# Installing Bliss on a QNAP NAS using the Docker and the Container Station

To create Docker containers on a QNAP NAS, you must have the Container Station application installed. Container Station is available from the QTS App Center (QTS 4.2 or above) for x86-based TS-x51, TS-x53, TS/TVS-x63, TVS-x70, TVS-x71, TS/SS-x79 and TS/TVS-x80 series and the ARM-based TS-x31+ series with a minimum of 1GB RAM. (From <https://www.qnap.com/solution/container_station/en/)>

Note: The Docker build instructions are completely generic and should work on any system which supports Docker.

## An Introduction to Docker/Container Terminology

Containers are a light-weight virtualisation technology, rather than installing a complete OS image as one does with Virtualisation, Containers run on top of the host operating system in an isolated environment created by the Docker Engine. With Docker we need to be familiar with 2 concepts; Images – the pre-configured software image from which we create the Container, and the Container – the actual execution environment where the program runs. For the Bliss install, we’ll first create a Docker image which contains the latest version of Bliss and all its dependencies, and then we’ll use this Image to create the Container which will run Bliss on the QNAP.

More details on what Docker is and how it works can be found here: <https://docs.docker.com/index.html>

## Docker Build Steps

First, connect to the Public share of your QNAP NAS and create a folder called “docker”, now copy the “Dockerfile” and “bliss-runner.sh” files to this directory.

To kick-off the Container Image build process we need to login to the QNAP NAS - Connect to your QNAP NAS using SSH (this needs to be enabled in the QTS Control Panel, under the Network Services section)

Once you have logged on, run the following commands to build the Bliss container image:

**cd /share/Public/docker**

**docker build --rm=true --tag="bliss-image" .**

This will take a while to complete as it downloads the Ubuntu Trusty image and installs the various components into the container image.

Once the build is complete you will see a message like this:

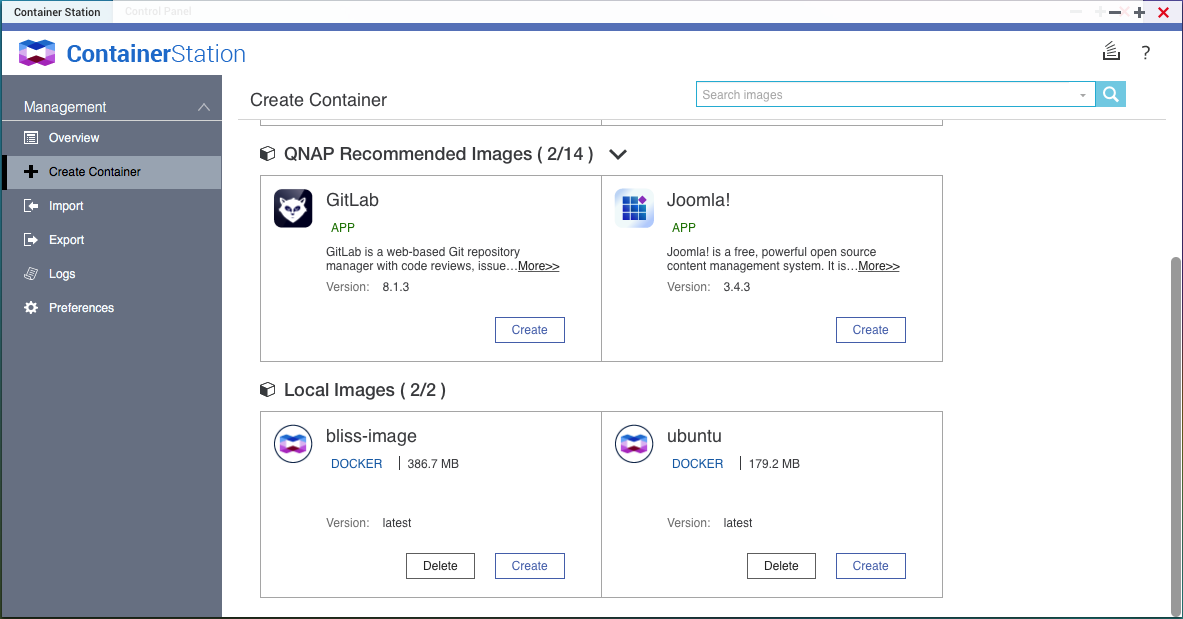
**Removing intermediate container 99af97f56a24**

**Successfully built 272f8cba8682**

Your numbers will be different as these are unique container IDs which are automatically generated by the installation process.

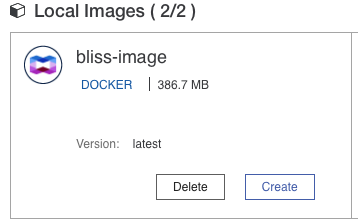
At this point, we can log off QNAP ssh session and move to the QNAP UI for all the other build steps. In the QNAP UI open Container Station and you’ll now see 2 new images available under the Create Container option:

1. **Ubuntu** – this is the Ubuntu Trusty image which our Bliss image is based on.
2. **Bliss-image** – this is the pre-installed image for Bliss from which we will build the actual container.

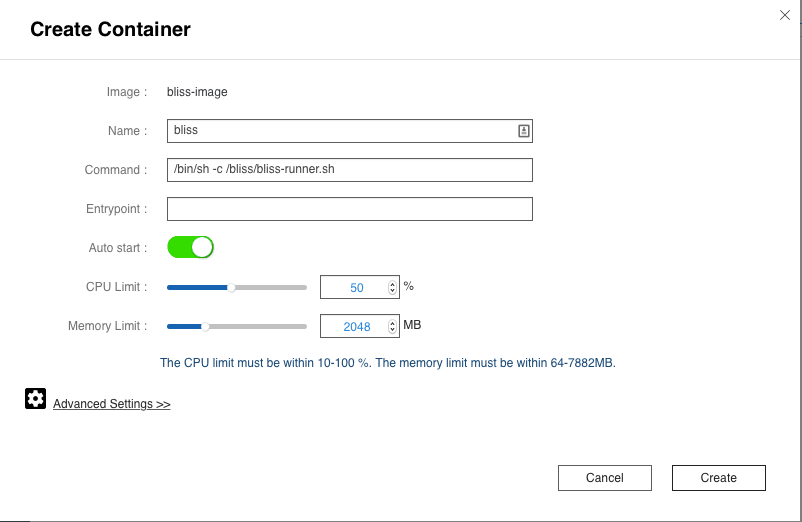


So far we have only created the image for the bliss container, next we need to create the actual container which will host the running instance of Bliss.

Next click “Create” on the bliss-image.



This will open the “Create Container” dialog:

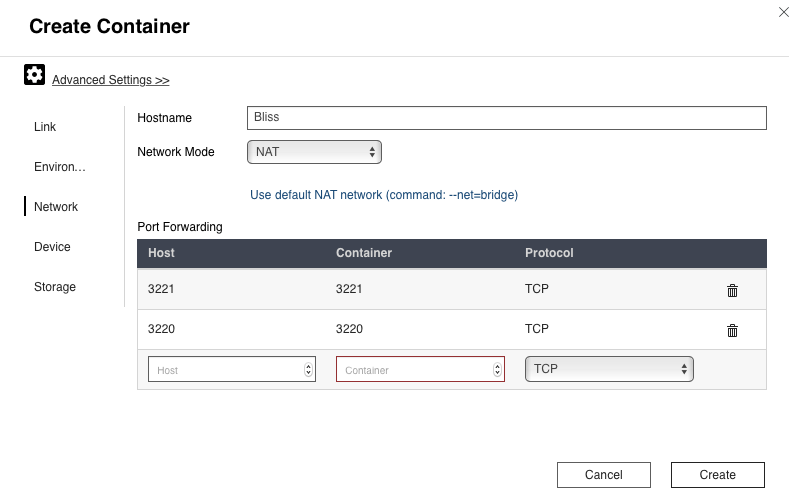


* Set the name to “Bliss”
* Set CPU limit to 50%
* Set Memory Limit to 2048MB (or what ever is appropriate for your NAS - this simply sets the maximum amount of memory the Container is allowed to use, it does not exclusively allocate this memory to the Container. You can change this value and the CPU limit later if needed.)
* **Note:** Because we are running bliss in a container we need to wrap the bliss.sh script in another script which will restart bliss.sh after an update. This is because containers are cleaned up when the process they are running terminates, so if we just started bliss.sh then the container would automatically terminate when bliss exited during the update process. As a result, we need create a wrapper script called bliss-runnner.sh which will restart bliss.sh if it exits with a zero return code (as happens during an update). If bliss.sh exits with a non-zero return code (indicating an error) then the runner script exits to ensure we don’t get into an endless loop of fail and restart.

Now click “Advanced Settings”

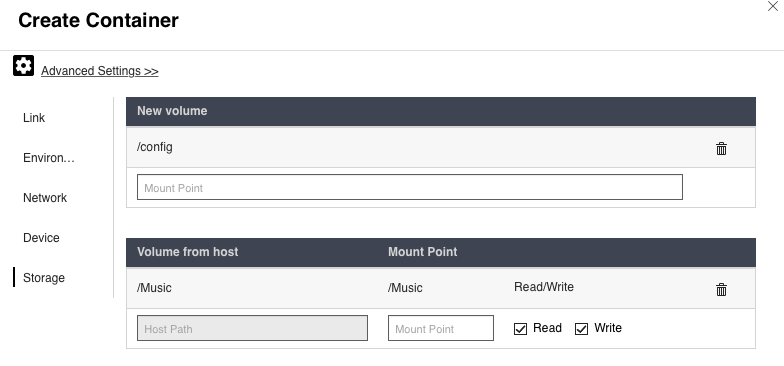
Screen%20Shot%202015-11-14%20at%2016.32.45.png

Under the Network tab, set the Hostname to Bliss and set the Host ports to match the 3220 & 3221 ports exposed by the Bliss container.



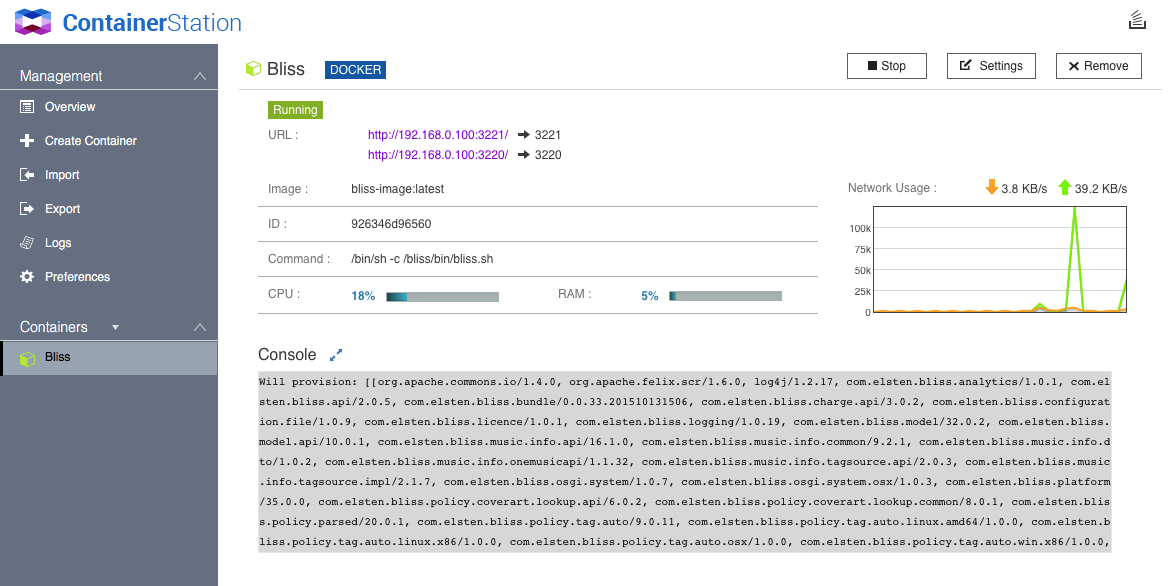
Storage check that New Volume is set to /config, this is a mount within the container where Bliss will store working files and configuration information.

Now finally, under “**Volume from host**” you need to map the location of your Music library on the QNAP to the /Music mount point inside the container. In my example my music is stored in the **/Music** share on the QNAP, so my mapping is **/Music -> /Music**. This mapping must be Read+Write to enable Bliss to update the album art.



Now click the “Create” button to create the Bliss Container.

Once the create process completes, you will see a new container listed in the Container Station UI:



Text in the Console window will say something like: Deploying...

Starting

Bliss will also echo any error messages to this console window – the best way to check Bliss is running is to point your browser at http://<your-QNAP-ip>:3220 and configure Bliss as normal.

## Debugging

You should not need this, but I have included it for the curious…

Once the container is up and running, Bliss is running in the isolated Linux container in much the same way as it would be if you installed it directly on Linux. However, there is one difference when it comes to “logging in” to the container environment if you want to debug or access the command line for the Container.

First you need to find the ID of the running container. To do this, connect to you QNAP via SSH and run the following command:

docker ps

You’ll now see something like this:

CONTAINER ID IMAGE … NAMES

2b7868288ae9 bliss-image:latest … Bliss

We are interested in the Container ID running Bliss – in this case 2b7868288ae9 (you ID will be different).

To access the shell within this container we need to tell docker to execute a shell within the running container using the ID you saw from the docker ps

command:

docker exec -i -t <your Container ID> bash

For example:

docker exec -i -t 2b7868288ae9 bash

You’ll then see a prompt like this:

root@Bliss:/bliss#

And at this point you are logged into the shell within the running container, which will function just like a normal Linux shell. When you are done simply exit the shell.

## Dockerfile

FROM ubuntu:trusty

ENV DEBIAN\_FRONTEND noninteractive

VOLUME /config

EXPOSE 3220 3221

RUN mkdir /bliss

ADD bliss-runner.sh /bliss/

RUN chmod +x /bliss/bliss-runner.sh

RUN apt-get update

RUN apt-get install -y wget

RUN apt-get install -y openjdk-7-jre-headless

RUN wget -qO- http://www.blisshq.com/app/latest-linux-version | xargs wget -O bliss-install-latest.jar -nv

RUN echo INSTALL\_PATH=/bliss > auto-install.properties

RUN java -jar bliss-install-latest.jar -console -options auto-install.properties

WORKDIR /bliss

CMD /bliss/bliss-runner.sh

## bliss-runner.sh

#!/bin/bash

#

# Script to restart bliss after an update

# Only needed for the Docker build, to stop the container exiting after an update

# We redirect the bundle storage to a persistent volume, and

# disable the bliss launcher options so that this script controls restarting bliss

# after an upgrade.

export VMARGS="-Dbliss\_working\_directory=/config -Dorg.osgi.framework.storage=/config/felix-cache"

export BLISS\_OSGI\_BUNDLE\_STORAGE=/config/felix-cache

export BLISS\_LAUNCHER\_PROPERTY=''

while [ true ]

do

sh /bliss/bin/bliss.sh

if [ "$?" = "0" ]; then

echo "bliss.sh exited, restarting..."

sleep 2

else

echo "bliss.sh exited with a non-zero value, exiting..."

exit 1

fi

done