Table of Contents

* [Disconnected Install Lab](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_disconnected_install_lab)
* [1. Provision Lab Environment](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_provision_lab_environment)
  + [1.1. Share Public Key with OPENTLC](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_share_public_key_with_opentlc)
  + [1.2. Start Environment After Shut Down](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_start_environment_after_shut_down)
  + [1.3. Test Server Connections](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_test_server_connections)
  + [1.4. Prepare GUID environment variable across all hosts](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_prepare_guid_environment_variable_across_all_hosts)
* [2. Set Up Isolated Host: RPM Repository and Container Registry](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#labexercises)
  + [2.1. Verify Proxied RPM Repository](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_verify_proxied_rpm_repository)
  + [2.2. Install and Populate Image Registry](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_install_and_populate_image_registry)
* [3. Install OpenShift](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_install_openshift)
  + [3.1. Prepare OpenShift Cluster Hosts](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_prepare_openshift_cluster_hosts)
  + [3.2. Configure Ansible Inventory File for Disconnected Install](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_configure_ansible_inventory_file_for_disconnected_install)
  + [3.3. Execute OpenShift Installer](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_execute_openshift_installer)
  + [3.4. Configure Cluster Post-Installation](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_configure_cluster_post_installation)
  + [3.5. Configure Load Balancer](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_configure_load_balancer)
* [4. Deploy CI/CD Tools](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_deploy_ci_cd_tools)
  + [4.1. Deploy Nexus3](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_deploy_nexus3)
  + [4.2. Prepare Repository in Nexus](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_prepare_repository_in_nexus)
  + [4.3. Deploy Gogs](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_deploy_gogs)
  + [4.4. Push **openshift-tasks**Source Code into Gogs](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_push_code_openshift_tasks_code_source_code_into_gogs)
  + [4.5. Run Build from Bastion](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_run_build_from_bastion)
* [5. Reconfigure as Proxied Environment](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_reconfigure_as_proxied_environment)
  + [5.1. Set Up Isolated Node as Proxy](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_set_up_isolated_node_as_proxy)
  + [5.2. Configure RHOCP to use Proxy](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#_configure_rhocp_to_use_proxy)

Disconnected Install Lab

**Overview**

In a disconnected environment, the installer and applications have access to all of the required components of their services, without requiring access to the Internet.

A disconnected install usually occurs in a private network environment dedicated to the cluster. Other networks dedicated to particular purposes access the private network in a highly specified, regulated, and audited manner. Typical networks in a highly secure architecture are likely to include separate networks for general operations, application access, cluster management, development, storage, etc.

**Lab Infrastructure**

The lab infrastructure is a vastly simplified version of this, but it still approximates the installation and operational context of a true disconnected install.

| **Hosts** | | | | | |
| --- | --- | --- | --- | --- | --- |
| **Host** | **Count** | **Network** | **Ports Open to Internet** | **Service** | **Purpose** |
| Bastion | 1 | Public | 22 | * Ansible | * Installation * Ops management command |
| Isolated | 1 | Public | all | * RPM repository (HTTP reverse proxy) * Isolated image registry | * Mimics laptop or file source for installation |
| Load Balancer | 1 | Private | * 80 * 443 | * HAProxy | * OpenShift master API high availability * Application access (disconnected only) |
| Masters | 3 | Private | None | * OpenShift node * OpenShift master API and controllers * **etcd** datastore * **oc** command line tool * Logging and stats collection | * Cluster management * Test application deployment |
| Nodes | 3 | Private | None | * OpenShift node * Application pods/containers * Logging and stats collection | * Deployment targets for pods |
| Infra Nodes | 2 | Private | None | * OpenShift node * OpenShift integrated registry * OpenShift router * Logging and stats collection | * Application access * Internal integrated image registry for application artifacts |
| Support Node | 1 | Private | None | * NFS | * Remote storage for OpenShift persistent volumes |

There are only two networks in this lab.

| **Networks** | | | |
| --- | --- | --- | --- |
| **Network** | **Hosts** | **Services** | **Purpose** |
| Private (ClusterLAN) | Load Balancer, Masters, Nodes, Storage | OpenShift, NFS | Private LAN for all OpenShift and storage traffic |
| Public LAN (Ops, Dev, DMZ) | Bastion, Isolated | Ansible | Installations and ops management |

**Goals**

* Set up an isolated host for the RPM repository and container registry
* Install OpenShift Container Platform
* Deploy CI/CD tools to build and deploy applications
* Reconfigure as a proxied environment

1. Provision Lab Environment

|  |  |
| --- | --- |
|  | If you previously set up an environment for this class, you can skip this section and go directly to the [Set Up Isolated Host: RPM Repository and Container Registry](https://www.opentlc.com/labs/ocp_advanced_deployment/05_1_Disconnected_Install_Solution_Lab.html#labexercises) section. |

In this section, you provision the lab environment to provide access to all of the components required to perform the labs, including access to the {open\_shared\_ocp}. The lab environment is a shared cloud-based environment, so that you can access it over the Internet from anywhere. However, do not expect performance to match a dedicated environment.

1. Go to the [OPENTLC lab portal](https://labs.opentlc.com/) and use your OPENTLC credentials to log in.

|  |  |
| --- | --- |
|  | If you do not remember your password, go to the [OPENTLC Account Management page](https://www.opentlc.com/account/) to reset your password. |

1. Navigate to **Services → Catalogs → All Services → OPENTLC OpenShift Labs**.
2. On the left, select **OpenShift Disconnected HA Lab**.
3. On the right, click **Order**.
4. If you are not in North America, click **Lab Parameters** and select your appropriate region.
5. On the bottom right, click **Submit**.

|  |  |
| --- | --- |
|  | Do not select **App Control → Start** after ordering the lab. Ordering the lab automatically begins the build process. Selecting **Start** may corrupt the lab environment or cause other complications. |

* + After a few minutes, expect to receive an email with instructions on how to connect to the environment.

1. You will receive three e-mails:

**Provisioning started**

Provisioning has started on your Red Hat OPENTLC OTLC-LAB-<OPENTLC Username>-PROD\_OCP\_HA\_DISCONNECTED\_LAB-<GUID> environment.

You will receive more information via email during the build process within the next 30 minutes.

**Provisioning has updated (this means that the VMs have been requested successfully)**

Provisioning is continuing for Red Hat OPENTLC service OTLC-LAB-<OPENTLC Username>-PROD\_OCP\_HA\_DISCONNECTED\_LAB-<GUID>6.

You will receive another email when the environment is available within the next 30 minutes.

**Provisioning completed**

Your environment for OTLC-LAB-<OPENTLC Username>-PROD\_OCP\_HA\_DISCONNECTED\_LAB-e176 has been provisioned.

Your unique identifier, GUID, is: e176

Access to the environment will be granted as soon as the environment deployment is complete.

Please refer to your lab instructions on next steps and how to access your environment.

If your lab/demo instructions indicate connecting with SSH, use your OPENTLC private key, and user name wkulhane-redhat.com.

Here is an example:

$ ssh -i ~/.ssh/your\_private\_key\_name <OPENTLC Username>@bastion.e176.example.opentlc.com

Troubleshooting and Access issues:

If you need help setting up SSH client on your workstation, use http://www.opentlc.com/ssh.html.

If you did not share your public key with OPENTLC (For ssh connections) or if you need to reset your OPENTLC password (For Web Console access), you can use this link: https://account.opentlc.com/account/

NOTICE: Your environment will expire and be deleted in 7 day(s) at 2018-04-26 00:00:00 -0400 and will be idled down after 8 hour(s).

In order to conserve resources, we cannot archive or restore any data in this environment. All data will be lost upon expiration.

1. Wait until you receive the final e-mail before trying to connect to your environment.
2. Note the GUID (a 4 character ID) of your environment.

1.1. Share Public Key with OPENTLC

To access any of your lab systems via SSH, you must use your personal OPENTLC SSO username and public SSH key.

If you have not already done so, you must provide a public SSH key to the OPENTLC authentication system:

1. Go to the [OPENTLC Account Management page](https://www.opentlc.com/account/).
2. Click **Update SSH Key** and log in using your OPENTLC credentials.
3. Paste your public key in that location.

|  |  |
| --- | --- |
|  | For more information on generating SSH keys, see [Setting Up an SSH Key Pair](https://www.opentlc.com/ssh.html). |

1.2. Start Environment After Shut Down

To conserve resources, the lab environment shuts down automatically after eight hours. In this section, you restart the lab environment for this course after it has shut down automatically.

1. Go to the [OPENTLC lab portal](https://labs.opentlc.com/) and use your OPENTLC credentials to log in.
2. Navigate to **Services → My Services** (this should be the screen shown right after logging in).
3. In the list of your services, select your lab environment.
4. Select **App Control → Start** to start your lab environment.
5. Select **Yes** at the **Are you sure?** prompt.
6. On the bottom right, click **Submit**.

After a few minutes, expect to receive an email letting you know that the lab environment has been started.

1.3. Test Server Connections

The **bastion** administration host serves as an access point into the environment and is not part of the OpenShift environment.

1. Connect to your administration host and make sure you can access each of your provisioned hosts:

ssh -i ~/.ssh/yourprivatekey.key opentlc-username@bastion.$GUID.example.opentlc.com

|  |  |
| --- | --- |
|  | * In the labs, "$GUID" is used to represent your 4-character unique identifier. Your GUID is located at the top of your lab provisioning email. * Make sure to replace "$GUID" with your GUID as shown in the example below:   ssh -i ~/.ssh/yourprivatekey.key shacharb-redhat.com@bastion.r2d2.example.opentlc.com   * Red Hat recommends that you create an environment variable so you do not have to type your GUID each time. |

1.4. Prepare GUID environment variable across all hosts

To make running commands specific to your environment easier, run the following ansible command to create a GUID environment variables on all hosts in your cluster that are managed by ansible.

* All GUID env variable to **$HOME/.bashrc** by connecting to the **bastion** host and executing the ad-hoc ansible command:
* ssh -i ~/.ssh/yourprivatekey.key opentlc-username@bastion.$GUID.example.opentlc.com
* sudo -i

ansible localhost,all -m shell -a 'export GUID=`hostname | cut -d"." -f2`; echo "export GUID=$GUID" >> $HOME/.bashrc'

2. Set Up Isolated Host: RPM Repository and Container Registry

In this lab, you must set up the isolated host to serve the proper RPMs and OpenShift container images prior to launching the OpenShift installation. You set up a simple HTTP reverse proxy on the isolated host to serve up RPM packages, and a registry to serve the container images.

This lab covers in detail how the container images must be preloaded into the registry. The RPM packages do not need to be preloaded.

In this lab, you access all of your hosts through the **bastion** host. First, you examine the portion of your environment that is addressable by Ansible.

1. Use SSH to access the **bastion** host and switch to **root**:
2. ssh bastion.$GUID.internal

sudo -i

1. Examine the environment available to you via Ansible:

ansible all -m ping

**Sample Output**

master2.GUID.internal | SUCCESS => {

"changed": false,

"failed": false,

"ping": "pong"

}

master1.GUID.internal | SUCCESS => {

"changed": false,

"failed": false,

"ping": "pong"

}

infranode1.GUID.internal | SUCCESS => {

"changed": false,

"failed": false,

"ping": "pong"

}

loadbalancer1.GUID.internal | SUCCESS => {

"changed": false,

"failed": false,

"ping": "pong"

}

master3.GUID.internal | SUCCESS => {

"changed": false,

"failed": false,

"ping": "pong"

}

infranode2.GUID.internal | SUCCESS => {

"changed": false,

"failed": false,

"ping": "pong"

}

support1.GUID.internal | SUCCESS => {

"changed": false,

"failed": false,

"ping": "pong"

}

node1.GUID.internal | SUCCESS => {

"changed": false,

"failed": false,

"ping": "pong"

}

node3.GUID.internal | SUCCESS => {

"changed": false,

"failed": false,

"ping": "pong"

}

node2.GUID.internal | SUCCESS => {

"changed": false,

"failed": false,

"ping": "pong"

}

|  |  |
| --- | --- |
|  | Your **isolated1** node does not appear here—it is not managed by Ansible. |

1. Use SSH to access the isolated host and switch to **root**:
2. ssh isolated1.$GUID.internal

sudo -i

2.1. Verify Proxied RPM Repository

1. Examine the configuration file of the HTTP reverse proxy:

cat /etc/httpd/conf.d/default.conf

**Sample Output**

<VirtualHost \*:\*>

ProxyPreserveHost On

ProxyPass / http://admin.na.shared.opentlc.com/

ProxyPassReverse / http://admin.na.shared.opentlc.com/

ServerName isolated1.$GUID.internal

*# parameterize me for more storage*

CacheRoot "/var/cache/httpd/proxy/"

CacheEnable disk /

CacheDirLevels 2

CacheDirLength 1

CustomLog "/var/log/httpd/cached-requests.log" common env=cache-hit

CustomLog "/var/log/httpd/uncached-requests.log" common env=cache-miss

CustomLog "/var/log/httpd/revalidated-requests.log" common env=cache-revalidate

CustomLog "/var/log/httpd/invalidated-requests.log" common env=cache-invalidate

</VirtualHost>

1. Note the following settings:
   * It proxies the known good repository server
   * It caches non-aggressively (it does not override cache hints from upstream)
   * It caches at **/var/cache/httpd/proxy/**
   * It logs at **/var/log/httpd/**

2.2. Install and Populate Image Registry

1. Install the Docker image registry, **skopeo**, and **jq**:
   * To use EPEL, you need to issue the **yum makecache** command for the repository where metadata is downloaded.
   * rpm -i https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
   * yum makecache

yum install -y docker-distribution skopeo jq

1. Edit **/etc/docker-distribution/registry/config.yml** and replace the **rootdirectory** line:

sed -i 's/^.\*rootdirectory.\*$/ rootdirectory: \/srv\/repohost\/registry/' /etc/docker-distribution/registry/config.yml

1. Add the following lines to enable logging to **journalctl**:
2. cat << EOF >> /etc/docker-distribution/registry/config.yml
3. log:
4. accesslog:
5. disabled: false
6. level: info
7. formatter: text
8. fields:
9. service: registry
10. environment: staging

EOF

1. Create the directory to hold images in the registry and use **systemd** to enable, start, and check the status of the Docker registry:
2. mkdir -p /srv/repohost/registry
3. systemctl enable docker-distribution
4. systemctl start docker-distribution

systemctl status docker-distribution

2.2.1. Copy Images from Internet to Isolated Registry

In this section, you copy all of the images from the Red Hat registry to the local Docker registry, and make sure that they are written to the local registry with tags that match the requirements of the current release of RHOCP. This can take quite some time. These commands run concurrently in their own loops.

1. Run each of the following four steps in separate shells on the **isolated1** node:
   1. Make sure the images already tagged as **v3.9.14** keep **v3.9.14** tags:
   2. RHT\_TAG=v3.9.14
   3. LOCAL\_TAG=v3.9.14
   4. IMAGES\_SAME\_PATCH="ose-ansible ose-cluster-capacity ose-deployer ose-docker-builder ose-docker-registry ose-egress-http-proxy ose-egress-router ose-haproxy-router ose-pod ose-sti-builder ose container-engine efs-provisioner node openvswitch oauth-proxy logging-auth-proxy logging-curator logging-elasticsearch logging-fluentd logging-kibana metrics-cassandra metrics-hawkular-metrics metrics-heapster oauth-proxy ose ose-service-catalog prometheus-alert-buffer prometheus-alertmanager prometheus registry-console ose-web-console ose-template-service-broker ose-ansible-service-broker logging-deployer metrics-deployer ose-service-catalog mediawiki-apb postgresql-apb mariadb-apb mysql-apb"
   5. time for image in ${IMAGES\_SAME\_PATCH}
   6. do
   7. latest\_version=`skopeo inspect --tls-verify=false docker://registry.access.redhat.com/openshift3/$image | jq ".RepoTags | map(select(startswith((\"${RHT\_TAG}\")))) |.[] "| sort -V | tail -1 | tr -d '"'`
   8. if [[ "$latest\_version" == "" ]]; then latest\_version='latest';fi
   9. echo "Copying image: $image version: $latest\_version"
   10. skopeo copy --dest-tls-verify=false docker://registry.access.redhat.com/openshift3/${image}:${latest\_version} docker://localhost:5000/openshift3/${image}:${LOCAL\_TAG}
   11. echo "Copied image: $image version: $latest\_version"

done

* 1. Copy images with **latest** tags to **v3.9.14** tags:
  2. *# 7 minutes*
  3. RHT\_TAG='latest'
  4. LOCAL\_TAG='v3.9.14'
  5. IMAGES\_LATEST\_TO\_PATCH="ose-recycler prometheus-node-exporter"
  6. time for image in ${IMAGES\_LATEST\_TO\_PATCH}
  7. do
  8. latest\_version=`skopeo inspect --tls-verify=false docker://registry.access.redhat.com/openshift3/$image | jq ".RepoTags | map(select(startswith((\"${RHT\_TAG}\")))) |.[] "| sort -V | tail -1 | tr -d '"'`
  9. if [[ "$latest\_version" == "" ]]; then latest\_version='latest';fi
  10. echo "Copying image: $image version: $latest\_version"
  11. skopeo copy --dest-tls-verify=false docker://registry.access.redhat.com/openshift3/${image}:${latest\_version} docker://localhost:5000/openshift3/${image}:${LOCAL\_TAG} &

done

* 1. Copy images with **v3.9** tags to **latest** tags:
  2. RHT\_TAG='v3.9'
  3. LOCAL\_TAG='latest'
  4. *# Latest tags point to older releases. Need to use version-specific tag::*
  5. IMAGES\_MAJOR\_LATEST="jenkins-2-rhel7 jenkins-slave-base-rhel7 jenkins-slave-maven-rhel7 jenkins-slave-nodejs-rhel7"
  6. time for image in ${IMAGES\_MAJOR\_LATEST}
  7. do
  8. latest\_version=`skopeo inspect --tls-verify=false docker://registry.access.redhat.com/openshift3/$image | jq ".RepoTags | map(select(startswith((\"${RHT\_TAG}\")))) |.[] "| sort -V | tail -1 | tr -d '"'`
  9. if [[ "$latest\_version" == "" ]]; then latest\_version='latest';fi
  10. echo "Copying image: $image version: $latest\_version"
  11. skopeo copy --dest-tls-verify=false docker://registry.access.redhat.com/openshift3/${image}:${latest\_version} docker://localhost:5000/openshift3/${image}:${LOCAL\_TAG}

done

* 1. Copy the images for applications (including **etcd** for OpenShift Ansible Broker):
  2. *# Nexus and Gogs (latest) from docker.io*
  3. for image in sonatype/nexus3 wkulhanek/gogs
  4. do
  5. skopeo copy --dest-tls-verify=false docker://docker.io/${image}:latest docker://localhost:5000/${image}:latest
  6. done
  7. *# from registry.access.redhat.com*
  8. for image in rhel7/etcd rhscl/postgresql-96-rhel7 jboss-eap-7/eap70-openshift
  9. do
  10. skopeo copy --dest-tls-verify=false docker://registry.access.redhat.com/$image:latest docker://localhost:5000/${image}:latest

done

1. After all of the copies from the Internet to your isolated registry are complete, log out of the **isolated1** host and switch back to your **bastion** host.

3. Install OpenShift

3.1. Prepare OpenShift Cluster Hosts

You need to perform these tasks as **root** on **bastion.$GUID.internal**.

1. Make sure that all of your masters and nodes have Docker running:

ansible nodes -mshell -a'systemctl status docker| grep Active'

**Sample Output**

node2.$GUID.internal | SUCCESS | rc=0 >>

Active: active (running) since Fri 2018-03-09 18:18:41 UTC; 3 days ago

master1.$GUID.internal | SUCCESS | rc=0 >>

Active: active (running) since Fri 2018-03-09 18:18:39 UTC; 3 days ago

infranode1.$GUID.internal | SUCCESS | rc=0 >>

Active: active (running) since Fri 2018-03-09 18:18:40 UTC; 3 days ago

node1.$GUID.internal | SUCCESS | rc=0 >>

Active: active (running) since Fri 2018-03-09 18:18:40 UTC; 3 days ago

... output omitted ...

... output omitted ...

... output omitted ...

1. Make sure that the yum repository configurations are pointing to the isolated host on all of the hosts in the deployment:

ansible all -mshell -a'yum repolist -v| grep baseurl'

**Sample Output**

[WARNING]: Consider using yum module rather than running yum

loadbalancer1.$GUID.internal | SUCCESS | rc=0 >>

Repo-baseurl : http://isolated1.$GUID.internal/repos/ocp/3.9.14/rhel-7-fast-datapath-rpms

Repo-baseurl : http://isolated1.$GUID.internal/repos/ocp/3.9.14/rhel-7-server-extras-rpms

Repo-baseurl : http://isolated1.$GUID.internal/repos/ocp/3.9.14/rhel-7-server-optional-rpms

Repo-baseurl : http://isolated1.$GUID.internal/repos/ocp/3.9.14/rhel-7-server-ose-3.9-rpms

Repo-baseurl : http://isolated1.$GUID.internal/repos/ocp/3.9.14/rhel-7-server-rh-common-rpms

Repo-baseurl : http://isolated1.$GUID.internal/repos/ocp/3.9.14/rhel-7-server-rpms

... omitted output ...

... omitted output ...

... omitted output ...

1. From the **bastion** host, connect to the support server to create additional NFS exports to support applications:
2. ssh support1.$GUID.internal
3. sudo -i
4. mkdir -p /srv/nfs/user-vols/pv{1..200}
5. for pvnum in {1..50} ; do
6. echo /srv/nfs/user-vols/pv${pvnum} \*(rw,root\_squash) >> /etc/exports.d/openshift-uservols.exports
7. chown -R nfsnobody.nfsnobody /srv/nfs
8. chmod -R 777 /srv/nfs
9. done
10. systemctl restart nfs-server
11. exit

exit

3.2. Configure Ansible Inventory File for Disconnected Install

1. On the **bastion** host, edit your **/etc/ansible/hosts** Ansible inventory file to address the following requirements:
   * Deploy a three-master HA OpenShift deployment with a load balancer for API and web console load balancing.
   * Make sure the load balancer listens on port 443 (and not 8443).
   * Set a default node selector of **env=app**.
   * Set two nodes to be infra nodes by setting the **env=infra** tag on them.
   * Deploy OpenShift routers with wildcard domains on both of the infra nodes.
   * Deploy an integrated OpenShift image registry on your infra nodes.
   * Deploy Fluentd logging and Hawkular metrics with persistent storage for all nodes in the cluster.
   * Deploy Prometheus-based metrics and alerting.
   * Deploy the template service broker and service catalog.
   * Deploy the OpenShift Ansible Broker and storage for the **etcd** pod.
   * Set up HTTP authorization via an **htaccess** file.
     + See the sample file on your **bastion** host at **/root/htpasswd.openshift**.
   * Set up **ovs-networkpolicy** as your SDN plug-in.
2. Finally, and most importantly, configure RHOCP to access the following:
   * Additional registries
   * Custom image prefixes for:
     + metrics
     + logging
     + service catalog
     + cockpit deployer
     + template service broker
     + OpenShift Ansible Broker
     + Prometheus
     + Prometheus **alertmanager**
     + Prometheus **alertbuffer**
     + Prometheus OAuth proxy

vi /etc/ansible/hosts

|  |  |
| --- | --- |
|  | A fully populated and working solution is available on your **bastion** host in **/var/preserve/hosts**. |

3.3. Execute OpenShift Installer

In this section, you execute the OpenShift installer and monitor the installation process.

1. Run the prerequisites playbook:

ansible-playbook -f 20 /usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml

1. Run the installation playbook:

ansible-playbook -f 20 /usr/share/ansible/openshift-ansible/playbooks/deploy\_cluster.yml

1. On your isolated host, use **tmux** to create two windows and then view the logs of your registry and HTTP reverse proxy as follows:
   1. View HTTP reverse proxy logs showing RPM repository access:
   2. ssh isolated1.$GUID.internal
   3. sudo -i

tail -f /var/log/httpd/\*

* + - You can observe which packages your **httpd-reverse-proxy-cache** was able to cache.
  1. After about 15 minutes, observe the image pulls from the server side by examining the **systemd** journal:

journalctl -f | grep Pull

**Sample Output:**

2018-03-13 18:05:47 +0000 UTC 2018-03-13 18:05:47 +0000 UTC 1 docker-registry-1-fppsj Pod spec.containers{registry} Normal Pulling kubelet, infranode1.$GUID.internal pulling image "isolated1.$GUID.internal:5000/openshift3/ose-docker-registry:v3.9.14"

2018-03-13 18:05:49 +0000 UTC 2018-03-13 18:05:49 +0000 UTC 1 docker-registry-1-fppsj Pod spec.containers{registry} Normal Pulled kubelet, infranode1.$GUID.internal Successfully pulled image "isolated1.$GUID.internal:5000/openshift3/ose-docker-registry:v3.9.14"

1. Observe OpenShift’s event logging:
2. ssh master1.$GUID.internal
3. sudo -i

oc get events --all-namespaces -w

* 1. After about 15 minutes, you can observe how OpenShift pulls the images it needs from the **isolated1.$GUID.internal** host.

1. Allow the Ansible playbooks to run to completion. The entire process takes about 20 minutes. In the end, expect to see output similar to the following:

**Sample Output**

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

infranode1.$GUID.internal : ok=192 changed=60 unreachable=0 failed=0

infranode2.$GUID.internal : ok=192 changed=60 unreachable=0 failed=0

loadbalancer1.$GUID.internal : ok=73 changed=15 unreachable=0 failed=0

localhost : ok=14 changed=0 unreachable=0 failed=0

master1.$GUID.internal : ok=1019 changed=397 unreachable=0 failed=0

master2.$GUID.internal : ok=422 changed=151 unreachable=0 failed=0

master3.$GUID.internal : ok=422 changed=152 unreachable=0 failed=0

node1.$GUID.internal : ok=192 changed=60 unreachable=0 failed=0

node2.$GUID.internal : ok=192 changed=60 unreachable=0 failed=0

node3.$GUID.internal : ok=192 changed=60 unreachable=0 failed=0

support1.$GUID.internal : ok=71 changed=13 unreachable=0 failed=0

INSTALLER STATUS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Initialization : Complete

Health Check : Complete

etcd Install : Complete

NFS Install : Complete

Load balancer Install : Complete

Master Install : Complete

Master Additional Install : Complete

Node Install : Complete

Hosted Install : Complete

Metrics Install : Complete

Logging Install : Complete

Prometheus Install : Complete

Service Catalog Install : Complete

3.4. Configure Cluster Post-Installation

1. From the **bastion** host, connect to the **master1** host and switch to **root**:
2. ssh master1.$GUID.internal

sudo -i

1. Create PVs:
2. mkdir /root/pvs
3. for volume in pv{1..200} ; do
4. cat << EOF > /root/pvs/${volume}
5. {
6. "apiVersion": "v1",
7. "kind": "PersistentVolume",
8. "metadata": {
9. "name": "${volume}"
10. },
11. "spec": {
12. "capacity": {
13. "storage": "10Gi"
14. },
15. "accessModes": [ "ReadWriteOnce" ],
16. "nfs": {
17. "path": "/srv/nfs/user-vols/${volume}",
18. "server": "support1.$GUID.internal"
19. },
20. "persistentVolumeReclaimPolicy": "Recycle"
21. }
22. }
23. EOF
24. echo "Created def file for ${volume}";
25. done;

cat /root/pvs/\* | oc create -f -

1. Disconnect from the **master1** host.

3.5. Configure Load Balancer

In this section, you configure **loadbalancer1** to serve the **\*.apps** wildcard domain.

1. From the **bastion** host, connect to **loadbalancer1**:
2. ssh loadbalancer1.$GUID.internal

sudo -i

1. Open port 80 on **loadbalancer1**:
2. // not firewalld
3. iptables -A OS\_FIREWALL\_ALLOW -p tcp -m tcp --dport 80 -j ACCEPT
4. service iptables save
5. // firewalld
6. firewall-cmd --permanent --add-service=http

firewall-cmd --reload

1. Add entries in **/etc/haproxy/haproxy.cfg** to configure HAProxy to forward to your infra nodes:
2. export GUID=`hostname | awk -F. '{print $2}'`
3. MASTER1=`host master1.$GUID.internal | cut -f4 -d" "`
4. MASTER2=`host master2.$GUID.internal | cut -f4 -d" "`
5. MASTER3=`host master3.$GUID.internal | cut -f4 -d" "`
6. INFRANODE1=`host infranode1.$GUID.internal | cut -f4 -d" "`
7. INFRANODE2=`host infranode2.$GUID.internal | cut -f4 -d" "`
8. cat <<EOF > /etc/haproxy/haproxy.cfg
9. *# Global settings*
10. *#---------------------------------------------------------------------*
11. global
12. maxconn 20000
13. log /dev/log local0 info
14. chroot /var/lib/haproxy
15. pidfile /var/run/haproxy.pid
16. user haproxy
17. group haproxy
18. daemon
19. *# turn on stats unix socket*
20. stats socket /var/lib/haproxy/stats
21. *#---------------------------------------------------------------------*
22. *# common defaults that all the 'listen' and 'backend' sections will*
23. *# use if not designated in their block*
24. *#---------------------------------------------------------------------*
25. defaults
26. mode http
27. log global
28. option httplog
29. option dontlognull
30. *# option http-server-close*
31. option forwardfor except 127.0.0.0/8
32. option redispatch
33. retries 3
34. timeout http-request 10s
35. timeout queue 1m
36. timeout connect 10s
37. timeout client 300s
38. timeout server 300s
39. timeout http-keep-alive 10s
40. timeout check 10s
41. maxconn 20000
42. listen stats :9000
43. mode http
44. stats enable
45. stats uri /
46. frontend atomic-openshift-all-the-things-http
47. bind \*:80
48. mode tcp
49. option tcplog
50. default\_backend atomic-openshift-apps-http
51. frontend atomic-openshift-all-the-things-https
52. bind \*:443
53. mode tcp
54. option tcplog
55. tcp-request inspect-delay 5s
56. tcp-request content accept if { req\_ssl\_hello\_type 1 }
57. acl host\_masters req\_ssl\_sni -i loadbalancer1.${GUID}.example.opentlc.com loadbalancer1.${GUID}.internal
58. use\_backend atomic-openshift-api if host\_masters
59. default\_backend atomic-openshift-apps-https
60. frontend atomic-openshift-all-the-things-http
61. bind \*:80
62. mode tcp
63. option tcplog
64. default\_backend atomic-openshift-apps-http
65. backend atomic-openshift-api
66. balance source
67. mode tcp
68. server master0 $MASTER1:443 check
69. server master1 $MASTER2:443 check
70. server master2 $MASTER3:443 check
71. backend atomic-openshift-apps-https
72. balance source
73. mode tcp
74. server infranode1 $INFRANODE1:443 check
75. server infranode2 $INFRANODE2:443 check
76. backend atomic-openshift-apps-http
77. balance source
78. mode tcp
79. server infranode1 $INFRANODE1:80 check
80. server infranode2 $INFRANODE2:80 check

EOF

1. Restart the HAProxy service and verify that ports 80 and 443 are open:
2. systemctl restart haproxy ; systemctl status haproxy

ss -lntp

1. Disconnect from the **loadbalancer1** host.
2. Test that your configuration worked by going to your OpenShift web console and to**https://hawkular-metrics.apps.GUID.example.opentlc.com/hawkular/metrics**.

4. Deploy CI/CD Tools

In this section, you deploy CI/CD tools to build and deploy applications.

* Gogs is a source code repository manager with a web front end.
* Nexus is an artifact repository that stores build dependencies (among other capabilities).

Because your cluster has no outside connectivity—for example, to GitHub or the Maven repositories—you launch these tools in your disconnected private cluster network, and use them to supply the source code and dependent packages for your application builds.

Use the **bastion** host to execute all of these tasks.

1. As **root**, fetch the **.kube** directory from one of the masters to your **bastion** host:

ansible masters[0] -b -m fetch -a "src=/root/.kube/config dest=/root/.kube/config flat=yes"

* + This enables you to use **oc** commands as **system:admin**.

1. Import all images that may be necessary—such as PostgreSQL, Gogs, and Nexus3—from the **isolated1.$GUID.internal** host to the OpenShift integrated **docker-registry** service that is running pods on your infra nodes:
2. oc import-image docker-registry.default.svc:5000/gogs:latest --from=isolated1.$GUID.internal:5000/wkulhanek/gogs:latest --confirm --insecure=true -n openshift
3. oc import-image docker-registry.default.svc:5000/sonatype/nexus3:latest --from=isolated1.$GUID.internal:5000/sonatype/nexus3:latest --confirm --insecure=true -n openshift
4. oc import-image docker-registry.default.svc:5000/rhscl/postgresql:latest --from=isolated1.$GUID.internal:5000/rhscl/postgresql-96-rhel7:latest --confirm --insecure=true -n openshift
5. oc tag postgresql:latest postgresql:9.6 -n openshift
6. oc import-image docker-registry.default.svc:5000/openshift/jboss-eap70-openshift:latest --from=isolated1.$GUID.internal:5000/jboss-eap-7/eap70-openshift:latest --confirm --insecure=true -n openshift

oc tag jboss-eap70-openshift:latest jboss-eap70-openshift:1.7 -n openshift

* + Note the convenient host names for internal OpenShift services.
  + Also note that some images need tags other than **latest** because the templates are looking for specific tags.

4.1. Deploy Nexus3

1. Deploy Nexus3 in OpenShift in a new **cicd** project:
2. oc new-project cicd
3. echo "apiVersion: v1
4. kind: PersistentVolumeClaim
5. metadata:
6. name: nexus-pvc
7. spec:
8. accessModes:
9. - ReadWriteOnce
10. resources:
11. requests:
12. storage: 10Gi" | oc create -f -
13. oc new-app openshift/nexus3:latest
14. oc rollout pause dc nexus3
15. oc patch dc nexus3 --patch='{ "spec": { "strategy": { "type": "Recreate" }}}'
16. oc set resources dc nexus3 --limits=memory=2Gi --requests=memory=1Gi
17. oc set volume dc/nexus3 --add --overwrite --name=nexus3-volume-1 --mount-path=/nexus-data/ --type persistentVolumeClaim --claim-name=nexus-pvc
18. oc set probe dc/nexus3 --liveness --failure-threshold 3 --initial-delay-seconds 60 -- echo ok
19. oc set probe dc/nexus3 --readiness --failure-threshold 3 --initial-delay-seconds 60 --get-url=http://:8081/repository/maven-public/
20. oc rollout resume dc nexus3

oc expose svc nexus3

4.2. Prepare Repository in Nexus

In this section, you prepare a repository in Nexus for build artifacts using the web console.

Because this is a fully offline environment, Nexus can not act as a proxy repository. Therefore it is necessary to create a hosted Maven2 repository and then copy all of the artifacts necessary for building any given application into the repository. All of the necessary artifacts are already made available for you in a zip file.

1. Log in to Nexus from your laptop at **http://nexus3-cicd.apps.$GUID.example.opentlc.com** with the username **admin** and password **admin123**.
2. In Nexus, create a hosted Maven2 repository called **offline**:
   1. Click the gear icon and select **Repositories → Create Repositories → maven2 (hosted)**.
   2. Name it **offline** and click **Create Repository**
3. Download the dependencies zip file and extract it to your **$HOME/repository** directory:
4. cd $HOME
5. wget http://admin.na.shared.opentlc.com/openshift-tasks-repository.zip

unzip -o openshift-tasks-repository.zip -d $HOME

* 1. This ensures that all of the dependencies are in your Maven repository.

1. Create the following **nexusimport.sh** script in **$HOME/repository**:
2. cd $HOME/repository
3. cat << EOF > ./nexusimport.sh
4. #!/bin/bash
5. *# copy and run this script to the root of the repository directory containing files*
6. *# this script attempts to exclude uploading itself explicitly so the script name is important*
7. *# Get command line params*
8. while getopts ":r:u:p:" opt; do
9. case \$opt in
10. r) REPO\_URL="\$OPTARG"
11. ;;
12. u) USERNAME="\$OPTARG"
13. ;;
14. p) PASSWORD="\$OPTARG"
15. ;;
16. esac
17. done
18. find . -type f \\
19. -not -path './mavenimport\.sh\*' \\
20. -not -path '\*/\.\*' \\
21. -not -path '\*/\^archetype\-catalog\.xml\*' \\
22. -not -path '\*/\^maven\-metadata\-local\*\.xml' \\
23. -not -path '\*/\^maven\-metadata\-deployment\*\.xml' | \\
24. sed "s|^\./||" | \\
25. xargs -t -I '{}' curl -s -S -u "\$USERNAME:\$PASSWORD" -X PUT -T {} \${REPO\_URL}/{} ;

EOF

* 1. This simplifies loading dependencies into Nexus.

1. Make sure the script is executable:

chmod +x $HOME/repository/nexusimport.sh

1. Now run the **nexusimport.sh** script to upload the entire repository into Nexus:
2. cd $HOME/repository

./nexusimport.sh -u admin -p admin123 -r http://$(oc get route nexus3 --template='{{ .spec.host }}' -n cicd)/repository/offline/

* 1. This can take a minute or two.

4.3. Deploy Gogs

1. Create a PostgreSQL database for Gogs:
2. oc project cicd

oc new-app postgresql-persistent --param POSTGRESQL\_DATABASE=gogs --param POSTGRESQL\_USER=gogs --param POSTGRESQL\_PASSWORD=gogs --param VOLUME\_CAPACITY=4Gi -lapp=postgresql\_gogs

1. Make sure to wait until **postgresql-persistent** is running, then create a Gogs template file called **$HOME/gogs.yaml** with the following content:
2. echo 'kind: Template
3. apiVersion: v1
4. metadata:
5. annotations:
6. description: The Gogs git server. Requires a PostgreSQL database.
7. tags: instant-app,gogs,datastore
8. iconClass: icon-github
9. name: gogs
10. objects:
11. - kind: Service
12. apiVersion: v1
13. metadata:
14. annotations:
15. description: The Gogs servers http port
16. labels:
17. app: ${APPLICATION\_NAME}
18. name: ${APPLICATION\_NAME}
19. spec:
20. ports:
21. - name: 3000-tcp
22. port: 3000
23. protocol: TCP
24. targetPort: 3000
25. selector:
26. app: ${APPLICATION\_NAME}
27. deploymentconfig: ${APPLICATION\_NAME}
28. sessionAffinity: None
29. type: ClusterIP
30. - kind: Route
31. apiVersion: v1
32. id: ${APPLICATION\_NAME}-http
33. metadata:
34. annotations:
35. description: Route for applications http service.
36. labels:
37. app: ${APPLICATION\_NAME}
38. name: ${APPLICATION\_NAME}
39. spec:
40. host: ${GOGS\_ROUTE}
41. to:
42. name: ${APPLICATION\_NAME}
43. - kind: DeploymentConfig
44. apiVersion: v1
45. metadata:
46. labels:
47. app: ${APPLICATION\_NAME}
48. name: ${APPLICATION\_NAME}
49. spec:
50. replicas: 1
51. selector:
52. app: ${APPLICATION\_NAME}
53. deploymentconfig: ${APPLICATION\_NAME}
54. strategy:
55. rollingParams:
56. intervalSeconds: 1
57. maxSurge: 25%
58. maxUnavailable: 25%
59. timeoutSeconds: 600
60. updatePeriodSeconds: 1
61. type: Rolling
62. template:
63. metadata:
64. labels:
65. app: ${APPLICATION\_NAME}
66. deploymentconfig: ${APPLICATION\_NAME}
67. spec:
68. containers:
69. - image: \"\"
70. imagePullPolicy: Always
71. name: ${APPLICATION\_NAME}
72. ports:
73. - containerPort: 3000
74. protocol: TCP
75. resources: {}
76. terminationMessagePath: /dev/termination-log
77. volumeMounts:
78. - name: gogs-data
79. mountPath: /data
80. - name: gogs-config
81. mountPath: /opt/gogs/custom/conf
82. readinessProbe:
83. httpGet:
84. path: /
85. port: 3000
86. scheme: HTTP
87. initialDelaySeconds: 3
88. timeoutSeconds: 1
89. periodSeconds: 20
90. successThreshold: 1
91. failureThreshold: 3
92. livenessProbe:
93. httpGet:
94. path: /
95. port: 3000
96. scheme: HTTP
97. initialDelaySeconds: 3
98. timeoutSeconds: 1
99. periodSeconds: 10
100. successThreshold: 1
101. failureThreshold: 3
102. dnsPolicy: ClusterFirst
103. restartPolicy: Always
104. terminationGracePeriodSeconds: 30
105. volumes:
106. - name: gogs-data
107. persistentVolumeClaim:
108. claimName: gogs-data
109. - name: gogs-config
110. configMap:
111. name: gogs-config
112. items:
113. - key: app.ini
114. path: app.ini
115. test: false
116. triggers:
117. - type: ConfigChange
118. - imageChangeParams:
119. automatic: true
120. containerNames:
121. - ${APPLICATION\_NAME}
122. from:
123. kind: ImageStreamTag
124. name: ${GOGS\_IMAGE}
125. namespace: openshift
126. type: ImageChange
127. - kind: PersistentVolumeClaim
128. apiVersion: v1
129. metadata:
130. name: gogs-data
131. spec:
132. accessModes:
133. - ReadWriteOnce
134. resources:
135. requests:
136. storage: ${GOGS\_VOLUME\_CAPACITY}
137. - kind: ConfigMap
138. apiVersion: v1
139. metadata:
140. name: gogs-config
141. data:
142. app.ini: |
143. APP\_NAME = Gogs
144. RUN\_MODE = prod
145. RUN\_USER = gogs
146. [database]
147. DB\_TYPE = postgres
148. HOST = postgresql:5432
149. NAME = ${DATABASE\_NAME}
150. USER = ${DATABASE\_USER}
151. PASSWD = ${DATABASE\_PASSWORD}
152. SSL\_MODE = disable
153. [repository]
154. ROOT = /data/repositories
155. [server]
156. ROOT\_URL=http://${GOGS\_ROUTE}
157. [security]
158. INSTALL\_LOCK = true
159. [mailer]
160. ENABLED = false
161. [service]
162. ENABLE\_CAPTCHA = false
163. REGISTER\_EMAIL\_CONFIRM = false
164. ENABLE\_NOTIFY\_MAIL = false
165. DISABLE\_REGISTRATION = false
166. REQUIRE\_SIGNIN\_VIEW = false
167. [picture]
168. DISABLE\_GRAVATAR = false
169. ENABLE\_FEDERATED\_AVATAR = true
170. [webhook]
171. SKIP\_TLS\_VERIFY = true
172. parameters:
173. - name: APPLICATION\_NAME
174. description: The name for the application.
175. required: true
176. value: gogs
177. - name: GOGS\_ROUTE
178. description: The route for the Gogs Application
179. required: true
180. - name: GOGS\_VOLUME\_CAPACITY
181. description: Volume space available for data, e.g. 512Mi, 2Gi
182. required: true
183. value: 4Gi
184. - name: DATABASE\_USER
185. displayName: Database Username
186. required: true
187. value: gogs
188. - name: DATABASE\_PASSWORD
189. displayName: Database Password
190. required: true
191. value: gogs
192. - name: DATABASE\_NAME
193. displayName: Database Name
194. required: true
195. value: gogs
196. - name: GOGS\_IMAGE
197. displayName: Gogs Image and tag
198. required: true

value: gogs:latest' > $HOME/gogs.yaml

1. Create Gogs from the template:

oc process -f $HOME/gogs.yaml --param GOGS\_ROUTE=gogs-cicd.apps.$GUID.example.opentlc.com|oc create -f -

1. Watch **oc get pods** within the **cicd** project until the deployment finishes.
2. When Gogs is running, navigate to the Gogs home page in your browser.

|  |  |
| --- | --- |
|  | If you are not sure of the URL, you can determine the route using **oc get route -n cicd**. |

1. Register yourself as a new user.
2. Log in to Gogs as the user you just registered as.
3. Once logged in, create an organization named **CICDLabs**.
4. Under the **CICDLabs** organization, create a repository called **openshift-tasks**.

|  |  |
| --- | --- |
|  | Do not change the visibility setting. This repository must be **Public**. |

You have now created an empty source code repository in Gogs that OpenShift has access to. Next, you push code into this repository and build from that code and the dependencies that are in Nexus.

4.4. Push **openshift-tasks** Source Code into Gogs

1. Clone the **openshift-tasks** repository from GitHub and push it into the Gogs repository:
2. cd $HOME
3. git clone https://github.com/wkulhanek/openshift-tasks.git

cd $HOME/openshift-tasks

1. Set up the remote Git repository location in your local Git repository, and push it to Gogs by executing the following:

|  |  |
| --- | --- |
|  | **git** prompts you for your Gogs username and password when you execute the **push** command—make sure to use the username and password that you registered in Gogs. |

1. git remote add gogs http://gogs-cicd.apps.$GUID.example.opentlc.com/CICDLabs/openshift-tasks.git
2. git push -u gogs master
3. **Sample Output**
4. Counting objects: 314, done.
5. Delta compression using up to 2 threads.
6. Compressing objects: 100% (209/209), done.
7. Username for 'http://gogs-cicd.apps.$GUID.example.opentlc.com': wkulhanek
8. Password for 'http://wkulhanek@gogs-cicd.apps.$GUID.example.opentlc.com':
9. Writing objects: 100% (314/314), 3.88 MiB | 305.00 KiB/s, done.
10. Total 314 (delta 69), reused 313 (delta 69)
11. To http:*//gogs-cicd.apps.$GUID.example.opentlc.com/CICDLabs/openshift-tasks.git*
12. \* [new branch] master -> master
13. Branch master set up to track remote branch master from gogs.
    * This concludes the setup portion:
      + The correct image stream for the builder image (EAP 7.0) is in your disconnected environment.
      + The image is available in the disconnected registry.
      + The source code is in the disconnected Gogs repository.
      + All of the Maven build dependencies are in Nexus.

4.5. Run Build from Bastion

Now that all of the prerequisites are set up, it is time to execute a fully disconnected build. You are using the **eap70-basic-s2i**template to create the **openshift-tasks** application.

1. On your **bastion** host as **root**, create the **tasks** project and then the application:

|  |  |
| --- | --- |
|  | You are specifying the Nexus proxy repository as a parameter to the builder image. Every Red Hat xPaaS builder image understands the variable **MAVEN\_MIRROR\_URL**. Furthermore, because the template has a few non-sensible defaults for the branch and context directory, you need to explicitly set the branch to **master** and the context directory to empty. |

1. oc new-project tasks
2. oc new-app eap70-basic-s2i \
3. --param MAVEN\_MIRROR\_URL=http://nexus3.cicd.svc.cluster.local:8081/repository/offline \
4. --param SOURCE\_REPOSITORY\_URL=http://gogs.cicd.svc.cluster.local:3000/CICDLabs/openshift-tasks \
5. --param SOURCE\_REPOSITORY\_REF=master \
6. --param CONTEXT\_DIR= --param APPLICATION\_NAME=tasks
   * If you did everything correctly, this application builds and then runs.
7. Follow the build to make sure it is working properly and that the source code can be cloned and the build dependencies can be pulled from Nexus:

oc logs -f tasks-1-build

**Sample Output**

Pulling image "isolated1.2a77.internal:5000/jboss-eap-7/eap70-openshift@sha256:9b99d822f4cbcb1d133dfbe1494d018a88a314385622b248265fd5b05719a27e" ...

Found pom.xml... attempting to build with 'mvn -e -Popenshift -DskipTests -Dcom.redhat.xpaas.repo.redhatga package --batch-mode -Djava.net.preferIPv4Stack=true '

Using MAVEN\_OPTS '-XX:+UseParallelGC -XX:MinHeapFreeRatio=20 -XX:MaxHeapFreeRatio=40 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90 -XX:MaxMetaspaceSize=100m -XX:+ExitOnOutOfMemoryError'

Using Apache Maven 3.3.9 (Red Hat 3.3.9-2.8)

Maven home: /opt/rh/rh-maven33/root/usr/share/maven

Java version: 1.8.0\_161, vendor: Oracle Corporation

Java home: /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.161-0.b14.el7\_4.x86\_64/jre

Default locale: en\_US, platform encoding: ANSI\_X3.4-1968

OS name: "linux", version: "3.10.0-693.el7.x86\_64", arch: "amd64", family: "unix"

[INFO] Error stacktraces are turned on.

[INFO] Scanning for projects...

[INFO] Downloading: http:*//nexus3.cicd.svc.cluster.local:8081/repository/offline/org/jboss/bom/eap/jboss-javaee-6.0-with-tools/6.4.0.GA/jboss-javaee-6.0-with-tools-6.4.0.GA.pom*

[INFO] Downloaded: http:*//nexus3.cicd.svc.cluster.local:8081/repository/offline/org/jboss/bom/eap/jboss-javaee-6.0-with-tools/6.4.0.GA/jboss-javaee-6.0-with-tools-6.4.0.GA.pom (8 KB at 45.8 KB/sec)*

[INFO] Downloading: http:*//nexus3.cicd.svc.cluster.local:8081/repository/offline/org/jboss/bom/eap/jboss-eap-bom-parent/6.4.0.GA/jboss-eap-bom-parent-6.4.0.GA.pom*

[INFO] Downloaded: http:*//nexus3.cicd.svc.cluster.local:8081/repository/offline/org/jboss/bom/eap/jboss-eap-bom-parent/6.4.0.GA/jboss-eap-bom-parent-6.4.0.GA.pom (7 KB at 252.4 KB/sec)*

[....]

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 16.548 s

[INFO] Finished at: 2018-04-03T20:24:23+00:00

[INFO] Final Memory: 21M/60M

[INFO] ------------------------------------------------------------------------

Copying all war artifacts from /tmp/src/target directory into /opt/eap/standalone/deployments for later deployment...

Copying all ear artifacts from /tmp/src/target directory into /opt/eap/standalone/deployments for later deployment...

Copying all rar artifacts from /tmp/src/target directory into /opt/eap/standalone/deployments for later deployment...

Copying all jar artifacts from /tmp/src/target directory into /opt/eap/standalone/deployments for later deployment...

Copying all war artifacts from /tmp/src/deployments directory into /opt/eap/standalone/deployments for later deployment...

'/tmp/src/deployments/ROOT.war' -> '/opt/eap/standalone/deployments/ROOT.war'

Copying all ear artifacts from /tmp/src/deployments directory into /opt/eap/standalone/deployments for later deployment...

Copying all rar artifacts from /tmp/src/deployments directory into /opt/eap/standalone/deployments for later deployment...

Copying all jar artifacts from /tmp/src/deployments directory into /opt/eap/standalone/deployments for later deployment...

Copying config files from project...

'/tmp/src/configuration/application-roles.properties' -> '/opt/eap/standalone/configuration/application-roles.properties'

'/tmp/src/configuration/application-users.properties' -> '/opt/eap/standalone/configuration/application-users.properties'

Pushing image docker-registry.default.svc:5000/tasks/tasks:latest ...

Pushed 0/7 layers, 1% complete

Pushed 1/7 layers, 22% complete

Pushed 2/7 layers, 50% complete

Pushed 3/7 layers, 58% complete

Pushed 4/7 layers, 82% complete

Pushed 5/7 layers, 87% complete

Pushed 6/7 layers, 91% complete

Pushed 7/7 layers, 100% complete

Push successful

* + When the build finishes, the application deploys.

1. Again, follow the pod logs to make sure the application started successfully (replacing **xxxxx** with your specific pod name):

oc logs -f tasks-1-xxxxx

**Sample Output**

WARNING: Service account has insufficient permissions to view pods in kubernetes (HTTP 403). Clustering might be unavailable. Please refer to the documentation for configuration.

INFO: Configuring JGroups discovery protocol to openshift.KUBE\_PING

Using PicketBox SSL configuration.

Missing SSO\_URL. Unable to properly configure SSO-enabled applications

Access log is disabled, ignoring configuration.

Running jboss-eap-7/eap70-openshift image, version 1.7

-XX:+UseParallelGC -XX:MinHeapFreeRatio=20 -XX:MaxHeapFreeRatio=40 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90 -XX:MaxMetaspaceSize=100m -XX:+ExitOnOutOfMemoryError

=========================================================================

JBoss Bootstrap Environment

JBOSS\_HOME: /opt/eap

[...]

20:25:23,193 INFO [org.jboss.as.server] (ServerService Thread Pool -- 38) WFLYSRV0010: Deployed "activemq-rar.rar" (runtime-name : "activemq-rar.rar")

20:25:23,317 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0060: Http management interface listening on http://127.0.0.1:9990/management

20:25:23,317 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0054: Admin console is not enabled

20:25:23,317 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0025: JBoss EAP 7.0.9.GA (WildFly Core 2.1.20.Final-redhat-1) started in 13863ms - Started 593 of 892 services (515 services are lazy, passive or on-demand)

1. Access the newly deployed application from your web browser at **http://tasks-tasks.apps.$GUID.example.opentlc.com**.

5. Reconfigure as Proxied Environment

Some environments have less strict security policies, and may access the Internet through a proxy. Development environments are often like this.

You have already set up a caching proxy to handle RPM images. Next, you configure that proxy to handle all other outbound communication from the OpenShift cluster. Then you reconfigure RHOCP and Docker to access the Internet through the proxy.

* Set up the isolated node as a web and container image proxy
* Configure RHOCP to use the proxy
* Build and deploy an application from Internet sources

5.1. Set Up Isolated Node as Proxy

1. Update SELinux rules on the isolated node to allow **httpd** on port 8080 for proxying:
2. ssh isolated1.$GUID.internal
3. sudo -i

semanage port -m -t http\_port\_t -p tcp 8080

1. Configure **httpd** to run in proxy mode for local addresses only:
2. cat >> /etc/httpd/conf.d/default.conf << EOF
3. Listen 8080
4. <VirtualHost \*:8080>
5. ProxyRequests On
6. ProxyVia On
7. <Proxy "\*">
8. Require ip 192.168.2.0/24 192.168.1.0/24
9. </Proxy>
10. LogLevel trace5
11. ServerName isolated1.$GUID.internal
12. *# parameterize me for more storage*
13. CacheRoot "/var/cache/httpd/proxy/"
14. CacheEnable disk /
15. CacheDirLevels 2
16. CacheDirLength 1
17. CustomLog "/var/log/httpd/cached-requests.log" common env=cache-hit
18. CustomLog "/var/log/httpd/uncached-requests.log" common env=cache-miss
19. CustomLog "/var/log/httpd/revalidated-requests.log" common env=cache-revalidate
20. CustomLog "/var/log/httpd/invalidated-requests.log" common env=cache-invalidate
21. </VirtualHost>
22. EOF

systemctl restart httpd

5.2. Configure RHOCP to use Proxy

1. Edit your Ansible inventory file to set proxies in RHOCP and Docker:
2. ssh bastion.$GUID.internal
3. sudo -i
4. cat > /tmp/HereFile <<HEREDOC
5. *######################*
6. *# Global Proxy Configuration*
7. *# These options configure HTTP\_PROXY, HTTPS\_PROXY, and NOPROXY environment*
8. *# variables for docker and master services.*
9. openshift\_http\_proxy=http://isolated1.$GUID.internal:8080
10. openshift\_https\_proxy=http://isolated1.$GUID.internal:8080
11. openshift\_no\_proxy=".internal,172.30.0.0/16"
12. *#*
13. *# Most environments do not require a proxy between OpenShift masters, nodes, and*
14. *# etcd hosts. So automatically add those host names to the openshift\_no\_proxy list.*
15. *# If all of your hosts share a common domain you may wish to disable this and*
16. *# specify that domain above.*
17. openshift\_generate\_no\_proxy\_hosts=False
18. *######################*
19. HEREDOC
20. sed -i '/debug\_level/ {
21. r /tmp/HereFile
22. }' /etc/ansible/hosts

rm /tmp/HereFile

1. Next, permit RHOCP and Docker to access image registries that were blocked by the disconnected install:

sed -i 's/\(openshift\_docker\_blocked\_registries=\).\*$/\1nonsense/' /etc/ansible/hosts

|  |  |
| --- | --- |
|  | Setting this value to **nonsense** is required because of a known bug. As implemented, this command only replaces all of the values; it cannot be used to remove the parameters from the Docker configuration. |

1. Skip reconfiguring the load balancer by updating the playbook to add a tag as follows:

|  |  |
| --- | --- |
|  | The **openshift-ansible** playbooks normally overwrite your customer load-balancer configuration. It is possible to skip the load-balancer reconfiguration when running the installer by adding a tag to the appropriate task and then running **ansible-playbook** with **--skip-tags**. |

* 1. In the **/usr/share/ansible/openshift-ansible/playbooks/common/openshift-cluster/roles/openshift\_loadbalancer/tasks/main.yml**file, add the **tags: - configure\_haproxy\_template** lines:
  2. - name: Configure haproxy
  3. template:
  4. src: haproxy.cfg.j2
  5. dest: /etc/haproxy/haproxy.cfg
  6. owner: root
  7. group: root
  8. mode: 0644
  9. notify: restart haproxy
  10. tags:

- configure\_haproxy\_template

1. Before running the playbook to reconfigure the system, examine the proxy settings for RHOCP and Docker and note what is configured:
2. cd /etc/sysconfig
3. grep -i proxy atomic-openshift-master atomic-openshift-master-api atomic-openshift-master-controllers atomic-openshift-node | egrep -v '#'

ansible all -mshell -a'grep -i proxy atomic-openshift-master atomic-openshift-master-api atomic-openshift-master-controllers atomic-openshift-node | egrep -v \#'

1. Also examine the Docker process to see which registries are blocked:

ansible masters[0] -mshell -a 'ps -ef | grep block-registry'

1. Run the **openshift-ansible** installer to set the proxy values:

ansible-playbook -f20 /usr/share/ansible/openshift-ansible/playbooks/deploy\_cluster.yml -vvv --skip-tags=configure\_haproxy\_template

1. Make sure it runs successfully to completion:

**Sample Output:**

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

infranode1.$GUID.internal : ok=192 changed=60 unreachable=0 failed=0

infranode2.$GUID.internal : ok=192 changed=60 unreachable=0 failed=0

loadbalancer1.$GUID.internal : ok=73 changed=15 unreachable=0 failed=0

localhost : ok=14 changed=0 unreachable=0 failed=0

master1.$GUID.internal : ok=1019 changed=397 unreachable=0 failed=0

master2.$GUID.internal : ok=422 changed=151 unreachable=0 failed=0

master3.$GUID.internal : ok=422 changed=152 unreachable=0 failed=0

node1.$GUID.internal : ok=192 changed=60 unreachable=0 failed=0

node2.$GUID.internal : ok=192 changed=60 unreachable=0 failed=0

node3.$GUID.internal : ok=192 changed=60 unreachable=0 failed=0

support1.$GUID.internal : ok=71 changed=13 unreachable=0 failed=0

INSTALLER STATUS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Initialization : Complete

Health Check : Complete

etcd Install : Complete

NFS Install : Complete

Load balancer Install : Complete

Master Install : Complete

Master Additional Install : Complete

Node Install : Complete

Hosted Install : Complete

Metrics Install : Complete

Logging Install : Complete

Prometheus Install : Complete

Service Catalog Install : Complete

1. Set the image registry back to the default, external registry:

|  |  |
| --- | --- |
|  | When you installed RHOCP, you asked it to modify the example image streams to pull from the isolated registry. You achieved this with **openshift\_examples\_modify\_imagestreams=true**. Now that you are back online, albeit through a proxy, you need to change it back to the external registry. |

1. oc get is -n openshift -o yaml | sed -e 's/isolated1.$GUID.internal/registry.access.redhat.com/' | oc apply -f -
2. Make sure your inventory file is reconfigured so it does not modify the image streams again:

sed -i 's/openshift\_examples\_modify\_imagestreams=true/openshift\_examples\_modify\_imagestreams=false/' /etc/ansible/hosts

1. Use the web console to try to deploy sample applications.
2. Examine the proxy logs to make sure you are accessing the Internet through it:
3. ssh isolated1.$GUID.internal
4. sudo -i

less /var/log/httpd/error\_log

**Sample Output**

[Wed Apr 11 18:27:36.384425 2018] [core:trace5] [pid 874] protocol.c(647): [client 192.168.2.232:38708] Request received from client: CONNECT registry.access.redhat.com:443 HTTP/1.1

[Wed Apr 11 18:27:36.384449 2018] [core:debug] [pid 874] vhost.c(1170): [client 192.168.2.232:38708] AH02417: Replacing host header 'registry.access.redhat.com:443' with host 'registry.access.redhat.com:443' given in the request uri

[Wed Apr 11 18:27:36.384459 2018] [http:trace4] [pid 874] http\_request.c(312): [client 192.168.2.232:38708] Headers received from client:

[Wed Apr 11 18:27:36.384461 2018] [http:trace4] [pid 874] http\_request.c(316): [client 192.168.2.232:38708] Host: registry.access.redhat.com:443

[Wed Apr 11 18:27:36.384463 2018] [http:trace4] [pid 874] http\_request.c(316): [client 192.168.2.232:38708] User-Agent: Go-http-client/1.1

[Wed Apr 11 18:27:36.384485 2018] [authz\_core:debug] [pid 874] mod\_authz\_core.c(809): [client 192.168.2.232:38708] AH01626: authorization result of Require ip 192.168.2.0/24 192.168.1.0/24: granted

[Wed Apr 11 18:27:36.384488 2018] [authz\_core:debug] [pid 874] mod\_authz\_core.c(809): [client 192.168.2.232:38708] AH01626: authorization result of <RequireAny>: granted

This completes the lab. You are now able to successfully do the following:

* Deploy a disconnected OpenShift cluster
* Configure CI/CD tools on it
* Get appropriate dependencies and push them to the CI/CD tools
* Build a JEE application and access it
* Reconfigure an isolated node to act as a proxy

Build Version: c3147ce9f77191e30b447cc423f2f68a0c40fc03 : Last updated 2018-07-31 01:29:02 EDT