# Upgrading Ingress Rules and Adding TLS Certificates

This lesson explains how to upgrade Ingress rules and add TLS certificates.

**We'll cover the following**



All the applications we installed so far are accessible through a plain HTTP protocol. As I'm sure you're aware, that is not acceptable. All public-facing applications should be available through HTTPS only, and that means that we need TLS certificates. We could generate them ourselves for each of the applications, but that would be too much work. Instead, we'll try to figure out how to create and manage the certificates automatically. Fortunately, Jenkins X already solved that and quite a few other **Ingress**-related challenges. We just need to learn how to tell `jx` what exactly we need.

All the `jx upgrade` commands we explored so far have followed the same pattern, they upgrade components to a specific or the latest release. **Ingress** is the exception. We can use the `jx upgrade ingress` command to change a variety of things.

- We can change the domain of the cluster or a namespace.
- We can add TLS certificates to all **Ingress** endpoints.
- We can also change the template Jenkins X is using to auto-generate addresses of applications.

## Confirming the deployment #

Let's start by checking the applications we currently have in our cluster.

```
jx get applications
```

The output is as follows.

```
APPLICATION  STAGING PODS URL
jx-go-demo-6 1.0.110 3/3  http://go-demo-6.cd-staging.35.243.230.195.nip.io
```

> ⚠ If the output is empty, you were probably reading too fast, and the pipeline run initiated when you imported *go-demo-6* hasn't finished and, therefore, the release was not deployed to the staging environment.

You already saw that output quite a few times before. There's nothing special about it, except that *go-demo-6* is accessible through an auto-generated HTTP address. We must change that to HTTPS since no serious applications should be reachable without TLS. But, before we do that, let's confirm that our application can indeed be reached.

> ⚠ Make sure to replace `[...]` with the address from the `URL` column from the previous output before executing the commands that follow.

```
STAGING_ADDR=[...]

curl "$STAGING_ADDR/demo/hello"
```

The output should show `hello, PR!` so we confirmed that the application is working and that it can be reached through the insecure HTTP protocol. Feel free to send a request using `https`, and you'll see that the output will state that there is an `SSL certificate problem`.

So far, we used a `nip.io` domain for all our examples. That was useful for the exercises since it saved us from purchasing a "real" domain, from reconfiguring DNS with our domain registrar, and from a long wait until changes are propagated. But I'm sure that you already wondered how to make Jenkins X use a custom domain. When you start using it "for real", you will surely want Jenkins X and your applications to be accessible through your domain instead of `nip.io`. We'll try to remedy that as well.

So, we have three issues to solve.

1. We should redirect all HTTP requests to HTTPS.
2. We should make sure that SSL/TLS certificates are in place.
3. We should switch from `nip.io` to any domain we want to use.

We should be able to make any of those changes on the level of the whole cluster, a Namespace, or an application. We'll start with cluster-wide changes.

## Finding the IP of the cluster #

Now you need to choose whether or not you'd like to use a "real" domain. If you do not have one available, or you do not want to mess with DNS, you can continue using `nip.io` as a simulation of what would happen if we'd change a domain. In either case, we need to find the IP of the cluster first.

```
LB_IP=$(kubectl \
    --namespace kube-system \
    get svc jxing-nginx-ingress-controller \
    -o jsonpath="{.status.loadBalancer.ingress[0].ip}")

echo $LB_IP
```

> ⚠ Execute the command that follows only if you do NOT have a fully qualified domain. If that's your case, you should know that you will not be able to experience TLS. Later on, when we upgrade **Ingress**, the addresses **will NOT be changed to HTTPS**, and you will have to "imagine" how it would look like.

```
DOMAIN=$LB_IP.nip.io
```

If you do have a domain available, as I hope you do, please change your DNS records in your domain registrar to the IP of the cluster (output of `echo $LB_IP`). It might take an hour or even more until DNS records are propagated so you'll need to wait for a while. This would be an excellent time to have lunch, do some exercise, or see a movie. But, before you go, please execute the command that follows.

> ⚠ Execute the command that follows only if you do have a fully qualified domain pointing to your cluster. Make sure to replace `[...]` with the domain (e.g., play-with-jx.com).

```
DOMAIN=[...]
```

## Upgrading Ingress

Now we're ready to upgrade **Ingress** (remember that you had to wait for DNS to propagate if you're using a custom domain). We'll change the domain across the whole cluster. If you're using nip.io, it'll stay the same, and we'll only simulate the process.

```
jx upgrade ingress \
    --cluster true \
    --domain $DOMAIN
```

It'll take a while until the upgrade is finished and we'll have to answer a few questions since, this time, we did not specify `--batch-mode`.

First, we're asked to `confirm to delete all and recreate` **Ingress** rules. Please stick with the default value not only for this question but for all others. This time, the default value is `Y`.

Next, we are asked whether we want to expose `Ingress` or `Route`. Unless you're using OpenShift, `Ingress` is the correct (and the default) answer.

Next, we are asked to define the domain. Since we already specified it through the `--domain` argument, the default it is already predefined and we can simply press the enter key.

Now comes the critical question if you are using a "real" domain. `Would you like to enable cluster-wide TLS?` Again, the default answer `Y` is correct. Who wouldn't want TLS if it comes at no additional cost? Please note that nip.io users will not be asked that question, nor the next few ones related to certificates.

## Enabling TLS certificates

Jenkins X supports Let's Encrypt implemented through cert-manager out of the box. You can use some other means to provide certificates but, unless you have exceptional requirements, cert-manager is the easiest and probably the most

reliable dynamic TLS certification we can use. So, we'll stick with it.

We're asked whether to use `staging` or `production` version of Let's Encrypt. Staging allows more retry attempts, so it is suitable for practice purposes when we iterate over the process frequently. But, in our case, we're likely going to succeed from the first attempt, so we'll go with the default value `production`.

Since Let's Encrypt needs your email address to validate your identity, you'll be asked to provide one.

Next, we can specify `UrlTemplate`, which we'll explore it in more detail later. For now, please press the enter key to keep the current value.

Finally, we are asked to confirm that the selected choices are what we really want. Press the enter key to select the default answer `Y`.

Since this is the first time we're using the cert-manager, `jx` will ask us whether we want to install it. Be brave and say yes (press the enter key).

As you can see, we kept all the default options. The main reason we did not run the command in the batch mode was for you to get familiar with the things we can change through `jx upgrade ingress`. For now, we're focusing only on applying a cluster-wide change of the domain and adding TLS certificates to all publicly available applications. We'll explore some of the other possibilities later on.

Now you're in for a wait. It'll take a while for all the changes are propagated throughout the whole cluster, and you should be able to follow through logs in which resources were created, deleted, or updated.

If you are running static Jenkins X, you might be asked for your `Jenkins user name` and GitHub `API Token`. Ignore the instructions for creating the token. Instead, use the one you have from before or go to GitHub and create a new one.

After a while, you will be asked a few more questions like whether you want to `update all existing webhooks` (say that you do) and what your GitHub organization is. Assuming that you did choose to update webhooks, `jx` will go through all your GitHub repositories and update those currently used by Jenkins X. After all, we might have changed the domain, and we have undoubtedly changed the protocol to HTTPS, and those changes need to be applied to GitHub webhooks, or Jenkins X will not receive any new notifications that something changed and, therefore, will not execute new pipeline runs.

therefore, will not execute new pipeline runs.

That's it. We changed the domain (if you had one), and we switched to HTTPS across the whole cluster (if you're not using *nip.io*). In this context, "whole" means "controlled by Jenkins X". If you had applications deployed through some other means, they would be left intact.

## Confirming updates #

Now that we mentioned applications, let's see what we got after the upgrade.

```
jx get applications --env staging
```

The output is as follows.

```
APPLICATION  STAGING PODS URL
jx-go-demo-6 1.0.110 3/3  https://go-demo-6.cd-staging.play-with-jx.com
```

If you changed the domain, you'll notice that the change is reflected in the `URL`. The other important difference is that the protocol is now `https`. **Does it work?**

> ⚠ Make sure to replace `[...]` with the address from the `URL` column from the previous output before executing the commands that follow.

```
STAGING_ADDR=[...]

curl "$STAGING_ADDR/demo/hello"
```

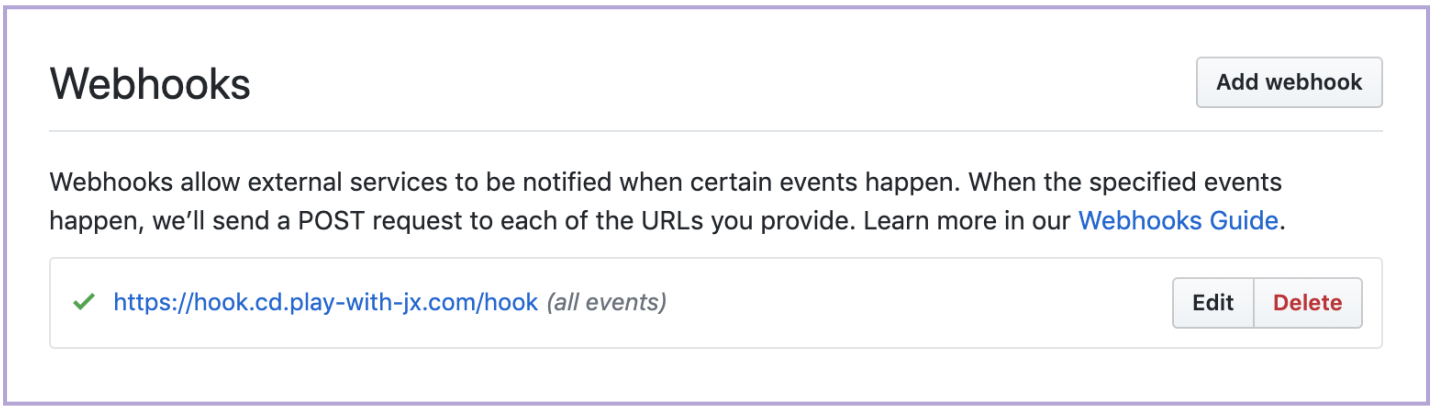
If you tried sending a request to HTTPS before, you probably noticed that the response was an error complaining that the certificate is invalid. That is not the case anymore. This time we got the familiar `hello, PR!` output. From now on, all our applications respond to HTTPS with proper certificates. On top of that, all requests sent to HTTP will be redirected to HTTPS.

Now that we saw that **Ingress** resources associated with our applications were modified correctly, let's see whether our GitHub webhooks were updated as well.

```
cd go-demo-6

jx repo --batch-mode
```

Please make sure that you are logged in and open *Settings* followed by *Webhooks*. You'll see that `jx upgrade ingress` modified it to reflect the new domain and that it now uses HTTPS.




Updated webhook with the new domain and HTTP protocol

The fact that webhooks were modified does not necessarily mean that they work. We can check that out by pushing a change. For brevity, we'll skip creating a pull request and simply make a change to the master branch.

> ⚠ If you changed the domain DNS, make sure that a considerable amount of time has passed so that the update was propagated across the Web. One hour should be enough. Otherwise, GitHub might not yet know that your DNS was updated.

We'll make a silly change to the code and push it to the repository. That will allow us to test whether the updated webhooks work correctly.

```
echo "I am too lazy to write a README" \
    | tee README.md

git add .

git commit -m "Checking webhooks"

git push
```

Now, go back to the GitHub webhooks screen and confirm that the newly updated one is still green.

To be on the safe side, we'll take a look at the activities. If a new one was initiated through the last push, we'll know that the webhook updates were done correctly.

> 🔍 Static Jenkins X periodically pulls Git for changes apart from accepting webhook requests. So, a new pipeline run will execute even if the webhook was not reconfigured correctly.

```
jx get activities \
    --filter go-demo-6 \
    --watch
```

The output should display the progress of a new pipeline run, thus confirming that webhooks are healthy in case of serverless Jenkins X (the static flavor could accomplish the same by pulling info from GitHub).

Feel free to stop watching the activity progress by pressing *ctrl+c*.

> 🔍 We could have saved us from running the `jx upgrade ingress` command to define the "real" domain if we specified `--domain` argument when we executed `jx create cluster` or `jx install` commands. We didn't do that simply because I wanted to show you how `jx upgrade ingress` accomplishes the change of the domain.

---

Next, let's see how we can change the URL patterns of our applications in staging and production.