Changing URL Patterns

This lesson explains how we can define a custom URL template and change the URL patterns.

We'll cover the following

Jenkins X's ability to auto-generate addresses of our applications running in different environments is beneficial. It allows us to stop worrying about whether we'll introduce a conflict (the same address in multiple environments) and concentrate on the applications themselves.

Auto-generated address pattern

It does that by auto-generating **Ingress** resources with addresses that follow the pattern which is set by default to the value `{{.Service}}.{{.Namespace}}.{{.Domain}}`. So, if we have a service called `my-service` deployed to the Namespace `pre-prod` and in a cluster accessible through the domain `acme.com`, the application would be accessible through the auto-generated address `my-service.pre-prod.acme.com`. That is great, but sometimes we might want to change the template, either on the level of the whole cluster or for individual Namespaces. Sometimes we might not even want autogenerated addresses when applications are deployed to specific environments (e.g., production).

Fortunately, Jenkins X allows us to control all that, and we're about to explore how to establish that control through a single command.

Getting the address of the application in staging

Let’s take a look at the address of our application running in the staging environment.

```
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
```

The pattern should be easy to deduce. The address consists of the name of the application (`go-demo-6`), the name of the environment (`cd-staging`), and the domain (in my case, it’s `play-with-jx.com`). There is often no need even to retrieve the address of an application if we know the pattern. You can easily guess that if we’d have an application called *my-app* deployed to the namespace *cd-staging*, it would be accessible through *https://my-app.cd-staging.play-with-jx.com*. But what if we do not like the pattern Jenkins X created for us? Can we change it?

Currently, all applications contain `cd-staging` in the address (or `jx-staging` is using the static flavor) and that might not we very intuitive to our users. Wouldn’t it be nicer if it is merely `staging` (without *cd-* or *jx-* prefix)? In such a case, our *go-demo-6* application running in staging would be accessible through `https://go-demo-6.staging.play-with-jx.com`.

Changing the URL template for the staging environment

So, our next mission is to change the URL template that defines how Jenkins X generates addresses of the applications deployed to staging. We’ll execute yet another `jx upgrade ingress` command but, this time, we’ll limit it to the staging Namespace.

```
NAMESPACE=$(kubectl config view \
    --minify \
    --output jsonpath="{..namespace}")

jx upgrade ingress \
    --namespaces $NAMESPACE-staging \
    --urltemplate "{{.Service}}.staging.{{.Domain}}"
```

We initiated the process that will change the URL template. By default, it is set to the value of `{{.Service}}.{{.Namespace}}.{{.Domain}}`. We replaced `{{.Namespace}}` with `staging` hoping that will make it easier for other users to know the address of the applications.

We are presented with the same questions as when we executed `jx upgrade ingress` the first time. Since the necessary changes are already defined in the command arguments, you can select the default answers to all by simply pressing the enter key.

Confirming the changes

Once the upgrade process is finished, we can confirm that the change was indeed successful.

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

The output is as follows.

```
APPLICATION STAGING PODS URL
go-demo-6   1.0.134 3/3  https://go-demo-6.staging.play-with-jx.com
```

We can see from the `URL` column that we now have `staging` instead of the Namespace.

To be on the safe side, we'll send a request to the application and confirm that it is accessible through the new address.

> ⚠ Replace the first occurrence of `[...]` with the release version (we'll need it later) and the second with the `URL`.

```
VERSION=[...]

STAGING_ADDR=[...]

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

It should come as no surprise that we got the response (`hello, PR!`). It worked!

Having `staging` in the addresses of the applications deployed to that environment is a good idea because it guarantees that they will be unique and will not clash with the same application running in other environments. However, we might

want to simplify it even more for production that is accessible to our end users. But, before we do that, let's first promote the latest release so that we are sure that we have something to work with.

Promoting the latest release

```
jx promote go-demo-6 \
    --version $VERSION \
    --env production \
    --batch-mode
```

Let's take a look at the address of our application running in production.

```
jx get applications --env production
```

The output is as follows.

```
APPLICATION PRODUCTION PODS URL
go-demo-6   1.0.134         http://go-demo-6.cd-production.40.85.160.33.nip.io
```

The address follows the same pattern `{{.Service}}.{{.Namespace}}.{{.Domain}}`. While we could certainly simplify it by replacing `{{.Namespace}}` with `production`, that might still be too unfriendly to our end users. **Wouldn't it be better if we get rid of the namespace altogether?**

> ⚠ There is a bug (undocumented feature) in `jx upgrade ingress` that causes changes to the domain to apply only to existing applications. If we upgrade them, the new domain is maintained. However, if we install an application for the first time it resets to the "original" domain we specified during the Jenkins X installation. That's what happened when we promoted *go-demo-6*. It's a new install, and the "original" domain was used. I opened issue 4114 that you can track to see whether the problem is resolved. While it is not convenient to work through this bug, it is still not a critical one since we do not install new applications to new environments that frequently. Most of the time, we upgrade those that are already running. The workaround is to re-run the `jx upgrade ingress` command (just as we're about to anyway) with the `--domain` argument.

⚠ If you do see the new domain, the previous warning must sound like the talk of a madman. That means that the issue is either resolved or that you reused the cluster from the earlier chapters and you already have an older release of the application running in production, so it wasn't a new install but an upgrade.

Changing the URL template for the production environment

Let's get back to the task at hand. We'll change the URL pattern used in the production environment so that applications are accessible through their (service) name as a subdomain (without anything in between). The command is as follows.

```
jx upgrade ingress \
    --domain $DOMAIN \
    --namespaces $NAMESPACE-production \
    --urltemplate "{{.Service}}.{{.Domain}}"
```

Keep pressing the enter key to answer all the questions.

Let's see the address of the production deployment of *go-demo-6*.

```
jx get applications --env production
```

The output is as follows.

```
APPLICATION PRODUCTION PODS URL
go-demo-6   1.0.134    3/3  https://go-demo-6.play-with-jx.com
```

As you can see, the address is now using the new template consisting of the application (service) name and the domain. The namespace we had in between is gone and we can safely say that this is much more user-friendly than before.

We could have changed the template to `{{.Service}}.com`, which would assume that the name of the service of an application is the same as the domain we own. Unless you do own *go-demo-6.com*, we'll skip changing the template to `{{.Service}}.com` and you'll need to trust me that it would work as long as you do own that domain.

Redefining addresses for the production environment

Ultimately, we might choose not to let Jenkins X generate addresses dynamically. While I cannot imagine a reason why we would do that in staging, previews, and other non-production environments, that might be an excellent strategy for production. That would allow us to control the address of each application. It would introduce a slight overhead since we'd need to set it up ourselves, but it might be worth it for production. **How would we do that?**

If you take a closer look at the files in the charts/go-demo-6/templates directory, you'll notice that there is no **Ingress**. The assumption is that Jenkins X will create it for us using the URL template we've been modifying. But, if we do want to have application-level control over addresses we might need to add **Ingress**. Given that we cannot have the same addresses in different environments (e.g., staging, production), that **Ingress** needs to be created only under certain conditions.

If we'd set the goal to redefine the address of *go-demo-6* only when it's running in production, we could create *charts/go-demo-6/templates/ing.yaml* that would have the following content.

```
# If could be `jx-production` if you're using static JX.
{{- if eq .Release.Namespace "cd-production" }}
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: go-demo-6-prod
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  - host: go-demo-6.com
    http:
      paths:
      - backend:
          serviceName: go-demo-6
          servicePort: 80
{{- end }}
```

The key is in the first and the last line that envelops the whole `YAML` inside a conditional statement. **Ingress** will be created if the `Namespace` is `cd-production` (or `jx-production`, or whatever is the condition).

If we would push that `YAML` to the repository and deploy a new release to production, we would end up with two addresses through which the application

could be accessed. One would be the same dynamically created URL, and the other would be the **Ingress** we just defined. If we want to remove the former, we'd need to get rid of `service.annotations` defined in charts/go-demo-6/values.yaml. The `fabric8.io/expose` tells Jenkins X that it should dynamically expose our application. But we cannot simply remove it from the application chart since that would affect deployments to all the environments, not only production. Instead, we should go to the production environment repository and add values that would overwrite those annotations. That way, dynamic exposing would work anywhere but in production.

Let's wrap up this discussion and free up the used resources in the next lesson.