Article
Eduard Lebedyuk · Mar 13, 2018 4m read

Continuous Delivery of your InterSystems solution using GitLab - Part III: GitLab installation and configuration

In this series of articles, I'd like to present and discuss several possible approaches toward software development with InterSystems technologies and GitLab. I will cover such topics as:

- Git 101
- Git flow (development process)
- GitLab installation
- GitLab Workflow
- Continuous Delivery
- **GitLab installation and configuration**
- GitLab CI/CD

In the first article, we covered Git basics, why a high-level understanding of Git concepts is important for modern software development, and how Git can be used to develop software.

In the second article, we covered GitLab Workflow - a complete software life cycle process and Continuous Delivery.

I this article we'll discuss:

- GitLab installation and configuration
- Connecting your environments to GitLab

GitLab installation

We're going to install GitLab on premise. There are many ways to install GitLab - from source, package, in a container. I'm not actually going to describe all the steps here, there is a guide for that. But still, some notes.

Prerequisites:

- Separate server - since it's a web application and a fairly resource intensive it's better to run it on a separate server
- Linux
- (Optional but strongly recommended) Domain - required to run pages and secure the whole setup

Configuration

First of all, you'd probably need to send emails with notifications.

Next, I recommend installing Pages. As we discussed in the previous article - artifacts from the script can be uploaded into GitLab. User can download them but it's useful to be able to open them directly in browser and for that we need pages.

Why do you need pages:

- To display some generated wiki or a static page for your project
- To display html artifacts
- Other stuff you can do with pages

And as html pages can have an onload redirect, they can be used to send the user wherever we need. For example here's the code that generates html page which sends a user to the last executed unit test (at the moment of html generation):

```
ClassMethod writeTestHTML()
{
  set text = ##class(%Dictionary.XDataDefinition).IDKEYOpen($classname(), "html").Dat
a.Read()
  set text = $replace(text, "!!!", ..getURL())

  set file = ##class(%Stream.FileCharacter).%New()
  set name = "tests.html"
  do file.LinkToFile(name)
  do file.Write(text)
  quit file.%Save()
}

ClassMethod getURL()
{
  set url = "http://host:57772"
  set url = url _ $system.CSP.GetDefaultApp("%SYS")
  set url = url_"/%25UnitTest.Portal.Indices.cls?Index="_ $g(^UnitTest.Result, 1) _ "
&$NAMESPACE=" _ $zconvert($namespace,"O","URL")
  quit url
}

XData html
{
<html lang="en-US">
  <head>
    <meta charset="UTF-8"/>
    <meta http-equiv="refresh" content="0; url=!!!"/>
    <script type="text/javascript">window.location.href = "!!!"</script>
  </head>
  <body>
    If you are not redirected automatically, follow this <a href='!!!'>link to tests<
/a>.
  </body>
</html>
}
```

There's is a bug I encountered using pages (502 error when browsing artifacts) - here's a fix.

Connecting your environments to GitLab

To run CD scripts you need environments - configured servers to run your application. Assuming you have a Linux server with InterSystems product installed (let's say InterSystems IRIS, but it would work with Caché and Ensemble too) these steps would connect environment to GitLab:

1. Install GitLab runner
2. Register runner with GitLab
3. Allow runner to call InterSystems IRIS

Important note on installing GitLab runner - DON'T clone servers after you installed GitLab runner. Results are unpredictable and most unwelcome.

Register runner with GitLab

After you run the initial:

```
sudo gitlab-runner register
```

you'd be presented with several prompts and While most of the steps are fairly straightforward several are not:

Please enter the gitlab-ci token for this runner

There are several tokens available:

- One for the whole system (available in administration settings)
- One for each project (available in project settings)

As you connect a runner to run CD for a specific project you need to specify a token for this project.

Please enter the gitlab-ci tags for this runner (comma separated):

In your CD configuration, you can filter which scripts run on which tags. So in the most simple case specify one tag which would be environment name.

Please enter the executor: ssh, docker+machine, docker-ssh+machine, kubernetes, docker, parallels, virtualbox, docker-ssh, shell:
docker

If you're using the usual server without docker choose shell. Docker would be discussed in the later parts.

Allow runner to call InterSystems IRIS

After connecting runner to GitLab we need to allow it to interact with InterSystems IRIS, to do that:

1. **gitlab-runner** user should be able to call csession. To do that add him to cacheusr group:
    - `usermod -a -G cacheusr gitlab-runner`
2. Create **gitlab-runner** user in InterSystems IRIS and give him roles to do CD tasks (write to DB, etc)
3. Allow OS-level authentication

For 2 and 3 other approaches can be used, such as passing user/pass but I think that OS authentication is preferable.

Conclusion

In this part:

- Installed GitLab
- Connected environments to GitLab

Links

- Installation instructions
- Pages
- Runner
- Part I: Git
- Part II: GitLab workflow

What's next

In the next part, we'd write our Continuous Delivery configuration.

#Beginner #Continuous Integration #Deployment #Git #System Administration #Caché

Source URL: https://community.intersystems.com/post/continuous-delivery-your-intersystems-solution-using-gitlab-part-iii-gitlab-installation-and