# How-To: how to use the IBM (or any other repository) with IBM Cloud private in an internet-disconnected environment

## Assaf Kendler, chief architect, IBM Cloud Israel – assafk@il.ibm.com

## Preface

First, let's understand what Helm is. Helm can be best described as a package manager for kubernetes (aka K8S). As you know, K8S can deploy workloads based on docker container images. Helm is leveraging this capability, and makes it easier and more convenient to deploy workloads, that are typical comprised of various components and K8S resources, such as pods, replica sets, secrets, services etc.

Helm works using a client-server topology, Helm being the client side utilizing a very light weight client-side software (roughly 50KB) and a server-side component named "tiller" embedded in K8S. Helm is using a "chart", which can be described as a blueprint or recipe for the complete deployment of the requested workload.

A Helm repository is a collection of Helm charts made available by either an external provider of workloads (such as IBM, Google and others) or an internal provider so within the organization you would find teams authoring helm charts along with container images for other teams within the organization to consume for the purpose of deploying the workload based on these container images. The use of external repositories is over internet, while the use of internal repositories is usually over intranet. Also, important to note is that while internal repositories point to internally accessible image registries to pull the images from, external repositories point to internet accessible image registries.

## Problem statement

As more users of ICp are operating in an internet-disconnected environment, mainly due to regulation or security constraints, these users cannot enjoy the benefits on the internet accessible software catalog from IBM (or other vendors), and that's exactly what we are to fix. ICP or IBM Cloud private provides the following features relevant for our purpose:

1. An empty internal helm repository.
2. A method of connecting to external repositories to deploy workloads in an application center style.
3. Internal (trusted) image registry.

## The procedure

So, as every recipe begins, we need the following ingredients to provide the above-mentioned ability:

1. An internet connected linux system, and another linux system on the intranet side
2. Access to the vendor's (in our example IBM, but can be any vendor) helm repository
3. The helm chart for the requested workload
4. The collection of images needed for deployment (usually out of dockerhub, but not necessarily).
5. An operating ICp environment

The procedure will comprise of these steps:

1. Obtaining the helm chart(s)
2. Obtaining the container image(s)
3. Modifying the helm chart to fit our purpose
4. Packaging the required components to be imported into the internal environment
5. Pushing the container images to the appropriate registry
6. Loading the helm chart into the ICp provided repository
7. Test

Steps 1-4 should be performed on the internet facing system, with the following components installed – git, tree, helm, docker.

Steps 5-7 are to be performed on the internal system, with the following components installed – ICp cli client (for instructions use this [link](https://www.ibm.com/support/knowledgecenter/SSBS6K_2.1.0/manage_cluster/install_cli.html)), helm, docker.

### Step 1 – obtaining the helm charts

As helm repositories are basically a collection http(s) accessible files or directories, we will use wget to get the files we need. In IBM's case it's even easier, as IBM publishes its repository on GitHub so the process would look like this:

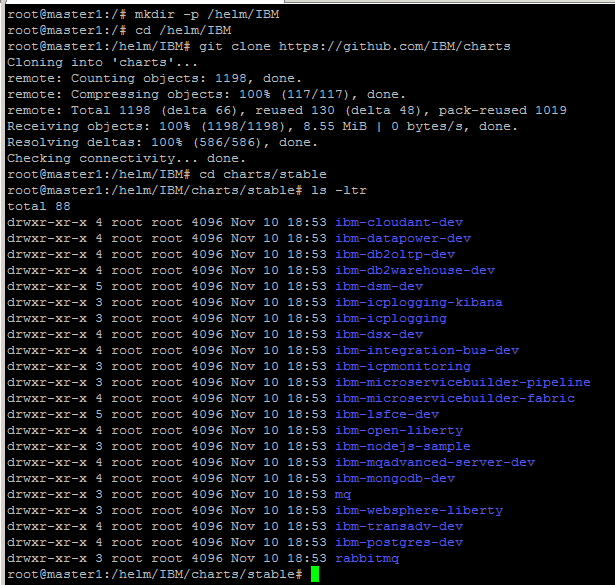
*mkdir -p /helm/IBM*

*cd /helm/IBM*

*git clone* [*https://github.com/IBM/charts*](https://github.com/IBM/charts)

*cd charts/stable*

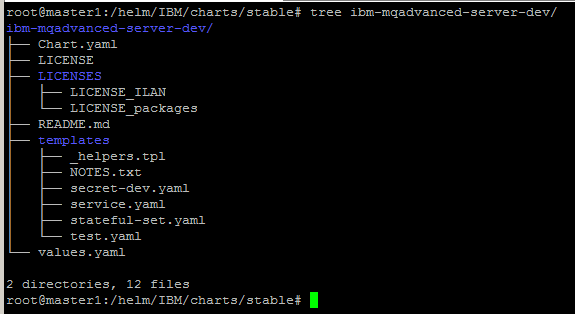
*ls -ltr*



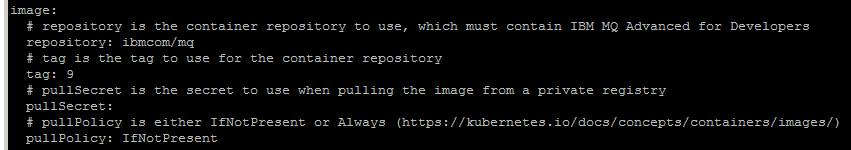
### Step 2 – obtain the container images

In order to obtain the required container images, first we need to understand which container images we need, and for that purpose we need to understand how the helm chart is built and what information can be found (or changed) where.

A helm chart has a defined folder hierarchy, starting from a top-level folder by the name of our workload, under it a folder called "templates" where you will find some relevant yaml files to be used during deployment (called a helm release). Going through these files you will probably find yaml file(s) that describe the deployment or stateful-set itself and within it the referenced image to be pulled and later used by the running containers – log it. Usually you will find these references come from the values.yaml file that has defaults that the user may change during the helm release process (i.e. deploy a workload using helm).



We'll cat the values.yaml file and we will find that in our case, we use the ibmcom/mq:9 from dockerhub.



Now, once you have the container images, it's time to pull it locally so we'll use

*Docker pull <insert your required image name here>*

And in our case –



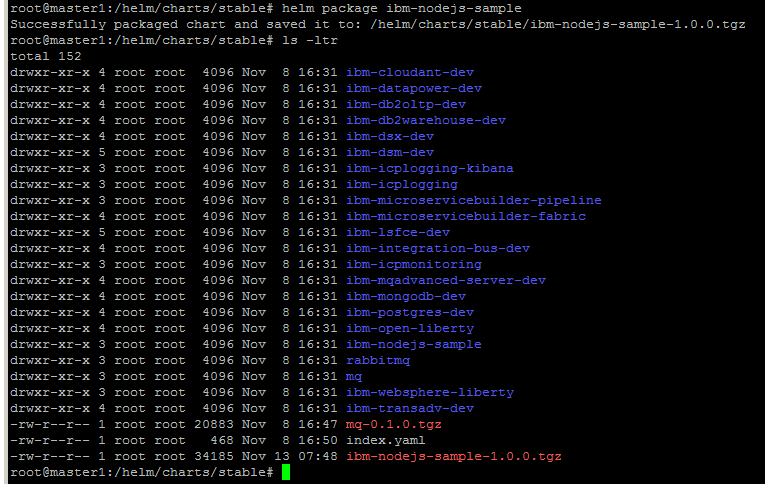
you can verify the images have been pulled using docker images command.

### Step 3 – modify and package the helm charts

In this step, you may want to tweak and change the helm chart. For that purpose, you may want to look into the Chart.yaml and values.yaml files. The Chart.yaml file includes some meta-data about the chart, such as name, version, icon etc.

The values.yaml is more of a functional nature, and it includes values for the release that you may change. You would probably want to change the reference to the image, as after the load you would want the deployment to pull it from your ICp registry and not anything on the internet such as dockerhub which is not accessible. In any case, take into account that once the consumer wishes to deploy using a helm chart, they can override the default values, and as an example change the image name to use the internal image registry, name and tag.

Once you have completed updating the helm chart, it's time to package it, all you need to do is use the helm package command, the input is the directory location, and the output is a tgz file you can transfer to your internal environment.



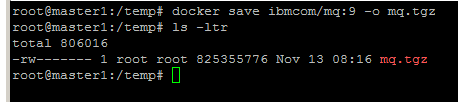
As you can see, we have a new file which is actually a helm chart (ibm-nodejs-sample-1.0.0.tgz).

### Step 4 - Packaging the required components

Now, it's time to ready our files. We already have the helm chart(s) ready as tgz files, and all we need to do now, is ready the container images, we will do that using the command:

*Docker save <image name and tag> -o <destination file>*

Do it for your container images, and you'll have a set of container images ready to ship to the offline environment.



Now, gather your files and get them into the internal system.

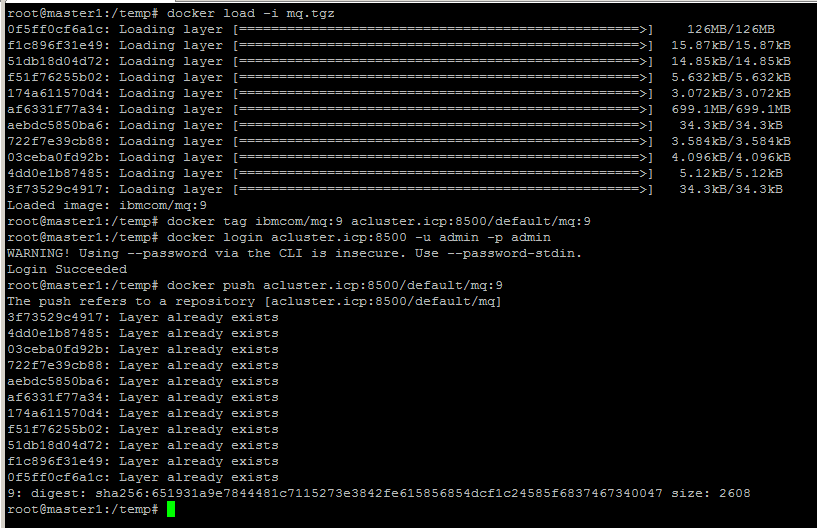
### Step 5 – push the images

On the internal system, we have a set of tgz files (images and charts), we will now load them into the local registry and from there to the ICp registry. The steps are actually :

1. *docker load -I tgz file*
2. *docker tag <image:tag> <targetregistry>/<namespace>/<image:tag>*
3. *docker login <targetregistry>*
4. *docker push <targetregistry>/<namespace>/<image:tag>*

(in my example – the cluster name is acluster.icp, and the target image name and tag are "acluster.icp:8500/default/mq:9") .

Let's look at the example:



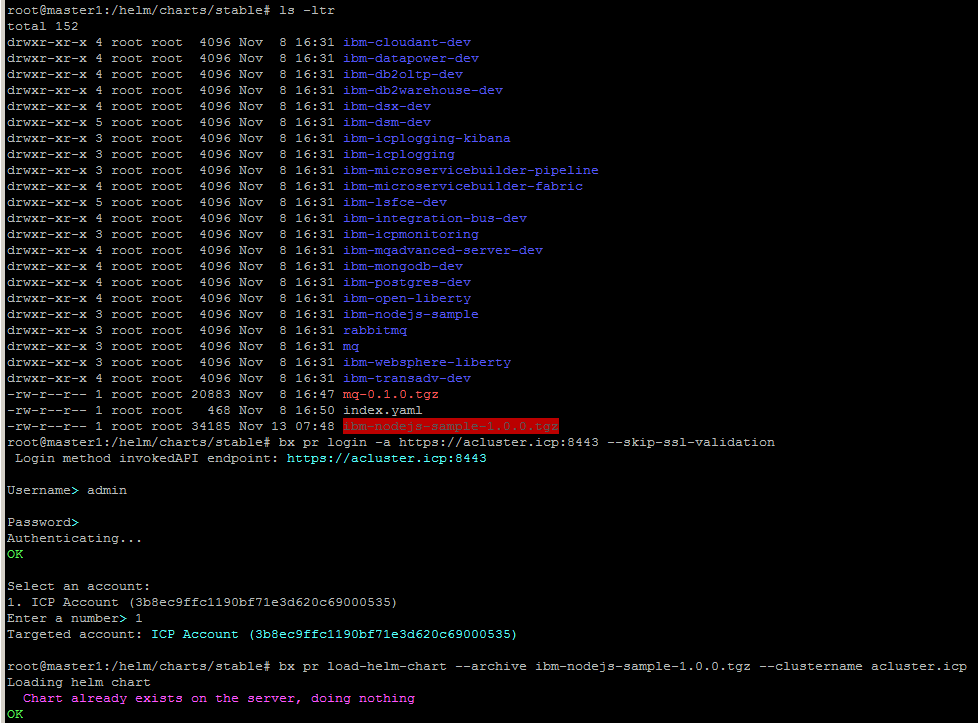
### Step 5 – load the new helm chart

At this stage we will use the bx pr CLI to interact with ICP.

You can find the installation procedure for it here – <https://www.ibm.com/support/knowledgecenter/en/SSBS6K_2.1.0/manage_cluster/install_cli.html>

And the steps are:

1. login to ICp using - *bx pr login -a https://<clustername>:8443 --skip-ssl-validation*
2. load the helm chart using - *bx pr load-helm-chart --archive <path to the chart tgz file> --clustername <clustername>*



### Step 6 – test

Now, you should see our newly loaded helm chart in the ICp catalog just waiting for you to deploy it. Go ahead and test it, and that's it, you're done (don't forget to verify you are targeting the right image for the pull).

