Windows image builder helpers

Currently, the image building process implies using a Linux node with docker buildx, which allows us to build multiarch images, as well as building Windows images. An additional benefit to this approach is that we wouldn't have to create a new Windows node for each new Windows release (you cannot spawn containers with an OS version newer than the host OS version), simplifying the building process.

However, there are few constraints when it comes to building Windows images using docker buildx: you cannot use any `RUN` or `WORKDIR` instructions in the Windows stage in the image Dockerfile, but there are a few cases in which we need to `RUN` some commands. As a workaround to this, we can simply build some helper images using a Windows docker node, publish it, and then use them in the regular Image Builder which will only have to use a Linux node. This folder contains such helper images.

Windows node(s) setup

In order to build your own helper images, a node with Windows 10 or Windows Server 2019 with the latest updates installed is required. The node will have to have Docker installed, preferably version 18.06.0 or newer.

Remote management must be configured for the node's Docker daemon. Exposing the Docker daemon without requiring any authentication is not recommended, and thus, it must be configured with TLS to ensure that only authorised users can interact with it. For this, the following `powershell` script can be executed:

```
mkdir .docker
docker run --isolation=hyperv --user=ContainerAdministrator --rm `
  -e SERVER_NAME=$(hostname) `
  -e IP_ADDRESSES=127.0.0.1,YOUR_WINDOWS_BUILD_NODE_IP `
  -v "c:\programdata\docker:c:\programdata\docker" `
  -v "$env:USERPROFILE\.docker:c:\users\containeradministrator\.docker" stefanscherer/docker
# restart the Docker daemon.
Restart-Service docker
```

For more information about the above commands, you can check here.

A firewall rule to allow connections to the Docker daemon is necessary:

```
New-NetFirewallRule -DisplayName 'Docker SSL Inbound' -Profile @('Domain', 'Public', 'Privat
```

If your Windows build node is hosted by a cloud provider, make sure the port `2376` is open for the node. For example, in Azure, this is done by running the following command:

```
az vm open-port -g GROUP-NAME -n NODE-NAME --port 2376
```

The `ca.pem`, `cert.pem`, and `key.pem` files that can be found in `$env:USERPROFILE\.docker` will have to copied to the `~/.docker/` on the Linux build node.

```
scp.exe $env:USERPROFILE\.docker\*.pem ubuntu@YOUR_LINUX_BUILD_NODE:/home/ubuntu/.docker/
```

After all this, the Linux build node should be able to connect to the Windows build node:

```
docker --tlsverify --tlscacert ~/.docker/ca.pem --tlscert ~/.docker/cert.pem --tlskey ~/.doc
```

For more information and troubleshooting about enabling Docker remote management, see here

Finally, the node must be able to push the images to the desired container registry, make sure you are authenticated with the registry you're pushing to.

Additional configuration

The `powershell-helper` image uses `mcr.microsoft.com/windows/nanoserver:1809` as a base image. Note that `docker buildx` has an issue pulling cross-registry images when building images, and in order to circumvent this issue, the make target `all-push-as-linux` will push a Linux cache image which will contain only the necessary bits, and this cache image can then be used in the regular image building process. As an additional benefit, using a Linux cache image will be faster.

In order to build the Linux cache image, `docker buildx` is needed. For more information about it can be read here.

Building images

The images are built through `make`:

```
make REGISTRY=foo_registry REMOTE_DOCKER_URL=$REMOTE_DOCKER_URL all-push-as-linux
```