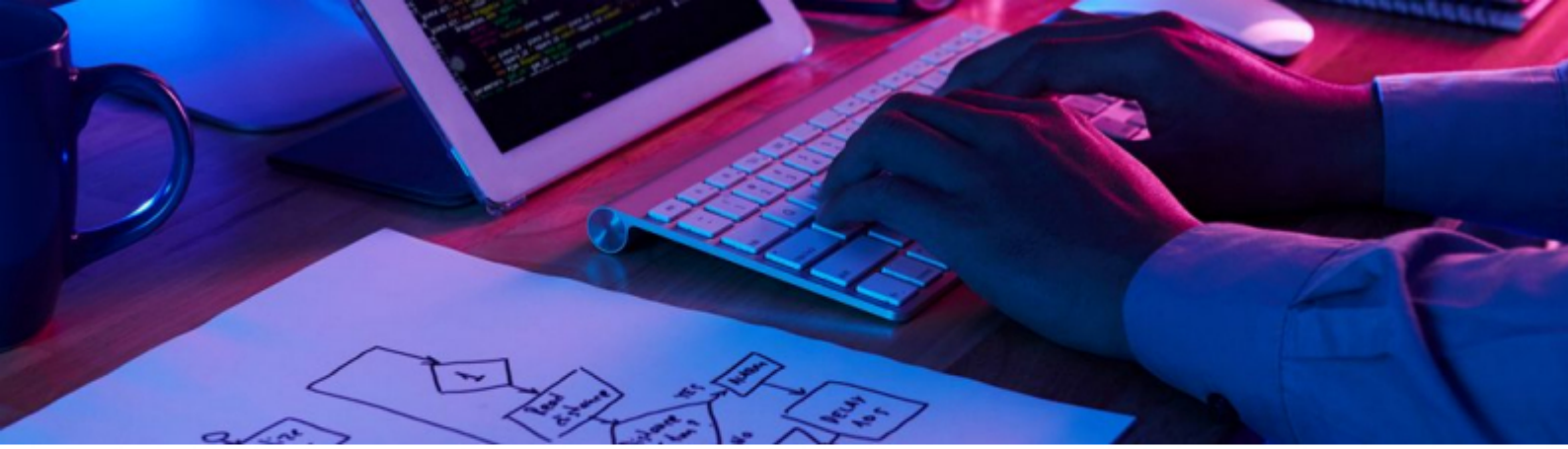


Configuration Management and the Cloud

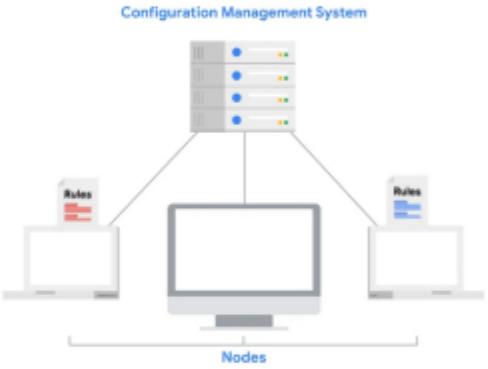
Course 5 of 6 in Google IT Automation with Python Specialization.

bangkit

Automating with Configuration Management

Introduction to Automation at Scale

- Being able to **scale** means that the system able to keep achieving larger impacts with the same amount of effort. In short, a **scalable system** is a flexible system. **Automation** is an essential tool for keeping up with the infrastructure needs of a growing business. Automation is what lets the system scale.
- A configuration management tool can take defined rules and apply them to the system that it manages, making changes efficient and consistent. Some popular **Configuration Management System** include Puppet, Chef, Ansible, and CFEngine.



Introduction to Automation at Scale

- Using **Infrastructure as Code** (IaC) means all the configuration necessary to deploy and manage a node in the infrastructure stored in version control. So the fleet of nodes are consistent, versioned, reliable, and repeatable.
- It brings all the benefit of version control, include audit trail of changes, quickly rollback, review mechanism and collaboration.
- Automation and configuration management can help the IT team adapt and stay flexible to embrace the technology industry which constantly changing and evolving.

Introduction to Puppet

- Puppet is a cross-platform, open source configuration management system, created in 2005.
- Puppet is the currently industry standard for managing the configuration of the computers in a fleet of machines.
- In the client-server architecture, the **client is Puppet agent** and the **server is Puppet master**.

Introduction to Puppet

- With client-server architecture:
  - The agent connects to the master and sends facts that describe the computer.
  - The master then processes the information, generates the list of rules and send the list back to the agent.
  - Then the agent making any necessary changes on the computer.

Simple Puppet Rule

```
class sudo {

  package { 'sudo':
    ensure => present,
  }

}
```

Introduction to Puppet

- In Puppet, **resources** are the basic unit for modeling the configuration to manage. Each resource specifies one configuration to manage, like a service, a package, or a file.
- For example, a file resource, in simple rule that ensures that /etc/sysctl.d exists and is a directory. The **resource title** is '/etc/sysctl.d', written before colon (:). Followed by **attributes** to set for the resources.

Simple Rule with File Resource

```
class sysctl {

  # Make sure the directory exists
  file { '/etc/sysctl.d':
    ensure => directory,
  }

}
```

Introduction to Puppet

- The Puppet agent applies puppet rules (including the desired state of the resource) to the system using **providers**.
- When the puppet agent processes a resource, it first decides which provider it needs for user, then passes along the attributes that configured in the resource to the provider.
- The code of each provider is in charge of making the computer reflects the state requested in the resource.

Introduction to Puppet

- Use **class** in Puppet to collect the resources needed to achieve a goal in single place.
- For example, there lies class with three resources, a package, a file and a service. All of them is related to the Network Time Protocol or NTP. This would make it easier to exert changes in the future since all the related resources together.

Puppet Classes

```
class ntp {
  package { 'ntp':
    ensure => latest,
  }
  file { '/etc/ntp.conf':
    source =>
'puppet:///modules/ntp/ntp.conf',
    replace => true,
  }
  service { 'ntp':
    enable => true,
    ensure => running,
  }
}
```

The Building Blocks of Configuration Management

- **Puppet's domain specific language** or **DSL**, is limited to operations related to when and how to apply configuration management rules to devices.
- On top of the basic resources types which previously checked, Puppet's DSL includes **variables**, **conditional statements**, and **functions**.
- Puppet's facts are variable representing characteristics of the system. Facts generated by a program called "**facter**" in Puppet agent , which analyzes the current system and stores the information in facts.

The Building Blocks of Configuration Management

- **facts** is **variable**, preceded by a dollar sign in Puppet's DSL. The facts variable known as **hash** in DSL, like Python's dictionary. **is_virtual** is one of the built-in facts. It reports if puppet runs on virtual or physical machine.
- The **if** and **else** blocks declared as DSL **conditional statement**.
- Use equals greater than (**=>**) to assign values to **attributes**. Each attribute ends with a **comma**.

Sample Puppet Code with Built-in Facts

```
if $facts['is_virtual'] {
  package { 'smartmontools':
    ensure => purged,
  }
} else {
  package { 'smartmontools':
    ensure => installed,
  }
}
```

The Building Blocks of Configuration Management

- Puppet uses a declarative language by declaring the state to achieve rather than the steps to get there.
- Another important aspect of configuration management is the operations should be **idempotent**. An **exception** is the **exec** resource that runs commands. Since a command might modify the system each time it's executed, or use **onlyif** attribute.

Using onlyif to change exec attribute become idempotent

```
exec { 'move example file':

  command => 'mv /home/bangkit/example.txt
/home/bangkit/Desktop',
  onlyif => 'test -e
/home/bangkit/example.txt',

}
```

The Building Blocks of Configuration Management

- Another important aspect of how configuration management works are the **test and repair paradigm**. This means that actions are taken only when necessary to achieve a goal.
- And lastly, an important characteristic is that Puppet is **stateless**, which means there's no state being kept between runs of the agent. Each Puppet run is independent to the previous one and the next one.

Deploying Puppet Locally

- When testing new configurations, instead of using Puppet as client-server architecture, use it as stand-alone application run from the command line.
- Install puppet-master on computer (this course use Linux operating system)

Install puppet-master on debian based Linux

```
$ sudo apt install puppet-master
```

Deploying Puppet Locally

- Create manifests file to store the rules, with **.pp** extension.
- For example, create rule to make package "htop" installed.
- Catalog is the list of rules generated for one specific computer once the server has evaluated all variables, conditionals, and functions.

Puppet manifest tools.pp

```
package { 'htop':
  ensure => present,
}
```

Apply rules in tools.pp

get verbosity output

```
$ sudo puppet apply -v tools.pp
…
Notice: Applied catalog in 2.32 seconds
```

Deploying Puppet Locally

- In resource relationships, the resource types in lowercase when declaring them, but capitalizes them when referring to them from another resource's attribute.
- To apply the rules described in a class, call **include** command. Typically, the class is defined in one file and include it in another.

Resource Relationships

```
class ntp {
  package { 'ntp':
    ensure => latest,
  }
  file { '/etc/ntp.conf':
    source => '/home/bangkit/ntp.conf',
    replace => true,
    require => Package['ntp'],
    notify => Service['ntp'],
  }
  service { 'ntp':
    enable => true,
    ensure => running,
    require => File['/etc/ntp.conf'],
  }
}

include ntp
```

Deploying Puppet Locally

- Puppet module is a collection of manifests and associated data. Here's some of the common files and directories.
  - All manifests gets stored in a directory called **manifests**. It includes special file **init.pp** which defines a class with the same name as the module.
  - The rest of the data is stored in different directories depending on what it does.
  - The **files** directory includes files copied into the client machines without any changes.
  - The **templates** directory includes files that had already been preprocessed before they were copied into the client machines.

Deploying Puppet Locally

- There's a large collection of prepackaged modules that are shipped and ready to user.
- Let's try to install the Apache module provided by Puppet Labs and include it in a module. Use a double colon (::) before the module name to let's Puppet know that this is a global module.

The Apache prepackaged module

```
$ sudo apt install
puppet-module-puppetlabs-apache
# the module installed in directory
/usr/share/puppet/modules.available/puppetlabs-
apache
```

Use the Apache module

```
$ cat webserver.pp
include ::apache

$ sudo puppet apply -v
webserver.pp
```

Deploying Puppet to Clients

- In Puppet terminology, a node is any system where Puppet agent can run. Node definition typically stored in file called **site.pp**.
- When setting up Puppet, usually there's a **default** node definition that lists down classes that should be included for all the nodes.
- To apply for specific nodes, identify by its FQDNs (Fully Qualified Domain Names). It's also list the same classes from default.

Default node definition

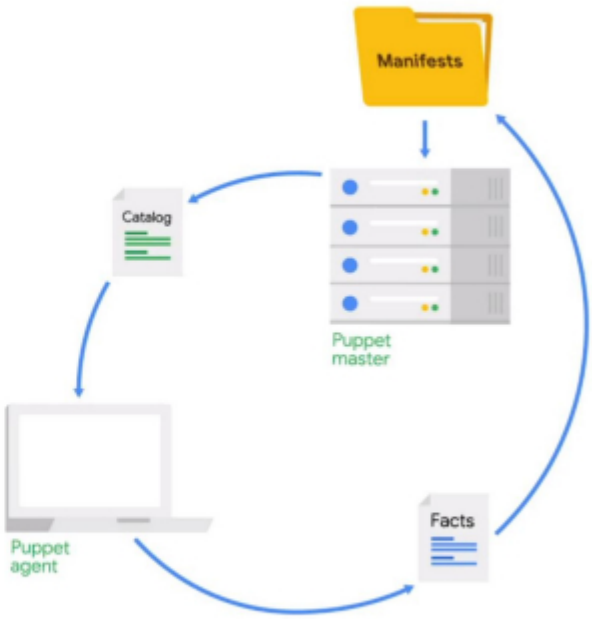
```
node default {
  class { 'sudo': }
  class { 'ntp':
    servers => ['ntp1.com', 'ntp2.com']
  }
}
```

web server node definition

```
node webserver.example.com {
  class { 'sudo': }
  class { 'ntp':
    servers => ['ntp1.com', 'ntp2.com']
  }
  class { 'apache': }
}
```

Deploying Puppet to Clients

- Puppet uses public key infrastructure (PKI), to establish secure connections between the server and the clients, using SSL.
- Puppet comes with its own certificate authority (CA), which can be used to create certificates for each clients.



Deploying Puppet to Clients

- Only for demo purpose, configure Puppet Master to automatically sign the certificate.
- Set node definitions in /etc/puppet/code/environments/productions/manifests/site.pp
- Sample case, to deploy Apache web server to Puppet node webmaster.example.com

Configuration on Puppet Master

```
$ sudo puppet config --section master set
autosign true

# edit node definitions site.pp, the output:
$ cat
/etc/puppet/code/environments/productions
/manifests/site.pp
node webserver.example.com {
  class {'apache':}
}
node default {}
```

Deploying Puppet to Clients

- Install package puppet on computer as Puppet Client. And configure as client with config set server to master.example.com.
- Test run the Puppet agent by passing --test. Make sure to get a catalog to install and configure the Apache package.
- Set the Puppet running automatically, so if the configuration on master changes, clients will automatically apply.

Configuration on Puppet Client

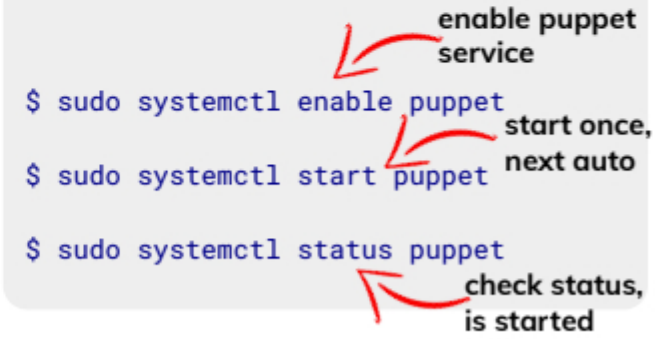
```
$ sudo apt install puppet
$ sudo puppet config set server
master.example.com

$ sudo puppet agent -v --test

$ sudo systemctl enable puppet

$ sudo systemctl start puppet

$ sudo systemctl status puppet
```



Updating Deployments

- To avoid doing mistakes on deployments, there's some options:
  - Test manually on specific test machine.
  - Use puppet parser validate command that checks that the syntax of the manifests is correct.
  - Run the rules using --noop. Means no operations, makes puppet simulate what it would do without actually doing it.
  - Puppet lets test manifests automatically by using R-Spec tests.

Updating Deployments

- In Infrastructure context, **production** is part of the infrastructure where a service is executed and served to its users.
- One of the keys to roll out changes safely is running them through a test environment first.
- Also try to isolate the configurations that the agents see, depending on what Puppet environment they're running. Eg: mark some users as early adopter or canaries these nodes detect potential issues first.
- Pushing changes in batches, instead of pushing the changes to all nodes.
- It’s a good idea to break the changes to be **small and self-contained**.

Managing Cloud

Cloud Computing

- A service running in the cloud, simply means that the service is running somewhere else either in a data center or in other remote servers that we can reach over the Internet.
- Software as a Service (SaaS), is when a Cloud provider delivers an entire application or program to the customer.
- Platform as a Service (PaaS), is when a Cloud provider offers a preconfigured platform to the customer.
- Infrastructure as a Service (IaaS), is when a Cloud provider supplies only the bare-bones computing experience.

Cloud Computing

- There's the lift and shift phrase. Related to a migration from traditional server configurations to the Cloud, there is a lift of the current configuration, and a shift to a virtual machine.
- Public cloud is the cloud services provided by third party. Private cloud is when the company owns the services and the rest of your infrastructure, whether that’s on-site or in a remote data center. Hybrid cloud is a mixture of both public and private Clouds. Multi-cloud is a mixture of public and/or private clouds across vendors.

Managing Instances in the Cloud

- There's a variety of different Cloud providers, each with some specific advantages. Some terms used by one provider might not exactly match with others, while some other concepts are the same. This course use Google Cloud Platform to demonstrate the examples.
- To create VM resources, it is available via the web interface or the command line interface. The web UI is great for experimenting, like comparing different options and event show the cost estimation. The command line interface is a significant step towards automation.
- For automation, there's also **reference images** feature that store the contents of a machine in a reusable format there lies also **templating** feature, which is the process of capturing all of the system configuration to create VMs in a repeatable way and usually saved in **disk image**.

Automating Cloud Deployments

- A load balancer ensures that each node receives a balanced number of requests. The simplest strategy is just to give each node one request, called round robin.
- Auto-scaling allows the service to increase or reduce capacity as needed while the service owner only pays for the cost of the machines that are in use at any given time.
- Orchestration is the automated configuration and coordination of complex IT systems and services.
- Related to infrastructure as Code, (similar to Puppet) there's Terraform, also use Domain-specific language to specify the Cloud infrastructure to look like and able to interact with many Cloud providers and automation vendor.

Building Software for the Cloud

- Storage services:
  - Block storage in the Cloud acts almost exactly like a hard drive. As virtual disks it can easily move the data around.
    - Persistent storage used for instances that are long lived and need to keep data across reboots and upgrades.
    - Ephemeral storage used for instances that are only temporary and only need to keep local data while it's running.
  - Object storage or Blob (binary large object) storage to place and retrieve objects in a storage bucket.

Building Software for the Cloud

- Cloud providers typically offer different classes of storage at different prices. It depends on performance, availability, or how often the data is accessed. Hot data is accessed frequently, cold data is otherwise.
- Performance storage solution is influenced by a number of factors:
  - Throughput is the amount of data that can be read and write in a given amount of time.
  - IOPS or input/output operations per second measures how many reads or writes it can do in one second, no matter how much data to access.
  - Latency is the amount of time it takes to complete a read or write operation.

Building Software for the Cloud

- Database services:
  - SQL (relational) database employs the traditional database format and query language. Data is stored in tables with columns and rows that can be indexed. Retrieved the data by writing SQL queries.
  - NoSQL databases, has multiple advantages related to scale. It is designed to be distributed across tons of machines and ultra fast when retrieving results. Use specific API provided by the database.

Building Software for the Cloud

- Load balancer services:
  - A quite common load balancing technique is round robin DNS. DNS configured with round robin will give each client asking for the translations a group of IP addresses in a different order.
  - Setup a server as a dedicated load balancer, acts as a proxy between the clients and the servers. Use sticky sessions to keep track requests from the same client. Such clients always go to the same backend server. It also checks the health of the backend servers and serves the geographically-nearest clients by using Geo DNS and Geo IP.
  - Some providers dedicated to bring content of your services as close as possible to the users by using content delivery networks or CDNs.

Building Software for the Cloud

- Change management is how to make changes in a controlled & safe way.
- Make sure everything is well-tested. Running unit tests and integration tests, running these tests whenever there's a change. With Continuous Integration (CI) the system will build and test every time there's a change. Then use Continuous Deployment (CD) to automatically deploy the results of the build or build artifacts.
- To test in production with real customers, experiment using A/B testing.
- Learn from failure and build the new knowledge into the change management.

Building Software for the Cloud

- Cloud providers often enforce rate limits on resource-hungry service calls to prevent on service from overloading the entire system. Alternatively, for example, try to group all of the calls in to one batch, or switch to a different service.
- For some of the limits, there's an option to ask for a quota increase from the providers for additional capacity, or set a smaller quota to avoid overspending.
- One of the tricks is to have a good monitoring and alerting around the behaviour.

Monitoring and Alerting

- To understand how the service is performing, use monitoring. Monitoring check into the history of current status of a system, by evaluating a bunch of different metrics.
- Some monitoring system like AWS Cloudwatch, Google Stackdriver, or Azure Metrics are offered directly by the Cloud providers. Other systems like Prometheus, Datadog, or Nagios can be used across vendors.
- Some systems use a pull model, which means that the monitoring infrastructure periodically queries the service to get the metrics. Others use a push model, which means that the service needs to periodically connect to the system and send the metrics.
- Whitebox monitoring checks the behaviour of the system from the inside. Blackbox monitoring checks the behaviour of the system from the outside.

Monitoring and Alerting

- Create automation that checks the health of the system and notifies when things don't behave as expected. The most basic approach is to run a job periodically that monitors the health of the system and sends out an email if the system isn't healthy. On Linux, use **cron**, as the tool to schedule periodic jobs, then pair with a simple Python script.
- Not all alerts are equally urgent. They are typically divided into two groups, namely those in need of urgent immediate attention and those in need of attention in the near future.

Monitoring and Alerting

- Service-Level Objectives (SLOs) are pre-established performance goals for a specific service. SLOs need to be measurable, which means that there should be metrics that track how the service is performing and able to check if it's meeting the objectives or not.
- Many SLOs are expressed as how much time a service will behave as expected. For example, a service might promise to be available 99% of the time. It can be down up to 1% of the time, up to 3.65 days per year.
- Service level agreement (SLAs) is a commitment between a provider and a client.

Troubleshooting and Debugging

- Understanding logs is essential to solve problems in IT. With services in Cloud, learn where to find the logs. Some Cloud providers offer centralized solutions to collect all the logs in one place.
- The more the faulty behavior can be isolated, the easier to fix it. So start from finding the root cause of a problem in the service, is it caused by an error on our side or on the provider side.
- Some other alternatives are to bring up the service in other regions, try on the different machine type, or try to rollback for the system that has recently changed.

Troubleshooting and Debugging

- If operating a service that stores any kind of data, it's critical to implement automatic backups, thus periodically check that those backups are working correctly by performing restores.
- It’s common practice to have enough servers running at any time so that if one of them goes down, others can still handle the traffic. On a large scale, you must ensure that the service runs on different data centers or different providers. Therefore, if one of the data centers has a problem, the service can still be provided by the other data centers or providers.


bangkit