lvm
    └─docker--vg-docker--pool     253:2    0   51G  0 lvm
sr0                                   11:0    1  1,1M  0 rom
```

Tip

Swap is disabled automatically in the installation with the git repository scripts in nodes where pods will run as a best practice

Note

For more detail about the **`emptyDir`** and container image storage, see the Management of Maximum Pod Size section

The bastion host only has the default **`operating system disk`** and **`temporary disk`**, where in the **`Standard_DS1_v2`** virtual machine size are:

Table 2.2. Instance Storage details for bastion by default

Type	Name	Mountpoint	Size	Purpose
`operating system disk`	sda	**`/boot`** & **`/`**	32GB	Root filesystem
`temporary disk`	sdb	**`/mnt/resource`**	128 GB	Temporary storage

All the disks created by this reference architecture for the virtual machines use the **Azure Premium Disk** to performance reasons (high throughput and IOPS).

Note

For more information, see about disks and VHDs for Azure Linux VMs

2.2. MICROSOFT AZURE LOAD BALANCER DETAILS

Two **Azure Load Balancers** (**ALB**) are used in this reference environment. The table below describes the **ALB**, the load balancer **DNS** name, the instances in which the **Azure Load Balancers** (**ALB**) is attached, and the port monitored by the load balancer to state whether an instance is in or out of service.

Table 2.3. Microsoft Azure Load Balancer

ALB	DNS name	Assigned Instances	Port
External load balancer	<resourcegroupname>.<region>.cloudapp.azure.com	master1-3	8443
Router load balancer	<wildcardzone>.<region>.cloudapp.azure.com	infra-nodes1-3	80 and 443

The **External load balancer** utilizes the **OpenShift Container Platform** master API port for communication internally and externally. The **Router load balancer** uses the public subnets and maps to infrastructure nodes. The infrastructure nodes run the router pod which then directs traffic directly from the outside world into pods when external routes are defined.

To avoid reconfiguring DNS every time a new route is created, an external wildcard A **DNS** entry record must be configured pointing to the **Router load balancer** IP.

For example, create a wildcard DNS entry for **cloudapps.example.com** that has a low time-to-live value (TTL) and points to the public IP address of the **Router load balancer**:

```
*.cloudapps.example.com. 300 IN A 192.168.133.2
```

2.3. SOFTWARE VERSION DETAILS

The following tables provide the installed software versions for the different servers that make up the **Red Hat OpenShift Container Platform** highly available reference environment.

Table 2.4. RHEL OSEv3 Details

Software	Version
Red Hat Enterprise Linux 7.3 x86_64	kernel-3.10.0-327
Atomic-OpenShift{master/clients/node/sdn-ovs/utils}	3.5
Docker	1.12.x
Ansible	2.2.1

2.4. REQUIRED CHANNELS

A subscription to the following channels is required in order to deploy this reference environment's configuration.

Table 2.5. Required Channels - OSEv3 Master and Node Instances

Channel	Repository Name
Red Hat Enterprise Linux 7 Server (RPMs)	rhel-7-server-rpms
Red Hat OpenShift Enterprise 3.5 (RPMs)	rhel-7-server-ose-3.5-rpms
Red Hat Enterprise Linux 7 Server - Extras (RPMs)	rhel-7-server-extras-rpms
Red Hat Enterprise Linux 7 Server - Fast Datapath (RPMs)	rhel-7-fast-datapath-rpms

The subscriptions are accessed via a **`pool id`**, which is a required parameter in the **`ARM`** template that will deploy the VMs in the **Microsoft Azure** environment and it is located in the **`reference-architecture/azure-ansible/azuredeploy.parameters.json`** file in the **`openshift-ansible-contrib`** repository

Note

The **`pool id`** can be obtained in the **Subscriptions** section of the **Red Hat Customer Portal**, by selecting the appropriate subscription that will open a detailed view of the subscription, including the Pool ID

2.5. PREREQUISITES

This section describes the environment and setup needed to execute the **`ARM`** template and perform post installation tasks.

2.5.1. GitHub Repositories

The code in the **`openshift-ansible-contrib`** repository referenced below handles the installation of **OpenShift Container Platform** and the accompanying infrastructure. The **`openshift-ansible-contrib`** repository is not explicitly supported by Red Hat but the Reference Architecture team performs testing to ensure the code operates as defined and is secure.

https://github.com/openshift/openshift-ansible-contrib/tree/master/reference-architecture/azure-ansible

For this reference architecture, the scripts are accessed and used directly from **GitHub**. There is no requirement to download the code, as it's done automatically once the script is started.

2.6. MICROSOFT AZURE SUBSCRIPTION

In order to deploy the environment from the template, a **Microsoft Azure** subscription is required. A trial subscription is not recommended, as the reference architecture uses significant resources, and the typical trial subscription does not provide adequate resources.

The deployment of **OpenShift Container Platform** requires a user that has the proper permissions by the **Microsoft Azure** administrator. The user must be able to create accounts, storage accounts, roles, policies, load balancers, and deploy virtual machine instances. It is helpful to have delete permissions in order to be able to redeploy the environment while testing.

2.7. MICROSOFT AZURE REGION SELECTION

An **OpenShift Container Platform** cluster is deployed with-in one **`Azure Region`**. In order to get the best possible availability in **Microsoft Azure**, availability sets are implemented.

In **Microsoft Azure**, virtual machines (VMs) can be placed in to a logical grouping called an **`availability set`**. When creating VMs within an availability set, the **Microsoft Azure** platform distributes the placement of those VMs across the underlying infrastructure. Should there be a planned maintenance event to the **Microsoft Azure** platform or an underlying hardware/infrastructure fault, the use of availability sets ensures that at least one VM remains running. The **Microsoft Azure** SLA requires two or more VMs within an availability set to allow the distribution of VMs across the underlying infrastructure.

2.8. SSH PUBLIC AND PRIVATE KEY

SSH keys are used instead of passwords in the **OpenShift Container Platform** installation process. These keys are generated on the system that will be used to login and manage the system. In addition, they are automatically distributed by the **ARM** template to all virtual machines that are created.

In order to use the template, **SSH** public and private keys are needed. To avoid asking for the passphrase, do not not apply a passphrase to the key.

The public key will be injected in the **~/.ssh/authorized_keys** file in all the hosts, and the private key will be copied to the **~/.ssh/id_rsa** file in all the hosts to allow **SSH** communication within the environment (i.e.- from the bastion to master1 without passwords).

2.8.1. SSH Key Generation

If **SSH** keys do not currently exist then it is required to create them. Generate an RSA key pair by typing the following at a shell prompt:

```
$ ssh-keygen -t rsa -N '' -f /home/USER/.ssh/id_rsa
```

A message similar to this will be presented indicating they key has been successful created

```
Your identification has been saved in /home/USER/.ssh/id_rsa.
Your public key has been saved in /home/USER/.ssh/id_rsa.pub.
The key fingerprint is:
e7:97:c7:e2:0e:f9:0e:fc:c4:d7:cb:e5:31:11:92:14
USER@sysdeseng.rdu.redhat.com
The key's randomart image is:
+--[ RSA 2048]----+
|             E.  |
|            . .  |
|             o . |
|              . .|
|        S .    . |
|         + o o ..|
|          * * +oo|
|           O +..=|
|           o*  o.|
+-----------------+
```

2.9. RESOURCE GROUPS AND RESOURCE GROUP NAME

In the **Microsoft Azure** environment, resources such as storage accounts, virtual networks and virtual machines (VMs) are grouped together in **resource groups** as a single entity and their names must be unique to an **Microsoft Azure** subscription. Note that multiple **resource groups** are supported in a region, as well as having the same **resource group** in multiple regions but a **resource group** may not span resources in multiple regions.

Note

For more information about **Microsoft Azure** Resource Groups, check the Azure Resource Manager overview documentation

2.10. MICROSOFT AZURE VIRTUAL NETWORK (VNET)

An **`Azure VNet`** provides the ability to set up custom virtual networking which includes subnets, and IP address ranges. In this reference implementation guide, a dedicated **`VNet`** is created with all its accompanying services to provide a stable network for the **OpenShift Container Platform** deployment.

A **`VNet`** is created as a logical representation of a networking environment in the **Microsoft Azure** cloud. The following subnets and CIDR listed below are used.

Important

Substitute the values if needed to ensure no conflict with an existing CIDR or subnet in the environment. The values are defined in the template https://github.com/openshift/openshift-ansible-contrib/tree/master/reference-architecture/azure-ansible/azuredeploy.json

Table 2.6. VNet Networking

CIDR/Subnet	Values
CIDR	10.0.0.0/16
Master Subnet	10.0.0.0/24
Node Subnet	10.0.1.0/24
Infra Subnet	10.0.2.0/24

The **`VNet`** is created and a human readable tag is assigned. Three subnets are created in the **`VNet`**. The bastion instance is on the Master Subnet. The two internal load balancers allow access to the **OpenShift Container Platform** API and console and the routing of application traffic. All the VMs are able to communicate to the internet for packages, container images, and external git repositories.

Note

For more information see Azure Virtual Networks documentation

2.11. OPENSHIFT SDN

OpenShift Container Platform uses a software-defined networking (**SDN**) approach to provide a unified cluster network that enables communication between pods across the **OpenShift Container Platform** cluster. This pod network is established and maintained by the OpenShift SDN, which configures an overlay network using **Open vSwitch** (**OVS**).

There are three different plug-ins available in **OpenShift Container Platform** 3.5 for configuring the pod network:

- The **redhat/ovs-subnet** plug-in which provides a "flat" pod network where every pod can communicate with every other pod and service.
- The **redhat/ovs-multitenant** plug-in which provides **OpenShift Container Platform** project level isolation for pods and services. Each project receives a unique Virtual Network ID (**VNID**) that identifies traffic from pods assigned to the project. Pods from different projects cannot send packets to or receive packets from pods and services of a different project.
- The **redhat/ovs-networkpolicy** plug-in (currently in Tech Preview) allows project administrators to configure their own isolation policies using **NetworkPolicy** objects.

The plugin used in this reference architecture can be specified among the supported ones at deployment time using the **ARM** template. The default value is **redhat/ovs-multitenant** that allows multitenant isolation for pods per project.

For more information about **OpenShift Container Platform** networking, see OpenShift SDN documentation.

2.12. MICROSOFT AZURE NETWORK SECURITY GROUPS

The purpose of the **Microsoft Azure** Network security groups (**NSG**) is to restrict traffic from outside of the **VNet** to servers inside of the **VNet**. The Network security groups also are used to restrict server to server communications inside the **VNet**. Network security groups provide an extra layer of security similar to a firewall: in the event a port is opened on an instance, the security group will not allow the communication to the port unless explicitly stated in a Network security group.

NSG are grouped depending on the traffic flow (inbound or outbound) and every **NSG** contains rules where every rule specify:

- priority
- source
- destination
- service (network port and network protocol)
- action on the traffic (allow or deny)

NSG rules are processed by priority meaning the first rule matching the traffic it is applied.

All the security groups contains default rules to block connectivity coming from outside the **VNet**, where the default rules allow and disallow traffic as follows:

- Virtual network: Traffic originating and ending in a virtual network is allowed both in inbound and outbound directions.
- Internet: Outbound traffic is allowed, but inbound traffic is blocked.
- Load balancer: Allow **Microsoft Azure** load balancer to probe the health of the **VMs**.

Once the Network security group is created, it should be associated to an infrastructure component where using the resource manager mechanism to deploy infrastructure in **Microsoft Azure** they can be associated to a NIC or a subnet.

Note

In this reference architecture, every **VM** is associated to a single NIC, and the Network security groups are associated to NICs, therefore there will be a 1:1:1 relationship between **VM**, NIC and Network security group. For more information about the**Microsoft Azure** Network security groups, see Filter network traffic with Network security groups

The Network security groups are specified on each node type **json** file, located in https://github.com/openshift/openshift-ansible-contrib/tree/master/reference-architecture/azure-ansible/ (like **master.json** for master instances)

2.12.1. Bastion Security Group

The bastion Network security group allows **SSH** connectivity from the outside to the bastion host. Any connectivity via **SSH** to the master, application or infrastructure nodes must go through the bastion host.

The Network security group applied to the bastion host NIC is called **bastionnsg** and contains the following rules:

NSG rule name	Type	Source	Destination	Service	Action
default-allow-ssh	Inbound	Any	Any	SSH (TCP/22)	Allow

2.12.2. Master Nodes Security Group

The master nodes Network security group allows inbound access on port 8443 from the internet to the virtual network. The traffic is then allowed to be forwarded to the master nodes.

The Network security group applied to every master node instances' NIC are called **master1nsg**, **master2nsg** and **master3nsg** and contain the following rules:

NSG rule name	Type	Source	Destination	Service	Action
default-allow-openshift-master	Inbound	Any	Any	Custom (TCP/8443)	Allow

2.12.3. Infrastructure nodes Security Group

The infrastructure nodes Network security group allows inbound access on port 80 and 443. If the applications running on the **OpenShift Container Platform** cluster are using different ports this can be adjusted as needed.

The Network security group applied to every infrastructure node instances' NIC are called **`infranode1nsg`**, **`infranode2nsg`** and **`infranode3nsg`** and contain the following rules:

NSG rule name	Type	Source	Destination	Service	Action
default-allow-openshift-router-http	Inbound	Any	Any	HTTP (TCP/80)	Allow
default-allow-openshift-router-https	Inbound	Any	Any	HTTPS (TCP/443)	Allow

2.13. MICROSOFT AZURE DNS

DNS is an integral part of a successful **OpenShift Container Platform** environment. **Microsoft Azure** provides a DNS-as-a-Service called **`Azure DNS`**, per Microsoft; "The Microsoft global network of name servers has the scale and redundancy to ensure ultra-high availability for your domains. With **Microsoft Azure** DNS, you can be sure that your DNS will always be available."

Microsoft Azure provides the DNS for public zone, as well as internal host resolution. These are configured automatically during the execution of the reference architecture scripts.

Note

For more information see Azure DNS documentation

2.13.1. Public Zone

When the reference architecture is deployed, **Microsoft Azure** supplied domains are used. The domains consists of: <hostname>.<region>.cloudapp.azure.com.

For each **OpenShift Container Platform** deployment on **Microsoft Azure**, there are three created domain names:

- <resourcegroup>.<region>.cloudapp.azure.com - The API and **OpenShift Container Platform** web console load balancer
- <resourcegroup>b.<region>.cloudapp.azure.com - The bastion host for ssh access
- <wildcardzone>.<region>.cloudapp.azure.com - The DNS of the applications load balancer

Important

Due to the current **Microsoft Azure** limitations on creating subdomains and wildcards, the **nip.io** service is used for the application load balancer. For more information about current **Microsoft Azure** limitations with subdomains, check the following link subdomain cloudapp.net rather than having a global namespace

Note

In order to have a proper wildcard DNS entry with a proper subdomain like **`*.apps.mycompany.com`**, it is recommended to create a wildcard A record externally with your **`DNS`** domain provider and configure it to the applications load balancer IP like: **`*.apps.mycompany.com. 300 IN A 192.168.133.2`** To reflect those modifications in **OpenShift Container Platform**, modify the **`routingConfig.subdomain`** parameter in the **`/etc/origin/master/master-config.yaml`** file in all the masters and restart the **`atomic-openshift-master`** service, or modify the ansible hosts file and rerun the installation.

2.13.2. Internal resolution

This reference architecture uses Azure-provided name resolution mechanism which:

- creates an internal subdomain per **`resource group`** like **`fesj5eh111uernc5jfpnxi33kh.dx.internal.cloudapp.net`**
- creates an A **`DNS`** record on that internal subdomain of every instance deployed in that **`resource group`**
- configures the proper resolution in the **`/etc/resolv.conf`** file on every VM

Using this, instances can be reached using just the VM shortname:

```
$ cat /etc/resolv.conf
# Generated by NetworkManager
search fesj5eh114uebnc5jfpnxi33kh.dx.internal.cloudapp.net
nameserver 168.63.129.16

$ nslookup master1
Server:  168.63.129.16
Address: 168.63.129.16#53

Name: master1.fesj5eh114uebnc5jfpnxi33kh.dx.internal.cloudapp.net
Address: 10.0.0.5
```

2.13.3. Microsoft Azure VM Images

`Azure Virtual Machines Images` provide different virtual machine images to launch instances. In this reference architecture, the **`On-Demand Red Hat Enterprise Linux`** (**`RHEL`**) image in the **Azure Marketplace** is used.

Important

The **Red Hat Enterprise Linux** image carries an additional charge in addition to the base **`Linux VM`** price. For more information on **Microsoft Azure** pricing for Red Hat images see Azure Documentation.

2.13.4. Microsoft Azure VM Sizes

Microsoft Azure offers different VM sizes that can be used to deploy the **OpenShift Container Platform** environment. Further, all the nodes, have been selected with **`premium storage`**, to allow the best performance.

Note

The sizes provided in this reference architecture are a guide but it is advised to see the OpenShift Container Platform 3 Sizing Considerations for more information

The VM sizes are specified as parameters in the template file **`reference-architecture/azure-ansible/azuredeploy.json`** and the following table shows the specific parameter of each VM type and its default value:

Table 2.7. Default VM sizes

Type	Parameter	Default size
Bastion	**`bastionVMSize`**	**`Standard_DS1_v2`**
Masters	**`masterVMSize`**	**`Standard_DS4_v2`**
Infrastructure nodes	**`infranodeVMSize`**	**`Standard_DS4_v2`**
Application nodes	**`nodeVMSize`**	**`Standard_DS4_v2`**

Application node VM size is an important parameter for selecting how many containers as well as how big containers will be. The current default value for application nodes size allocates 8 CPU cores and 28 Gigabytes of memory for each VM. If the containers are memory intensive then it is advised to either increase the node count, or increase the node memory size. For these applications, choosing a **`Standard_D14_v2`** size will give 112 Gigabytes of memory or another VM size with more memory if needed.

2.13.5. Identity and Access Management

For this reference account, a **Microsoft Azure** account is required. Ideally this is either a pay-as-you-go account, or a **Microsoft Enterprise Agreement**.

You must have enough resources to deploy the reference architecture, otherwise the installation will fail.

During the installation of **OpenShift Container Platform** using the reference architecture scripts and playbooks, six storage accounts are created automatically per cluster. The following table shows the name of every storage account and its purpose:

Table 2.8. Storage accounts

Name	Purpose
`samas<resourcegroup>`	Masters storage
`sanod<resourcegroup>`	Application nodes storage
`sainf<resourcegroup>`	Infrastructure nodes storage
`sareg<resourcegroup>`	Registry persistent volume storage
`sapv<resourcegroup>`	The **`generic`** storage class
`sapvlm<resourcegroup>`	Storage account where the metrics and logging volumes will be stored

Note

For more information about the **Microsoft Azure** identity management and storage accounts, see The fundamentals of Azure identity management and About Azure storage accounts

2.14. BASTION

The bastion server implements mainly two distinct functions. One is that of a secure way to connect to all the nodes, and second that of the "installer" of the system. The information provided to the **ARM `template`** is passed to the bastion host and from there, playbooks and scripts are automatically generated and executed, resulting in **OpenShift Container Platform** being installed.

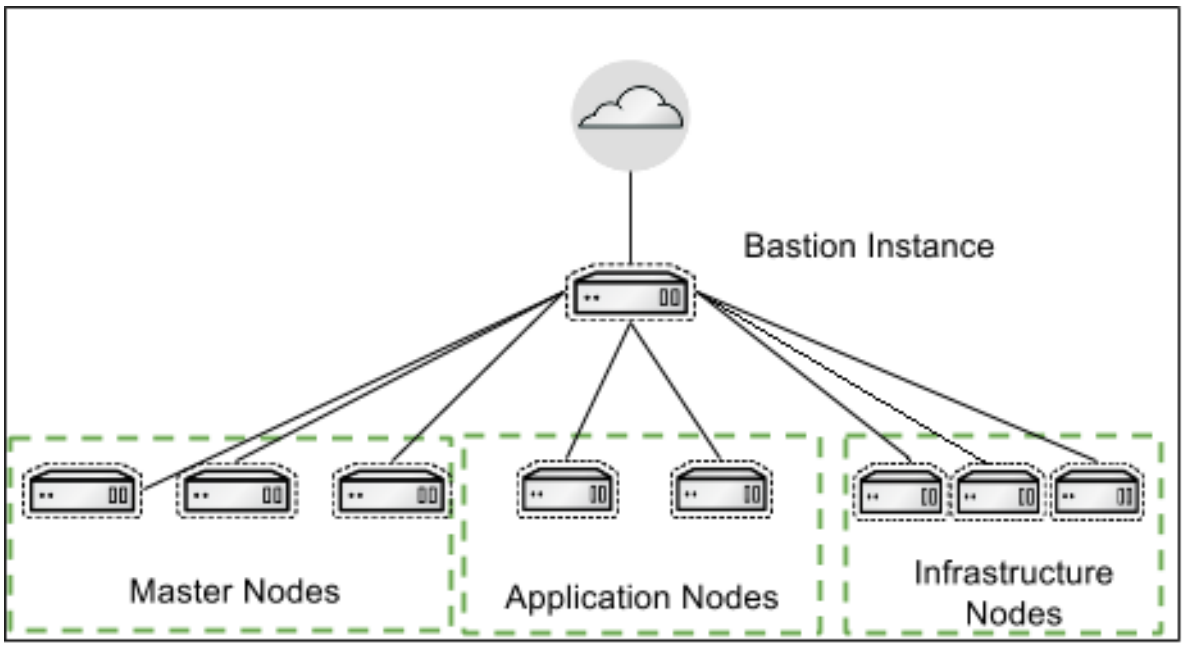
As shown in the Figure 2.1, “bastion diagram” the bastion server in this reference architecture provides a secure way to limit **SSH** access to the **Microsoft Azure** environment. The master and node security groups only allow for **SSH** connectivity between nodes inside of the Security Group

while the bastion allows **SSH** access from everywhere. The bastion host is the only ingress point for **SSH** in the cluster from external entities. When connecting to the **OpenShift Container Platform** infrastructure, the bastion forwards the request to the appropriate server.

Note

Connecting to other VMs through the bastion server requires specific **SSH** configuration which is outlined in the deployment section of the reference architecture guide.

Figure 2.1. bastion diagram



2.15. GENERATED INVENTORY

Ansible relies on inventory files and variables to perform playbook runs. As part of the reference architecture provided Ansible playbooks, the inventory is created during the boot of the bastion host. The **`Azure Resource Manager`** templates (**ARM**) passes parameters via a script extension to **RHEL** on the bastion. On the bastion host a bastion.sh script generates the inventory file in **`/etc/ansible/hosts`**.

Dynamic Inventory Script within bastion.sh

```
[OSEv3:children]
masters
nodes
etcd
new_nodes
new_masters

[OSEv3:vars]
osm_controller_args={'cloud-provider': ['azure'], 'cloud-config':
['/etc/azure/azure.conf']}
osm_api_server_args={'cloud-provider': ['azure'], 'cloud-config':
['/etc/azure/azure.conf']}
```

```
openshift_node_kubelet_args={'cloud-provider': ['azure'], 'cloud-config':
['/etc/azure/azure.conf'], 'enable-controller-attach-detach': ['true']}
debug_level=2
console_port=8443
docker_udev_workaround=True
openshift_node_debug_level="{{ node_debug_level | default(debug_level,
true) }}"
openshift_master_debug_level="{{ master_debug_level | default(debug_level,
true) }}"
openshift_master_access_token_max_seconds=2419200
openshift_hosted_router_replicas=3
openshift_hosted_registry_replicas=3
openshift_master_api_port="{{ console_port }}"
openshift_master_console_port="{{ console_port }}"
openshift_override_hostname_check=true
osm_use_cockpit=false
openshift_release=v3.5
openshift_cloudprovider_kind=azure
openshift_node_local_quota_per_fsgroup=512Mi
azure_resource_group=${RESOURCEGROUP}
rhn_pool_id=${RHNPOOLID}
openshift_install_examples=true
deployment_type=openshift-enterprise
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login':
'true', 'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider',
'filename': '/etc/origin/master/htpasswd'}]
openshift_master_manage_htpasswd=false

os_sdn_network_plugin_name=${OPENSHIFTSDN}

# default selectors for router and registry services
openshift_router_selector='role=infra'
openshift_registry_selector='role=infra'

# Select default nodes for projects
osm_default_node_selector="role=app"
ansible_become=yes
ansible_ssh_user=${AUSERNAME}
remote_user=${AUSERNAME}

openshift_master_default_subdomain=${WILDCARDNIP}
osm_default_subdomain=${WILDCARDNIP}
openshift_use_dnsmasq=true
openshift_public_hostname=${RESOURCEGROUP}.${FULLDOMAIN}

openshift_master_cluster_method=native
openshift_master_cluster_hostname=${RESOURCEGROUP}.${FULLDOMAIN}
openshift_master_cluster_public_hostname=${RESOURCEGROUP}.${FULLDOMAIN}

openshift_metrics_install_metrics=false
openshift_metrics_cassandra_storage_type=pv
openshift_metrics_cassandra_pvc_size="${METRICS_CASSANDRASIZE}G"
openshift_metrics_cassandra_replicas="${METRICS_INSTANCES}"
openshift_metrics_hawkular_nodeselector={"role":"infra"}
openshift_metrics_cassandra_nodeselector={"role":"infra"}
openshift_metrics_heapster_nodeselector={"role":"infra"}
```

```
openshift_logging_install_logging=false
openshift_logging_es_pv_selector={"usage":"elasticsearch"}
openshift_logging_es_pvc_dynamic="false"
openshift_logging_es_pvc_size="${LOGGING_ES_SIZE}G"
openshift_logging_es_cluster_size=${LOGGING_ES_INSTANCES}
openshift_logging_fluentd_nodeselector={"logging":"true"}
openshift_logging_es_nodeselector={"role":"infra"}
openshift_logging_kibana_nodeselector={"role":"infra"}
openshift_logging_curator_nodeselector={"role":"infra"}

openshift_logging_use_ops=false
openshift_logging_es_ops_pv_selector={"usage":"opselasticsearch"}
openshift_logging_es_ops_pvc_dynamic="false"
openshift_logging_es_ops_pvc_size="${OPSLOGGING_ES_SIZE}G"
openshift_logging_es_ops_cluster_size=${OPSLOGGING_ES_INSTANCES}
openshift_logging_es_ops_nodeselector={"role":"infra"}
openshift_logging_kibana_ops_nodeselector={"role":"infra"}
openshift_logging_curator_ops_nodeselector={"role":"infra"}

[masters]
master1 openshift_hostname=master1 openshift_node_labels="{'role':
'master'}"
master2 openshift_hostname=master2 openshift_node_labels="{'role':
'master'}"
master3 openshift_hostname=master3 openshift_node_labels="{'role':
'master'}"

[etcd]
master1
master2
master3

[new_nodes]
[new_masters]

[nodes]
master1 openshift_hostname=master1 openshift_node_labels="
{'role':'master','zone':'default','logging':'true'}"
openshift_schedulable=false
master2 openshift_hostname=master2 openshift_node_labels="
{'role':'master','zone':'default','logging':'true'}"
openshift_schedulable=false
master3 openshift_hostname=master3 openshift_node_labels="
{'role':'master','zone':'default','logging':'true'}"
openshift_schedulable=false
infranode1 openshift_hostname=infranode1 openshift_node_labels="{'role':
'infra', 'zone': 'default','logging':'true'}"
infranode2 openshift_hostname=infranode2 openshift_node_labels="{'role':
'infra', 'zone': 'default','logging':'true'}"
infranode3 openshift_hostname=infranode3 openshift_node_labels="{'role':
'infra', 'zone': 'default','logging':'true'}"
node[01:${NODECOUNT}] openshift_hostname=node[01:${NODECOUNT}]
openshift_node_labels="{'role':'app','zone':'default','logging':'true'}"
```

Note

Those are the values chosen for the **OpenShift Container Platform** installation in this reference architecture, for more information about those parameters and their values, see the OpenShift documentation

For the **OpenShift Container Platform** installation, the **`ARM`** template collects the needed parameters, creates the virtual machines, and passes the parameters to the virtual machines, where a node type specific script in bash will take the parameters and generate the needed playbooks and automation. During this process each VM is assigned an ansible tag, that allows the playbooks to address the different node types.

Note

For more information about the automation procedures on **Microsoft Azure**, see Azure Linux Automation blog post and for more information about the Ansible inventory, see Ansible Host Inventory

2.16. NODES

Nodes are **Microsoft Azure** instances that serve a specific purpose for **OpenShift Container Platform**. **OpenShift Container Platform** masters are also configured as nodes as they are part of the **`SDN`**. Nodes deployed on **Microsoft Azure** can be vertically scaled before or after the **OpenShift Container Platform** installation using the **Microsoft Azure** dashboard. There are three types of nodes as described below.

2.16.1. Master nodes

The master nodes contain the master components, including the API server, web console, controller manager server and **`etcd`**. The master maintains the cluster configuration within **`etcd`**, manages nodes in its **OpenShift Container Platform** cluster, assigns pods to nodes and synchronizes pod information with service configuration. The master is used to define routes, services, and volume claims for pods deployed within the **OpenShift Container Platform** environment. The users interact with the **OpenShift Container Platform** environment via the masters and using the API, web console or the **`oc`** command line interface.

Note

Even if master nodes would be able to run pods, they are configured as **`unschedulable`** to ensure that the masters are not burdened with running pods

2.16.2. Application nodes

The application nodes are the instances where non **OpenShift Container Platform** infrastructure based containers run. Depending on the application, **Microsoft Azure** specific storage can be applied such as an **`Azure VHD`** which can be assigned using a **`Persistent Volume Claim`** for application data that needs to persist between container restarts. A configuration parameter is set on the masters which ensures that **OpenShift Container Platform** user containers will be placed on the application nodes by default.

2.16.3. Infrastructure nodes

The infrastructure nodes are just regular nodes but with different labels so they are only used to host the optional infrastructure components for **OpenShift Container Platform** like the routers, registries, metrics or the aggregated logging to isolate those components of the regular applications. The storage for the registry deployed on the infrastructure nodes uses **`Azure Blob Storage`** which allows for multiple pods to use the same storage at the same time (**`ReadWriteMany`** or **`RWX`**). Since the registry does the lookup of the metadata, and then the download is handed off to azure to handle, this creates better scaling than other options.

Note

This reference architecture is emerging and components like the aggregated logging or metrics will be described in future revisions.

2.16.4. Node labels

All **OpenShift Container Platform** nodes are assigned a **`role`** label. This allows the scheduler decide of certain pods to be deployed on specific nodes.

Label	Nodes	Pods
`role=master`	Master nodes	None
`role=app`	Application nodes	Application pods
`role=infra`	Infrastructure nodes	Infrastructure pods

Note

The configuration parameter **`'defaultNodeSelector: "role=app"`** in **`/etc/origin/master/master-config.yaml`** file ensures all projects automatically are deployed on application nodes.

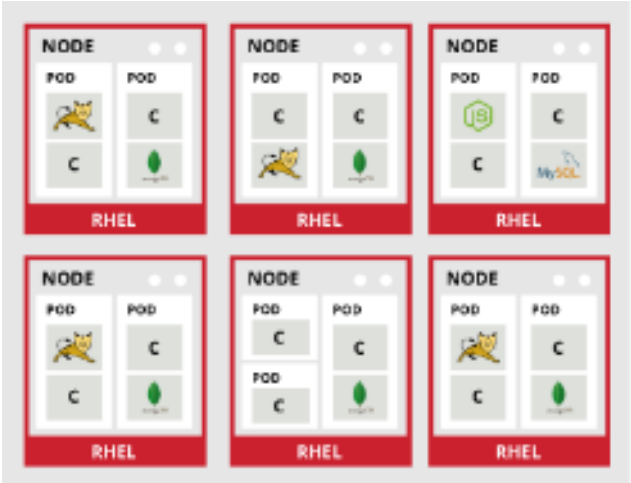
2.17. OPENSHIFT PODS

OpenShift Container Platform leverages the **Kubernetes** concept of a pod, which is one or more containers deployed together on one host, and the smallest compute unit that can be defined, deployed, and managed.

A pod could be just a single container that runs a php application connecting to a database outside of the **OpenShift Container Platform** environment, or a pod could be two containers, one of them runs a php application and the other one runs an ephemeral database. Pods have the ability to be scaled at runtime or at the time of launch using the **OpenShift Container Platform** web console, the **OpenShift Container Platform** API or the **oc** CLI tool.

Note

The **OpenShift Container Platform** infrastructure components like the router and registry are deployed as pods in the **OpenShift Container Platform** environment in the installation procedure or after the installation. Even though they are not required for the **OpenShift Container Platform** environment to run, they provide very useful features so this reference architecture will assume they will be deployed.



For more information about the **OpenShift Container Platform** architecture and components, see the OpenShift Architecture

2.18. OPENSHIFT ROUTER

Pods inside of an **OpenShift Container Platform** cluster are only reachable via their IP addresses on the cluster network. To be able to access pods from outside the **OpenShift Container Platform** cluster, **OpenShift Container Platform** provides a few options:

- Router
- Load Balancer Service
- Service ExternalIP
- Service NodePort
- Virtual IPs
- Non-Cloud Edge Router Load Balancer
- Edge Load Balancer

OpenShift Container Platform routers provide external hostname mapping and load balancing to services exposed by users over protocols that pass distinguishing information directly to the router; the hostname must be present in the protocol in order for the router to determine where to send it. Routers currently support the following protocols:

- HTTP
- HTTPS (with SNI)
- WebSockets

- TLS with SNI

There are two different router plug-ins that can be deployed in **OpenShift Container Platform**:

- HAProxy template router
- F5 router

Note

This reference architecture uses HAProxy template routers as the main mechanism to access the pods from outside the **OpenShift Container Platform** cluster. For more information on the different options, see Getting Traffic into the Cluster documentation and Router Overview

The HAProxy template router enable routes created by developers to be used by external clients. To avoid reconfiguration of the **DNS** servers every time a route is created, the suggested method is to define a wildcard **DNS** entry that will redirect every hostname to the router.

Note

For high availability purposes, this reference architecture deploys two router pods and creates an **Azure Load Balancer** which performs a health check and forwards traffic to router pods on port 80 and 443.

Important

Due to the current **Microsoft Azure** limitations on subdomains, the default wildcard entry uses the nip.io service. This can be modified after the installation and it is explained in the Microsoft Azure DNS section

2.19. REGISTRY

OpenShift Container Platform can build container images from source code, deploy them, and manage their lifecycle. To enable this it is required to deploy the Integrated OpenShift Container Platform Registry

The registry stores container images and metadata. For production environment, persistent storage is recommended to be used by the registry, otherwise any images built or pushed into the registry would disappear if the pod were to restart.

The registry is scaled to 3 pods/instances to allow for HA, and load balancing. The default load balancing settings use the source ip address to enforce session stickiness. Failure of a pod may result in a pull or push operation to fail, but the operation may be restarted. The failed registry pod will be automatically restarted.

Using the installation methods described in this document the registry is deployed using **Azure Blob Storage**, an **Microsoft Azure** service that provides object storage. In order to use **Azure Blob Storage** the registry configuration has been extended. The procedure used is detailed in the OpenShift and Docker documentation, and it is to modify the registry **deploymentconfig** to add the **Azure Blob Storage** service details as:

```
$ oc env dc docker-registry -e REGISTRY_STORAGE=azure -e
REGISTRY_STORAGE_AZURE_ACCOUNTNAME=<azure_storage_account_name> -e
REGISTRY_STORAGE_AZURE_ACCOUNTKEY=<azure_storage_account_key> -e
REGISTRY_STORAGE_AZURE_CONTAINER=registry
```

This will be done automatically as part of the installation by the scripts provided in the git repository.

Note

For more information about the **Microsoft Azure** Blob Storage, see: https://azure.microsoft.com/en-us/services/storage/blobs/

2.20. AUTHENTICATION

There are several options when it comes to authentication of users in **OpenShift Container Platform**. **OpenShift Container Platform** can leverage an existing identity provider within an organization such as **LDAP** or can use external identity providers like **GitHub**, **Google**, and **GitLab**. The configuration of identification providers occurs on the **OpenShift Container Platform** master instances and multiple identity providers can be specified. The reference architecture document uses **`htpasswd`** as the authentication provider but any of the other mechanisms would be an acceptable choice. Roles can be customized and added to user accounts or groups to allow for extra privileges such as the ability to list nodes or assign persistent storage volumes to a project.

Note

For more information on **`htpasswd`** and other authentication methods see the Configuring authentication documentation.

Note

For best practice on authentication, consult the Red Hat Single Sign-On (SSO) documentation. **Red Hat Single Sign-On** (SSO) allows a fully federated central authentication service that can be used by both developers and end-users across multiple identity providers, using a simple user interface.

2.21. MICROSOFT AZURE STORAGE

For the use cases considered in this reference architecture, including **OpenShift Container Platform** applications that connect to containerized databases or need some basic persistent storage, we need to consider multiple storage solutions in the cloud and different architectural approaches. **Microsoft Azure** offers a number of storage choices that offer high durability with three simultaneous replicas, including Standard and Premium storage.

Furthermore, a use case requirement is to implement "shared storage" where the volume should allow simultaneous read and write operations. Upon reviewing multiple options supported by **OpenShift Container Platform** and the underlying Red Hat Enterprise Linux infrastructure, a choice was made to use the **`Azure VHD`** based storage server to give the **Microsoft Azure OpenShift Container Platform** nodes the best match of performance and flexibility, and use **`Azure Blob Storage`** for the registry storage.

Note

The reference architecture will be updated with further storage choices (such as Managed Disks) as they are evaluated.

2.21.1. Microsoft Azure VHD

The **Microsoft Azure** cloud provider plugin for **OpenShift Container Platform** will dynamically allocate storage in the pre-created storage accounts based on requests for PV. In order to make this work, the **OpenShift Container Platform** environment should be configured properly, including creating a **`/etc/azure/azure.conf`** file with the **Microsoft Azure** account data, modifying the **`/etc/origin/master/master-config.yaml`** and **`/etc/origin/node/node-config.yaml`** files and restart the **`atomic-openshift-node`** and **`atomic-openshift-master`** services. The scripts provided with this reference architecture do this procedure automatically.

Note

For more information about the **Microsoft Azure** configuration for **OpenShift Container Platform** storage, see Configuring for Azure documentation

This reference architecture creates an install a default storage class, so any persistent volume claim will result in a new **`VHD`** being created in the selected storage account. The **`VHD`** will be created, mounted, formatted, and allocated to the container making the request.

Note

For more information about **`Azure VHD`** see About disks and VHDs for Azure Linux VMs documentation and see Dynamic Provisioning and Creating Storage Classes for more information about storage classes

2.22. RED HAT OPENSHIFT CONTAINER PLATFORM METRICS

Red Hat OpenShift Container Platform environments can be enriched by deploying an optional component named **Red Hat OpenShift Container Platform metrics** that collect metrics exposed by the **`kubelet`** from pods running in the environment and provides the ability to view CPU, memory, and network-based metrics and display the values in the user interface.

Note

Red Hat OpenShift Container Platform metrics is a required component for the **`horizontal pod autoscaling`** feature that allows the user to configure autoscaling pods on a certain capacity thresholds. For more information about pod autoscaling, see Pod Autoscaling.

Red Hat OpenShift Container Platform metrics it is composed by a few pods running on the **Red Hat OpenShift Container Platform** environment:

- Heapster: Heapster scrapes the metrics for CPU, memory and network usage on every pod, then exports them into Hawkular Metrics.
- Hawkular Metrics: A metrics engine which stores the data persistently in a Cassandra database.
- Cassandra: Database where the metrics data is stored.

It is important to understand capacity planning when deploying metrics into an OpenShift environment regarding that one set of metrics pods (Cassandra/Hawkular/Heapster) is able to monitor at least 25,000 pods.

Red Hat OpenShift Container Platform metrics components can be customized for longer data persistence, pods limits, replicas of individual components, custom certificates, etc.

Note

For more information about different customization parameters, see Enabling Cluster Metrics documentation.

Within this reference environment, metrics are deployed optionally on the infrastructure nodes depending on the "metrics" parameter of the **`ARM template`**. When "true" is selected, it deploys one set of metric pods (Cassandra/Hawkular/Heapster) on the infrastructure nodes (to avoid using resources on the application nodes) and uses persistent storage to allow for metrics data to be preserved for 7 days.

2.22.1. Horizontal pod Autoscaler

If **Red Hat OpenShift Container Platform** metrics has been deployed the **horizontal pod autoscaler** feature can be used. A horizontal pod autoscaler, defined by a **`HorizontalPodAutoscaler`** object, specifies how the system should automatically increase or decrease the scale of a replication controller or deployment configuration, based on metrics collected from the pods that belong to that replication controller or deployment configuration.

Note

For more information about the pod autoscaling feature, see the official documentation

2.23. RED HAT OPENSHIFT CONTAINER PLATFORM AGGREGATED LOGGING

One of the **Red Hat OpenShift Container Platform** optional components named **Red Hat OpenShift Container Platform aggregated logging** collects and aggregates logs for a range of **Red Hat OpenShift Container Platform** services enabling **Red Hat OpenShift Container Platform** users view the logs of projects which they have view access using a web interface.

Red Hat OpenShift Container Platform aggregated logging component it is a modified version of the **`ELK`** stack composed by a few pods running on the **Red Hat OpenShift Container Platform** environment:

- Elasticsearch: An object store where all logs are stored.
- Fluentd: Gathers logs from nodes and feeds them to Elasticsearch.

- Kibana: A web UI for Elasticsearch.
- Curator: Elasticsearch maintenance operations performed automatically on a per-project basis.

Once deployed in a cluster, the stack aggregates logs from all nodes and projects into Elasticsearch, and provides a Kibana UI to view any logs. Cluster administrators can view all logs, but application developers can only view logs for projects they have permission to view. To avoid users to see logs from pods in other projects, the Search guard plugin for Elasticsearch is used.

A separate Elasticsearch cluster, a separate Kibana, and a separate Curator components can be deployed to form the **OPS cluster** where logs for the **`default`**, **`openshift`**, and **`openshift-infra`** projects as well as **`/var/log/messages`** on nodes are automatically aggregated and grouped into the **`.operations`** item in the Kibana interface.

Red Hat OpenShift Container Platform aggregated logging components can be customized for longer data persistence, pods limits, replicas of individual components, custom certificates, etc.

Note

For more information about different customization parameters, see Aggregating Container Logs documentation.

Within this reference environment, aggregated logging components are deployed optionally on the infrastructure nodes depending on the "logging" parameter of the **`ARM template`**. When "true" is selected, it deploys on the infrastructure nodes (to avoid using resources on the application nodes) the following elements:

- 3 Elasticsearch replicas for HA using dedicated persistent volumes each one
- Fluentd as a **`daemonset`** on all the nodes that includes the "logging=true" selector (all nodes and masters by default)
- Kibana
- Curator

Also, there is an "opslogging" parameter that can optionally deploy the same architecture but for operational logs:

- 3 Elasticsearch replicas for HA using dedicated persistent volumes each one
- Kibana
- Curator

Note

Fluentd pods are configured automatically to split the logs for the two Elasticsearch clusters in case the ops cluster is deployed.

Table 2.9. Red Hat OpenShift Container Platform aggregated logging components

Parameter	Deploy by default	Fluentd	Elasticsearch	Kibana	Curator
logging	true	Daemonset ("logging=true " selector)	3 replicas	1	1
opslogging	false	Shared	3 replicas	1	1

CHAPTER 3. PROVISIONING THE INFRASTRUCTURE

This chapter focuses on Phase 1 of the process. The prerequisites defined below are required for a successful deployment of infrastructure and the installation of **OpenShift Container Platform** on **Microsoft Azure**.

3.1. PROVISIONING PREREQUISITES

The script and playbooks provided within the git repository deploys infrastructure, installs and configures **OpenShift Container Platform**, and performs post installation tasks such as scaling the router and registry. The playbooks create specific roles, policies, and users required for cloud provider configuration in **OpenShift Container Platform**, as well as storage accounts on **Microsoft Azure**. In order to deploy **OpenShift Container Platform** on **Microsoft Azure**, an **ARM** template must be created. The **ARM** template takes the basic information, admin user name, password, **Red Hat Subscription Manager** username and password, **SSH** public and private keys, as well as offering the ability to size the virtual machines.

3.1.1. Authentication Prerequisites

The template creates an **OpenShift Container Platform** user using the supplied values **adminUsername** and **adminPassword** in the **ARM** template parameters when creating the infrastructure and it will be granted the **cluster-admin** role for the **OpenShift Container Platform** environment.

3.1.2. SSH Prerequisites

3.1.2.1. SSH Configuration

The **SSH** service will be configured during the deployment to allow connections using a public key. To make this work, the same public and private key that were created before will be used for the admin user in the instances. Before beginning the deployment of the **Microsoft Azure** infrastructure and the deployment of **OpenShift Container Platform**, the **SSHprivate key** must be converted into a base64 string. Otherwise, the **Microsoft Azure** infrastructure cannot be configured manually or automatically.

Note

The reason to convert the private key to a base64 string is to avoid possible issues with multiple lines and carriage return characters

This task is performed locally, on the machine that will run the **ARM** template. In this example, the private key is converted to base64 encoding, and placed on the clipboard.

- For RHEL/CentOS:

```
$ cat ~/.ssh/id_rsa | base64 | tr -d '\n' | xclip -selection clipboard
```

- For OSX:

```
$ cat ~/.ssh/id_rsa | base64 | tr -d '\n' | pbcopy
```

- For Windows use a Base 64 Encoder app or website to perform the conversion

3.1.3. Red Hat Subscription Prerequisites

A **Red Hat account** is required to use this template. This account must have appropriate subscriptions available in the account in order to use the template. Either a username and password may be supplied, or an **`organization ID`** and **`activation key`**.

To be able to access the RPM packages the **`Pool ID`** for the subscriptions is required. The next section will show how to locate the **`Pool ID`**.

3.1.3.1. Red Hat Subscription Pool ID

In order to assign a subscription, a **`Pool ID`** is required. Using a system that currently has a valid subscription to Red Hat products, by using **`subscription-manager`** the **`Pool ID`** can be acquired. Perform the following command, and take note of the **`Pool ID`**.

```
# subscription-manager list --available

+-------------------------------------------+
    Available Subscriptions
+-------------------------------------------+
Subscription Name:   Red Hat OpenShift Container Platform, Premium (1-2
Sockets)
Provides:            Red Hat OpenShift Enterprise Application Node
                     Red Hat OpenShift Container Platform
                     Red Hat OpenShift Enterprise Client Tools
                     Red Hat Enterprise Linux Server
                     Red Hat Enterprise Linux Atomic Host
                     Red Hat OpenShift Enterprise Infrastructure
                     Red Hat CloudForms
SKU:                 MCT2862
Contract:            38154141
Pool ID:             1b152951269116311123bc522341e1
Provides Management: No
Available:           64
Suggested:           1
Service Level:       Premium
Service Type:        L1-L3
Subscription Type:   Stackable
Ends:                25/08/2017
System Type:         Virtual
```

Note

The **`Pool ID`** is also available in the **Subscriptions** section of the **Red Hat Customer Portal**, by selecting the appropriate subscription that will open a detailed view of the subscription, including the **`Pool ID`**

3.1.4. Organization ID and Activation Key

Instead using username/password combination, an **`activation key`** and **`organization ID`** can

be used. The **activation key** is a simple string, chosen by the end-user that contains the subscriptions the user attaches to it to avoid using passwords in the registration procedure. The **organization ID** is the Red Hat **ID** for the customer organizations and it is required when using activation keys to identify the activation keys within the organization.

Perform the following steps to obtain the **organization ID** and activation key.

3.1.4.1. Generating Activation Key

1. Go to the **Red Hat Customer Portal** located in http://access.redhat.com and login with your Red Hat username and password. Select **[Subscriptions]** section in the top left corner then the tab "Activation keys"
2. Create your key selecting the "New Activation Key" button

Once in there give your key a name, set service level to standard or premium, and add the subscriptions containing **OpenShift Container Platform**.

3.1.4.2. Getting the Organization ID

In order to find the **organization ID**, a **RHEL** server is required. Perform the following on a existing **RHEL** server.

```
$ subscription-manager identity
system identity: 8c85ab6d-495c-451e-b38e-5f09bc3342a0
name: bastion
org name: 6616399
org ID: 6616399
```

Find the value labeled **org ID:** and save this somewhere for use during the deployment.

3.1.4.3. Using the Organization ID and Activation Key

When running the **ARM** template, use the **organization ID** as the RHN User Name, and use the **activation key** as the RHN Password.

3.1.5. Azure Active Directory Credentials

It is required to create a **Microsoft Azure** service principal in order to be able to deploy the infrastructure using the **ARM** template. The service principal object defines the policy and permissions for an application's use in a specific tenant, providing the basis for a security principal to represent the application at run-time.

Note

For more information about the service principal objects in **Azure Active Directory** see Use Azure CLI to create a service principal to access resources

In order to create the **Azure Active Directory** (**AAD**) client id, and password, the **Node.js Azure CLI** package is required.

Follow this instructions to create it in a RHEL/CentOS/Fedora system:

1. Install **Node.js**

```
$ sudo yum -y install npm
```

2. Install the **`Azure CLI` Node.js** package:

```
$ sudo npm install -g azure-cli
```

3. Login to **Microsoft Azure**:

```
$ azure login
```

4. Create a service principal:

```
$ azure ad sp create -n <service_principal_name> -p <password>
```

The following is an example output:

```
$ azure ad sp create -n openshiftcloudprovider -p Pass@word1
info:    Executing command ad sp create
+ Creating application openshift demo cloud provider
+ Creating service principal for application 198c4803-1236-4c3f-
ad90-46e5f3b4cd2a
data:    Object Id:               00419334-174b-41e8-9b83-
9b5011d8d352
data:    Display Name:            openshiftcloudprovider
data:    Service Principal Names:
data:                             198c4803-1236-4c3f-ad90-
46e5f3b4cd2a
data:                             http://myhomepage
info:    ad sp create command OK
```

Save the **`Object Id`** and the **`Service Principal Names GUID`** values from the command output.

- The **`Object Id`** will be used to create the role assignment.
- The **`Service Principal Names GUID`** will be used as the **`aadClientId`** parameter value (Application ID/Client ID) in the template.
- The password entered as part of the CLI command will be the **`aadClientSecret`** paramter value in the template.

5. Show the **Microsoft Azure** account data:

```
$ azure account show
```

The following is an example output:

```
$ azure account show
info:    Executing command account show
data:    Name                        : Microsoft Azure
```

```
Sponsorship
data:    ID                                : 2581564b-56b4-4512-a140-
012d49dfc02c
data:    State                             : Enabled
data:    Tenant ID                         : 77ece336-c110-470d-a446-
757a69cb9485
data:    Is Default                        : true
data:    Environment                       : AzureCloud
data:    Has Certificate                   : Yes
data:    Has Access Token                  : Yes
data:    User name                         : ssysone@something.com
data:
info:    account show command OK
```

Save the command output **`ID`** value that will be used for the provisioning.

6. Grant the service principal the access level of **`contributor`** to allow **OpenShift Container Platform** to create/delete resources using the **`Object ID`** and **`ID`** parameters from the previous steps

```
$ azure role assignment create --objectId <objectID> -o
contributor -c /subscriptions/<id>/
```

The following is an example output:

```
# azure role assignment create --objectId 00419334-174b-41e8-
9b83-9b5011d8d352 -o contributor -c /subscriptions/2581564b-56b4-
4512-a140-012d49dfc02c/
info:    Executing command role assignment create
+ Finding role with specified name
/data:    RoleAssignmentId     : /subscriptions/2586c64b-38b4-
4527-a140-
012d49dfc02c/providers/Microsoft.Authorization/roleAssignments/49
0c9dd5-0bfa-4b4c-bbc0-aa9af130dd06
data:    RoleDefinitionName   : Contributor
data:    RoleDefinitionId     : b24988ac-6180-42a0-ab88-
20f7382dd24c
data:    Scope                : /subscriptions/2586c64b-38b4-
4527-a140-012d49dfc02c
data:    Display Name         : openshiftcloudprovider
data:    SignInName           : undefined
data:    ObjectId             : 00419334-174b-41e8-9b83-
9b5011d8d352
data:    ObjectType           : ServicePrincipal
data:
+
info:    role assignment create command OK
```

3.2. INTRODUCTION TO THE MICROSOFT AZURE TEMPLATE

`Azure Resource Manager` templates consist of json files that describes the objects that will be deployed in **Microsoft Azure**. The main template file for this reference architecture is located in the **`reference-architecture/azure-ansible/azuredeploy.json`** file in the git repository. This file is the main **`ARM`** template that launches all the other templates under **`azure-ansible`**.

There are four types of virtual machines created by the template (bastion, master node, infrastructure node and application node) and for each of these types there is a additional json file that defines each VM type.

Virtual Machine type	Template file
Bastion	**reference-architecture/azure-ansible/bastion.json**
Master	**reference-architecture/azure-ansible/master.json**
Infrastructure node	**reference-architecture/azure-ansible/infranode.json**
Application node	**reference-architecture/azure-ansible/node.json**

The **ARM** template for each type, automatically starts a bash shell script that does part of the initial setup. The main shell script is the **reference-architecture/azure-ansible/bastion.sh** that handles the generation of the ansible host inventory, as well as the setup and running of ansible across all the hosts. The bastion host also provides isolation of all the hosts in the resource group from the public internet for the purpose of **SSH** access.

3.3. ALTERNATIVE SINGLE VM MICROSOFT AZURE TEMPLATE

In addition to the production template of **azuredeploy.json**, a single virtual machine version is also available. This template is located at: **reference-architecture/azure-ansible/allinone.json** This provides for early prototypes and tests of applications in a **Red Hat OpenShift Container Platform** environment. Note that the single VM does not support the high-availability and load-balancing features of the full **azuredeploy.json** template. The single virtual machine template only allows you to choose the vm size, and uses a single public ip for console and applications. The **Number of Nodes** and **Wildcard Zone** are removed.

3.4. PARAMETERS REQUIRED

In order to provision the **OpenShift Container Platform** environment using the **ARM** template, the following information is required:

- A **Microsoft Azure** subscription, with appropriate core and VM quota limits.
- **Resource Group** - Used as the name of the **OpenShift Container Platform** Cluster - All the assets of a single cluster use the **Azure Resource Group** to organize and group the assets. This name needs to be unique for each cluster per **Azure Region** (Location). As some resources will be created using the resource group name, it must be 3-18 characters in length and use numbers and lower-case letters only.

- **Admin username and password** - This will be the admin user, used for multiple purposes:
    - As the **SSH** user to be able to connect to the bastion host, and administer the cluster.
    - As an **OpenShift Container Platform** administrative user, able to create and control **OpenShift Container Platform** from the command line, or the user interface.
- **SSH Key Data** - This is the **public key** (**`~/.ssh/id_rsa.pub`**), generated for the user that will be used to **SSH** access to all the VMs. During the creation and installation of **OpenShift Container Platform** virtual machines, the key will automatically be added to each host. This assures proper security and access. This key must be backed up, as its the only principal way to access the cluster for administration.
- **SSH Private Data** - This is the **private key** **`~/.ssh/id_rsa`** file contents that has been base64 encoded. During the creation and installation of **OpenShift Container Platform** virtual machines, the key will automatically be added to each host. This data should be backed up.
- **Wildcard Zone** - Subdomain for applications in the **OpenShift Container Platform** cluster (required by the load balancer, but **nip.io** will be used). It is just the subdomain, not the full FQDN. Example wildcardzone parameter will be "refarchapps" and the FQDN will be created as **`refarchapps.<region>.cloudapp.azure.com`**
- **Number of Nodes** - The template supports the creation of 3 to 30 nodes during greenfield creation of a cluster. Note that the quota of the **Microsoft Azure** account must support the number chosen.
- **Image** - The template supports **`RHEL`** (Red Hat Enterprise Linux) 7.3 or later. The image will be upgraded during the installation process to the latest release.
- **Master VM Size** (default: **`Standard_DS4_v2`**) - The default value gives 8 CPU Cores, 28 Gigabytes of memory, with 56 GB of premium storage local disk. This is used for **OpenShift Container Platform** master nodes, as well as the bastion host.
- **Infranode VM Size** (default: **`Standard_DS4_v2`**) - The default value gives 8 CPU Cores, 28 Gigabytes of memory, with 56 GB of premium storage local disk. Infrastructure nodes run the **OpenShift Container Platform** routers and the **OpenShift Container Platform** registry pods. As the infrastructure nodes provide the ingress for all applications, its recommended that *DS2* be the smallest node used for the infrastructure nodes.
- **Node VM Size** (default: **`Standard_DS4_v2`**) - The default value gives 8 CPU Cores, 28 Gigabytes of memory, with 56 GB of premium storage local disk. Application nodes is where the application containers run.
- **RHN Username** - This should be the username used for the **Red Hat Subscription Account** that has **OpenShift Container Platform** entitlements, or the **`Organization ID`** if using activation keys.
- **RHN Password** - This should be the password for the Red Hat Subscription Account, or the **`activation key`** if using activation keys.
- **Subscription Pool ID** - This is a number sequence that uniquely identifies the subscriptions that are to be used for the **OpenShift Container Platform** installation.
- **AAD Client Id** - This gives **OpenShift Container Platform** the **`Active Directory ID`**, needed to be able to create, move and delete persistent volumes.
- **AAD Client Secret** - The **`Active Directory Password`** to match the AAD Client ID. Required to create persistent volumes.

- **OpenShiftSDN** - The SDN plugin to be used in the environment (ovs-multitenant by default)
- **Metrics** - Deploy **Red Hat OpenShift Container Platform** metrics components (true by default)
- **Logging** - Deploy **Red Hat OpenShift Container Platform** aggregated logging components (true by default)
- **OPS Logging** - Deploy **Red Hat OpenShift Container Platform** ops aggregated logging components (false by default)

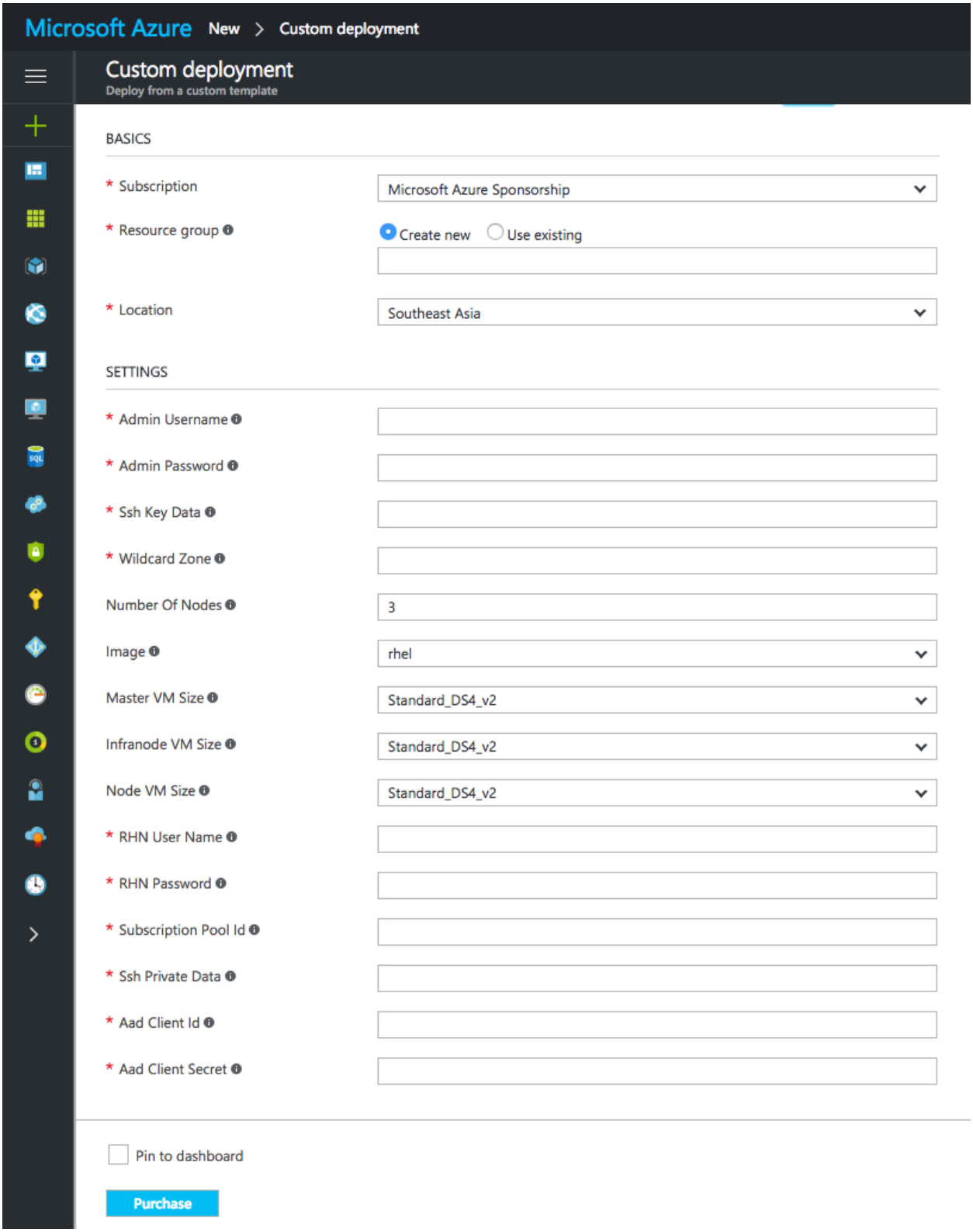
3.5. PROVISION OPENSHIFT CONTAINER PLATFORM ENVIRONMENT

There are two ways to provision the **OpenShift Container Platform** environment. Using a Web Interface, by filling out a form generated by the template for the needed parameters. And the alternate way, is using a ansible playbook to deploy the cluster. The ansible method is ideal when you wish to deploy clusters in a repeatable way, or when you wish to have more than one cluster.

3.5.1. Provisioning ARM Template by using the Web Interface

With the above information ready, go to https://github.com/openshift/openshift-ansible-contrib/tree/master/reference-architecture/azure-ansible and click the **[Deploy To Azure]** button near the bottom of the page. This will then show the form, to allow the deployment to be started.

Figure 3.1. ARM Template



3.5.2. Provisioning ARM Template by using Ansible

The ARM templates may be deployed via Ansible playbook when you wish to have a repeatable method of creating a cluster, or you wish to create multiple clusters.

In the reference scripts, a additional directory is provided, **`ansibledeployocp`**. This provides example playbooks to directly create clusters using Ansible in a Linux environment.

First, install git and ansible.

```
$ sudo yum -y install ansible git
```

Then clone the openshift-ansible-contrib repository to a RHEL based host.

```
$ git clone https://github.com/openshift/openshift-ansible-contrib
$ cd openshift-ansible-contrib/reference-architecture/azure-
ansible/ansibledeployocp/
```

Next, install the dependencies using the prepare playbook:

```
$ ansible-playbook playbooks/prepare.yml
```

Microsoft Azure credentials needs to be stored in a file at **`~/.azure/credentials`** with the following format (do not use quotes or double quotes):

```
[default]
subscription_id=00000000-0000-0000-0000-000000000000
tenant=11111111-1111-1111-1111-111111111111
client_id=33333333-3333-3333-3333-333333333
secret=ServicePrincipalPassword
```

Where **`subscription_id`** and **`tenant`** parameters can be obtained from the **Microsoft Azure** cli:

```
$ sudo yum install -y nodejs
$ sudo npm install -g azure-cli
$ azure login
$ azure account show
info:    Executing command account show
data:    Name                        : Acme Inc.
data:    ID                          : 00000000-0000-0000-0000-
000000000000
data:    State                       : Enabled
data:    Tenant ID                   : 11111111-1111-1111-1111-
111111111111
data:    Is Default                  : true
data:    Environment                 : AzureCloud
data:    Has Certificate             : Yes
data:    Has Access Token            : Yes
data:    User name                   : youremail@yourcompany.com
data:
info:    account show command OK
```

The **`client_id`** is the "Service Principal Name" parameter when you create the serviceprincipal:

```
$ azure ad sp create -n azureansible -p ServicePrincipalPassword

info:    Executing command ad sp create
+ Creating application ansiblelab
+ Creating service principal for application 33333333-3333-3333-3333-
333333333
data:    Object Id:               44444444-4444-4444-4444-444444444444
data:    Display Name:            azureansible
data:    Service Principal Names:
data:                             33333333-3333-3333-3333-333333333
data:                             http://azureansible
info:    ad sp create command OK
```

The **secret** is the serviceprincipal password.

Ansible Parameters required

The ansible playbook needs some parameters to be specified. There is a **vars.yaml** example file included in this repository that should be customized with your environment data.

PARAMETERS

```
$ cp vars.yaml.example vars.yaml
$ vim vars.yaml
```

- **sshkeydata** id_rsa.pub content
- **sshprivatedata** id_rsa content in base64 without \n characters (cat ~/.ssh/id_rsa | base64 | tr -d '\n')
- **adminusername** User that will be created to login via ssh and as **Red Hat OpenShift Container Platform** cluster-admin
- **adminpassword** Password for the user created (in plain text)
- **rhsmusernamepasswordoractivationkey** This should be "usernamepassword" or "activationkey". If "usernamepassword", then the username and password should be specified If "activationkey", then the activation key and organization id should be specified
- **rhnusername** The RHN username where the instances will be registered or "activationkey" if activation key method has been chosen
- **rhnpassword** The RHN password where the instances will be registered in plain text
- **rhnpassword** "organizationid" if activation key method has been chosen else password.
- **subscriptionpoolid** The subscription pool id the instances will use
- **resourcegroupname** The **Microsoft Azure** resource name that will be created
- **aadclientid** Active Directory ID needed to be able to create, move and delete persistent volumes
- **aadclientsecret** The Active Directory Password to match the AAD Client ID
- **wildcardzone** Subdomain for applications in the OpenShift cluster (required by the load balancer, but nip.io will be used). It is just the subdomain, not the full FQDN.

Optional (default values are set in **playbooks/roles/azure-deploy/default/main.yaml**)

- **templatelink** - The ARM template that will be deployed
- **numberofnodes** - From 3 to 30 nodes
- **image** - The operating system image that will be used to create the instances
- **mastervmsize** - Master nodes VM size
- **infranodesize** - Infrastructure nodes VM size
- **nodevmsize** - Application nodes VM size

- **`location`** - westus by default
- **`openshiftsdn`** - "redhat/openshift-ovs-multitenant" by default
- **`metrics`** - Deploy **Red Hat OpenShift Container Platform** metrics components (true by default)
- **`logging`** - Deploy **Red Hat OpenShift Container Platform** aggregated logging components (true by default)
- **`opslogging`** - Deploy **Red Hat OpenShift Container Platform** ops aggregated logging components (false by default)

Running the deploy

```
$ ansible-playbook -e @vars.yaml playbooks/deploy.yaml

PLAY [localhost]

TASK [Destroy Azure Deploy]  changed: [localhost] TASK [Destroy Azure
Deploy]
ok: [localhost]

TASK [Create Azure Deploy]
changed: [localhost]

PLAY RECAP **
localhost                  : ok=3    changed=2    unreachable=0
failed=0
```

3.6. POST DEPLOYMENT

Once the playbooks have successfully completed the next steps will be to perform the steps defined in Chapter 4, *Operational Management*. In the event that OpenShift Container Platform failed to install, follow the steps in Appendix C: Chapter 8, *Installation Failure* to restart the installation of OpenShift Container Platform.

3.7. POST PROVISIONING RESULTS

At this point the infrastructure and Red Hat OpenShift Container Platform have been deployed. Log into the `Azure web console` and check the resources. In the `Azure web console`, check for the following resources:

- **3 master nodes**
- **3 infrastructure nodes**
- **3 or more application nodes**
- **1 unique virtual network**
- **3 public IPs**
- **10 network interfaces**
- **4 network security groups**

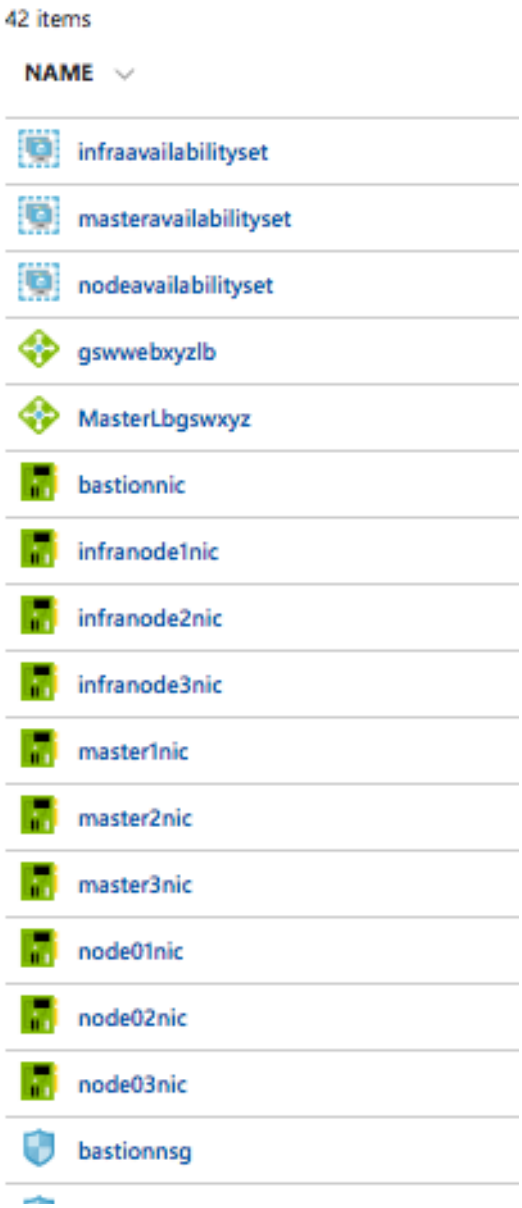
- 5 storage accounts
- 2 load balancer profiles
- 2 load balancer DNS entries
- 3 routers
- 3 registries

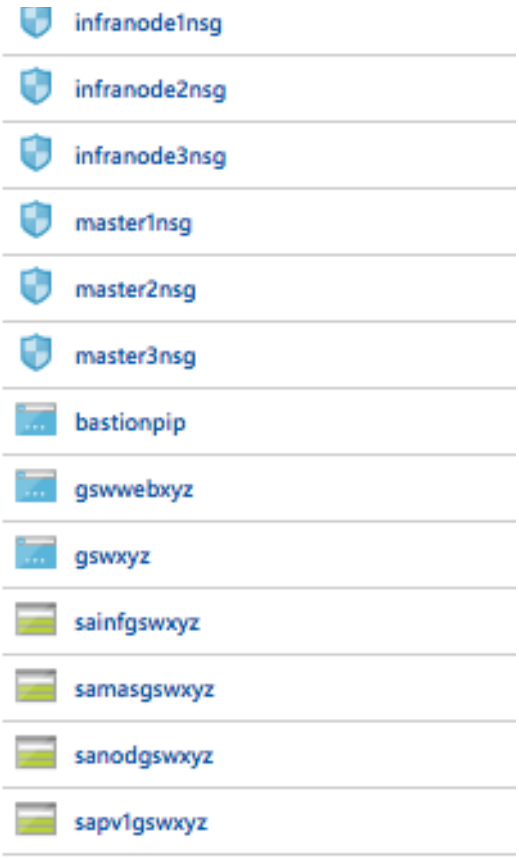
After the `Azure Resource Manager` template is submitted, and the `ARM` deployment succeeds, the ansible install is started automatically.

A wildcard DNS entry must be created if a custom domain is required, by default the nip.io service is be used as explained in the Microsoft Azure DNS section.

Note

When installing using this method the browser certificate must be accepted three times due to the number of masters in the cluster. Failure to accept the certificate can cause disconnect issues and the appearance of network failures.




infranode1nsg
infranode2nsg
infranode3nsg
master1nsg
master2nsg
master3nsg
bastionpip
gswwebxyz
gswxyz
sainfgswxyz
samasgswxyz
sanodgswxyz
sapv1gswxyz

CHAPTER 4. OPERATIONAL MANAGEMENT

With the successful deployment of OpenShift Container Platform, the following section demonstrates how to confirm proper functionality of the Red Hat OpenShift Container Platform.

4.1. SSH CONFIGURATION

Optionally, to be able to connect easily to the VMs, the following `SSH` configuration file can be applied to the workstation that will perform the `SSH` commands:

```
$ cat /home/<user>/.ssh/config

Host bastion
     HostName                <resourcegroup>b.
<region>.cloudapp.azure.com
     User                    <user>
     StrictHostKeyChecking   no
     ProxyCommand            none
     CheckHostIP             no
     ForwardAgent            yes
     IdentityFile            /home/<user>/.ssh/id_rsa

Host master? infranode? node??
     ProxyCommand            ssh <user>@bastion -W %h:%p
     user                    <user>
     IdentityFile            /home/<user>/.ssh/id_rsa
```

To connect to any VM it is only needed the hostname as:

```
$ ssh infranode3
```

4.2. GATHERING HOSTNAMES

With all of the steps that occur during the installation of OpenShift Container Platform, it is possible to lose track of the names of the instances in the recently deployed environment. One option to get these hostnames is to browse to the `Azure Resource Group` dashboard and select Overview. The filter shows all instances relating to the reference architecture deployment.

To help facilitate the Chapter 4, *Operational Management* chapter the following hostnames will be used.

- **master1**
- **master2**
- **master3**
- **infranode1**
- **infranode2**
- **infranode3**

- **node01**
- **node02**
- **node03**

4.3. RUNNING DIAGNOSTICS

To run diagnostics, SSH into the first master node (master1), via the bastion host using the admin user specified in the template:

```
$ ssh <user>@<resourcegroup>b.<region>.cloudapp.azure.com
$ ssh <user>@master1
$ sudo -i
```

Connectivity to the first master node (master1.<region>.cloudapp.azure.com) as the root user should have been established. Run the diagnostics that are included as part of the OpenShift Container Platform installation:

```
# oadm diagnostics
[Note] Determining if client configuration exists for client/cluster
diagnostics
Info:  Successfully read a client config file at '/root/.kube/config'
Info:  Using context for cluster-admin access: 'default/sysdeseng-
westus-cloudapp-azure-com:8443/system:admin'
[Note] Performing systemd discovery

[Note] Running diagnostic: ConfigContexts[default/sysdeseng-westus-
cloudapp-azure-com:8443/system:admin]
       Description: Validate client config context is complete and has
connectivity

Info:  The current client config context is 'default/sysdeseng-westus-
cloudapp-azure-com:8443/system:admin':
       The server URL is
'https://sysdeseng.westus.cloudapp.azure.com:8443'
       The user authentication is 'system:admin/sysdeseng-westus-
cloudapp-azure-com:8443'
       The current project is 'default'
       Successfully requested project list; has access to project(s):
         [default gsw kube-system logging management-infra openshift
openshift-infra]

[Note] Running diagnostic: DiagnosticPod
       Description: Create a pod to run diagnostics from the
application standpoint

       [Note] Running diagnostic: PodCheckDns
              Description: Check that DNS within a pod works as
expected

       [Note] Summary of diagnostics execution (version v3.5.5.5):
       [Note] Warnings seen: 0
       [Note] Errors seen: 0
```

```
[Note] Running diagnostic: NetworkCheck
       Description: Create a pod on all schedulable nodes and run
network diagnostics from the application standpoint

       [Note] Running diagnostic: CheckExternalNetwork
              Description: Check that external network is accessible
within a pod

       [Note] Running diagnostic: CheckNodeNetwork
              Description: Check that pods in the cluster can access
its own node.

       [Note] Running diagnostic: CheckPodNetwork
              Description: Check pod to pod communication in the
cluster. In case of ovs-subnet network plugin, all pods should be able
to communicate with each other and in case of multitenant network
plugin, pods in non-global projects should be isolated and pods in
global projects should be able to access any pod in the cluster and
vice versa.

       [Note] Running diagnostic: CheckServiceNetwork
              Description: Check pod to service communication in the
cluster. In case of ovs-subnet network plugin, all pods should be able
to communicate with all services and in case of multitenant network
plugin, services in non-global projects should be isolated and pods in
global projects should be able to access any service in the cluster.

       [Note] Running diagnostic: CollectNetworkInfo
              Description: Collect network information in the cluster.

       [Note] Summary of diagnostics execution (version v3.5.5.5):
       [Note] Warnings seen: 0


       [Note] Running diagnostic: CheckNodeNetwork
              Description: Check that pods in the cluster can access
its own node.

       [Note] Running diagnostic: CheckPodNetwork
              Description: Check pod to pod communication in the
cluster. In case of ovs-subnet network plugin, all pods should be able
to communicate with each other and in case of multitenant network
plugin, pods in non-global projects should be isolated and pods in
global projects should be able to access any pod in the cluster and
vice versa.

       [Note] Running diagnostic: CheckServiceNetwork
              Description: Check pod to service communication in the
cluster. In case of ovs-subnet network plugin, all pods should be able
to communicate with all services and in case of multitenant network
plugin, services in non-global projects should be isolated and pods in
global projects should be able to access any service in the cluster.

       [Note] Running diagnostic: CollectNetworkInfo
              Description: Collect network information in the cluster.
```

```
       [Note] Summary of diagnostics execution (version v3.5.5.5):
       [Note] Warnings seen: 0


       [Note] Running diagnostic: CheckNodeNetwork
              Description: Check that pods in the cluster can access
its own node.

       [Note] Running diagnostic: CheckPodNetwork
              Description: Check pod to pod communication in the
cluster. In case of ovs-subnet network plugin, all pods should be able
to communicate with each other and in case of multitenant network
plugin, pods in non-global projects should be isolated and pods in
global projects should be able to access any pod in the cluster and
vice versa.

       [Note] Running diagnostic: CheckServiceNetwork
              Description: Check pod to service communication in the
cluster. In case of ovs-subnet network plugin, all pods should be able
to communicate with all services and in case of multitenant network
plugin, services in non-global projects should be isolated and pods in
global projects should be able to access any service in the cluster.

       [Note] Running diagnostic: CollectNetworkInfo
              Description: Collect network information in the cluster.

       [Note] Summary of diagnostics execution (version v3.5.5.5):
       [Note] Warnings seen: 0

[Note] Skipping diagnostic: AggregatedLogging
       Description: Check aggregated logging integration for proper
configuration
       Because: No LoggingPublicURL is defined in the master
configuration

[Note] Running diagnostic: ClusterRegistry
       Description: Check that there is a working Docker registry

[Note] Running diagnostic: ClusterRoleBindings
       Description: Check that the default ClusterRoleBindings are
present and contain the expected subjects

Info:  clusterrolebinding/cluster-readers has more subjects than
expected.

       Use the oadm policy reconcile-cluster-role-bindings command to
update the role binding to remove extra subjects.

Info:  clusterrolebinding/cluster-readers has extra subject
{ServiceAccount management-infra management-admin    }.
Info:  clusterrolebinding/cluster-readers has extra subject
{ServiceAccount default router    }.

Info:  clusterrolebinding/self-provisioners has more subjects than
expected.
```

```
       Use the oadm policy reconcile-cluster-role-bindings command to
update the role binding to remove extra subjects.

Info:  clusterrolebinding/self-provisioners has extra subject
{ServiceAccount management-infra management-admin    }.

[Note] Running diagnostic: ClusterRoles
       Description: Check that the default ClusterRoles are present and
contain the expected permissions

[Note] Running diagnostic: ClusterRouterName
       Description: Check there is a working router

[Note] Running diagnostic: MasterNode
       Description: Check if master is also running node (for Open
vSwitch)

WARN:  [DClu3004 from diagnostic
MasterNode@openshift/origin/pkg/diagnostics/cluster/master_node.go:164]
       Unable to find a node matching the cluster server IP.
       This may indicate the master is not also running a node, and is
unable
       to proxy to pods over the Open vSwitch SDN.

[Note] Skipping diagnostic: MetricsApiProxy
       Description: Check the integrated heapster metrics can be
reached via the API proxy
       Because: The heapster service does not exist in the openshift-
infra project at this time,
       so it is not available for the Horizontal Pod Autoscaler to use
as a source of metrics.

[Note] Running diagnostic: NodeDefinitions
       Description: Check node records on master

WARN:  [DClu0003 from diagnostic
NodeDefinition@openshift/origin/pkg/diagnostics/cluster/node_definition
s.go:112]
       Node master1 is ready but is marked Unschedulable.
       This is usually set manually for administrative reasons.
       An administrator can mark the node schedulable with:
           oadm manage-node master1 --schedulable=true

       While in this state, pods should not be scheduled to deploy on
the node.
       Existing pods will continue to run until completed or evacuated
(see
       other options for 'oadm manage-node').

WARN:  [DClu0003 from diagnostic
NodeDefinition@openshift/origin/pkg/diagnostics/cluster/node_definition
s.go:112]
       Node master2 is ready but is marked Unschedulable.
       This is usually set manually for administrative reasons.
       An administrator can mark the node schedulable with:
           oadm manage-node master2 --schedulable=true
```

```
       While in this state, pods should not be scheduled to deploy on
the node.
       Existing pods will continue to run until completed or evacuated
(see
       other options for 'oadm manage-node').

WARN:  [DClu0003 from diagnostic
NodeDefinition@openshift/origin/pkg/diagnostics/cluster/node_definition
s.go:112]
       Node master3 is ready but is marked Unschedulable.
       This is usually set manually for administrative reasons.
       An administrator can mark the node schedulable with:
           oadm manage-node master3 --schedulable=true

       While in this state, pods should not be scheduled to deploy on
the node.
       Existing pods will continue to run until completed or evacuated
(see
       other options for 'oadm manage-node').

[Note] Running diagnostic: ServiceExternalIPs
       Description: Check for existing services with ExternalIPs that
are disallowed by master config

[Note] Running diagnostic: AnalyzeLogs
       Description: Check for recent problems in systemd service logs

Info:  Checking journalctl logs for 'atomic-openshift-node' service
Info:  Checking journalctl logs for 'docker' service

[Note] Running diagnostic: MasterConfigCheck
       Description: Check the master config file

WARN:  [DH0005 from diagnostic
MasterConfigCheck@openshift/origin/pkg/diagnostics/host/check_master_co
nfig.go:52]
       Validation of master config file '/etc/origin/master/master-
config.yaml' warned:
       assetConfig.loggingPublicURL: Invalid value: "": required to
view aggregated container logs in the console
       assetConfig.metricsPublicURL: Invalid value: "": required to
view cluster metrics in the console
       auditConfig.auditFilePath: Required value: audit can now be
logged to a separate file

[Note] Running diagnostic: NodeConfigCheck
       Description: Check the node config file

Info:  Found a node config file: /etc/origin/node/node-config.yaml

[Note] Running diagnostic: UnitStatus
       Description: Check status for related systemd units
```

```
[Note] Summary of diagnostics execution (version v3.5.5.5):
[Note] Warnings seen: 5
[Note] Errors seen: 0
```

Note

The warnings will not cause issues in the environment

Based on the results of the diagnostics, actions can be taken to alleviate any issues.

4.4. CHECKING THE HEALTH OF ETCD

This section focuses on the `etcd` cluster. It describes the different commands to ensure the cluster is healthy. The internal `DNS` names of the nodes running `etcd` must be used.

`SSH` into the first master node (master1) as before:

```
$ ssh <user>@<resourcegroup>b.<region>.cloudapp.azure.com
$ ssh <user>@master1
$ sudo -i
```

Using the output of the command `hostname` issue the `etcdctl` command to confirm that the cluster is healthy.

```
# etcdctl --endpoints
https://master1:2379,https://master2:2379,https://master3:2379 --ca-
file /etc/etcd/ca.crt --cert-file=/etc/origin/master/master.etcd-
client.crt --key-file=/etc/origin/master/master.etcd-client.key
cluster-health
member 82c895b7b0de4330 is healthy: got healthy result from
https://10.0.0.4:2379
member c8e7ac98bb93fe8c is healthy: got healthy result from
https://10.0.0.5:2379
member f7bbfc4285f239ba is healthy: got healthy result from
https://10.0.0.6:2379
```

Note

In this configuration the `etcd` services are distributed among the OpenShift Container Platform master nodes.

4.5. DEFAULT NODE SELECTOR

As explained in Nodes section, node labels are an important part of the OpenShift Container Platform environment. By default of the reference architecture installation, the default node selector is set to `role=apps` in `/etc/origin/master/master-config.yaml` on all of the master nodes. This configuration parameter is set during the installation of OpenShift on all masters.

SSH into the first master node (master1) to verify the `defaultNodeSelector` is defined.

```
$ ssh <user>@<resourcegroup>b.<region>.cloudapp.azure.com
$ ssh <user>@master1
$ sudo -i
# vi /etc/origin/master/master-config.yaml
... [OUTPUT ABBREVIATED] ...
projectConfig:
  defaultNodeSelector: "role=app"
  projectRequestMessage: ""
  projectRequestTemplate: ""
... [OUTPUT ABBREVIATED] ...
```

Note

If making any changes to the master configuration then the master API service must be restarted or the configuration change will not take place. Any changes and the subsequent restart must be done on all masters.

4.6. MANAGEMENT OF MAXIMUM POD SIZE

Quotas are set on ephemeral volumes within pods to prohibit a pod from becoming too large and impacting the node. There are three places where sizing restrictions should be set. When persistent volume claims are not set a pod has the ability to grow as large as the underlying filesystem will allow. The required modifications are set by automatically.

OpenShift Volume Quota

At launch time a script creates a XFS partition on the block device, adds an entry in `/etc/fstab`, and mounts the volume with the option of `gquota`. If `gquota` is not set the OpenShift Container Platform node will not be able to start with the `perFSGroup` parameter defined below. This disk and configuration is done on the master, infrastructure, and application nodes.

SSH into the first infrastructure node (infranode1) to verify the entry exists within `/etc/fstab`

```
$ ssh <user>@<resourcegroup>b.<region>.cloudapp.azure.com
$ ssh <user>@infranode1
$ grep "/var/lib/origin/openshift.local.volumes" /etc/fstab
/dev/sdc1 /var/lib/origin/openshift.local.volumes xfs gquota 0 0
```

OpenShift Emptydir Quota

During installation a value for `perFSGroup` is set within the node configuration. The `perFSGroup` setting restricts the ephemeral `emptyDir` volume from growing larger than 512Mi. This `emptyDir` quota is done on the master, infrastructure, and application nodes.

SSH into the first infrastructure node (infranode1) to verify `/etc/origin/node/node-config.yml` matches the information below.

```
$ ssh <user>@<resourcegroup>b.<region>.cloudapp.azure.com
$ ssh <user>@infranode1
```

```
$ sudo grep -B2 perFSGroup /etc/origin/node/node-config.yaml
volumeConfig:
  localQuota:
     perFSGroup: 512Mi
```

Docker Storage Setup

The `/etc/sysconfig/docker-storage-setup` file is created at launch time by the bash script on every node. This file tells the Docker service to use a specific volume group for containers. The extra Docker storage options ensures that a container can grow no larger than 3G. Docker storage setup is performed on all master, infrastructure, and application nodes.

SSH into the first infrastructure node (infranode1) to verify `/etc/sysconfig/docker-storage-setup` matches the information below.

```
$ ssh <user>@<resourcegroup>b.<region>.cloudapp.azure.com
$ ssh <user>@infranode1
$ cat /etc/sysconfig/docker-storage-setup
DEVS=/dev/sdd
VG=docker-vg
DATA_SIZE=95%VG
EXTRA_DOCKER_STORAGE_OPTIONS="--storage-opt dm.basesize=3G"
```

4.7. YUM REPOSITORIES

In section Required Channels the specific repositories for a successful OpenShift Container Platform installation were defined. All systems except for the bastion host should have the same repositories configured. To verify subscriptions match those defined in Required Channels perform the following. The repositories below are enabled during the rhsm-repos playbook during the installation. The installation will be unsuccessful if the repositories are missing from the system.

SSH into the first infrastructure node (infranode1) and verify the command output matches the information below.

```
$ ssh <user>@<resourcegroup>b.<region>.cloudapp.azure.com
$ ssh <user>@infranode1
$ yum repolist
Loaded plugins: langpacks, product-id, search-disabled-repos
repo id                                  repo name
status
rhel-7-fast-datapath-rpms/7Server/x86_64 Red Hat Enterprise Linux Fast
Datapath (RHEL 7 Server) (RPMs) 27
rhel-7-server-extras-rpms/x86_64         Red Hat Enterprise Linux 7
Server - Extras (RPMs)             461+4
rhel-7-server-ose-3.5-rpms/x86_64        Red Hat OpenShift Container
Platform 3.5 (RPMs)               437+30
rhel-7-server-rpms/7Server/x86_64        Red Hat Enterprise Linux 7
Server (RPMs)                      14.285
repolist: 15.210
```

4.8. CONSOLE ACCESS

This section will cover logging into the OpenShift Container Platform management console via the GUI and the CLI. After logging in via one of these methods applications can then be deployed and managed.

4.8.1. Log into GUI console and deploy an application

Perform the following steps from the local workstation.

Open a browser and access the OpenShift Container Platform web console located in https://<resourcegroupname>.<region>.cloudapp.azure.com/console The `resourcegroupname` was given in the `ARM` template, and `region` is the Microsoft Azure zone selected during install. When logging into the OpenShift Container Platform web console, use the user login and password specified during the launch of the `ARM` template.

Once logged, to deploy an example application:

- Click on the [New Project] button
- Provide a "Name" and click [Create]
- Next, deploy the `jenkins-ephemeral` instant app by clicking the corresponding box.
- Accept the defaults and click [Create]. Instructions along with a URL will be provided for how to access the application on the next screen.
- Click [Continue to Overview] and bring up the management page for the application.
- Click on the link provided as the route and access the application to confirm functionality.

4.8.2. Log into CLI and Deploy an Application

Perform the following steps from the local workstation.

Install the oc CLI by visiting the public URL of the OpenShift Container Platform deployment. For example, https://resourcegroupname.region.cloudapp.azure.com/console/command-line and click latest release. When directed to https://access.redhat.com, login with the valid Red Hat customer credentials and download the client relevant to the current workstation operating system. Follow the instructions located on documentation site for getting started with the cli.

A token is required to login to OpenShift Container Platform. The token is presented on the https://resourcegroupname.region.cloudapp.azure.com/console/command-line page. Click to show token hyperlink and perform the following on the workstation in which the oc client was installed.

```
$ oc login https://resourcegroupname.region.cloudapp.azure.com --
token=fEAjn7LnZE6v5SOocCSRVmUWGBNIIEKbjD9h-Fv7p09
```

Note

oc command also supports logging with username and password combination. See `oc help login` output for more information

After the oc client is configured, create a new project and deploy an application, in this case, a php sample application (CakePHP):

```
$ oc new-project test-app
$ oc new-app https://github.com/openshift/cakephp-ex.git --name=php
--> Found image 2997627 (7 days old) in image stream "php" in project
"openshift" under tag "5.6" for "php"

    Apache 2.4 with PHP 5.6
    -----------------------
    Platform for building and running PHP 5.6 applications

    Tags: builder, php, php56, rh-php56

    * The source repository appears to match: php
    * A source build using source code from
https://github.com/openshift/cakephp-ex.git will be created
      * The resulting image will be pushed to image stream "php:latest"
    * This image will be deployed in deployment config "php"
    * Port 8080/tcp will be load balanced by service "php"
      * Other containers can access this service through the hostname
"php"

--> Creating resources with label app=php ...
    imagestream "php" created
    buildconfig "php" created
    deploymentconfig "php" created
    service "php" created
--> Success
    Build scheduled, use 'oc logs -f bc/php' to track its progress.
    Run 'oc status' to view your app.

$ oc expose service php
route "php" exposed
```

Display the status of the application.

```
$ oc status
In project test-app on server
https://resourcegroupname.region.cloudapp.azure.com

http://test-app.apps.13.93.162.100.nip.io to pod port 8080-tcp
(svc/php)
  dc/php deploys istag/php:latest <- bc/php builds
https://github.com/openshift/cakephp-ex.git with openshift/php:5.6
    deployment #1 deployed about a minute ago - 1 pod
```

Access the application by accessing the URL provided by oc `status`. The CakePHP application should be visible now.

4.9. EXPLORE THE ENVIRONMENT

4.9.1. List Nodes and Set Permissions

```
$ oc get nodes --show-labels
NAME          STATUS                     AGE
infranode1    Ready                      16d
infranode2    Ready                      16d
infranode3    Ready                      16d
master1       Ready,SchedulingDisabled   16d
master2       Ready,SchedulingDisabled   16d
master3       Ready,SchedulingDisabled   16d
node01        Ready                      16d
node02        Ready                      16d
node03        Ready                      16d
```

Running this command with a regular user should fail.

```
$ oc get nodes --show-labels
Error from server: User "nonadmin" cannot list all nodes in the cluster
```

The reason it is failing is because the permissions for that user are incorrect.

Note

For more information about the roles and permissions, see Authorization documentation

4.9.2. List Router and Registry

List the router and registry pods by changing to the `default` project.

Note

Perform the following steps from the local workstation.

```
$ oc project default
$ oc get all
NAME                      REVISION        DESIRED       CURRENT
TRIGGERED BY
dc/docker-registry        1               1             1
config
dc/router                 1               2             2
config
NAME                      DESIRED         CURRENT       AGE
rc/docker-registry-1      1               1             10m
rc/router-1               2               2             10m
NAME                      CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
svc/docker-registry       172.30.243.63   <none>        5000/TCP
10m
svc/kubernetes            172.30.0.1      <none>
443/TCP,53/UDP,53/TCP     20m
svc/router                172.30.224.41   <none>
80/TCP,443/TCP,1936/TCP   10m
NAME                      READY           STATUS        RESTARTS
```

```
AGE
po/docker-registry-1-2a1ho   1/1            Running       0
8m
po/router-1-1g84e            1/1            Running       0
8m
po/router-1-t84cy            1/1            Running       0
8m
```

Observe the output of `oc get all`

4.9.3. Explore the Docker Registry

The OpenShift Container Platform ansible playbooks configure three infrastructure nodes that have one registry running. In order to understand the configuration and mapping process of the registry pods, the command `oc describe` is used. `oc describe` details how registries are configured and mapped to the `Azure Blob Storage` using the `REGISTRY_STORAGE_*` environment variables.

Note

Perform the following steps from the local workstation.

```
$ oc describe dc/docker-registry
... [OUTPUT ABBREVIATED] ...
Environment Variables:
  REGISTRY_HTTP_ADDR:     :5000
  REGISTRY_HTTP_NET:     tcp
  REGISTRY_HTTP_SECRET:
7H7ihSNi2k/lqR0i5iINHtx+ItA2cGnpccBAz2URT5c=
  REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_ENFORCEQUOTA: false
  REGISTRY_HTTP_TLS_KEY:     /etc/secrets/registry.key
  REGISTRY_HTTP_TLS_CERTIFICATE:    /etc/secrets/registry.crt
  REGISTRY_STORAGE:      azure
  REGISTRY_STORAGE_AZURE_ACCOUNTKEY:
DUo2VfsnPwGl+4yEmye0iSQuHVrPCVmj7D+oIsYVlmaNJXS4YkZoXODvOfx3luLL6qb4j+1
YhV8Nr/slKE9+IQ==
  REGISTRY_STORAGE_AZURE_ACCOUNTNAME:   sareg<resourcegroup>
  REGISTRY_STORAGE_AZURE_CONTAINER:    registry
... [OUTPUT ABBREVIATED] ...
```

To see if the docker images are being stored in the `Azure Blob Storage` properly, save the `REGISTRY_STORAGE_AZURE_ACCOUNTKEY` value from the command output before and perform the following command on the host you installed the `Azure CLI` Node.js package:

```
$ azure storage blob list registry --account-name=sareg<resourcegroup>
--account-key=<account_key>
info:    Executing command storage blob list
+ Getting blobs in container registry
data:    Name
Blob Type   Length    Content Type              Last Modified
Snapshot Time
data:    ------------------------------------------------------------
------------------------------------------------------------------
```

```
----------------------------------  ----------  --------  --------
---------------  ---------------------------  ------------
data:
/docker/registry/v2/blobs/sha256/31/313a6203b84e37d24fe7e43185f9c8b12b7
27574a1bc98bf464faf78dc8e9689/data
AppendBlob  9624      application/octet-stream  Tue, 23 May 2017
15:44:24 GMT
data:
/docker/registry/v2/blobs/sha256/4c/4c1fa39c5cda68c387cfc7dd32207af1a25
b2413c266c464580001c97939cce0/data
AppendBlob  43515975  application/octet-stream  Tue, 23 May 2017
15:43:45 GMT
... [OUTPUT ABBREVIATED] ...
info:    storage blob list command OK
```

4.9.4. Explore Docker Storage

This section will explore the Docker storage on an infrastructure node.

The example below can be performed on any node but for this example the infrastructure node (infranode1) is used.

The output below verifies docker storage is using the `devicemapper` driver in the `Storage Driver` section and using the proper `LVM VolumeGroup`:

```
$ ssh <user>@<resourcegroup>b.<region>.cloudapp.azure.com
$ ssh <user>@infranode1
$ sudo -i
# docker info
Containers: 2
 Running: 2
 Paused: 0
 Stopped: 0
Images: 4
Server Version: 1.10.3
Storage Driver: devicemapper
 Pool Name: docker--vol-docker--pool
 Pool Blocksize: 524.3 kB
 Base Device Size: 3.221 GB
 Backing Filesystem: xfs
 Data file:
 Metadata file:
 Data Space Used: 1.221 GB
 Data Space Total: 25.5 GB
 Data Space Available: 24.28 GB
 Metadata Space Used: 307.2 kB
 Metadata Space Total: 29.36 MB
 Metadata Space Available: 29.05 MB
 Udev Sync Supported: true
 Deferred Removal Enabled: true
 Deferred Deletion Enabled: true
 Deferred Deleted Device Count: 0
 Library Version: 1.02.107-RHEL7 (2016-06-09)
Execution Driver: native-0.2
Logging Driver: json-file
Plugins:
```

```
 Volume: local
 Network: bridge null host
 Authorization: rhel-push-plugin
Kernel Version: 3.10.0-327.10.1.el7.x86_64
Operating System: Employee SKU
OSType: linux
Architecture: x86_64
Number of Docker Hooks: 2
CPUs: 2
Total Memory: 7.389 GiB
Name: ip-10-20-3-46.azure.internal
ID: XDCD:7NAA:N2S5:AMYW:EF33:P2WM:NF5M:XOLN:JHAD:SIHC:IZXP:MOT3
WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
Registries: registry.access.redhat.com (secure), docker.io (secure)
# vgs
  VG        #PV #LV #SN Attr   VSize   VFree
  docker-vg   1   1   0 wz--n- 128,00g 76,80g
```

If it was in loopback as Storage Mode, the output would list the loopback file. As the below output does not contain the word loopback, the docker daemon is working in the optimal way.

Note

For more information about the docker storage requirements, check Configuring docker storage documentation

4.9.5. Explore the Microsoft Azure Load Balancers

As mentioned earlier in the document two `Azure Load Balancers` have been created. The purpose of this section is to encourage exploration of the load balancers that were created.

Note

Perform the following steps from the `Azure web console`.

On the main Microsoft Azure dashboard, click on [Resource Groups] icon. Then select the resource group that corresponds with the OpenShift Container Platform deployment, and then find the [Load Balancers] within the resource group. Select the `AppLB` load balancer and on the [Description] page note the [Port Configuration] and how it is configured. That is for the OpenShift Container Platform application traffic. There should be three master instances running with a [Status] of `Ok`. Next check the [Health Check] tab and the options that were configured. Further details of the configuration can be viewed by exploring the `ARM` templates to see exactly what was configured.

4.9.6. Explore the Microsoft Azure Resource Group

As mentioned earlier in the document an `Azure Resource Group` was created. The purpose of this section is to encourage exploration of the `resource group` that was created.

Note

Perform the following steps from the `Azure web console`.

On the main Microsoft Azure console, click on [Resource Group]. Next on the left hand navigation panel select the [Your Resource Groups]. Select the `Resource Group` recently created and explore the [Summary] tabs. Next, on the right hand navigation panel, explore the [Virtual Machines], [Storage Accounts], [Load Balancers], and [Networks] tabs More detail can be looked at with the configuration by exploring the ansible playbooks and `ARM` json files to see exactly what was configured.

4.10. TESTING FAILURE

In this section, reactions to failure are explored. After a successful install and some of the smoke tests noted above have been completed, failure testing is executed.

4.10.1. Generate a Master Outage

Note

Perform the following steps from the `Azure web console` and the OpenShift public URL.

Log into the Microsoft Azure console. On the dashboard, click on the[Resource Group] web service and then click [Overview]. Locate the running master2 instance, select it, right click and change the state to `stopped`.

Ensure the console can still be accessed by opening a browser and accessing https://resourcegroupname.region.cloudapp.azure.com. At this point, the cluster is in a degraded state because only 2/3 master nodes are running, but complete functionality remains.

4.10.2. Observe the Behavior of etcd with a Failed Master Node

`SSH` into the first master node (master1) from the bastion. Using the output of the command `hostname` issue the `etcdctl` command to confirm that the cluster is healthy.

```
$ ssh <user>@<resourcegroup>b.<region>.cloudapp.azure.com
$ ssh <user>@master1
$ sudo -i
# etcdctl --endpoints
https://master1:2379,https://master2:2379,https://master3:2379 --ca-
file /etc/etcd/ca.crt --cert-file=/etc/origin/master/master.etcd-
client.crt --key-file=/etc/origin/master/master.etcd-client.key
cluster-health
failed to check the health of member 82c895b7b0de4330 on
https://10.20.2.251:2379: Get https://10.20.1.251:2379/health: dial tcp
10.20.1.251:2379: i/o timeout
member 82c895b7b0de4330 is unreachable: [https://10.20.1.251:2379] are
all unreachable
```

```
member c8e7ac98bb93fe8c is healthy: got healthy result from
https://10.20.3.74:2379
member f7bbfc4285f239ba is healthy: got healthy result from
https://10.20.1.106:2379
cluster is healthy
```

Notice how one member of the `etcd` cluster is now unreachable. Restart master2 by following the same steps in the `Azure web console` as noted above.

4.10.3. Generate an Infrastructure Node outage

This section shows what to expect when an infrastructure node fails or is brought down intentionally.

4.10.3.1. Confirm Application Accessibility

Note

Perform the following steps from the browser on a local workstation.

Before bringing down an infrastructure node, check behavior and ensure things are working as expected. The goal of testing an infrastructure node outage is to see how the OpenShift Container Platform routers and registries behave. Confirm the simple application deployed from before is still functional. If it is not, deploy a new version. Access the application to confirm connectivity. As a reminder, to find the required information to ensure the application is still running, list the projects, change to the project that the application is deployed in, get the status of the application which including the URL and access the application via that URL.

```
$ oc get projects
NAME               DISPLAY NAME   STATUS
openshift                         Active
openshift-infra                   Active
ttester                           Active
test-app1                         Active
default                           Active
management-infra                  Active

$ oc project test-app1
Now using project "test-app1" on server
"https://resourcegroupname.region.cloudapp.azure.com".

$ oc status
In project test-app1 on server
https://resourcegroupname.region.cloudapp.azure.com

http://test-app1.apps.13.93.162.100.nip.io to pod port 8080-tcp
(svc/php-prod)
  dc/php-prod deploys istag/php-prod:latest <-
    bc/php-prod builds https://github.com/openshift/cakephp-ex.git with
openshift/php:5.6
    deployment #1 deployed 27 minutes ago - 1 pod
```

Open a browser and ensure the application is still accessible.

4.10.3.2. Confirm Registry Functionality

This section is another step to take before initiating the outage of the infrastructure node to ensure that the registry is functioning properly. The goal is to push a image to the OpenShift Container Platform registry.

Note

Perform the following steps from a CLI on a local workstation and ensure that the oc client has been configured as explained before.

Important

In order to be able to push images to the registry, the docker configuration on the workstation will be modified to trust the docker registry certificate.

Get the name of the docker-registry pod:

```
$ oc get pods -n default | grep docker-registry
docker-registry-4-9r033    1/1       Running   0          2h
```

Get the registry certificate and save it:

```
$ oc exec docker-registry-4-9r033 cat /etc/secrets/registry.crt >>
/tmp/my-docker-registry-certificate.crt
```

Capture the registry route:

```
$ oc get route docker-registry -n default
NAME              HOST/PORT
PATH      SERVICES          PORT      TERMINATION   WILDCARD
docker-registry   docker-registry-default.13.64.245.134.nip.io
docker-registry   <all>     passthrough   None
```

Create the proper directory in `/etc/docker/certs.d/` for the registry:

```
$ sudo mkdir -p /etc/docker/certs.d/docker-registry-
default.13.64.245.134.nip.io
```

Move the certificate to the directory previously created and restart the docker service in the workstation

```
$ sudo mv /tmp/my-docker-registry-certificate.crt
/etc/docker/certs.d/docker-registry-default.13.64.245.134.nip.io/ca.crt
$ sudo systemctl restart docker
```

A token is needed so that the Docker registry can be logged into.

```
$ oc whoami -t
```

```
feAeAgL139uFFF_72bcJlboTv7gi_bo373kf1byaAT8
```

Pull a new docker image for the purposes of test pushing.

```
$ docker pull fedora/apache
$ docker images | grep fedora/apache
docker.io/fedora/apache  latest  c786010769a8  3 months ago  396.4 MB
```

Tag the docker image with the registry hostname

```
$ docker tag docker.io/fedora/apache docker-registry-
default.13.64.245.134.nip.io/openshift/prodapache
```

Check the images and ensure the newly tagged image is available.

```
$ docker images | grep openshift/prodapache
docker-registry-default.13.64.245.134.nip.io/openshift/prodapache
latest              c786010769a8        3 months ago        396.4 MB
```

Issue a Docker login.

```
$ docker login -u $(oc whoami) -e <email> -p $(oc whoami -t) docker-
registry-default.13.64.245.134.nip.io
Login Succeeded
```

Note

The email doesn't need to be valid and it will be deprecated in next versions of the docker cli

Push the image to the OpenShift Container Platform registry:

```
$ docker push docker-registry-
default.13.64.245.134.nip.io/openshift/prodapache
The push refers to a repository [docker-registry-
default.13.64.245.134.nip.io/openshift/prodapache]
3a85ee80fd6c: Pushed
5b0548b012ca: Pushed
a89856341b3d: Pushed
a839f63448f5: Pushed
e4f86288aaf7: Pushed
latest: digest:
sha256:e2a15a809ce2fe1a692b2728bd07f58fbf06429a79143b96b5f3e3ba0d1ce6b5
size: 7536
```

4.10.3.3. Get Location of Registry

Note

Perform the following steps from the CLI of a local workstation.

Change to the default OpenShift Container Platform project and check the registry pod location

```
$ oc get pods -o wide -n default
NAME                       READY     STATUS    RESTARTS   AGE
IP           NODE
docker-registry-4-9r033    1/1       Running   0          2h
10.128.6.5   infranode3
registry-console-1-zwzsl   1/1       Running   0          5d
10.131.4.2   infranode2
router-1-09x4g             1/1       Running   0          5d
10.0.2.5     infranode2
router-1-6135c             1/1       Running   0          5d
10.0.2.4     infranode1
router-1-l2562             1/1       Running   0          5d
10.0.2.6     infranode3
```

4.10.3.4. Initiate the Failure and Confirm Functionality

Note

Perform the following steps from the `Azure web console` and a browser.

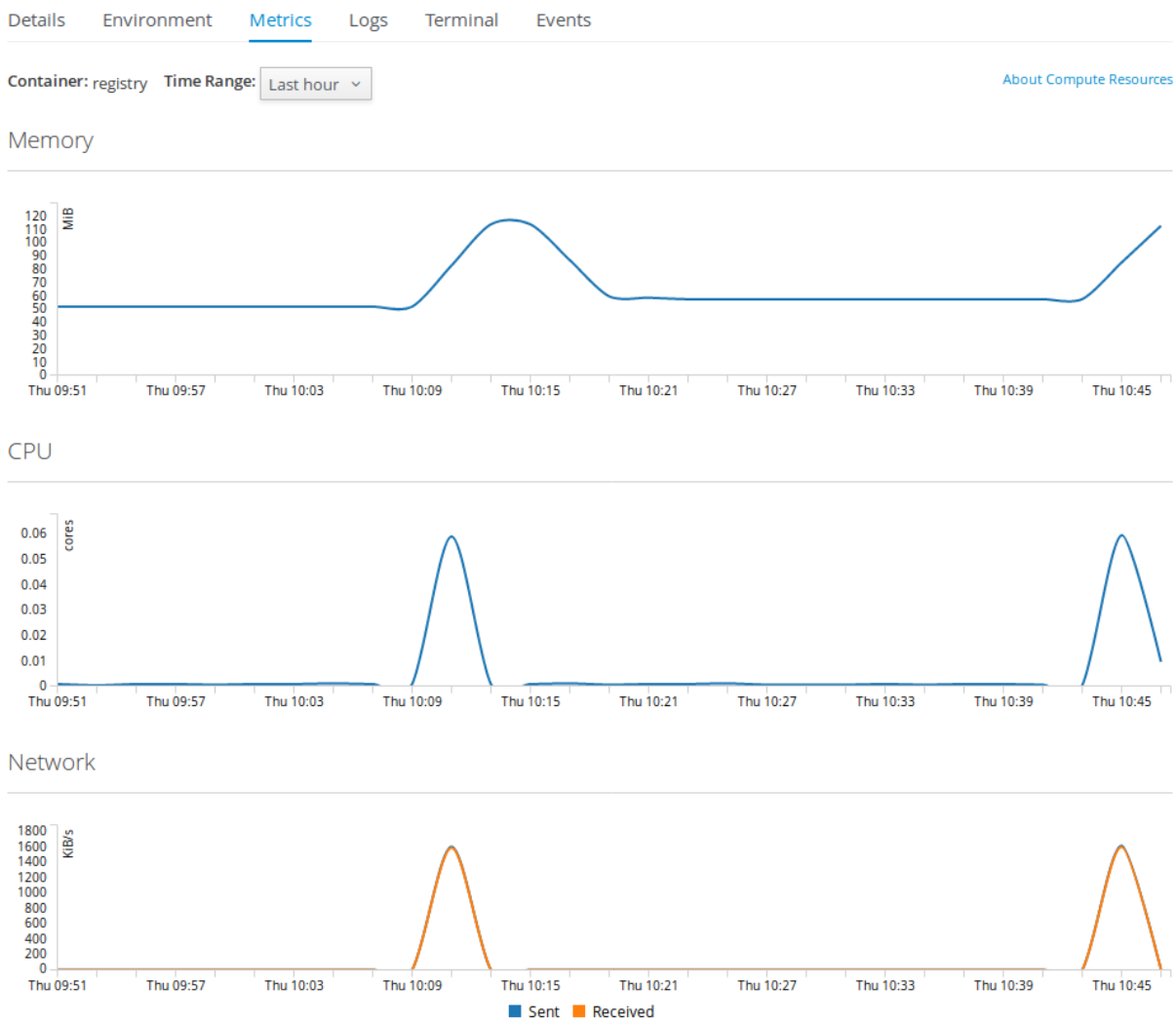
Log into the `Azure web console`. On the dashboard, click on the[Resource Group]. Locate the running instance where the registry pod is running (infranode3 in the previous example), select it, right click and change the state to `stopped`. Wait a minute or two for the registry pod to be migrate over to a different infranode. Check the registry location and confirm that it moved to a different infranode:

```
$ oc get pods -o wide -n default | grep docker-registry
docker-registry-4-kd40f    1/1       Running   0          1m
10.130.4.3   infranode1
```

Follow the procedures above to ensure a Docker image can still be pushed to the registry now that infranode3 is down.

4.11. METRICS EXPLORATION

Red Hat OpenShift Container Platform metrics components enable additional features in the Red Hat OpenShift Container Platform web interface. If the environment has been deployed choosing to deploy metrics, there will be a new tab in the pod section named "Metrics" where it shows usage data of CPU, memory and network resources for a period of time:



Note

If metrics don't show, check if the hawkular certificate has been trusted. Visit the metrics route using the browser and accept the self signed certificate warning and refresh the metrics tab to check if metrics are shown. Future revisions of this reference architecture document will include how to create proper certificates to avoid trusting self signed certificates.

Using the CLI, the cluster-admin can observe the usage of the pods and nodes using the following commands as well:

```
$ oc adm top pod --heapster-namespace="openshift-infra" --heapster-
scheme="https" --all-namespaces
NAMESPACE          NAME                                CPU(cores)
MEMORY(bytes)
openshift-infra    hawkular-cassandra-1-h9mrq          161m
1423Mi
logging            logging-fluentd-g5jqw               8m
92Mi
logging            logging-es-ops-b44n3gav-1-zkl3r     19m
861Mi
... [OUTPUT ABBREVIATED] ...
$ oc adm top node --heapster-namespace="openshift-infra" --heapster-
```

```
scheme="https"
NAME         CPU(cores)   CPU%      MEMORY(bytes)   MEMORY%
infranode3   372m         9%        4657Mi          33%
master3      68m          1%        1923Mi          13%
node02       43m          1%        1437Mi          5%
... [OUTPUT ABBREVIATED] ...
```

4.11.1. Using the Horizontal Pod Autoscaler

In order to be able to use the `HorizontalPodAutoscaler` feature, the metrics components should be deployed and limits should be configured for the pod in order to set the target percentage when the pod will be scaled.

The following commands shows how to create a new project, deploy an example pod and set some limits:

```
$ oc new-project autoscaletest
Now using project "autoscaletest" on server
"https://myocp.eastus2.cloudapp.azure.com:8443".
... [OUTPUT ABBREVIATED] ...

$ oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-
ex.git
--> Found Docker image d9c9735 (10 days old) from Docker Hub for
"centos/ruby-22-centos7"
... [OUTPUT ABBREVIATED] ...

$ oc patch dc/ruby-ex -p \'{"spec":{"template":{"spec":{"containers":
[{"name":"ruby-ex","resources":{"limits":{"cpu":"80m"}}}]}}}}'
"ruby-ex" patched

$ oc get pods
NAME              READY     STATUS      RESTARTS   AGE
ruby-ex-1-210l9   1/1       Running     0          2m
ruby-ex-1-build   0/1       Completed   0          4m

$ oc describe pod ruby-ex-1-210l9
Name:   ruby-ex-1-210l9
... [OUTPUT ABBREVIATED] ...
    Limits:
      cpu: 80m
    Requests:
      cpu:  80m
```

Once the pod is running, create the autoscaler:

```
$ oc autoscale dc/ruby-ex --min 1 --max 10 --cpu-percent=50
deploymentconfig "ruby-ex" autoscaled
$ oc get horizontalpodautoscaler
NAME      REFERENCE                  TARGET    CURRENT   MINPODS
MAXPODS   AGE
ruby-ex   DeploymentConfig/ruby-ex   50%       0%        1         10
53s
```

Access the pod and create some CPU load, as:

```
$ oc rsh ruby-ex-1-210l9

sh-4.2$ while true; do echo "cpu hog" >> mytempfile; rm -f mytempfile;
done
```

Observe the events and the pods running and after a while a new replica will be created:

```
$ oc get events -w
LASTSEEN                        FIRSTSEEN
COUNT     NAME      KIND                        SUBOBJECT   TYPE
REASON                    SOURCE                          MESSAGE
2017-07-13 13:28:35 +0000 UTC   2017-07-13 13:26:30 +0000 UTC   7
ruby-ex   HorizontalPodAutoscaler                Normal
DesiredReplicasComputed   {horizontal-pod-autoscaler }   Computed the
desired num of replicas: 0 (avgCPUutil: 0, current replicas: 1)
2017-07-13 13:29:05 +0000 UTC   2017-07-13 13:29:05 +0000 UTC   1
ruby-ex   HorizontalPodAutoscaler              Normal
DesiredReplicasComputed   {horizontal-pod-autoscaler }   Computed the
desired num of replicas: 2 (avgCPUutil: 67, current replicas: 1)
2017-07-13 13:29:05 +0000 UTC   2017-07-13 13:29:05 +0000 UTC   1
ruby-ex   DeploymentConfig              Normal
ReplicationControllerScaled   {deploymentconfig-controller }   Scaled
replication controller "ruby-ex-1" from 1 to 2
2017-07-13 13:29:05 +0000 UTC   2017-07-13 13:29:05 +0000 UTC   1
ruby-ex   HorizontalPodAutoscaler              Normal
SuccessfulRescale   {horizontal-pod-autoscaler }   New size: 2; reason:
CPU utilization above target
2017-07-13 13:29:05 +0000 UTC   2017-07-13 13:29:05 +0000 UTC   1
ruby-ex-1-zwmxd   Pod                  Normal    Scheduled   {default-
scheduler }   Successfully assigned ruby-ex-1-zwmxd to node02

$ oc get pods
NAME              READY     STATUS      RESTARTS   AGE
ruby-ex-1-210l9   1/1       Running     0          8m
ruby-ex-1-build   0/1       Completed   0          9m
ruby-ex-1-zwmxd   1/1       Running     0          58s
```

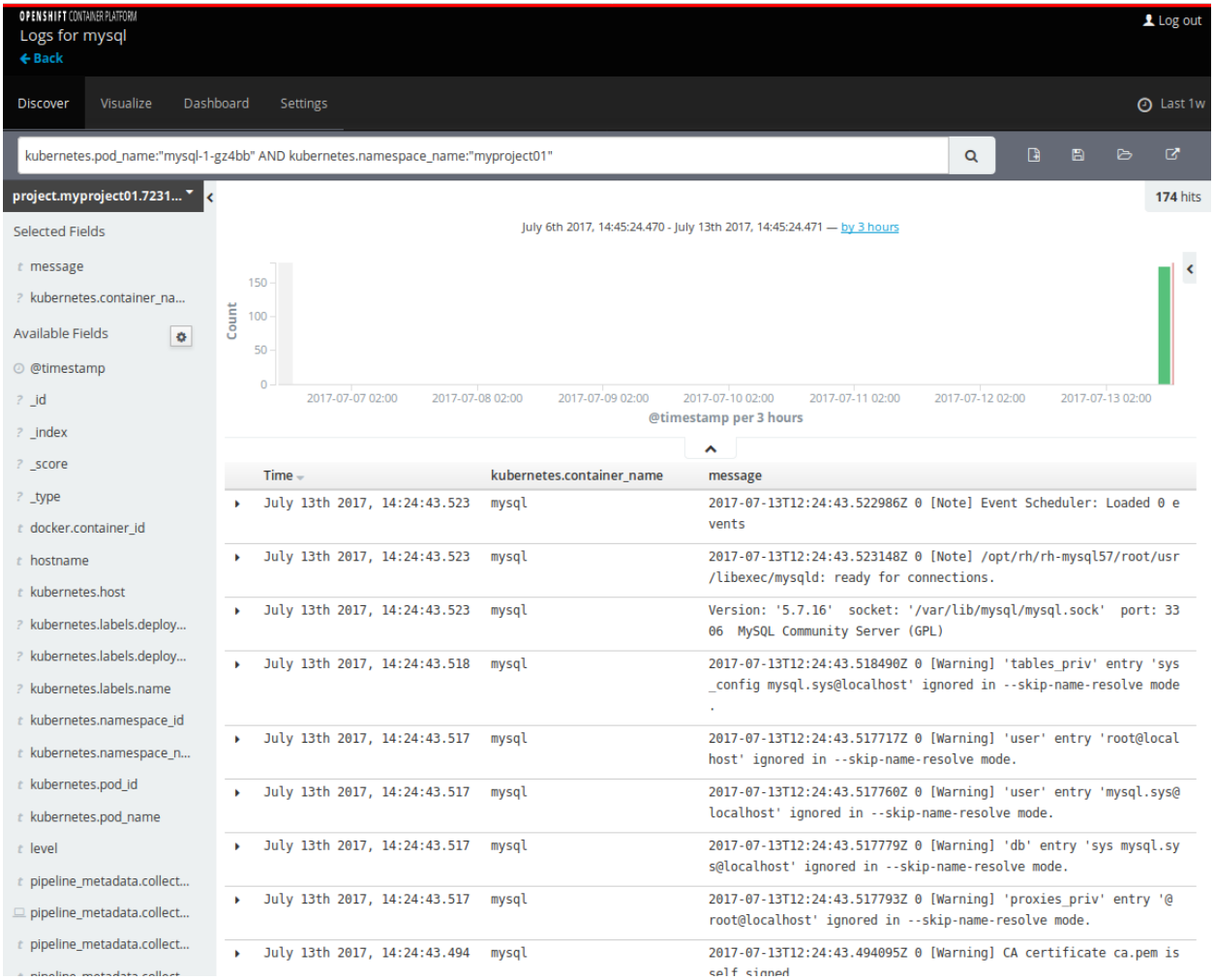
After canceling the CPU hog command, the events will show how the deploymentconfig returns to a single replica:

```
$ oc get events -w
LASTSEEN                        FIRSTSEEN
COUNT     NAME      KIND                        SUBOBJECT   TYPE
REASON                    SOURCE                          MESSAGE
2017-07-13 13:34:05 +0000 UTC   2017-07-13 13:34:05 +0000 UTC   1
ruby-ex   HorizontalPodAutoscaler              Normal
SuccessfulRescale   {horizontal-pod-autoscaler }   New size: 1; reason:
All metrics below target
2017-07-13 13:34:05 +0000 UTC   2017-07-13 13:34:05 +0000 UTC   1
ruby-ex   DeploymentConfig              Normal
ReplicationControllerScaled   {deploymentconfig-controller }   Scaled
replication controller "ruby-ex-1" from 2 to 1
```

```
2017-07-13 13:34:05 +0000 UTC   2017-07-13 13:34:05 +0000 UTC   1
ruby-ex-1   ReplicationController            Normal
SuccessfulDelete   {replication-controller }   Deleted pod: ruby-ex-1-
zwmxd
```

4.12. LOGGING EXPLORATION

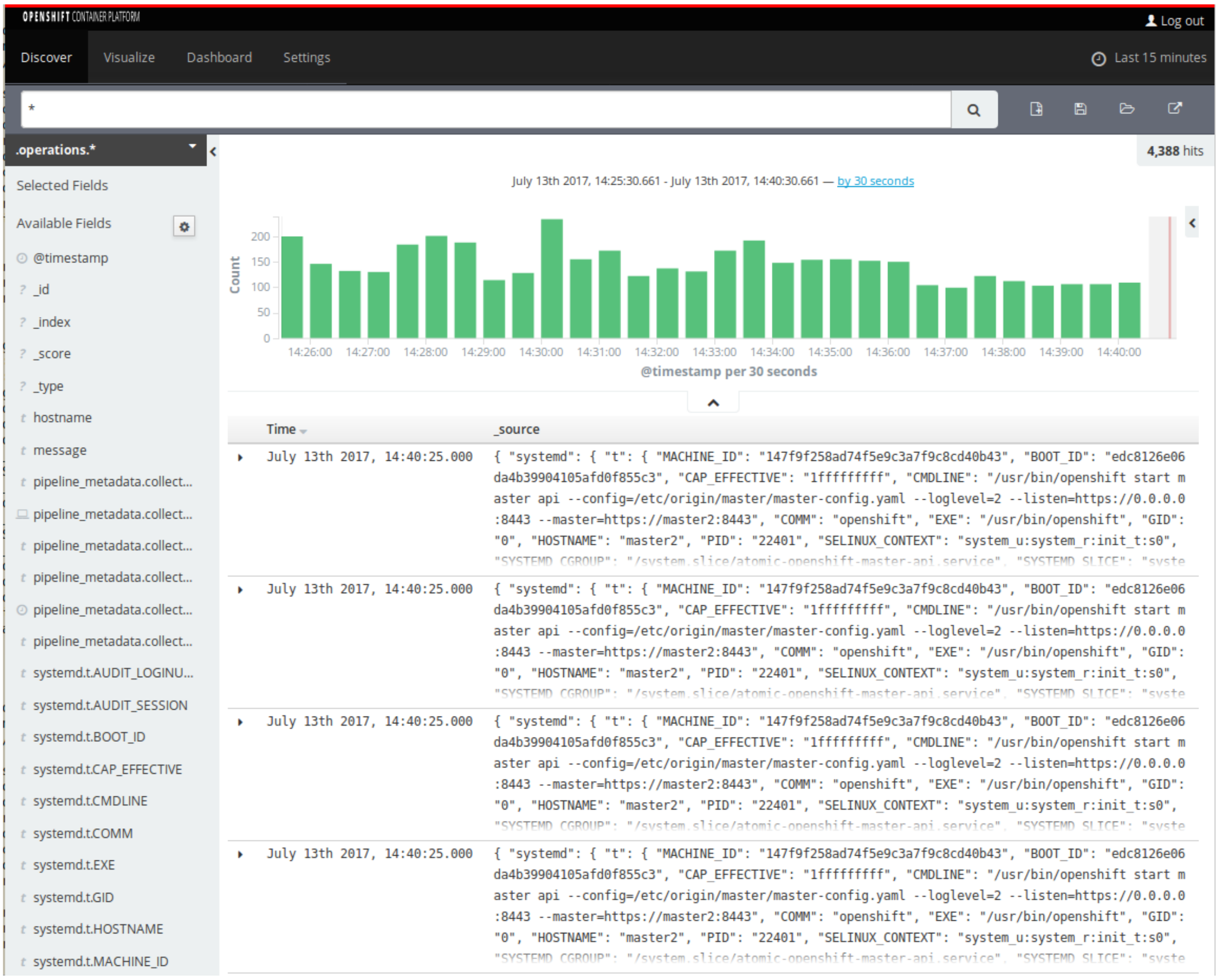
Red Hat OpenShift Container Platform aggregated logging components enable additional features in the Red Hat OpenShift Container Platform web interface. If the environment has been deployed choosing to deploy logging, there will be a new link in the pod logs section named "View Archive" that will redirect to the Kibana web interface for the user to see the pods logs, create queries, filters, etc.



Note

For more information about Kibana, see Kibana documentation

In case the "opslogging" cluster has been deployed, there will be a route "kibana-ops" in the "logging" project where cluster-admin users can browse infrastructure logs.


OPENSHIFT CONTAINER PLATFORM
Log out
Discover
Visualize
Dashboard
Settings
Last 15 minutes
.operations.*
4,388 hits
July 13th 2017, 14:25:30.661 - July 13th 2017, 14:40:30.661 — by 30 seconds
Selected Fields
Available Fields
@timestamp
_id
_index
_score
_type
hostname
message
systemd.t.AUDIT_SESSION
systemd.t.BOOT_ID
systemd.t.CAP_EFFECTIVE
systemd.t.CMDLINE
systemd.t.COMM
systemd.t.EXE
systemd.t.GID
systemd.t.HOSTNAME
systemd.t.MACHINE_ID
Count
@timestamp per 30 seconds
Time
_source
July 13th 2017, 14:40:25.000
{ "systemd": { "t": { "MACHINE_ID": "147f9f258ad74f5e9c3a7f9c8cd40b43", "BOOT_ID": "edc8126e06da4b39904105afd0f855c3", "CAP_EFFECTIVE": "1fffffffff", "CMDLINE": "/usr/bin/openshift start master api --config=/etc/origin/master/master-config.yaml --loglevel=2 --listen=https://0.0.0.0:8443 --master=https://master2:8443", "COMM": "openshift", "EXE": "/usr/bin/openshift", "GID": "0", "HOSTNAME": "master2", "PID": "22401", "SELINUX_CONTEXT": "system_u:system_r:init_t:s0",

CHAPTER 5. PERSISTENT STORAGE

Container storage by default is not persistent. For example, if a new container build occurs then data is lost because the storage is non-persistent or if a container terminates then of the all changes to its local filesystem are lost. OpenShift Container Platform offers many different types of persistent storage to avoid those situations. Persistent storage ensures that data that should persist between builds and container migrations is available.

Note

For more information about the available storage options in OpenShift Container Platform see Types of Persistent Volumes

When choosing a persistent storage backend ensure that the backend supports the scaling, speed, and redundancy that the project requires. This reference architecture will focus on cloud provider specific storage.

Note

This reference architecture is emerging and components like Container-Native Storage (CNS), and Container-Ready Storage(CRS) will be described in future revisions.

5.1. PERSISTENT VOLUMES

Container storage is defined by the concept of `persistent volumes` (pv) which are OpenShift Container Platform objects that allow for storage to be defined and then used by pods for data persistence. Requesting of `persistent volumes` is done by using a `persistent volume claim` (pvc) object. This claim, when successfully fulfilled by the system will also mount the persistent storage to a specific directory within a pod or multiple pods. This directory is referred to as the `mountPath` and facilitated using a concept known as `bind-mount`.

Note

For more information about the `persistent volumes` and its lifecycle, see Lifecycle of a Volume and Claim

`Persistent volumes` can be preprovisioned by the OpenShift Container Platform administrator by creating them in the underlying infrastructure and in OpenShift Container Platform manually, or the administrator can configure OpenShift Container Platform to create automatically the proper `persistent volumes` when users request them using the `dynamic provisioning` and `storage classes` capabilities of OpenShift Container Platform.

5.2. STORAGE CLASSES

The `StorageClass` resource object describes and classifies different types of storage that can be requested, as well as provides a means for passing parameters to the backend for

dynamically provisioned storage on demand. `StorageClass` objects can also serve as a management mechanism for controlling different levels of storage and access to the storage. `Cluster Administrators (cluster-admin)` or `Storage Administrators (storage-admin)` define and create the `StorageClass` objects that users can use without needing any intimate knowledge about the underlying storage volume sources. Because of this the naming of the `storage class` defined in the `StorageClass` object should be useful in understanding the type of storage it maps to (ie., `HDD` vs `SDD` or `Premium_LRS` vs `Standard_LRS`).

5.3. CLOUD PROVIDER SPECIFIC STORAGE

Cloud provider specific storage is storage that is provided from Microsoft Azure. This type of storage is presented as an `Data disk VHD` and can be mounted by one pod at a time. It is needed to configure OpenShift Container Platform with Microsoft Azure settings like the `resourceGroup` or `subscriptionID` in the `/etc/azure/azure.conf` file on masters and nodes as well as in the OpenShift Container Platform masters and nodes configuration file to be able to use `VHD` as `persistent storage` for pods. The settings needed are automatically configured as part of the installation process using the code provided in the openshift-ansible-contrib git repository.

Note

For more information about the required settings, see Configuring Azure

Cloud provider storage can be created manually and assigned as a `persistent volume` or a `persistent volume` can be created dynamically using a `StorageClass` object. Note that `VHD` storage can only use the access mode of `Read-Write-Once` (`RWO`).

The `VHDs` used in Microsoft Azure are `.vhd` files stored as page blobs in a standard or premium storage account in Microsoft Azure where standard delivers cost-effective storage and premium delivers high-performance, low-latency storage.

5.3.1. Creating a Storage Class

When requesting cloud provider specific storage in Microsoft Azure for OpenShift Container Platform, there are two options to define a `storage class`:

- Create a `service account` in the same `resource group` where the OpenShift Container Platform cluster has been deployed in Microsoft Azure where all the `VHDs` will be created.
- Provide a `skuName` and `location` to OpenShift Container Platform where all storage accounts associated with the resource group are searched to find one that matches.

Tip

In this reference architecture the first options has been chosen as it is simpler and avoids searching for matching `service accounts` where they will be provided before using them.

Besides the `/etc/azure/azure.conf` configuration file, it is required to create a `storage account` per `storage class` created in OpenShift Container Platform in order to be able to use `dynamic provisioning` of volumes for the pod storage.

Note

This reference architecture creates automatically two different `storage accounts` for pod storage that will be used in different `storage classes` to demonstrate the process.

There are two Microsoft Azure `storage accounts` created as part of the installation process using the `ARM` template:

- **`sapv<resourcegroup>` - For the `generic` storage class (using premium storage)**
- **`sapvlm<resourcegroup>` - To store metrics and logging volumes (using premium storage)**

To create more storage accounts, the azure-cli can be used as:

```
$ azure storage account create --sku-name <sku> --kind "Storage" -g
<resourcegroup> -l <region> <storage account name>
```

This example shows how to create a `sapv3sysdeseng storage class` using `standard storage` in the `westus` region:

```
$ azure storage account create --sku-name "LRS" --kind "Storage" -g
sysdeseng sapv3sysdeseng -l "westus"
info:    Executing command storage account create
+ Checking availability of the storage account name
+ Creating storage account
info:    storage account create command OK
```

Once the `storage account` has been created, a `StorageClass` OpenShift Container Platform object can be created to map it as:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: mystorageclass
provisioner: kubernetes.io/azure-disk
parameters:
  storageAccount: sapv3sysdeseng
```

The cluster-admin or storage-admin can then create the `StorageClass` object using the yaml file.

```
$ oc create -f my-storage-class.yaml
```

Multiple `StorageClassess` objects can be defined depending on the storage needs of the pods within OpenShift Container Platform.

5.3.2. Creating and using a Persistent Volumes Claim

The example below shows a dynamically provisioned volume being requested from the `StorageClass` named `mystorageclass`.

```
$ vi db-claim.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: db
 annotations:
   volume.beta.kubernetes.io/storage-class: mystorageclass
spec:
 accessModes:
  - ReadWriteOnce
 resources:
   requests:
     storage: 10Gi

$ oc create -f db-claim.yaml
persistentvolumeclaim "db" created
$ oc get pvc db
NAME      STATUS    VOLUME
CAPACITY   ACCESSMODES   AGE
db        Bound     pvc-be63668e-451e-11e7-b30b-000d3a36dea3   10Gi
RWO           1m
```

The `cluster-admin` role can also view more information about the `persistent volume`

```
$ oc describe pv pvc-be63668e-451e-11e7-b30b-000d3a36dea3
Name:  pvc-be63668e-451e-11e7-b30b-000d3a36dea3
Labels:  <none>
StorageClass: mystorageclass
Status:  Bound
Claim:  testdev/db
Reclaim Policy: Delete
Access Modes: RWO
Capacity: 10Gi
Message:
Source:
    Type:  AzureDisk (an Azure Data Disk mount on the host and bind
mount to the pod)
    DiskName:  kubernetes-dynamic-pvc-be63668e-451e-11e7-b30b-
000d3a36dea3.vhd
    DiskURI:
https://sapv3sysdeseng.blob.core.windows.net/vhds/kubernetes-dynamic-
pvc-be63668e-451e-11e7-b30b-000d3a36dea3.vhd
    FSType:  ext4
    CachingMode: None
    ReadOnly:  false
No events.
```

5.3.3. Deleting a PVC (Optional)

There may become a point in which a pvc is no longer necessary for a project. The following

can be done to remove the pvc.

```
$ oc delete pvc db
persistentvolumeclaim "db" deleted
$ oc get pvc db
No resources found.
Error from server: persistentvolumeclaims "db" not found
```

Note

Microsoft Azure does not support the Recycle reclaim policy, so all the data will be erased

CHAPTER 6. EXTENDING THE CLUSTER

By default, this reference architecture deploys 3 master, 3 infrastructure nodes, and 3 to 30 application nodes. This cluster size provides enough resources to get started with deploying a few test applications or a Continuous Integration Workflow example. However, as the cluster begins to be utilized by more teams and projects, it will be become necessary to provision more application or infrastructure nodes to support the expanding environment. To facilitate easily growing the cluster, the `add_host.sh` script is provided in the `openshift-ansible-contrib` repository. It will allow for provisioning either an application node, infrastructure node or master host per run and can be ran as many times as needed.

Note

The scale up procedure for masters includes the scale up procedure for nodes as the master hosts need to be part of the SDN.

6.1. PREREQUISITES FOR ADDING A NEW HOST

Verify the quantity and type of the nodes in the cluster by using the `oc get nodes` command. The output below is an example of a complete Red Hat OpenShift Container Platform environment after the deployment of the reference architecture environment.

```
$ oc get nodes
NAME         STATUS                     AGE
infranode1   Ready                      3m
infranode2   Ready                      3m
infranode3   Ready                      3m
master1      Ready,SchedulingDisabled   3m
master2      Ready,SchedulingDisabled   3m
master3      Ready,SchedulingDisabled   3m
node01       Ready                      3m
node02       Ready                      3m
node03       Ready                      3m
```

The script should be executed as the regular user created as part of the Red Hat OpenShift Container Platform reference architecture deployment on the bastion host.

Important

If manual changes in the Red Hat OpenShift Container Platform environment exist, ensure the inventory file reflects those changes prior to the scale up procedure. This includes changes to the Red Hat OpenShift Container Platform configuration files, for example, modifying the Red Hat OpenShift Container Platform masters configuration file to customize the Red Hat OpenShift Container Platform authentication provider as they may be overwritten.

6.2. ADD_HOST.SH

The bash script `add_host.sh` adds new hosts to the Red Hat OpenShift Container Platform cluster where it accepts a few parameters to customize the new host that is going to be

created in Microsoft Azure as part of the process. The script creates the required Microsoft Azure components such as `nic`, `vm`, `nsg`, attaches the new host to the load balancer if needed, deploys the `vm` as part of the same `availabilityset`, etc., run the prerrequisites playbooks to prepare the host, modifies the ansible inventory to fit the requirements and then runs the proper scale up playbook provided by the `atomic-openshift-utils` package.

Note

The VM name follows the reference architecture naming convention, so in case a new application node is added, its name is going to be the next nodeXY available (node04, node05,...)

Important

The script scales up the cluster one host per run but it can be ran as many times as needed.

Table 6.1. Parameters

Flag	Required	Description	Default value
`-t|--type`	No	Host type (node, master or infranode)	node
`-u|--user`	No	Regular user to be created on the host	Current user
`-p|--sshpub`	No	Path to the public `ssh` key to be injected in the host	`~/.ssh/id_rsa.pub`
`-s|--size`	No	VM size	Standard_DS12_v2 for node, Standard_DS12_v2 for infra node, Standard_DS3_v2 for master
`-d|--disk`	No	Extra disk size in GB (it can be repeated a few times)	2x128GB

6.3. ADDING AN APPLICATION NODE

To add an Application Node with the default values, run the `add_host.sh` script following the example below. Once the instance is launched, the installation of Red Hat OpenShift Container Platform will automatically begin.

```
$ ./add_host.sh
```

If some parameters need to be customized, use the proper flags. The example below adds a new Application Node with a different VM size, different user and ssh public key:

```
$ ./add_host.sh -s Standard_DS3_v2 -u user123 -p /my/other/ssh-id.pub
```

6.4. ADDING AN INFRASTRUCTURE NODE

The process for adding an Infrastructure Node is nearly identical to adding an Application Node. The only difference in adding an Infrastructure node is the type flag need to be set to "infranode". Follow the example steps below to add a new Infrastructure Node using the default values:

```
$ ./add_host.sh -t infranode
```

If some parameters need to be customized, use the proper flags. The example below adds a new Infrastructure Node with different disk sizes:

```
$ ./add_host.sh -t infranode -d 20 -d 200
```

6.5. ADDING A MASTER HOST

The process for adding a Master host is nearly identical to adding an Application Node. The only difference in adding a Master host is the type flag need to be set to "master". Follow the example steps below to add a new Master host using the default values:

```
$ ./add_host.sh -t master
```

If some parameters need to be customized, use the proper flags. The example below adds a new Master Host with different disk sizes and different user:

```
$ ./add_host.sh -t master -d 50 -d 20 -u adminxyz
```

Important

The current procedures for scaling up masters doesn't scale the etcd database if the masters contains etcd in the current environment. For a manual procedure on scale etcd, see adding new etcd hosts documentation.

6.6. VALIDATING A NEWLY PROVISIONED HOST

To verify a newly provisioned host has been added to the existing environment, use the `oc get nodes` command. In this example, 2 new Infrastructure nodes, 2 new Master hosts and 2 new Application Nodes have been added using the `add_host.sh` script by executing it a few times.

```
$ oc get nodes
NAME         STATUS                     AGE
infranode1   Ready                      5h
infranode2   Ready                      5h
infranode3   Ready                      5h
infranode4 Ready 1h
infranode5 Ready 4m
master1      Ready,SchedulingDisabled   5h
master2      Ready,SchedulingDisabled   5h
master3      Ready,SchedulingDisabled   5h
master4 Ready,SchedulingDisabled 2h
master5 Ready,SchedulingDisabled 1h
node01       Ready                      5h
node02       Ready                      5h
node03       Ready                      5h
node04 Ready 3h
node05 Ready 2h
```

The following procedure creates a new project and forces the pods of that project to run on the new host. This procedure validates the host is properly configured to run Red Hat OpenShift Container Platform pods:

Create a new project to test:

```
$ oc new-project scaleuptest
Now using project "scaleuptest" on server
"https://myocpdeployment.eastus2.cloudapp.azure.com:8443".
... [OUTPUT ABBREVIATED] ...
```

Patch the node-selector to only run pods on the new node:

```
$ oc patch namespace scaleuptest -p "{\"metadata\":{\"annotations\":
{\"openshift.io/node-selector\":\"kubernetes.io/hostname=node04\"}}}"
"scaleuptest" patched
```

Deploy an example app:

```
$ oc new-app openshift/hello-openshift
--> Found Docker image 8146af6 (About an hour old) from Docker Hub for
"openshift/hello-openshift"
... [OUTPUT ABBREVIATED] ...
```

Scale the number of pods to ensure they are running on the same host:

```
$ oc scale dc/hello-openshift --replicas=8
deploymentconfig "hello-openshift" scaled
```

Observe where the pods run:

```
$ oc get pods -o wide
```

```
hello-openshift-1-1ffl6   1/1       Running   0          3m
10.128.4.10   node04
hello-openshift-1-1kgpf   1/1       Running   0          3m
10.128.4.3    node04
hello-openshift-1-4lk85   1/1       Running   0          3m
10.128.4.4    node04
hello-openshift-1-4pfkk   1/1       Running   0          3m
10.128.4.7    node04
hello-openshift-1-56pqg   1/1       Running   0          3m
10.128.4.6    node04
hello-openshift-1-r3sjz   1/1       Running   0          3m
10.128.4.8    node04
hello-openshift-1-t0fmm   1/1       Running   0          3m
10.128.4.5    node04
hello-openshift-1-v659g   1/1       Running   0          3m
10.128.4.9    node04
```

Clean the environment:

```
$ oc delete project scaleuptest
```

In case the checks are mandatory before adding the host to the cluster, the labels can be set to avoid the default node-selector, run the checks then relabel the node:

```
... [OUTPUT ABBREVIATED] ...
[new_nodes]
node04.example.com openshift_node_labels="{\'role':
\'test',\'test':\'true'}"
```

Perform the scale up procedure, run the required tests, then relabel the node:

```
$ oc label node node04 "role=app" "zone=X" --overwrite
node "node04" labeled
$ oc label node node04 test-
node "node04" labeled
```

CHAPTER 7. CONCLUSION

Red Hat solutions involving the OpenShift Container Platform are created to deliver a production-ready foundation that simplifies the deployment process, shares the latest best practices, and provides a stable highly available environment on which to run production applications.

This reference architecture covered the following topics:

- **A completely provisioned infrastructure in Microsoft Azure public cloud**
- **OpenShift Container Platform three routers and three registry services deployed in dedicated infrastructure nodes**
- **Native integration with Microsoft Azure services as:**
  - **`Azure Load Balancer` for load balancing the API and console and for the applications running in OpenShift Container Platform**
  - **`Azure VHD` storage for persistent storage of container images**
  - **`Azure Premium Storage` for docker storage on each node**
  - **`Azure Blob Storage` for registry to increase scaling**
  - **`Azure Availability Zones` to increase reliability**
- **Creation of applications**
- **Validating the environment**
- **Testing failover**

For any questions or concerns, please email refarch-feedback@redhat.com and ensure to visit the Red Hat Reference Architecture page to find about all of our Red Hat solution offerings.

CHAPTER 8. INSTALLATION FAILURE

In the event of an OpenShift Container Platform installation failure use the following sections to diagnose and find the source of the problem. Note that the `resource group` can be deleted, reinstalled by running the script again.

8.1. DIAGNOSTIC AND CONTROL OF OPENSHIFT CONTAINER PLATFORM ON MICROSOFT AZURE

The OpenShift Container Platform installation can be controlled from the bastion host. This is a separate virtual machine that allows access to all VM's in the same resource group that defines the OpenShift Container Platform installation on Microsoft Azure.

As an example, assuming the resource group was named during creation to `ocpxenon1000`, with a username of `ocpadmin` on the `westus` region:

```
$ ssh ocpadmin@ocpxenon1000b.westus.cloudapp.azure.com
Last login: Sat Jan 21 04:32:47 2017 from 103.252.201.32
[ocpadmin@bastion ~]$
```

8.2. LOGGING OF INSTALLATION

The automation collects logs for various stages of the installation. All the logs are stored on the bastion host. Assuming `ocpadmin` has been chosen as the admin user when creating the install, there are some useful logs in the home directory of the user (`/home/ocpadmin`):

Table 8.1. Installation logs

File name	Content
`ansible-preinstall-ping.out`	Check connectitity of all hosts
`openshift-install.out`	Main **OpenShift Container Platform** installation

8.3. INVENTORY

The inventory for ansible is automatically generated by the bastion.sh script at the first boot of the bastion host, stored in the bastion itself at the default location (`/etc/ansible/hosts`) and can be used to run update scripts. In order to run updates, or to diagnose failures, it is necessary to ssh to the bastion host on Microsoft Azure.

```
$ sudo cat /etc/ansible/hosts
[OSEv3:children]
masters
etcd
```

```
nodes
misc

[OSEv3:vars]
azure_resource_group=ocpxenon1000
rhn_pool_id=8a85f98156724eaa0156728452003452
openshift_install_examples=true
deployment_type=openshift-enterprise
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login':
'true', 'challenge': 'true', 'kind':
'HTPasswdPasswordIdentityProvider', 'filename':
'/etc/origin/master/htpasswd'}]

# default selectors for router and registry services
openshift_router_selector='region=infra'
openshift_registry_selector='region=infra'

ansible_become=yes
ansible_ssh_user=ocpadmin
remote_user=ocpadmin

openshift_master_default_subdomain=52.163.224.147.xip.io
openshift_use_dnsmasq=False
openshift_public_hostname=ocpxenon1000.westus.cloudapp.azure.com

openshift_master_cluster_method=native
openshift_master_cluster_hostname=ocpxenon1000.westus.cloudapp.azure.co
m
openshift_master_cluster_public_hostname=ocpxenon1000.westus.cloudapp.a
zure.com

# Enable cockpit
osm_use_cockpit=true

# Set cockpit plugins
osm_cockpit_plugins=['cockpit-kubernetes']

# default storage plugin dependencies to install, by default the ceph
and
# glusterfs plugin dependencies will be installed, if available.
osn_storage_plugin_deps=['Azure VHD']

[masters]
master1 openshift_hostname=master1 openshift_node_labels="{'role':
'master'}"
master2 openshift_hostname=master2 openshift_node_labels="{'role':
'master'}"
master3 openshift_hostname=master3 openshift_node_labels="{'role':
'master'}"

[etcd]
master1
master2
master3

[nodes]
```

```
master1 openshift_node_labels="{'region':'master','zone':'default'}"
openshift_schedulable=false
master2 openshift_node_labels="{'region':'master','zone':'default'}"
openshift_schedulable=false
master3 openshift_node_labels="{'region':'master','zone':'default'}"
openshift_schedulable=false
node[01:03] openshift_node_labels="{'role': 'app', 'zone': 'default'}"
infranode1 openshift_hostname=infranode1 openshift_node_labels="
{'role': 'infra', 'zone': 'default'}"
infranode2 openshift_hostname=infranode2 openshift_node_labels="
{'role': 'infra', 'zone': 'default'}"
infranode3 openshift_hostname=infranode3 openshift_node_labels="
{'role': 'infra', 'zone': 'default'}"
```

8.4. UNINSTALLING AND DELETING

The uninstall playbook removes OpenShift Container Platform related packages, `etcd`, and removes any certificates that were created during the failed install. In case you need to do it, run the following from the bastion host:

```
$ ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/adhoc/uninstall.yml
```

After the playbook, the administrator should unsubscribe each host, to return the subscription back into the available pool, and then delete the resource group within Microsoft Azure portal, which will delete all resources.

8.5. MANUALLY LAUNCHING THE INSTALLATION OF OPENSHIFT

The `openshift-install.sh` script, located in the bastion host at `/home/user/openshift-install.sh`, can be used to automatically install OpenShift Container Platform. The script can be re-run to diagnose problems.

```
$ ./openshift-install.sh
```

8.6. GMAIL NOTIFICATION

The `bastion.sh` script can optionally notify the user via email during the installation about the steps that has been done. It creates a `/root/setup_ssmtp.sh` script with the username and password provided in the `ARM template` that will configure an `ssmtp` MTA service, and if the GMail account exists, it will notify the user periodically on the steps finished.

APPENDIX A. CONTRIBUTORS

Glenn West, content provider

Ryan Cook, content provider

Scott Collier, content provider

Jason DeTiberus, content provider

Matt Woodson, content reviewer

Harold Wong, content reviewer

Thomas Wiest, content reviewer

APPENDIX B. REVISION HISTORY

Revision 3.5.6-0	**2017-08-08**	**Glenn West (gwest@redhat.com)**

- **Added allinone single vm alternative template**

Revision 3.5.5-0	**2017-07-24**	**Eduardo Minguez (edu@redhat.com)**

- **Added extend cluster section and SDN option**

Revision 3.5.4-0	**2017-07-14**	**Eduardo Minguez (edu@redhat.com)**

- **Added metrics and aggregated logging**

Revision 3.5.3-0	**2017-07-10**	**Glenn West (gwest@redhat.com)**

- **Document the use of Ansible Playbook to Deploy ARM Template**

Revision 3.5.2-0	**2017-06-23**	**Eduardo Minguez (edu@redhat.com)**

- **Added more information about security groups**

Revision 3.5.1-0	**2017-06-01**	**Eduardo Minguez (edu@redhat.com)**

- **Added some sections to clarify some concepts**

Revision 3.5.0-0	**2017-01-25**	**Glenn West (gwest@redhat.com)**

- **Initial creation of document**