

WORKSHOP LPI DEVOPS TOOLS

15:35h a 16:40h

Integración y despliegue continuo mediante contenedores Docker sobre Kubernetes, Openshift y Jenkins

Ponente: Sergio Rodríguez de Guzmán, CTO at PUE



Linux
Professional
Institute

LPI Certification Programs





Linux
Professional
Institute

Software Development
Configuration Management
Log Management
Container
Virtual Machines
Monitoring
SCM
Cloud Instances
Cloud
Microservices
System Images
Immutable Servers
IT Operations
Cloud Services
Continuous Delivery
12 Factor App

So what?

- What does that mean for you?
- What does that mean for our candidates?
- How long will we need the Jack-of-all-trades sysadmin?
- Is it worth investing time in learning these things?

- To answer this we have to discover ...
 - ... what these buzzwords really mean
 - ... what their technological impact is
 - ... how we can embrace the relevant portion of these tools

- Let's take a tour through DevOps Tools Land!

HOW IT USED TO BE

The (not so) old days

- Companies buy computers / servers
 - Probably put some virtualization on them
 - Admins install os / prepare and enroll templates
 - New servers / VMs require manual configuration
 - There is usually a process for new servers / VMs
- Software is packaged with installation routines
 - Packages tend to be large
 - Specific requirements on the runtime environment
 - Conflicts with other installed software possible
 - Post-installation configuration required

The (not so) old days

- Implications
 - Admins do a lot of repetitive, boring work
 - Setting up new servers / VMs is hard
 - Testing systems live long
 - Prod / Dev parity
 - Updates / releases require downtime and work
 - Releases are risky and complicated
 - Bugs require time to get fixed
 - Large releases
 - It's all about servers
 - Stored data
 - “Nursing” sick servers back to health

Devs and Ops

- Administrators and Developers have contradictory requirements:
 - Daily fresh testing systems vs. repetitive work
 - Frequent deployments vs. uptime
 - Blaming platforms vs. blaming software
- Both have the same goals:
 - Deliver a service to clients...
 - ... without emergency calls at 4am in the morning

DevOps

- DevOps stands for collaboration and shared responsibility
- The same team develops and operates a service
 - Software is designed to be operational
 - Operations is aligned to software lifecycles
- This collaboration needs an implementation
 - High standardization
 - Need for a common set of tools
 - New skills for both developers and operators

Classes of tools

- There are several classes of tools which help to implement DevOps:
 - Configuration Automation
 - Machine Deployment
 - Container
 - Modern Software Architecture and Development, Continuous Integration / Delivery
 - Monitoring and Log Management
- These tools will significantly change the way IT works
- You should know them well enough to (not) like them

CONFIGURATION AUTOMATION

Manual Configuration

- Classic system administration involves
 - Issuing commands on servers
 - Using an editor to change config files
 - Review log files
- How to ...
 - ... keep multiple servers in sync?
 - ... verify a server is configured correctly?
 - ... keep track of changes?
 - ... configure a new server within minutes?
- If you maintain three servers manually in the same way, sooner or later they will end up being different

Configuration Automation

- Definition of how a server should be configured
 - Actions to be performed
 - Description of states
 - Examples:
 - SSH keys are in place
 - Package apache2 is installed
 - File index.html exists with a specific content
 - php.ini contains certain parameters
- Automate configuration
 - Shell scripts
 - Specialized tools
 - Try to never make ad-hoc changes again

Configuration Automation

- Configuration descriptions
 - are usually maintained as text files
 - use tool-specific tasks (e.g. “install a package”, “change a value in an ini file”, “start a service”)
 - can be split into functional elements (e.g. “web server”, “database server”)
 - support variables, secrets and templates
 - All this can be put under version control
- Assign a general configuration description to specific target nodes

Apply configurations

- Agents on the client
 - determine the node's current status
 - apply necessary changes to the node
 - avoid unnecessary changes
- Push approach
Server or management node contacts client nodes
- Pull approach
Client nodes periodically check for changes
- Idempotence
Run it as often as you want

Configuration Automation Tools

- **Puppet**

- Usually server / client based, initial setup effort
- Comprehensive, but rather complex, Ruby SDL
- Strong use of dependencies can be confusing

- **Chef**

- Usually server / client based, initial setup effort
- Comprehensive, but rather complex, Ruby SDL
- Programming skills make Chef a lot easier

- **Ansible**

- No central server, only SSH + Python on the nodes
- YaML configuration, no real programming
- Easy to get started

Benefits of Configuration

Automation

- All installation and configuration steps in code
- Developers and operators can optimize this code for all their needs
- Bundle configuration code with releases
- Configuration can be applied to numerous servers automatically
 - Server configurations can be validated
 - Environment specific configuration
 - Transparency due to versioning

MACHINE DEPLOYMENT

System Images

- Usually contain an operating system
- Used to install on client computers and cloud instances
- Cloud Providers
 - need to adapt images to their clouds
 - want to use one image for all instance types
- Administrators
 - need to configure the images
 - want to reduce the provisioning time
- Software vendors
 - need to bundle their software as appliances
 - want to ship one image to all kinds of clients

cloud-init

- Cloud providers need to adapt images to their clouds
 - Use one image in different types of instances
 - Resize disks and initialize storage
 - Inject SSH keys
 - Configure networking
- cloud-init
 - Used by most large cloud providers
 - Installed in the image, executed at boot time
 - Retrieves “vendor-data” and “user-data”
 - Executes commands and statements to prepare the instance
 - “user-data” can be used for custom configuration

Packer

- Packer creates custom system images
- Create images that are immediately usable
- Shift effort from deployment time to build time
- Deploy images to clients or public clouds
- Build the image by
 - modifying an existing image
(use an image that is already integrated into its target cloud!)
 - running an unattended installation

Packer

- Provision the image in different ways
 - Execute commands, install packages, create files, ...
 - Execute Ansible, Puppet, Chef, ...
- Pack the resulting image and make it available for new instances
- Building and provisioning is decoupled
 - Maintain one set of provisioning instructions
 - Apply to multiple provisioners, e.g. in different clouds

Vagrant

- Running images locally, mostly for development
- Vagrant manages VMs
 - Retrieve images from central repositories
 - Standard configuration scheme
 - Applies the configuration to different hypervisors
 - Sets up shared directories, port forwards, ...
- Eases the handling of VMs
 - Easy deployment
 - Simple start and stop
 - Can handle multiple VMs at a time

Benefits of system images

- Images can be prepared to be used without further configuration
 - Fast Deployments
 - Scaling
- Run the similar images in staging and production environment
- Use **Packer** to create images which ...
 - ... are available for different cloud providers
 - ... adapt to their runtime using **cloud-init**
 - ... are available to developers using **Vagrant**
- Ship appliances to clients

CONTAINERS

Containers

- VMs are heavy
 - Fixed resource allocation
 - A lot of supporting services
 - Long boot time
- “One application per server”
- Avoid conflicts between applications
- Linux namespaces allow simple process isolation
- Several implementations
 - “VPS style”: LXC, OpenVZ, Linux VServer
 - “Container style”: Docker, rkt

Docker

- Application container
 - Usually only one (main) process per container
 - Throw away containers
 - Volumes provide persistence
- Docker is more than just process isolation
 - Retrieve / upload images from / to a registry
 - Build / adjust images according to a Dockerfile
 - Connect containers to the network, forward ports
 - Provide access to persistent volumes
- Easy setup with docker-machine

Container Orchestration

- Applications are based on multiple containers
Database, Message Broker, Worker, ...
- Containers need to be started in certain combinations
Order, redundancy / replication, scaling, ...
- Applications need to be managed
Links between containers, service discovery, load balancing, ...
- Docker nodes need to be coordinated
Distribute containers, image availability, volumes, overlay networks, central API, ...

Docker Approach

- Docker Swarm
 - Builds a cluster of multiple Docker nodes
 - Multi-host networking, load balancing, service discovery, rolling updates
- Docker Compose
 - Defines how multiple container work together to provide a “service”
 - Manages volumes, networks and port forwards
 - Starts and coordinates these containers
 - Stack: Compose files can be deployed to a Swarm

Kubernetes

- Platform for container orchestration by Google
 - Forms a cluster out of multiple nodes
 - Provides an API to manage containers on the nodes
- Configuration elements
 - “Pods”
Container(s) which are always located together and share an IP address, can have volumes
 - “Controllers”
Manage the cluster and start pods
 - “Services”
Form applications out of multiple pods
 - Labels link these elements to each other

Openshift

- OpenShift Origin is a distribution of Kubernetes optimized for continuous application development and multi-tenant deployment.
- OpenShift adds developer and operations-centric tools on top of Kubernetes to enable rapid application development, easy deployment and scaling, and long-term lifecycle maintenance for small and large teams.
- OpenShift embeds Kubernetes and extends it with security and other integrated concepts. An OpenShift Origin release corresponds to the Kubernetes distribution - for example, OpenShift 1.7 includes Kubernetes 1.7.

Others Container Tools

- Mesos / Marathon
Task runner and container platform
- CoreOS Container Linux and RancherOS
Linux operating systems to build a container platform
- rkt
Alternative container tool
- systemd-nspawn
- flatpak
- “Old-School Container Tools”
OpenVZ, Linux VServer, FreeBSD Jails, Solaris Zones, ...



Benefits of Containers

- Containers contain the whole runtime environment
 - Relationship of containers in code
 - Including dependencies
 - Huge library of existing images
- Super-fast and safe deployment
 - Scale at runtime
- The same container images can ...
 - ... be used for development, testing and production
 - ... be used locally and in the cloud
 - ... parameterized with environment variables

**MODERN SOFTWARE
DEVELOPMENT**

CLOUD SERVICES

Software Platforms

- The tools discussed so far change
 - the deployment of software
 - the platform on which software runs
- To be beneficial, these changes have to be adopted
 - by applications and application architectures
 - by development and operations teams need
- Prominent approaches
 - 12 Factor App
 - Cloud Native Applications
 - DevOps

Software Platforms

- Deployment units get smaller
- Distributed systems already contain breakpoints
- Split an application into manageable pieces
 - Microservices
 - Stateless vs. Stateful services
- Leverage standardization
 - Use existing container / system images for services like databases, message brokers, ...
 - Most public clouds offer these things as a service
 - New classes of services, e.g. object storage, NoSQL

Software Development

- Put everything under version control
Program code, Dockerfiles, Ansible Playbooks, ...
- Continuous Integration and Deployment
 - Automate all steps from an SCM checkin, through building and testing to the deployment
 - Immediate feedback and overall status
 - Frequent and small releases
- Jenkins
Prominent CI/CD server, modular architecture
- DevOps

CI/CD requires skills from both “worlds”

Benefits of Modern Software Architectures

- Applications made for the cloud are easy to deploy
- Support containers and VMs
 - Specific places of persistent data ("Cats and Cattle")
 - Easy and fast scaling
- Standard services can be provided once for numerous applications
- CI/CD allows immediate testing and feedback
 - Use containers / VMs as build artifacts
 - Run the very same environment in testing and production

OPERATIONS

IT Infrastructure

- Even the cloud is composed of IT infrastructure
 - Computing nodes, database servers, storage, ...
 - Load balancing, service discovery, DNS
- Even if the cloud hides it well, an application relies on infrastructure
- Infrastructure can fail
 - Monitor your services, no matter where they run
 - Have a migration plan to another cloud or data center
 - Use modern deployment tools for disaster recovery (“Cats and Cattle”)

Logging

- All applications produce log files
- Log files are scattered around in VMs, containers, on multiple nodes and clouds
- SLG-Stack
 - Solr stores and searches log data
 - Logstash collects, normalizes and prepares log data
 - Grafana provides graphical access to the log data
- Ensure your logs are informative, well classified and in a structured format
- Log and monitor metrics that really reflect your service's health

Benefits of Logging and Monitoring

- Monitoring and logging provides essential information
- Good applications log senseful information
- One common “log warehouse” is beneficial for everyone
 - Operators are alerted of errors
 - Developers learn about application usage and behaviour

QUICK POLL



SAFE?

Are you using version control?

QUICK?

Can you release new version of your software in one day?

QUICK AND SAFE?

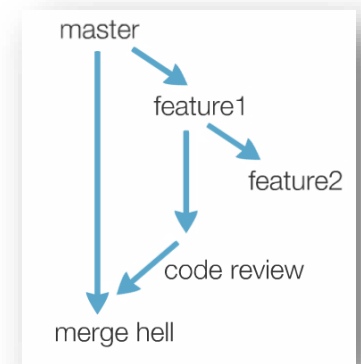
Can you release new, well-tested version of your software in one day?

DEVELOPMENT NOW



- **Each developer has feature branches**
 - *If the version control is used at all*
- **Features are deployed when completed**
- **Integration issues**

- **Small test suite**



PROBLEMS

- **Bringing software into production is hard**
- **Takes a lot of time**
- **Error prone**

SOLUTION – CONTINUOUS INTEGRATION



“Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.”

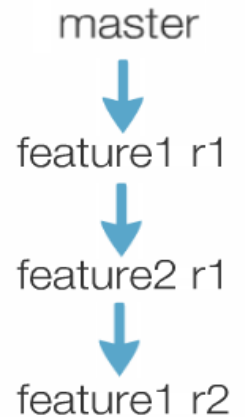
- Martin Fowler

CHANGE THE WORKFLOW!

- **Checkout/update**
- **Code**
- **Build & test locally**
- **Update (merge)**
 - Retest if changed
- **Commit**
- **Continuous Integration server takes over ...**

CHANGE THE VERSIONING!

- **No 'feature' branches**
- **Temporary 'developer' branches**
- **Good to test crazy ideas**
 - Branch and throw away
- **Trunk must always compile**
- **Avoid big scary merges**



- **No 'feature' branches**
- **Features can be toggled on and off via deployment or compilation configuration**
 - Also helps with Continuous Delivery/Deployment
- **Keep features small**
- **Improve features interactively**
 - Introduce early, then improve

SIDENOTE – VERSION CONTROL SYSTEMS

- **History**

- Log

- Blame

- **Revert to version**

- Bug-finding tool

- **Branches**

- Tag/mark every release

- **Always use VCS – even for single-person projects!**

- **Automate everything**

- If it hurts, do it more often. Continuously.

- Fail fast.

- **Integration testing**

- **Unit testing**

- **Functional testing**

- **Application testing**

- **Mobile testing**

- **Whatever you don't test against, will happen**

CONTINUOUS DELIVERY



“The essence of my philosophy to software delivery is to build software so that it is **always** in a **state** where it could be put into **production**. We call this *Continuous Delivery* because we are continuously running a *deployment pipeline* that tests if this software is in a state to be delivered.”

– Jez Humble, Thoughtworks

- **CD = CI + fully automated test suite**
- **Not every change is a release**
 - Manual trigger
 - Trigger on a key file (version)
 - Tag releases!
- **CD – It is all about testing!**

“How long would it take your organization to deploy a change that involves just one single line of code?”

- Mary and Tom Poppendieck,

Implementing Lean Software Development

CONT. DELIVERY VS. DEPLOYMENT

Continuous Delivery



Continuous Deployment



CONTINUOUS DEPLOYMENT



A WORD OF WARNING

- **Continuous delivery is doable.**
- **Continuous deployment is a hard problem.**

DEPLOYMENT SCHEDULE

- **Release when a feature is complete**
- **Release every day**

- **Zero-downtime deployment (and rollback)**
- **Blue-green**
 - Two environments
 - Install on one. Switch. Switch back on problems.
- **Canary release**
 - Deploy to subset of servers
- **Real-time application state monitor!**

PROBLEMS

- **Technical**

- Databases
 - Schema migration
 - Revert!
 - Change management software
- Configuration

- **Human**

- Even more important
- Automatic deployment = great fear
- Customers don't want software to constantly change

TRANSITION



- **Gain expertise**

- First step accomplished – you are here

- **Automate the build**

- **Introduce tests**

- **Prove the concept**

- Introduce CI system
- Run it in parallel to existing infrastructure
- Give it time
- Show the win-win

WHAT TO TAKE HOME?

Recap

- **Configuration Automation**
Put configuration into code and apply it automatically
- **Machine Deployment**
Prepare system images which are ready to run
- **Container**
Bundle and isolate runtime environments for a single application, orchestrate and scale containers
- **Modern Software Architecture and Development**
Software that leverages new tools, CI/CD
- **Operations**
Configure infrastructure, observe if services are up

Is it really worth it?

- New tools change the game, faster than ever
- Learning them takes time and effort
- The tools discussed today are actually used
 - Employers request them
 - More and more projects rely on them
 - Learning them frees you from boring tasks
- Think about your projects and your job:
Which of the tools discussed today could be beneficial for your daily life?

Suggestion on tools to start

- **Configuration Automation**

Ansible

- **Machine Deployment**

Vagrant, Packer, cloud-init

- **Container**

Docker, Docker Swarm, Docker Compose, Kubernetes

- **Modern Software Architecture and Development**

Git, Jenkins, build and testing tools for your platforms




- **Operations**

SLG-Stack, your (current?) monitoring system



BARCELONA – MADRID
www.pue.es

¡Gracias!

-  #PUEDAY18
-  educación@pue.es
-  93 206 02 49

