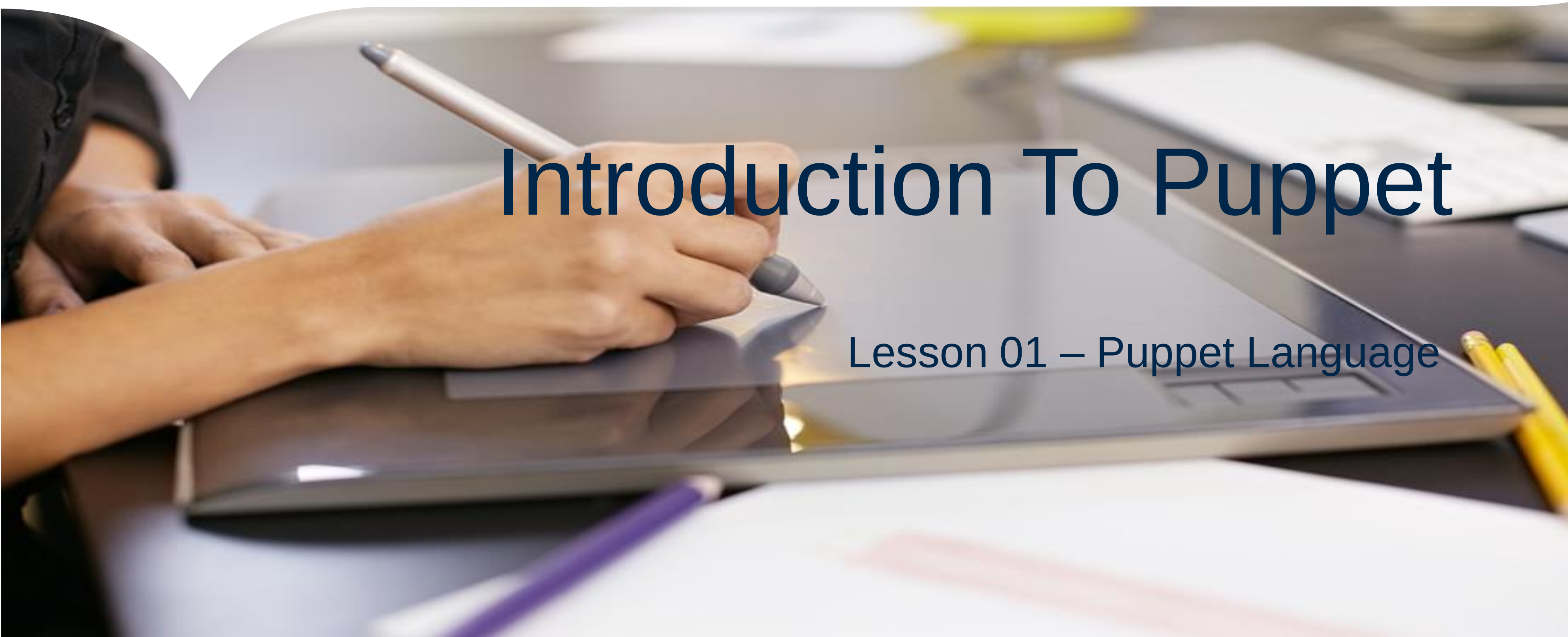


Introduction To Puppet

Lesson 01 – Puppet Language

Lesson Objectives

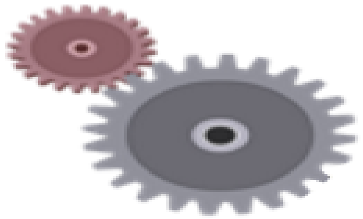
- Setting the Context for Puppet
  - IT world Challenges
  - Emergence of Puppet
- What is Puppet?
  - Infrastructure Management Platform
  - Declarative Configuration Language
- Understanding Puppet Language
  - Resources
  - Types and Providers
  - Sample Puppet Code
  - Classes, Manifests and Modules

Setting the context for Puppet : IT world Challenges

- The IT world challenges
  - Developers
  - QA Team
  - Operations Team
  - Environment Silos
  - No software can be a solution, this needs change in culture

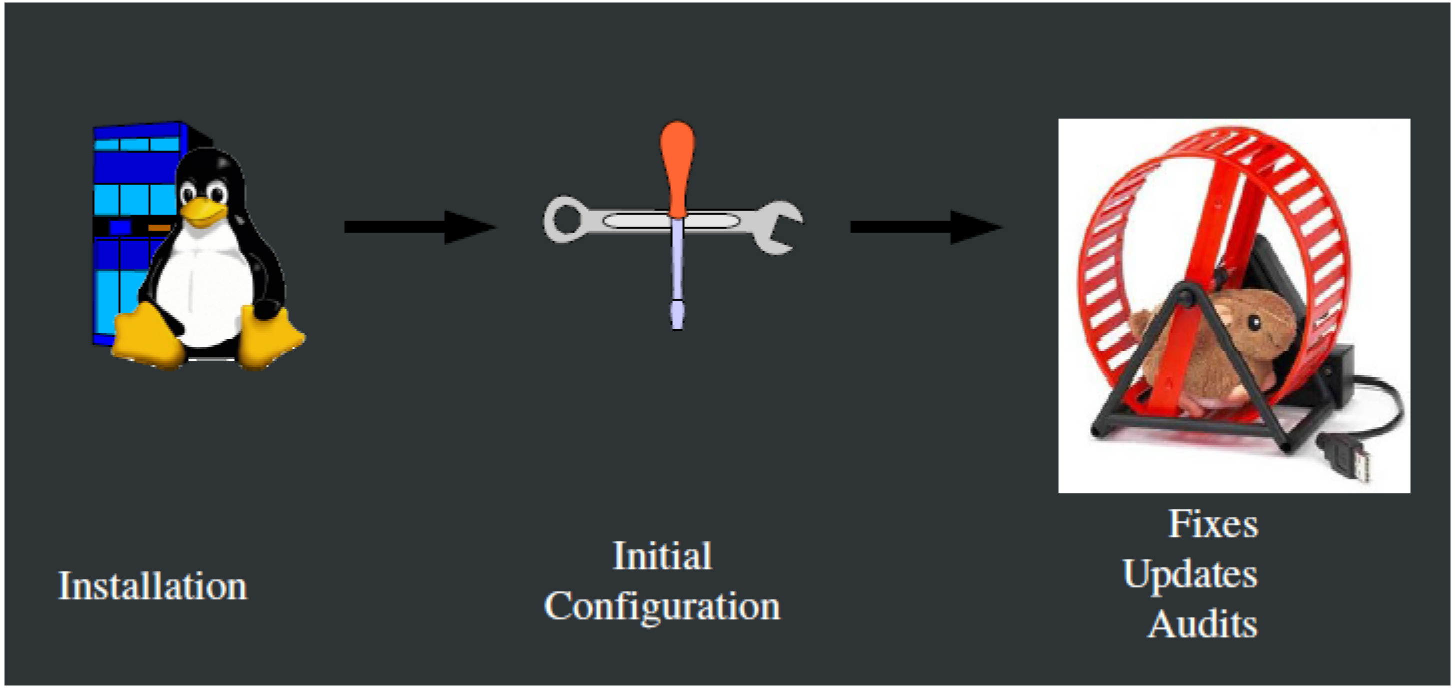
Dev Environment

Transition



Ops Environment

Typical System Life Cycle

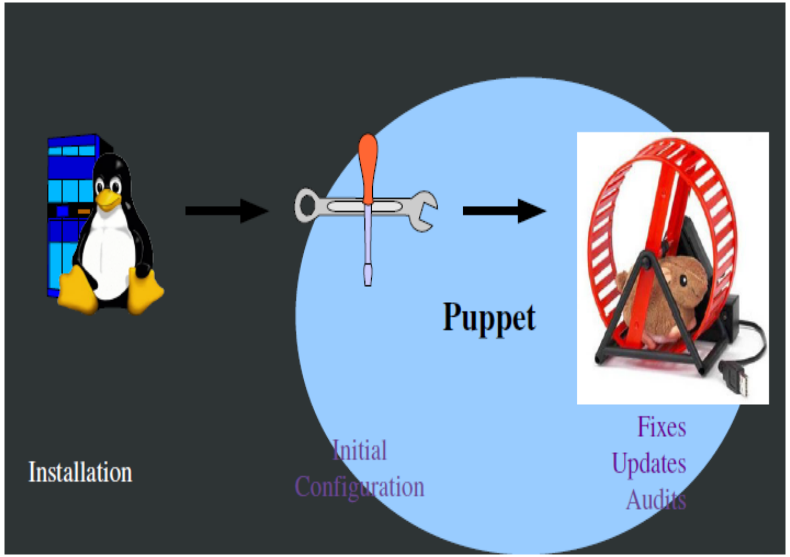


The Challenges

- Keep our systems "harmonized"
- Know whats going on on each system
- Replace a server if it dies or to be able to add another server that is exactly like it
- Similar Applications, different OS's
- Push out changes to all the servers that need a particular change
- Stop duplicating effort
- Go home early

How to solve the problem

- ● The Manual way
  - Log in and do it
  - Thats OK only for a limited set of machines...
- ● Install time auto configure
  - Kickstart, jumpstart etc, with a post installation scripts
- ● But then what?
  - How to push a change?
  - No History of changes, audit etc...
- Or....



Setting the context for Puppet : Emergence of Puppet

- The dissatisfaction and frustration of such an ops team member, Luke Kanies who was stuck working late nights in a data center, led him to write the software that became what we know it today as Puppet

- Puppet deals with only part of the IT world challenges by providing for Infrastructure Management

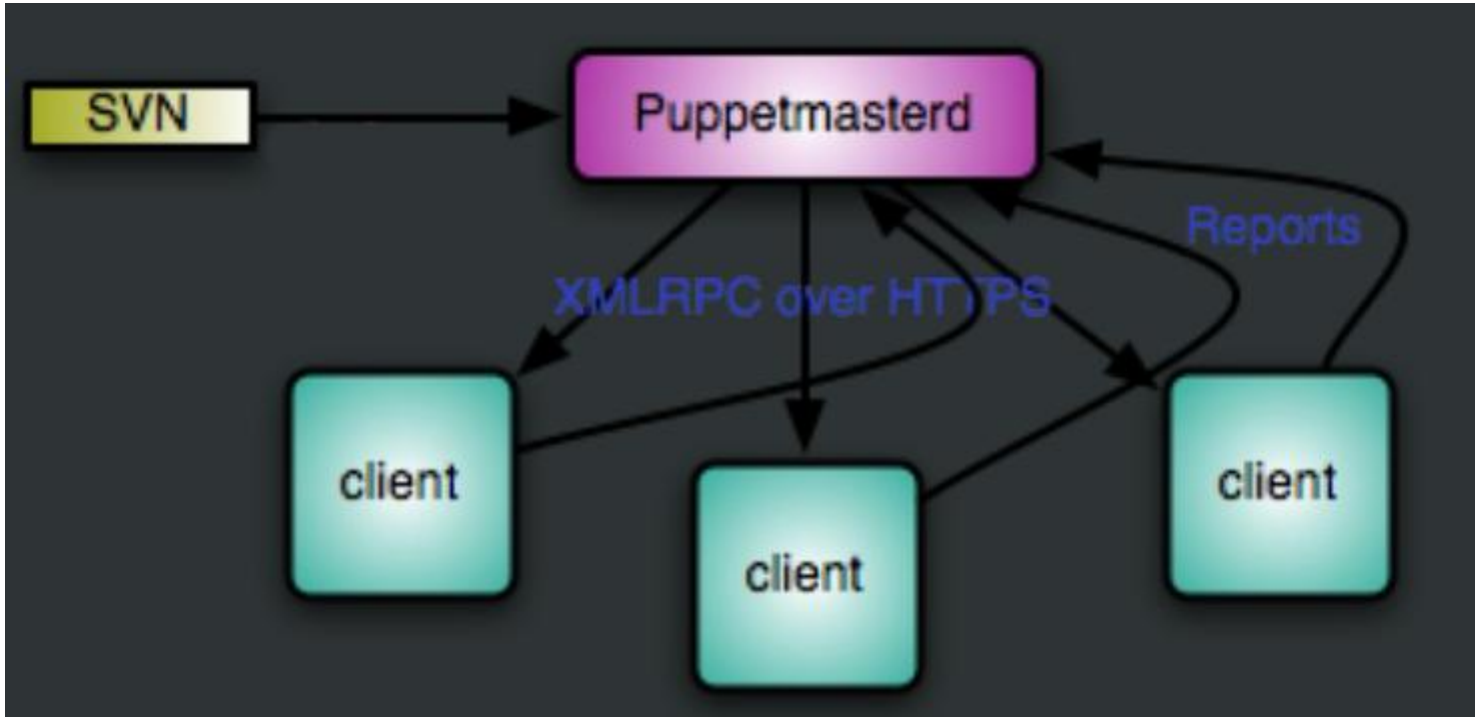
What is Puppet

- A GPL Open Source Project written in Ruby
- A declarative language for expressing system configuration
- A Client and server
- A library to realize the configuration
- Puppet is the abstraction layer between the system administrator and the system
- Puppet requires only Ruby and Facter
- Client runs every 30 minutes by default

What is Puppet

- Puppet is an Infrastructure management platform
- It can work on multiple operating systems based on Unix as well as on Microsoft Windows
- Puppet is Infrastructure management platform that allows automation of repeated tasks and it also includes its own declarative language to describe system configuration, the Puppet Language that is based on Ruby
- Puppet is arranged in a **master – agent architecture**. The master serves the manifests and files, and the agents poll the master at specific intervals of time to get their configuration. The master does not push anything into the client.
- Agents identify with the master using SSL, so the first time an agent tries to connect to the master, the agent certificate needs to be approved (in the default configuration), and that’s usually a source of problems.

Puppet Components



Understanding Puppet Language

- Puppet is a modeling language used to automate Infrastructure management
- Puppet language enables declarative Infrastructure management
- Using Puppet language you simply describe the end state of your nodes
- Puppet's language works across different platforms
- It abstracts state of a node away from implementation
- It allows code to manage systems easy to all role holders
- Teams can collaborate better, people can manage resources that would normally be outside their acquaintance, promoting shared responsibility amongst teams
- If the system is already in its desired state, Puppet will leave it in that state

Puppet Resources

- Puppet language is used for declaration of Puppet resources
- Resource describe component of a system
- Examples
  - A user account
  - A specific file
  - A directory of files
  - Any software package
  - Any running service
- Resources are used to model the desired state of the systems to be managed

Puppet Resource Types and Providers

- Puppet groups similar kinds of resources into types
- In order to configure a resource, we need to correctly describe its resource type and declare the desired state for that resource
- Providers implement resource types on a specific kind of system, using the system’s own tools
- Separation of types from their providers allows a single resource type to manage tasks on many different systems
- Provider, these are often simple Ruby wrappers around shell commands, so they are usually short and easy to create
- Types and providers enable Puppet to function across all major platforms, and allow Puppet to grow and evolve to support additional platforms beyond compute servers, such as networking and storage devices

Sample Puppet Code

Imperative shell code

```
if [ 0 -ne $(getent passwd elmo > /dev/null)$? ]
then
    useradd elmo --gid sysadmin -n
fi

GID=`getent passwd elmo | awk -F: '{print $4}'`
GROUP=`getent group $GID | awk -F: '{print $1}'`

if [ "$GROUP" != "$GID" ] && [ "$GROUP" != "sysadmin" ]
then
    usermod --gid $GROUP $USER
fi
```

```
if [ "`getent group sysadmin | awk -F: '{print $1}'`" == "" ]
then
    groupadd sysadmin
fi
```

Declarative Puppet Code

```
user { 'elmo':
  ensure => present,
  gid    => 'sysadmin',
}
```

```
group { 'sysadmin':
  ensure => present,
}
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Puppet Classes

- Classes are Puppet’s way of separating out chunks of code, combining resources into larger units of configuration
- A class could include all the Puppet code needed to install and configure NTP, for example. Classes can be created in one place and invoked in another.
- Different sets of classes are applied to nodes that serve different roles. We call this “node classification” and it’s a powerful capability that allows you to manage your nodes based on their capabilities, rather than based on their names.

Puppet Manifests

- Puppet language files are called **manifests**
- The simplest Puppet deployment is a lone manifest file with a few resources
- If we were to give the basic Puppet code in the previous example the filename “user-present.pp,” that would make it a manifest
- Puppet configuration files are called manifests, written in a ruby-like Domain Specific Language
- Puppet provides types and functions to manage typical resources (files, services, users, groups,…) and new ones can be defined through extensions called modules

Puppet Modules

- Puppet Modules are a collection of classes, resource types, files, and templates, organized around a particular purpose and arranged in a specific, predictable structure
- There are modules available for all kinds of purposes, from completely configuring an Apache instance to setting up a Rails application, and many, many more
- Including the implementation of sophisticated features in modules allows admins to have much smaller, more readable manifests that simply call modules.
- One huge benefit of Puppet modules is that they are reusable
- One can use modules written by other people, and Puppet has a large, active community of people who freely share modules they've written

Lesson Summary

- The Context of using Puppet
- What Puppet is as a platform and language
- Understanding Puppet Language
  - Resources
  - Types and Providers
  - Sample Puppet Code
  - Classes, Manifests and Modules