Some takeaway from the Getting Start with Puppet Course

Role and Profile

https://puppet.com/docs/pe/latest/the_roles_and_profiles_method.html

Role = Business classification e.g. Database Server vs. Web Server role.
Roles only ever contain *include* statements which reference profiles. i.e.

```
node someHost {
  include role::something
}
```

Profile = Technical classification e.g. database security profile, MySQL profile.
Profiles reference modules and pull in data from a selection of potential sources. Essentially, a Profile is a collection steps to achieve a specific technical goal.

Inject data in the Profile level and not the Role level or the module level!!!

Modules = A structured piece of functionality e.g. install php, add an admin user etc.
Modules are where user written puppet code resides.

Some useful puppet commands

#To show the different types of resources

```
puppet resource –types
```

run adhoc commands using resources

```
puppet resource user bob ensure=present
```

use describe to show help about that resource

```
puppet describe user
```

Query puppet agent status

```
puppet config print
```

Or query a specific parameter e.g. statedir path

```
puppet config print statedir
```

list the certificates needing to be signed. Use the –all switch to list all certs on the master.

```
puppet cert list
```

Sign a node cert

```
puppet cert sign nameOfNode
```

remove a node from the puppet master, including certs etc

```
puppet node purge nameOfNode
```

list installed modules and display module folder

```
puppet module list
```

find puppet modules in the forge

```
puppet module search someName
```

install a puppet module from the forge

```
puppet module install moduleName
```

lookup hiera data

```
puppet lookup message --explain
```

PDK related

To start creating a new resource/class/profile always use pdk to create the skeleton code for you!

```
pdk new module foo
cd /foo
pdk new class bar
```

Once the object is created by PDK, it will be able to validate its syntax

```
pdk validate manifest/bar.pp
pdk validate -a xxx    #can attempt to auto fix syntax error!
```

Once the syntax is correct, it can also test compile it

```
cd <module folder>
pdk test unit
```

When creating a new object, PDK uses a set of standard templates like the type of license to be used etc. These templates can be customized:

Default one is here: https://github.com/puppetlabs/pdk-templates

Pull down the default one, change it as it needs the standard folder structure, then use the pdk new module --template-url <path> to refer to it

Code Manager

https://puppet.com/docs/pe/2017.3/code_mgr_how_it_works.html

Code manager is the mechanism for deploying and syncing the code within the Puppet infrastructure using r10k

To find out the current code_manager settings:

PE Console -> Classification -> PE Infrastructure -> PE Master -> Configuration

Code Deployment

After checking in the code to git, the code needs to be deployed to Puppet Master for it to become live! This can be done using githook but the underlying commands are:

```
puppet access login -l 1h   #get a login token that last for 1 hour
puppet code deploy production –w    #deploy the production environment
```

Hiera data structure

https://puppet.com/blog/hiera-data-and-puppet-code-your-path-right-data-decisions

Hiera is for dynamic hidden secret, not static data!!

Yes – Password, keys
No – standard mysql path

Hiera store data at 3 levels, to show the 3 locations:

```
"sudo puppet lookup message --explain"
#need sudo as otherwise puppet cannot access the folders
```

And the Hiera data is as simple as a text file with key: value pair e.g.:

```
---
motd: 'Hello World! from %{name}'
#%{name} is another Hiera data to be substituted in at run time
```

Hierarchy entry "Other YAML hierarchy levels"

Path "/etc/puppetlabs/code/environments/production/data/common.yaml"

Original path: "common.yaml"

Interpolation on "Hello World! from %{name}"

Global Scope

Found key: "name" value: "main"

Found key: "motd" value: "Hello World! from main"

Puppet Coding

Environment/Group -> Role -> Profile -> Module

A good example of puppet code is Puppetlabs-MySQL:

https://github.com/puppetlabs/puppetlabs-mysql

When writing code –put Include before Declare

A good analogy for the definition vs. declaration

definition = pizza menu (define the options)
declaration = ordering a pizza (execute the option)

Accessing parameter within a class

$ssh::server

Referencing

Refer only between objects already defined within the module e.g. even Root will exist in the system, but if it is being referenced, need to have a ensure declaration for root in the module!

Addressing puppet resource from within the Files folder of a module.

puppet:///modules/profile/puppet_webapp.jar
the third / refer to the puppet master, modules is a mount point

A generic resource definition

```
Type {'Title'
  Param => $var
}
```

define resource vs class

class can only be applied once in a node, a define is reusable
e.g. MySQL DB as a defined resource

Control-Repo

- Control-repo is an object, which contains multiple environments.
- We are editing the Site environment – it is just a name, it can be called anything!
- Which contains two modules - Profile and Role
- Puppetfile declares which module to be deployed, or which external module to be included e.g. Time
- Each module has examples and manifests folders as standard structure (use pdk to generate them!)
- Manifests contains the definitions
- Examples contain the declarations

Additional notes

There is an open source Windows Package Manager works like Yum for windows called Chocolatey – Chocolate is yum!

https://chocolatey.org/docs/installation

Install Puppet Enterprise Client tools to enhance control (Think VMWare Tools)

https://puppet.com/docs/pe/2017.3/installing_pe_client_tools.html

There is no such concept as local roll back – once Puppet applied something, there is no default/automatic action to undo. To “Undo” you will need to prepare an undo task and define the steps that you see fit to undo the effect and then apply that task.