Article
Eduard Lebedyuk · Mar 26, 2018 5m read

Continuous Delivery of your InterSystems solution using GitLab - Part V: Why containers?

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
- **Why containers?**
- GitLab CI/CD using containers

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.

In the third article, we covered GitLab installation and configuration and connecting your environments to GitLab

In the fourth article, we wrote a CD configuration.

In this article, let's talk about containers and how (and why) they can be used.

This article assumes familiarity with the concepts of docker and container. Check out these articles by @Luca Ravazzolo if you want to read about containers and images.

Advantages

There are many advantages to using containers:

- Portability
- Efficiency
- Isolation
- Lightweight
- Immutability

Let's talk about them in detail.

Portability

A container wraps up an application with everything it needs to run, like configuration files and dependencies. This enables you to easily and reliably run applications on different environments such as your local desktop, physical servers, virtual servers, testing, staging, production environments and public or private clouds.

Another part of portability is that once you built your Docker image and checked that it runs correctly, it would run anywhere else that runs docker, which are Windows, Linux and MacOS servers today.

Efficiency

You really only need your application process to run not all the system soft, etc. And containers provide exactly that - they run only the processes you explicitly need and nothing else. Since containers do not require a separate operating system, they use up less resources. While a VM often measures several gigabytes in size, a container usually measures only a few hundred megabytes, making it possible to run many more containers than VMs on a single server. Since containers have a higher utilization level with regard to the underlying hardware, you require less hardware, resulting in a reduction of bare metal costs as well as datacenter costs.

Isolation

Containers isolate your application from everything else, and while several containers can run on the same server they can be completely independent of each other. Any interaction between containers should be explicitly declared as such. If one container fails it doesn't affect others and can be quickly restarted. Security also benefits from such isolation. For example, exploiting web server vulnerability on a bare metal can potentially give an attacker access to the whole server, but in the case of containers, attacker would only get access to web server container.

Lightweight

Since containers do not require a separate OS, they can be started, stopped or rebooted in a matter of seconds which speeds up all related development pipelines and time to production. You can start working sooner and spend zero time on configuring.

Immutability

Immutable infrastructure is comprised of immutable components that are replaced for every deployment, rather than being updated in-place. Those components are started from a common image that is built once per deployment and can be tested and validated. Immutability reduces inconsistency and allows replication and moving between different states of your application with ease. More on immutability.

New possibilities

All these advantages allow us to manage our infrastructure and workflow in the entirely new ways.

Orchestration

There is a problem with bare metal or VM environments - they gain *individuality* which brings many, usually unpleasant, surprises down the road. The answer to that is **Infrastructure as code** - management of infrastructure in a descriptive model, using the same versioning as DevOps team uses for source code.

With Infrastructure as Code a deployment command always sets the target environment into the same configuration, regardless of the environment ’ s starting state. It is achieved by either automatically configuring an existing target or by discarding the existing target and recreating a fresh environment.

Accordingly, with Infrastructure as Code, teams make changes to the environment description and version the configuration model, which is typically in well-documented code formats such as JSON. The release pipeline executes the model to configure target environments. If the team needs to make changes, they edit the source, not the target.

All this is possible and much easier to do with containers. Shutting down container and starting a new one takes a few seconds while provisioning a new VM takes a few minutes. And I'm not even talking about rolling back a server into a clean state.

Scaling

From the previous point, you may get an idea that infrastructure as code is static by itself. That's not so, as orchestration tools can also provide horizontal scaling (provisioning more of the same) based on current workload. You should only run what is currently required and scale your application accordingly. It can also reduce costs.

Conclusion

Containers can streamline your development pipeline. Elimination of inconsistencies between environments allows for easier testing and debugging. Orchestration allows you to build scalable applications. Deployment or rollback to any point of immutable history is possible and easy.

Organizations want to work at a higher level where all the above listed issues are already solved and where we find schedulers and orchestrators handling more things in an automated way for us.

What's next

In the next article, we'll talk about provisioning with containers and create a CD configuration that leverages InterSystems IRIS Docker container.

#Change Management #Containerization #Continuous Integration #Docker #Caché