

Running OpenShift Container Platform with IBM Cloud Infrastructure Center for KVM

IBM Hybrid Cloud Integration Test team

September 30, 2022

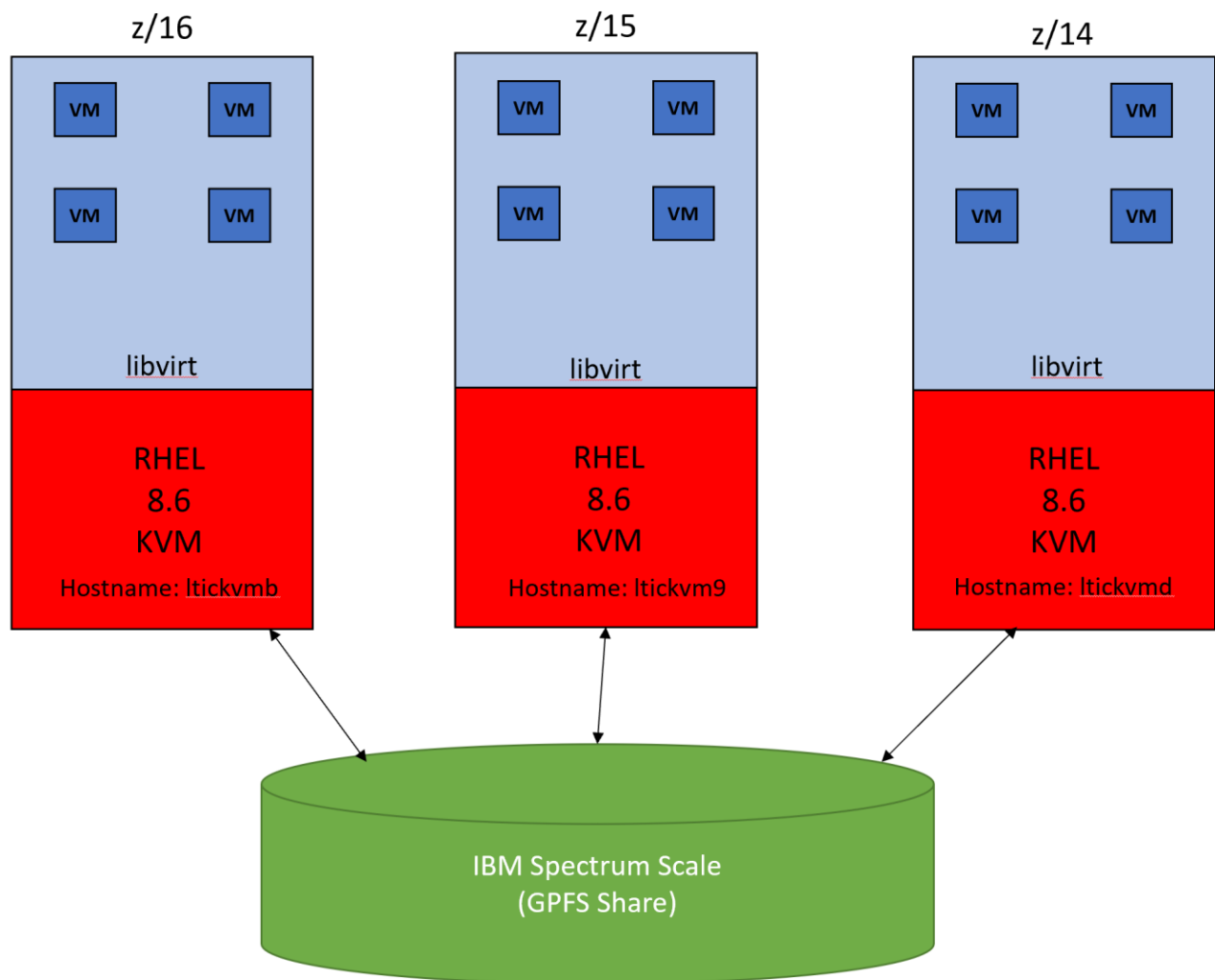
Overview

As the growing popularity of different cloud offerings and solutions grow in the tech industry, IBM has decided to throw their hat in the ring with an IaaS offering called IBM Cloud Infrastructure Center (ICIC). An infrastructure management solution that can manage and facilitate the on-premise cloud deployments of IBM z/VM based and KVM (Kernel Virtual Machine) based Linux virtual machines on the IBM Z and IBM LinuxONE platforms. This guide will strictly focus on the deployment of ICIC for KVM on IBM LinuxONE and outline the necessary steps I needed to take in order to properly configure ICIC for my production environment. In addition to that, I will also be covering how to install Red Hat's OpenShift Container Platform (OCP) using a running production instance of ICIC. OCP is Red Hat's enterprise container orchestration solution, backed by Kubernetes, which allows for clients to build, deploy, and manage their production applications. The goal of this document is to provide an overview of the procedure I followed to install an OCP cluster with ICIC, so that others that follow can learn from my experience and perform this installation in an efficient manner. This guide will be broken up into two parts, part one will cover how I went about installing ICIC itself and part two will cover how I used ICIC to install OCP with some fancy Ansible automation provided by the ICIC team.

Part 1 (ICIC)

The first thing worth noting about IBM Cloud Infrastructure Center is that it is an IBM packaged and managed implementation of Openstack. At a high level, Openstack is a popular open-source IaaS solution that can manage large amounts of different virtual environments and storage. It can essentially do anything ICIC can and because it is open source, it is free...so why would anyone want to use IBM Cloud Infrastructure Center? The answer really boils down to one main thing, simplicity. ICIC is a simplified version of Openstack, it is a single installable that allows you to quickly stand up an Openstack environment geared towards IBM Z and IBM LinuxONE platforms. Standalone Openstack is quite the opposite, there are **A LOT** of different pieces that need to be installed and configured properly for the software to operate as intended. Not to mention, all the installations and configurations are different depending on what platform you choose to install it on. I really can't emphasize enough, OpenStack is a complicated piece of technology, and there are numerous different processes, services, and networking pieces that go into operating the software correctly. Now although ICIC is a simplified version of Openstack, it is still Openstack, and complex things can still go wrong making it difficult to debug (Believe me it will). Thankfully, ICIC has a dedicated development team that you can rely on to ask questions and help you debug problems. I consulted them multiple times during this effort, and they proved to be very helpful. Google is also your friend, given the fact that a lot of the errors you might experience will be OpenStack related, the Openstack community will most likely have posted about a similar problem you are facing and hopefully have provided a solution.

ICIC manages a lot of this Openstack complexity for you under the covers and it tries its best to shield you from all the moving pieces that are running in the environment. In my case, the environment I started with before installing ICIC looks something like the diagram below:



My environment consists of three IBM LinuxONE machines, each operating across the above hardware levels (z16, z15, z14), and each machine is running an LPAR (Logical Partition) with RHEL8.6 installed. I already configured the LPARs in my environment to run KVM virtual machines prior to the installation of ICIC, as a result of prior work I had been doing with KVM. At this point, all the virtual machines in my environment are connected over a MacVtap network connection to each KVM host. I also preemptively configured each LPAR to have access to an eleven terabyte GPFS share, petKVM, which I plan on using as the backing storage for my provisioned ICIC guests. GPFS, currently known as IBM Spectrum Scale, is outside of the scope for this document but for context purposes, it is a cloud storage offering that allows for my KVM hosts to have concurrent access to a single filesystem which I will use to store my ICIC guest images.

IBM Cloud Infrastructure Center requires two types of nodes in order to operate. A management node, which is a Linux machine responsible for overseeing all the OpenStack processes running across your ICIC instances. A compute node, which is a Linux machine acting as a Hypervisor, KVM in my case, in charge of provisioning and managing virtual machines. The management node executes a series of Openstack processes and helps facilitate a web application. The web application is what is used to interface with ICIC and will forwards requests to the compute nodes that are also running a series of Openstack processes.

The management node is the first thing I needed for IBM Cloud Infrastructure Center to be installed. In my setup, I chose to do something a little more complicated, I decided to provision a KVM guest in my current environment and dedicate that KVM guest as my ICIC management node. I then planned to use my KVM guest's host and my other two remaining KVM hosts as ICIC compute nodes. The ICIC documentation refers to this configuration setup as the "all-in-one environment". It's not something they recommend doing because it adds a layer of complexity on top of an already complicated environment, but if I wanted to use all three of my current KVM hosts, this was my only solution. Alternatively, I could have chosen to not use my future ICIC management node's KVM host as a ICIC compute node and just settled for only running with two ICIC compute nodes. Or I could have done some searching and used a separate new LPAR as a KVM host, which would then serve as my third ICIC compute node. Both are fine options, at the time I wanted to continue using my existing KVM environment and see if I could get the "all-in-one" environment working plus I thought it would be relatively easy (Emphasis on thought).

Once I provisioned my future ICIC management node, the next thing I needed to do was configure the KVM guest to meet all the prerequisites for ICIC. At the time of writing this document, version 1.1.5 was the latest available version of ICIC and the tarball containing the installation package was downloaded from the [IBM Shopz store](#) located under the "Linux on z-Standalone products and fixes" package type. The online documentation for ICIC is fairly extensive and does a pretty good job at highlighting all the necessary requirements needed for your management and compute nodes. I highly recommend using their documentation as a reference when going through this installation process, as it proved to be very helpful for me when I was first setting up this environment. When I set up my management node, these are some the main takeaways I would keep in mind:

- Their documentation states the ICIC management node only supports RHEL 8.2 and RHEL 8.4. I ended up running with RHEL 8.4 on my management node. In later talks with the ICIC development team, which I touch on a little later, I probably could have gotten away with running RHEL 8.6 but I haven't tried it yet.
- Their documentation dumps an exorbitant number of packages required for the installation. I just updated all the latest packages on my system and went ahead with the installation. Yes, the installation did yell at me a couple times for missing a few packages, which I had to go back and install, in order to get the installation to finally complete. However, until they update their documentation to list the RPMs required, instead of doing an RPM dump of their entire environment. I don't know of a better way to go about this process, it just isn't feasible to go through each RPM they have listed and validate that every single one has been installed. In addition, given the OS release you are running with, the version of the RPM they have listed might not even be compatible with your system and/or break future RPMs that you need to complete the installation.
- After installing ICIC and setting up your management node, the installation will give you access to the Openstack CLI by downloading a binary to your `/usr/bin` directory. This is an extremely useful tool that can aid you when debugging things and allow for you to modify your ICIC configuration from the terminal. Openstack's online doc does a good job at outlining all the CLI commands and explaining their purpose. Now if your management node happens to be using PROXY environment variables (like mine), you are going to want to tell the management node to not use your proxy server when invoking the Openstack CLI. This is because the ICIC instance the CLI targets is running locally on your manager node. To do that you must set the `no_proxy` environment variable, mine looks like this:

`no_proxy=localhost,127.0.0.1,.fpet.pokprv.stglabs.ibm.com,10.0.0.0/8`
For completeness, I added in the subnet for anything running on a 10. IP address, because everything I care about in my environment uses a 10. IP address. I also added the domain name that every VM in my environment uses.

- The management node needs to have network connectivity to all the compute nodes you plan on using, specifically passwordless SSH access. Normally this would be a non-issue but in my current KVM environment by default, all my KVM guests that get provisioned are connected over a MacVtap network connection on every KVM host. Due to the nature of MacVtap, any KVM guest running on a MacVtap connection cannot communicate over the network to its KVM host, which in my case would be the ICIC compute node. In order to circumvent this problem, I needed to create a KVM kernel bridge network on my KVM host and attach that bridge as a network interface to my KVM guest which is representing my ICIC management node. This would allow for my ICIC management node to properly communicate with its KVM host and allow for a successful passwordless SSH connection. (I simply added my management node's public SSH key to all my KVM hosts in order to establish a passwordless SSH connection, `ssh-copy-id -i /path/to/my/public_key.pub computeuser@0.0.0.0`)

After feeling as though my management node met the prerequisites as listed in the ICIC documentation, I unzipped the tarball for ICIC version 1.1.5 that I had previously downloaded. I went into the install directory and issued the following command, `sudo ./install -k -i -c`, the `-k` option indicated that I wanted to start the installation, the `-i` option forced SELinux checking to be skipped because I had SELinux turned off on my VM, and the `-c` option indicated I wanted firewall configuration to be performed on my installation. I stumbled through the installation for a little bit because there were some packages I was missing, but after resolving those dependency issues I could see all the Openstack services running on my VM, and the ICIC UI was now reachable. Hooray! I also wanted to verify that the Openstack CLI was functioning properly. From the newly installed `/opt/ibm/icic` directory, I copied my `icircrc` file to my user's home directory. I then issued the source command on the `icircrc` file, `source icircrc`, this then prompted me for the username and password of my ICIC instance (By default they are the root credentials of your management node). Finally, I was able to issue Openstack commands successfully and verify the CLI was working as intended.

```
[lozcoc@cic4mgt icic]$ openstack user list
```

ID	Name
0688b01e6439ca32d698d20789d52169126fb41fb1a4ddafcebb97d854e836c9	root

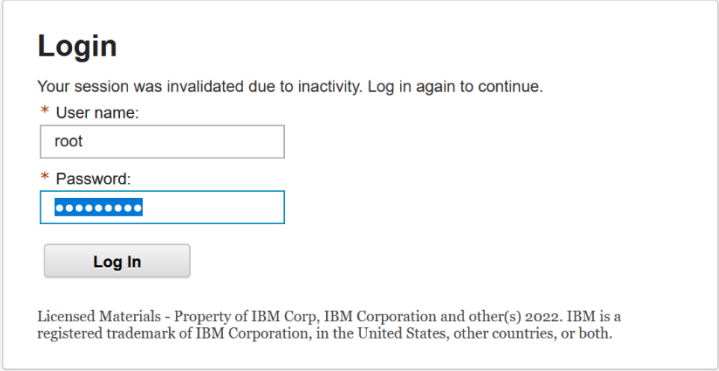
Now that my ICIC management node had the necessary Openstack processes running and I could access the GUI, it was time for me to prepare my ICIC compute nodes to be added to my installation. Keep in mind, I'm performing an ICIC "all-in-one-environment" installation, so this would require me adding my three original KVM hosts in the diagram above as, ICIC compute nodes. Essentially, this "all-in-one environment" allows for my existing KVM infrastructure to continue operating as normal and adds the ability for ICIC to use this existing infrastructure to also provision KVM guests. Before adding my existing KVM hosts as ICIC compute

nodes, I once again need to make sure they meet all the prerequisites for ICIC. The main takeaways I had when preparing my compute nodes were the following:

- Once again, their documentation states that ICIC compute nodes only support RHEL 8.2 and RHEL 8.4. The KVM hosts in my environment were running RHEL 8.6 and I did not want to downgrade my version of RHEL. After speaking with the ICIC team, they said I should not have a problem with RHEL 8.6 on my compute nodes, as long as I installed the following patch on each compute node after they had been successfully added to my ICIC management node (Patch should be included in ICIC version 1.1.6): [Patch for ICIC 1.1.5 \(RHEL 8.6\)](#)
- 80GB of disk is said to be required, but ICIC lets you modify the guest disk size to be anything you desire. I would just make sure you have enough backing storage to support the VMs you plan on provisioning. In my case, because I wanted to use my 11TB GPFS share to back all my ICIC virtual machines, I needed to create a symbolic link from `/var/lib/libvirt/images/nova/instances` to my GPFS share (petKVM). After you have added the compute node to the ICIC manager, the images for your virtual machines will be stored in the directory above. This is good to know if you plan on offloading that storage to an external data store of some kind.
- Once again, their doc dumps a ridiculous number of RPMs required for the compute node installation. I just made sure to have the libvirt, Qemu and python3 packages installed before I went ahead and added my compute nodes. The installation did break on me a couple times, and I did have to go back and install the packages I was missing but this time it was a little more painless. The documentation needs to be better at highlighting the RPMs truly needed for an ICIC compute node and stray away from outputting every RPM installed on a working machine.
- You need to have 2 Open Systems Adapter (OSA) Cards attached as network interfaces to the KVM host. Let's call them OSA1 and OSA2. The OSA1 network interface must have an IP address, the management node will use this IP to communicate with the compute node. Make sure you can ping your management node from OSA1's IP interface. The OSA2 network interface will be responsible for providing network connectivity to your ICIC virtual machines. OSA2 must have connectivity to the subnet you plan on using for your future ICIC VMs. It is also imperative that the OSA2 device has its `bridge_role` set to primary. The attributes on your OSA2 card should look something like this:

```
-----
card_type           : OSD_25GIG
cdev0               : 0.0.2a00
cdev1               : 0.0.2a01
cdev2               : 0.0.2a02
chpid               : 8C
online              : 1
portname            : no portname required
portno              : 0
state               : UP (LAN ONLINE)
priority_queueing   : disabled
buffer_count        : 64
layer2              : 1
isolation           : none
bridge_role         : primary
bridge_state        : active
bridge_hostnotify   : 0
bridge_reflect_promisc : none
switch_attrs        : unknown
vnicc/flooding      : n/a (BridgePort)
vnicc/learning       : n/a (BridgePort)
vnicc/learning_timeout : n/a (BridgePort)
vnicc/mcast_flooding : n/a (BridgePort)
vnicc/rx_bcast      : n/a (BridgePort)
vnicc/takeover_learning : n/a (BridgePort)
vnicc/takeover_setvmac : n/a (BridgePort)
```

At this point, I'm ready to access the ICIC web interface and attempt to add my compute nodes. The default login information is the root credentials of your management node:



Login

Your session was invalidated due to inactivity. Log in again to continue.

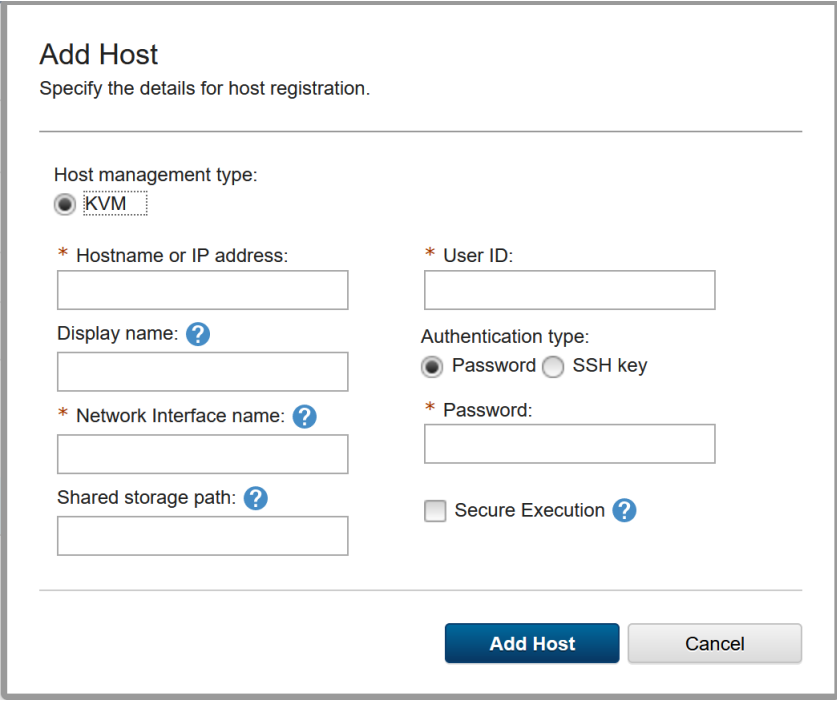
* User name:

* Password:

Log In

Licensed Materials - Property of IBM Corp, IBM Corporation and other(s) 2022. IBM is a registered trademark of IBM Corporation, in the United States, other countries, or both.

After successfully logging in, navigating to the host screen, and selecting the 'add host' button. I can begin filling out the information for my compute node and attempt to add it as a viable ICIC KVM host.



Add Host

Specify the details for host registration.

Host management type:
☒ KVM

* Hostname or IP address:

* User ID:

Display name: ?

Authentication type:
☒ Password ☐ SSH key

* Network Interface name: ?

* Password:


Shared storage path: ?







☐ Secure Execution ?

Add Host Cancel

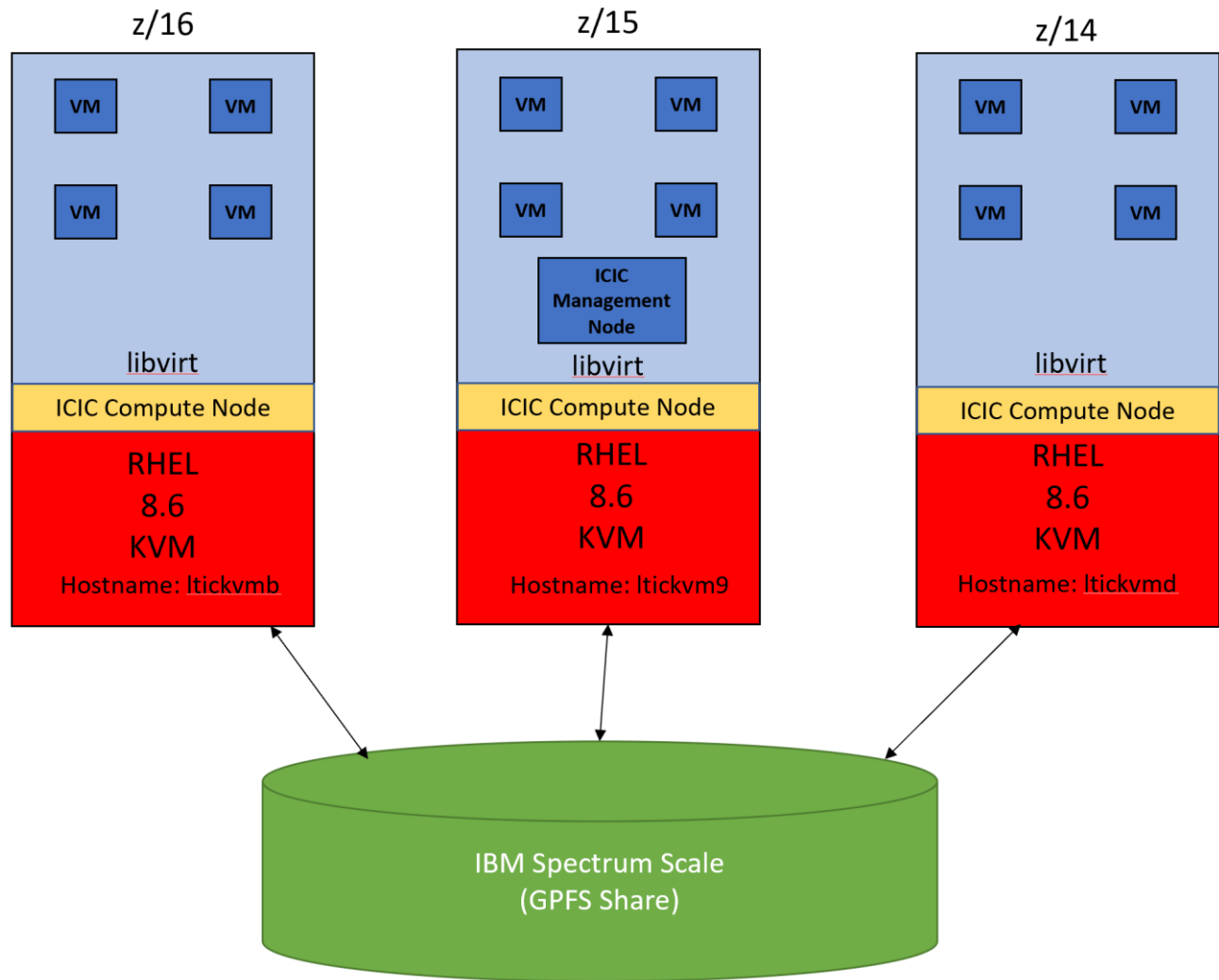
I ended up doing this process three times, once for each of my compute nodes that I wanted to add to ICIC. Each time, I entered the IP address of my compute node that my management node could successfully ping. I entered in a display name that accurately represented my compute node as well as the necessary authentication credentials (User ID & Password). It is important that the credentials entered for the compute node have passwordless SSH access from the previously configured ICIC management node. Lastly, I passed in the network interface running on my compute node that will provide network access to my newly provisioned virtual machines. The network interface chosen must have connectivity to the subnet you later define to ICIC, I will go into more detail about ICIC subnets in part two. It is worth noting that when ICIC attempts to add your KVM hosts as ICIC compute nodes, ICIC on KVM will configure Open vSwitch on each compute node and incorporate the network interface provided in the panel above in that Open vSwitch configuration. A little background, Open vSwitch is an open-source virtual switch software that can manage network traffic between a virtual machine and the physical host network. ICIC uses this software on each compute node to help manage network access for all its running virtual machines.

Finally, once the host information has been filled out above, ICIC will go through some processing and hopefully add your ICIC compute node without error. In my case, when all my compute nodes had been added successfully, the panel looked something like this:

 No filter applied

Name ▲	Virtual Machines *	Processors	Memory (GB)	Health
 LITIKVMB	0	0 Used (120 Total)	0 Used (503.326 Total)	 OK
 LTICKVM9	7	25 Used (120 Total)	100 Used (503.33 Total)	 OK
 LTICKVMD	0	0 Used (120 Total)	0 Used (503.342 Total)	 OK

Now that I have successfully installed ICIC, I'm ready to start configuring my KVM ICIC environment to install OCP. My current environment has now changed and can be accurately represented by the diagram below:



Part 2 (Installing OCP with ICIC)

Now that I have a prepared and installed a functioning ICIC environment, I can now use my ICIC instance to provision the infrastructure necessary to run Red Hat's OpenShift Container Platform (OCP). Now there are a few ways to install OpenShift, none are trivial, but the ICIC team has provided some useful automation in a public GitHub repository that can help make this installation process a little more painless. Keep in mind at the time of writing this report, their automation is in the earlier stages of development and is still undergoing improvements. As someone who just used the automation, I feel it is very much a "happy path" tool, meaning that if you use the automation in its most basic form and don't try to drive some of the more expanded functionality, it works great. However, if you want to use that expanded functionality (like me) expect to tailor some of the automation to meet your environment's specifications and iterate on that process to achieve what you want.

The automation provided by ICIC is written in the form of an Ansible playbook. An Ansible playbook is a series of scripting files written in YAML (yet another markup language) that accomplish a series of automatable tasks on a target machine. When I started this installation effort, I was already very familiar with Ansible and used it on many occasions in my job role on previous projects, this prior knowledge was a proponent for why I gravitated towards ICIC's automation. I'm going to assume anyone who attempts to use

ICIC's Ansible UPI automation to install OCP has prior experience with Ansible playbooks. As a result, I will not be covering how Ansible itself works in extreme detail as that is outside the scope of this report.

The first step I took after examining the ICIC Ansible automation in GitHub was create a fork of the entire repository so that I can make updates without impacting the ICIC team's ongoing development. Once the repository had been forked, I began looking over the README and started to compile a list of prerequisites my Ansible controller (the machine responsible for running the Ansible) would need to execute the playbooks successfully. The README in their repository has a lot of good information that I would highly recommend absorbing before going ahead with this installation. I decided to provision a KVM guest running RHEL 8.6 on one of my KVM hosts, this guest would serve as my Ansible controller. Next, I installed all the packages the README stated I needed (**Note:** Be sure to enable the ansible-2.8 repository listed in the ICIC README before installing the Ansible package. If you don't, by default your machine may install an incompatible version of Ansible that will cause you **A LOT** of problems later, yes this did happen to me):

- Python3
- Ansible
- jq
- wget
- git
- tar
- gzip
- firewalld

I then used a symbolic link to point my python binary file to the python version 3.6.8 binary file and prepared to install the necessary python packages using pip. This is where I had to stray away from the ICIC README documentation and do things a little different. Their README recommends using the python package manger, `pip`, with root privileges, `sudo`, to install and upgrade the required python packages. This is not recommended because doing so will mess with the system's python environment which could easily result in chaos. Python package version control and management can get out of hand very quickly, so it is important to limit the scope of your Python environment to either your user's home directory or a Python virtual environment. In my case, I chose to use my user's home directory by adding the `--user` flag on my pip command. I also prefixed all my pip commands with `python -m` so I could explicitly tell the system witch pip binary I wanted to use for the installation. In addition, I modified the `PATH` environment variable on my system such that the following bin folder was first in my concatenation, `/home/<user>/local/bin`. This ensured I did not use my system's `pip` binary file. At this point, this is what my python environment looks like:

```
[lozcoc@ocpControllerKVM ~]$ env | grep PATH
PATH=/home/lozcoc/.local/bin:/home/lozcoc/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
[lozcoc@ocpControllerKVM ~]$ python --version
Python 3.6.8
[lozcoc@ocpControllerKVM ~]$ python -m pip --version
pip 21.3.1 from /home/lozcoc/.local/lib/python3.6/site-packages/pip (python 3.6)
[lozcoc@ocpControllerKVM ~]$ |
```

After being satisfied with my python environment, I upgraded `pip` with the following command, `python -m pip install --user --upgrade pip`. After upgrading my local `pip` package, ICIC documents installing another list of system packages:

- `redhat-rpm-config`
- `gcc`
- `libffi-devel`
- `python3-devel`
- `openssl-devel`
- `cargo`

I'm not certain if I could have installed these system packages along with the previous list above, at this point I was just following their README and did not feel adventurous enough to try different things. Next it was time to install the python packages, these python packages would ultimately give me access to the Openstack CLI which ICIC's Ansible uses to communicate with the management node and setup the infrastructure necessary for OCP. As their README outlines, I created a `requirements.txt` file containing a list of all the python packages I needed. It is important to note that I had to pin certain versions of packages in my text file that were not pinned in ICIC's README. The `requirements.txt` I ended up using, after numerous trial and error attempts, looked like this:

```
[lozcoc@ocpControllerKVM ~]$ cat requirements.txt
# The order of packages is significant, because pip processes them in the order
# of appearance. Changing the order has an impact on the overall integration
# process, which may cause wedges in the gate later.
pbr!=2.1.0,>=2.0.0 # Apache-2.0
cliff>=3.5.0 # Apache-2.0
iso8601>=0.1.11 # MIT
openstacksdk==0.61 # Apache-2.0
osc-lib>=2.3.0 # Apache-2.0
oslo.i18n>=3.15.3 # Apache-2.0
oslo.utils>=3.33.0 # Apache-2.0
python-keystoneclient>=3.22.0 # Apache-2.0
python-novaclient>=17.0.0 # Apache-2.0
python-cinderclient>=3.3.0 # Apache-2.0
stevedore>=2.0.1 # Apache-2.0
netaddr>=0.8.0
python-openstackclient>=5.5.0
```

Using the requirements file above, I installed the python packages with the following command,

```
python -m pip install --user -r requirements.txt.
```

Once all the python packages have been successfully installed into my environment, I now have access to the Openstack CLI. However, the Openstack CLI is only useful if it can communicate with the Openstack services running on my ICIC manager node. The means I needed to transfer a copy of my `icircrc` file from my ICIC management node, that I created in part 1 of this document, to my OCP controller. After transferring over my `icircrc` file, I once again issued the source command, `source icircrc`, which then prompted me for my username and password of my ICIC management node. At this point I removed all proxy environment variables from my OCP controller's shell environment and exported the same `no_proxy` environment variable I had set in part 1 on my ICIC management node. Keep in mind, although the ICIC Ansible playbooks only target your local Ansible controller, passwordless SSH authentication must be

configured to your ICIC management node from the Ansible controller. This allows the Ansible playbook running on your controller, to issue Openstack CLI commands successfully against the ICIC management node:

```
[lozcoc@ocpControllerKVM ~]$ openstack user list
```

```
+-----+-----+
| ID                                           | Name |
+-----+-----+
| 0688b01e6439ca32d698d20789d52169126fb41fb1a4ddafcebb97d854e836c9 | root |
+-----+-----+
[lozcoc@ocpControllerKVM ~]$
```

Now that my OCP controller's environment had been setup, I turned my focus to configuring ICIC for the OCP Ansible installation. In my case all the ICIC configuration that needed to be done was network related, as that is where most of my problems existed during this installation effort. From the ICIC infrastructure UI, I needed to add a network and subnet that would serve out IP addresses to the newly provisioned machines supporting the infrastructure for my OCP cluster. It is VERY important that the network VLAN and subnet IP address range you specify is accessible by the network interface you entered when adding your ICIC compute node in part 1. If not, none of the VMs provisioned by the ICIC Ansible automation will have network connectivity and you will experience errors. It is also imperative that the network type you specify is of the type **DHCP**, this is a requirement when running the ICIC Ansible automation on KVM. This is something I discovered later in my installation process and at first my VMs were not being assigned IP addresses. ICIC will take the first IP address in the IP range you specify and use it for a DHCP server, the DHCP server will be responsible for assigning IP addresses to newly created VMs. At this point, I would HIGHLY recommend double checking that you do not have a DHCP server already running in your environment, more specifically on the subnet you want to use. I was unaware of a DHCP server already running in my environment, this resulted in messy race conditions on the network and a lot of confusion. When all was said and done, my network configuration in ICIC looked like this:

Edit network properties as needed. Changes apply to only future

* Name:

* Type:

* Virtualization type:

* VLAN ID:

☐ Shared across projects

Network type

IP address type: DHCP

Subnet

+ Add Subnet

Name
VLAN1287

Total: 1 Selected: 1

Edit Subnet

* Name

* Subnet mask:

* Gateway:

Primary DNS:

Secondary DNS:

* Starting IP address:

* Ending IP address:

[Add additional ranges](#)

Edit

Cancel

After my network was defined to ICIC, the next thing I needed to do was assign the DHCP agent running across all my compute nodes to this network definition. By default, ICIC only has one DHCP process running for a given network, in my experience I found this insufficient and needed to add my VLAN1287 network to all the DHCP processes running in my ICIC environment. Unfortunately, this is something that cannot be done via the ICIC UI, and I would need to use the Openstack CLI:

1. First, I need to see if the network I care about, VLAN1287, currently has any DHCP agent(s) associated. In order to do that, I need to grab the UUID of my network and list its DHCP agents:
 - a. `openstack network list` (Capture the UUID for my VLAN1287 network)

- b. `openstack network agent list --network <uuid>` (List the DHCP agents running for network VLAN1287)

```
[lozcoc@ocpControllerKVM ~]$ openstack network list
+-----+-----+-----+
| ID                               | Name   | Subnets |
+-----+-----+-----+
| 09ac62e8-c2bc-48d8-8a42-14ff2fe17ddb | VLAN1287 | 22c2cf48-7412-4d28-88ec-6d7ef7143626 |
| f4e13a74-4e76-4ad6-a5e7-e38c702b6060 | VLAN1284 | 51688967-9b2d-46af-9f9e-dd13b9fcb988 |
+-----+-----+-----+
[lozcoc@ocpControllerKVM ~]$ openstack network agent list --network 09ac62e8-c2bc-48d8-8a42-14ff2fe17ddb
```

2. I then grabbed the UUIDs of all the DHCP agents running in my environment, in my case I had three, one for each of my ICIC compute nodes (`openstack network agent list | grep DHCP`):

```
[lozcoc@ocpControllerKVM ~]$ openstack network agent list | grep DHCP
+-----+-----+-----+
| 58d10469-1dd5-424a-ba68-a4c29f2cb7af | DHCP agent | ltickvm |
+-----+-----+-----+
| 5c77678a-2c35-4197-a7b5-88571a0c352b | DHCP agent | ltickvm |
+-----+-----+-----+
| a0b2c30c-7aaa-4aa7-9095-3ac3fa93a625 | DHCP agent | ltickvm |
+-----+-----+-----+
```

3. Next, I added my newly defined network, VLAN1287, to the DHCP agents that did not have a VLAN1287 association by referencing the UUID I captured in step 2: (Step 1b revealed that DHCP agent a0b2c30c-7aaa-4aa7-9095-3ac3fa93a625 already had a VLAN1287 association and did not need to be added)

- a. `openstack network agent add network --dhcp 5c77678a-2c35-4197-a7b5-88571a0c352b VLAN1287`
- b. `openstack network agent add network --dhcp 58d10469-1dd5-424a-ba68-a4c29f2cb7af VLAN1287`

4. Finally, when listing the DHCP agents associated with VLAN1287's UUID a second time, I can now see all my compute nodes are shown: (`openstack network agent list --network <uuid>`)

```
+-----+-----+-----+
| ID                               | Agent Type | Host |
+-----+-----+-----+
| 58d10469-1dd5-424a-ba68-a4c29f2cb7af | DHCP agent | ltickvmb.pok.stglabs.ibm.com |
| 5c77678a-2c35-4197-a7b5-88571a0c352b | DHCP agent | ltickvm9.pok.stglabs.ibm.com |
| a0b2c30c-7aaa-4aa7-9095-3ac3fa93a625 | DHCP agent | ltickvmd.pok.stglabs.ibm.com |
+-----+-----+-----+
```

Now that I had the networking configured and sorted out across my ICIC instance, the last thing I wanted to configure was availability zones (AZ). An availability zone is just a fancy term for defining what ICIC compute nodes you want your ICIC provisioned guests to run on. ICIC's Ansible automation requires an AZ and ICIC's UI does not provide an easy way to create them, so I had to turn back to my old friend the Openstack CLI. In my environment, I have three ICIC compute nodes and I want to define an AZ for each one of them. From the Openstack CLI, I had to do the following:

1. Create three host aggregate groups that I want to assign for my future availability zones. The purpose of host aggregate groups is outside the scope of my knowledge, but I couldn't create an AZ without one. To make things easy, I named my host aggregate group the same as my future AZ.
 - a. `openstack aggregate create z14_AZ`
 - b. `openstack aggregate create z15_AZ`
 - c. `openstack aggregate create z16_AZ`
2. Next, I created all three of my AZs and associated them with the aggregate group that I created:
 - a. `openstack aggregate set -zone z14_AZ z14_AZ`
 - b. `openstack aggregate set -zone z15_AZ z15_AZ`
 - c. `openstack aggregate set -zone z16_AZ z16_AZ`
3. Finally, I listed my aggregate groups, and I could see my AZs had been created
(`openstack aggregate list`):

```
[lozcoc@ocpControllerKVM ~]$ openstack aggregate list
```

ID	Name	Availability Zone
6	z15_AZ	z15_AZ
7	z16_AZ	z16_AZ
5	z14_AZ	z14_AZ
1	Default_Group	Default_Group

Back in the ICIC UI, on the host page, clicking on the host groups tab allowed for me to see the newly created availability zones. Now I could edit each AZ and add ICIC compute nodes to the one I desire. This is what my z15_AZ availability zone looks like (It contains my ICIC compute node LTICKVM9):



Host Group: z15_AZ

Refresh Edit

Information

Name:	z15_AZ
Availability zone:	z15_AZ
Aggregate Instance Extra Specs:	{ "arr_enabled": "False", "availability_zone": "z15_AZ", "dro_enabled": "False", "hapolicy-id": "1", "hapolicy-run_interval": "1", "hapolicy-stabilization": "5", "initialpolicy-id": "1", "runtimepolicy-action": "migrate_vm_advise_only", "runtimepolicy-id": "5", "runtimepolicy-max_parallel": "10", "runtimepolicy-run_interval": "5", "runtimepolicy-stabilization": "2", "runtimepolicy-threshold": "70" }

Capacity

Processors:	Using: 25	21%
Memory:	Using: 102,400 MB	20%

Hosts

No filter applied

Name	Virtual Machines *	Processors	Memory (GB)	Health
LTICKVM9	7	25 Used (120 Total)	100 Used (503.33 Total)	OK

At this phase, my ICIC environment has now been configured and properly prepared for the OCP Ansible automation, before I can start executing the ICIC Ansible playbooks, I must first modify ICIC's `inventory.yml`. ICIC uses this inventory file as a way for users to pass parameters and specify how the OCP installation should be orchestrated. The README in their GitHub repository does a pretty good job outlining the purpose behind all the variables defined in this inventory file. I will mention this again, I STRONGLY recommend going through their README before starting this installation process, it is important to know what type of installation you want to perform and how the environment you are working with fits into that picture. In my case, I wanted to perform an installation that used a local registry to pull down the OCP installation media (did not use the global RedHat mirrors), I did not want to use a bastion server, I did not want to use an HA proxy, and my environment already had a NGINX load balancer and DNS server properly configured, so extra setup was not required in that regard. As a result, my inventory file required all the `localreg` variables to be set pertaining to my local registry, this included the `additional_certs` variable which is needed to authenticate with the local registry running in my environment. Using the ICIC README as reference, setting the inventory variables was straight forward once you have determined what type of installation you want to perform. Here are a couple of the things worth noting when filling out your inventory file:

- The `use_network_subnet` variable takes in the UUID of the subnet you want to use. I acquired mine by issuing the Openstack CLI command, `openstack network list`
- The `os_flavor_*` variables will use exactly what is defined in ICIC. This is not always the clearest and I recommend issuing the CLI command, `openstack flavor list`, to view the flavors available in your ICIC instance

For context purposes, my installation included the following:

- OCP version 4.10.16
- OCP Cluster name, `lzocpfm1`
- My OCP bootstrap, master, and compute nodes were set to be provisioned in my `z15_AZ` (`ltickvm9`)
- I manually set all IP addresses out of the range previously defined in my subnet

- (NOTE: I did not use the first IP address in my subnet definition because ICIC uses that IP address as a DHCP server)
- The disk type I chose was DASD.
- I chose to use my previously defined subnet, VLAN1287
- On the network, my team has a PowerDNS (PDNS) server running which handles domain name resolution for my environment. This is what my PDNS configuration entries looks like:

Name	Type	Status	TTL	Data
*.apps.lzocpfm1	CNAME	Active	3600	lzocpfm1.fpet.pokprv.stglabs.ibm.com.
api-int.lzocpfm1	CNAME	Active	3600	lzocpfm1.fpet.pokprv.stglabs.ibm.com.
api.lzocpfm1	CNAME	Active	3600	lzocpfm1.fpet.pokprv.stglabs.ibm.com.
bootstrp.lzocpfm1	A	Active	3600	10.20.84.40
lzocpfm1	A	Active	3600	10.20.84.44
manager0.lzocpfm1	A	Active	3600	10.20.84.41
manager1.lzocpfm1	A	Active	3600	10.20.84.42
manager2.lzocpfm1	A	Active	3600	10.20.84.43
worker0.lzocpfm1	A	Active	3600	10.20.84.38
worker1.lzocpfm1	A	Active	3600	10.20.84.39

- Previously, I provisioned a RHEL guest via ICIC and installed NGINX on the VM. This machine will serve as the external load balancer for my OCP cluster. This is what my NGINX configuration looks like:

#lzocpfm1 Configuration

```
upstream lzocpfm1_k8s_api_server {
    server bootstrp.lzocpfm1.fpet.pokprv.stglabs.ibm.com:6443;
    server manager0.lzocpfm1.fpet.pokprv.stglabs.ibm.com:6443;
    server manager1.lzocpfm1.fpet.pokprv.stglabs.ibm.com:6443;
    server manager2.lzocpfm1.fpet.pokprv.stglabs.ibm.com:6443;
}

upstream lzocpfm1_machine_config {
    server bootstrp.lzocpfm1.fpet.pokprv.stglabs.ibm.com:22623;
    server manager0.lzocpfm1.fpet.pokprv.stglabs.ibm.com:22623;
    server manager1.lzocpfm1.fpet.pokprv.stglabs.ibm.com:22623;
    server manager2.lzocpfm1.fpet.pokprv.stglabs.ibm.com:22623;
}

upstream lzocpfm1_route_http {
    server worker0.lzocpfm1.fpet.pokprv.stglabs.ibm.com:80;
    server worker1.lzocpfm1.fpet.pokprv.stglabs.ibm.com:80;
}

upstream lzocpfm1_route_https {
    server worker0.lzocpfm1.fpet.pokprv.stglabs.ibm.com:443;
    server worker1.lzocpfm1.fpet.pokprv.stglabs.ibm.com:443;
}

upstream test_route {
    server kz14s15dt.fpet.pokprv.stglabs.ibm.com:9080;
}

server {
    listen 10.20.84.44:6443;
    proxy_pass lzocpfm1_k8s_api_server;
}

server {
    listen 10.20.84.44:22623;
    proxy_pass lzocpfm1_machine_config;
}

server {
    listen 10.20.84.44:80;
    proxy_pass lzocpfm1_route_http;
}

server {
    listen 10.20.84.44:443;
    proxy_pass lzocpfm1_route_https;
}

server {
    listen 10.20.84.44:9080;
    proxy_pass test_route;
}
```

Once satisfied with my inventory file and current environment, I was just about ready to start kicking off the Ansible playbooks. Before doing so, I did make a modification to one of the python scripts used in the Ansible automation responsible for generating the ignition file for the bootstrap node. In the second phase of the ICIC Ansible automation, the OCP bootstrap node, along with the control nodes will be provisioned. In the event errors occur, it is impossible to access the bootstrap node's shell. The public SSH key variable you are asked to set in the inventory file, seems to only be applicable for your OCP compute and control nodes for the username, `core`. So, if you need to access the bootstrap node for debugging purposes, you will need to make a similar modification to the `generate-bootstrap-ignitionshim.py`:

```
19 19 bootstrap_ign_shim = {
20 20     "ignition": {
21 21         "config": {
22 22             "merge": [
23 23                 {
24 24                     "source": image_url,
25 25                     "httpHeaders": [
26 26                         {
27 27                             "name": "X-Auth-Token",
28 28                             "value": token
29 29                         }
30 30                     ]
31 31                 }
32 32             ]
33 33         },
34 34         "version": "3.1.0"
35 35     },
36 36 - }
36 36 + "passwd": {
37 37 +     "users": [
38 38 +         {
39 39 +             "name": "root",
40 40 +             "passwordHash": <Censoring Password Hash for Security Purposes>
41 41 +             "sshAuthorizedKeys": [
42 42 +                 "ssh-ed25519 AAAAC3NzaC11ZDI1NTE5AAAAIB1BQw4IBxyn9W2hF+NDTRW21fq91PT3e0fwnKa2aTQz fpmitaro@us.ibm.com"
43 43 +             ]
44 44 +         }
45 45 +     ]
46 46 + }
47 47 + }
```

The above code changes, add my public SSH key and set a hashed password for the root user of the bootstrap machine. This allows for me to gain access to the bootstrap machine and better debug problems that may occur during bootstrap processing.

Finally, after adjusting the ICIC Ansible automation I was prepared to kick off the playbooks and begin the OCP installation. The ICIC Ansible playbooks are broken up into three phases:

- The preparation phase, `ansible-playbook -i inventory.yaml 01-preparation.yaml`, this playbook is responsible for priming your ICIC environment with the necessary data for the OCP installation. This playbook will download the required packages, gather the required images from the registry you chose, generate the necessary manifests, and create the ignition files which will be passed into your OCP nodes during the later phases.

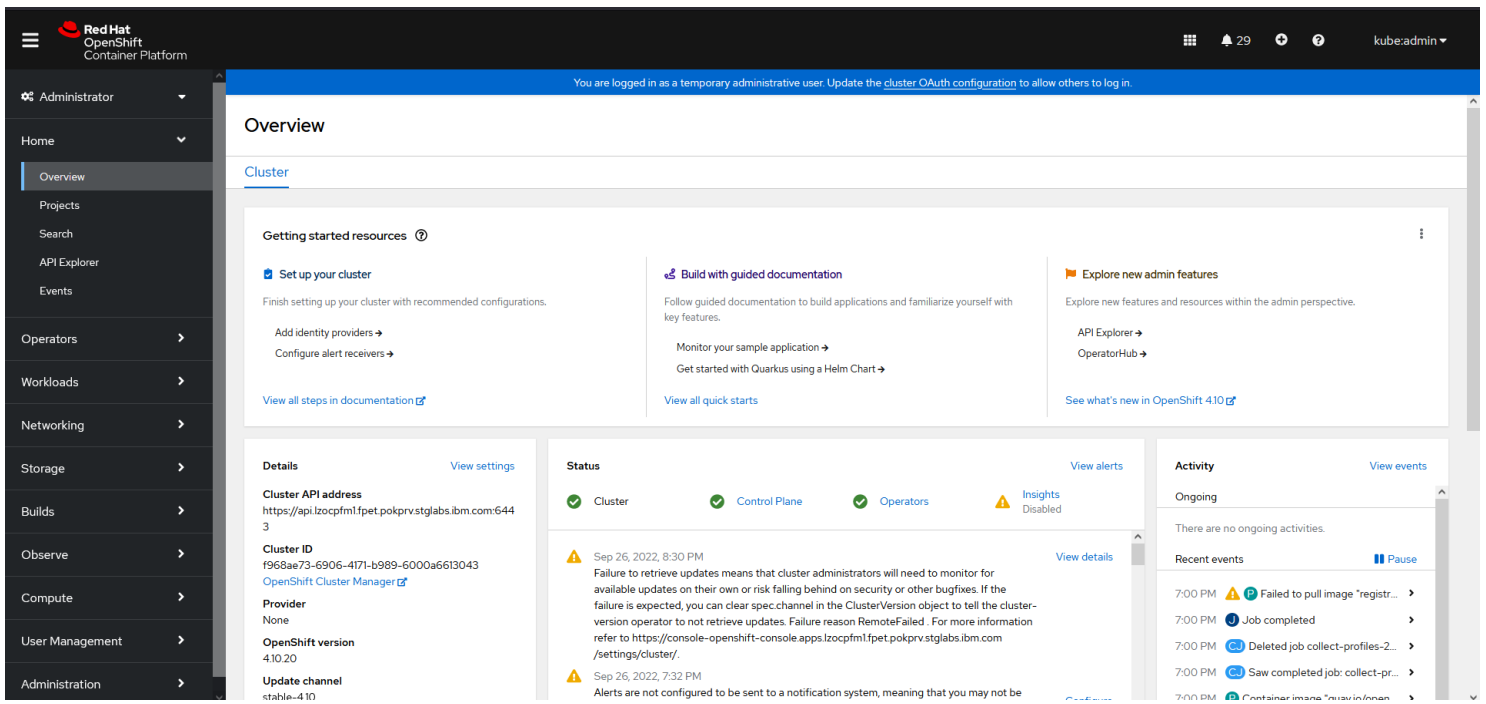
- The create cluster control phase, `ansible-playbook -i inventory.yaml 02-create-cluster-control.yaml`, this is where you will really find out if you set everything up correctly or not. During this phase the bootstrap node and control nodes, also known as the master nodes, are provisioned. At provision time, the bootstrap node and control nodes take in the ignition files previously generated during the preparation phase and begin bootstrap processing. This process takes some time, the automation will wait up to 20min for the bootstrap phase to complete and control nodes to be in sync. In my experience, after the bootstrap and manager nodes have been created, make sure you can ping them on the network. If you can't, something has gone wrong.
 - Note: If you experience a failure of any kind during phases one or two, ICIC recommends running the destroy playbook, `ansible-playbook -i inventory.yaml 04-destroy.yaml`, before rerunning things.
 - You can also monitor the progress of the installation by issuing the following OC command from the Ansible playbook directory:
 - `./oc get co --kubeconfig ./auth/kubeconfig`
- The create cluster compute phase, `ansible-playbook -i inventory.yaml 03-create-cluster-compute.yaml`, the third and final phase. At this point, if you are using a load balancer, you will want to comment out the entries for the bootstrap node. In this phase, your OCP worker nodes are provisioned and passed their generated ignition files from phase one. If everything is done correctly, your worker nodes will sync up with the rest of the cluster and your OCP installation will be complete.
 - Note: If an error is experienced during this phase, the ICIC team recommends running the destroy compute playbook before retrying: `ansible-playbook -i inventory.yaml destroy-computes.yaml`

This phase truly never completed successfully for me. At the time of writing this document, there is a known issue with this phase that the ICIC development is currently in the process of fixing. What happens is the playbook fails and times out when running the following code in the `approve.yaml` file: `Waiting more than {{ approve_nodes_csr }} minutes within poor performance node`. This resulted in me commenting out everything in this playbook, except for the following line of code:

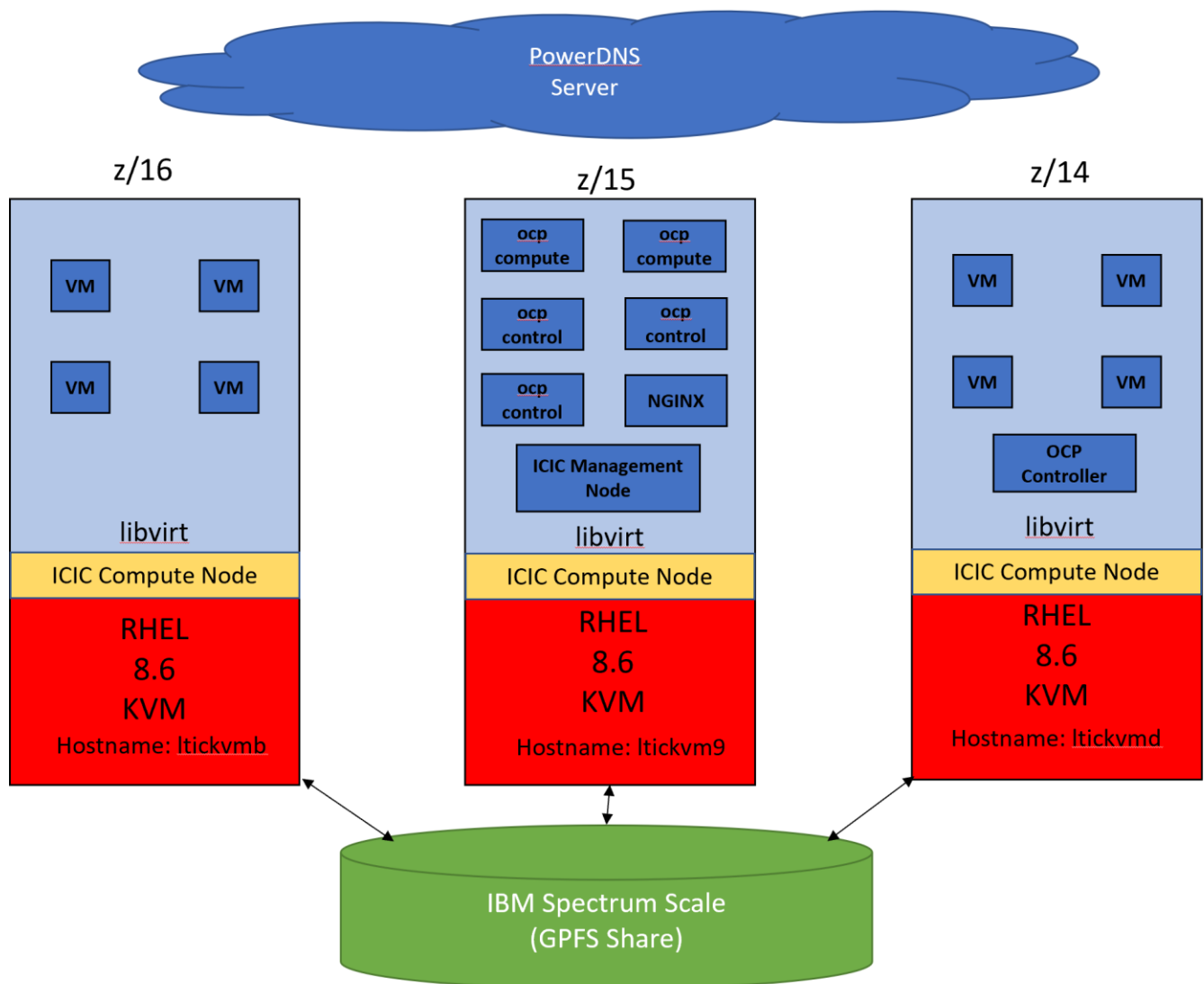
- `Import_playbook: wait-for-install-complete.yaml`

After making the code change documented above in phase three, I would go on to rerun the cluster compute playbook, without destroying my compute nodes. This time, I was able to receive successful output from my console running the Ansible automation. I was presented with the admin username and password for my newly created OCP cluster, as well as the URL it can be reached at from a web browser. Hooray!

Here is what my OCP cluster looks like from my web browser:



This is a final look at my environment topology following a successful OCP installation:



This is a list of the resources, Github repositories, and helpful links that guided me along this journey:

- ICIC Ansible automation, Github:
https://github.com/IBM/z_ansible_collections_samples/tree/main/z_infra_provisioning/cloud_infra_center/ocp_upi
- My personal forked ICIC Ansible automation Github:
https://github.com/frankmit11/z_ansible_collections_samples/tree/f517597b6370e06a3caf1620b99bf473a168102b/z_infra_provisioning/cloud_infra_center/ocp_upi
- ICIC 1.1.5 Documentation: <https://www.ibm.com/docs/en/cic/1.1.5>
- Openstack CLI command reference: <https://docs.openstack.org/python-openstackclient/pike/cli/command-list.html>
- Creating availability zones with the Openstack CLI: <https://www.linuxtechi.com/create-availability-zones-openstack-command-line/>
- OpenShift Mirror Repository: <https://mirror.openshift.com/pub/openshift-v4/s390x/dependencies/rhcos/>
- Red Hat OCP Overview: <https://www.redhat.com/en/technologies/cloud-computing/openshift/container-platform>
- Update logging for Openstack:
<https://docs.openstack.org/nova/pike/admin/manage-logs.html>

Contact Information:

- Email: fpmitaro@us.ibm.com
- Slack: @fpmitaro