Token

Because dockerhub does not implement authorization in their webhooks, we will need to generate a random string that is used in the catalyst webhook url.

First create a url-safe random string like so

```
$ dd if=/dev/urandom bs=1 count=64 2> /dev/null | base64 --wrap=0 | sed -e 's/+/-/g' -e 's/\//_/g' -e
's/=/~/g'
```

Now store that string in a secure way. You will need this when configuring webhooks in dockerhub or docker registry.

Add the string to docker secrets like this.
NB: if you don't use the default name `url.token`, you have to edit the compose file to reflect this.

```
$ docker secret create url.token [secure-string]
```

Dockerhub

If any repositories in use requires a docker login to pull images from it, you must create a secret containing the username and password of a user with read access to the repository. `$repo` refers to the repository in question.

```
cat << EOF | docker secret create $repo.login.json -
{
    "username": "cion",
    "password": "123456"
}
EOF
```

Docker

In order to add an external docker swarm we need to generate tls certificates. It is important to understand what is going on and to treat the generated files correctly. *They are equivalent to root access to the machine running the external swarm.* You can find the offical guide on securing the docker daemon here (https://docs.docker.com/engine/security/https/).

Generate artifacts

```
$ mkdir tls-certs && cd tls-certs
```

First we need to generate a Certificate Authority (https://en.wikipedia.org/wiki/Certificate_authority) which we will use to sign our certificates. Make sure the common name is the DNS name of the external docker swarm (referred to as `$HOST` from now on).

```
$ openssl genrsa -aes256 -out ca-key.pem 4096
> Generating RSA private key, 4096 bit long modulus
> e is 65537 (0x10001)
> Enter pass phrase for ca-key.pem:
> Verifying - Enter pass phrase for ca-key.pem:

$ openssl req -new -x509 -days 365 -key ca-key.pem -sha256 -out ca.pem
> Enter pass phrase for ca-key.pem:
> You are about to be asked to enter information that will be incorporated
> into your certificate request.
> What you are about to enter is what is called a Distinguished Name or a DN.
> There are quite a few fields but you can leave some blank
> For some fields there will be a default value,
> If you enter '.', the field will be left blank.
> -----
> Country Name (2 letter code) [AU]:
> State or Province Name (full name) [Some-State]:Queensland
> Locality Name (eg, city) []:Brisbane
> Organization Name (eg, company) [Internet Widgits Pty Ltd]:Docker Inc
> Organizational Unit Name (eg, section) []:Sales
> Common Name (e.g. server FQDN or YOUR name) []:$HOST
> Email Address []:Sven@home.org.au
```

Generate a server certificate and sign it with our CA.

```
$ openssl genrsa -out server-key.pem 4096
> Generating RSA private key, 4096 bit long modulus
> e is 65537 (0x10001)

$ openssl req -subj "/CN=$HOST" -sha256 -new -key server-key.pem -out server.csr

$ echo subjectAltName = DNS:$HOST,IP:10.10.10.20,IP:127.0.0.1 > extfile.cnf
$ echo extendedKeyUsage = serverAuth >> extfile.cnf

$ openssl x509 -req -days 365 -sha256 -in server.csr -CA ca.pem -CAkey ca-key.pem -CAcreateserial -out
server-cert.pem -extfile extfile.cnf
> Signature ok
> subject=/CN=your.host.com
> Getting CA Private Key
> Enter pass phrase for ca-key.pem:
```

Create the signed client certificates.

```
$ openssl genrsa -out key.pem 4096
> Generating RSA private key, 4096 bit long modulus
> e is 65537 (0x10001)

$ openssl req -subj '/CN=client' -new -key key.pem -out client.csr
$ echo extendedKeyUsage = clientAuth > extfile.cnf

$ openssl x509 -req -days 365 -sha256 -in client.csr -CA ca.pem -CAkey ca-key.pem -CAcreateserial -out
cert.pem -extfile extfile.cnf
> Signature ok
> subject=/CN=client
> Getting CA Private Key
> Enter pass phrase for ca-key.pem:
```

Delete unnecessary files and secure the artifacts.

```
$ rm -v client.csr server.csr
$ chmod -v 0400 ca-key.pem key.pem server-key.pem
$ chmod -v 0444 ca.pem server-cert.pem cert.pem
```

Artifact	Use by cion	Use by external docker swarm
ca.pem	Verify that servers are signed with this CA	Verify that clients are signed with this CA
server-cert.pem	None	Public key signed by CA
server-key.pem	None	Private key used to verify own identity to clients
cert.pem	Public key signed by CA	None
key.pem	Private key used to verify own identity to servers	None

Add the secrets to the docker swarm cion is running in

Cion needs access to the ca.pem, key.pem and cert.pem files from the previous section. `$env` refers to the name of the external swarm e.g. *qa*.

```
$ docker secret create $env.ca.pem /path/to/ca.pem
$ docker secret create $env.key.pem /path/to/key.pem
$ docker secret create $env.cert.pem /path/to/cert.pem
```

Configure external swarm to accept tls connections over https

The external machine needs access to the ca.pem, server-key.pem and server-cert.pem files from the previous section. We need to edit the docker daemon configuration. This is best done by editing the daemon.json (https://docs.docker.com/engine/reference/commandline/dockerd//#daemon-configuration-file) file usually found in `/etc/docker/daemon.json`. If `/etc/docker/daemon.json` does not exist, create it.

Open `/etc/docker/daemon.json` in a text editor. After setting up tls verification the configuration file should resemble this.

```
{
    "tlsverify": true,
    "tlscacert": "/path/to/ca.pem",
    "tlscert": "/path/to/server-cert.pem",
    "tlskey": "/path/to/server-key.pem",
    "hosts": [
        "fd://",
        "0.0.0.0:2376"
    ]
}
```

Unfortunately if you use systemd to start docker, the *hosts* option is already specified in the startup script's command line arguments. Because docker does not support a conflict between command line arguments and `daemon.json`, you need to resolve the conflict. If you are not running docker through systemd, you can skip this step and restart the docker daemon.

Open `/lib/systemd/system/docker.service` in a text editor. We need to modify `daemon.json` and ExecStart such that the hosts are only configured in one of them. In the following example they are kept in `daemon.json` and removed from ExecStart.

```
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
After=network-online.target docker.socket firewalld.service
Wants=network-online.target
Requires=docker.socket
[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
- ExecStart=/usr/bin/dockerd -H fd://
+ ExecStart=/usr/bin/dockerd
ExecReload=/bin/kill -s HUP $MAINPID
LimitNOFILE=1048576
# Having non-zero Limit*s causes performance problems due to accounting overhead
# in the kernel. We recommend using cgroups to do container-local accounting.
LimitNPROC=infinity
LimitCORE=infinity
# Uncomment TasksMax if your systemd version supports it.
# Only systemd 226 and above support this version.
#TasksMax=infinity
TimeoutStartSec=0
# set delegate yes so that systemd does not reset the cgroups of docker containers
Delegate=yes
# kill only the docker process, not all processes in the cgroup
KillMode=process
# restart the docker process if it exits prematurely
Restart=on-failure
StartLimitBurst=3
StartLimitInterval=60s
[Install]
WantedBy=multi-user.target
```

Now restart docker

```
$ systemctl daemon-reload
$ systemctl restart docker
```

Kubernetes

In order for the cluster to authorize cion, we need to create a service account.

```
$ cat > /tmp/serviceaccount.yaml << EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: cion
EOF
$ kubectl create -f /tmp/serviceaccount.yaml
> serviceaccount "cion" created
```

Kubernetes should automatically have created a secret token with the name `[serviceaccount]-token-[hash]` for the service account. This secret contains all the information needed to connect to the cluster.

```
$ kubectl get secrets
> NAME                  TYPE                                  DATA      AGE
> cion-token-79hrs      kubernetes.io/service-account-token   3         32d
> default-token-qxcdl   kubernetes.io/service-account-token   3         32d
```

First we will need the decoded CA file so our client can verify the tls connection. We also need the encoded token.

```
$ kubectl get secret cion-token-79hrs --output=json | jq '.data["ca.crt"]' --raw-output | base64 -d |
docker secret create $env.ca.crt -
$ kubectl get secret cion-token-79hrs --output=json | jq '.data.token' --raw-output | docker secret create
$env.token -
```