ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

ΑΝΩΤΑΤΗ ΣΧΟΛΗ ΠΑΙΔΑΓΩΓΙΚΗΣ



ΚΑΙ ΤΕΧΝΟΛΟΓΙΚΗΣ ΕΚΠΑΙΔΕΥΣΗΣ

**ΑΣΠΑΙΤΕ**

**ΤΜΗΜΑ ΕΚΠΑΙΔΕΥΤΙΚΩΝ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ &**

**ΕΚΠΑΙΔΕΥΤΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ**

**Πρακτικές Αυτοματοποίησης ΙΤ με Αnsible**

**Πτυχιακή Εργασία**

**Φοιτητές: Μακρής Δημήτριος**

**ΑΜ: 1353**

ΚΑΤΕΥΘΥΝΣΗ: **ΕΚΠ/ΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ**

Επιβλέπων Καθηγητής

**Μπάλλιου Αγγελική**

**Εργαστηριακή Συνεργάτης**

**Ημερομηνία: (Αντικαταστήστε με την ημερομηνία κατάθεσης)**

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

ΑΝΩΤΑΤΗ ΣΧΟΛΗ ΠΑΙΔΑΓΩΓΙΚΗΣ



ΚΑΙ ΤΕΧΝΟΛΟΓΙΚΗΣ ΕΚΠΑΙΔΕΥΣΗΣ

**ΑΣΠΑΙΤΕ**

**ΤΜΗΜΑ ΕΚΠΑΙΔΕΥΤΙΚΩΝ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ &**

**ΕΚΠΑΙΔΕΥΤΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ**

**ΙΤ Automation Practices with Ansible**

**Degree Thesis**

**Students: Makris Dimitrios**

**Registration Number: 1353**

**Degree Option: Electronic Engineering Educators**

Supervisor

**Balliou Aggeliki**

**Laboratory Associate**

**Date: (Αντικαταστήστε με την ημερομηνία κατάθεσης)**

**Μακρής Δημήτριος**

**Θεοδοσιάδης Αλέξανδρος-Σωτήριος**

**Copyright © Μακρής Δημήτριος 20/3/21**

Με επιφύλαξη παντός δικαιώματος, All rights reserved

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τους συγγραφείς.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τους συγγραφείς και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις της ΑΣΠΑΙΤΕ.

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

ΑΝΩΤΑΤΗ ΣΧΟΛΗ ΠΑΙΔΑΓΩΓΙΚΗΣ



ΚΑΙ ΤΕΧΝΟΛΟΓΙΚΗΣ ΕΚΠΑΙΔΕΥΣΗΣ

**ΑΣΠΑΙΤΕ**

**ΤΜΗΜΑ ΕΚΠΑΙΔΕΥΤΙΚΩΝ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ &**

**ΕΚΠΑΙΔΕΥΤΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ**

**Πρακτικές Αυτοματοποίησης ΙΤ με Αnsible**

**Πτυχιακή Εργασία**

Επιβλέπων Καθηγητής

**Μπάλλιου Αγγελική**

**Εργαστηριακή Συνεργάτης**

………..……………………… …………………..…………….. ……………………………….……..

|  |  |  |
| --- | --- | --- |
| **Εξεταστής**  **(Όνομα)**  **(Θέση / Τίτλος)** | **Εξεταστής**  **(Όνομα)**  **(Θέση / Τίτλος)** | **Εξεταστής**  **(Όνομα)**  **(Θέση / Τίτλος)** |

**Ημερομηνία: (Αντικαταστήστε με την ημερομηνία εξέτασης)**

**Περίληψη**

**Λέξεις – κλειδιά**

**Abstract**

**Keywords**

**Περιεχόμενα**

**1** **Εισαγωγή………………………………………………………………………………….. 9**

1.1 Αντικείμενο της πτυχιακής εργασίας……………………………………………………… 9

1.2 Μεθοδολογία………………………………………………………………………………. 9

1.3 Δομή……………………………………………………………………………………….. 9

**2 Ansible……………………………………………………………………………………... 10**

2.1 Εισαγωγή………………………………………………………………………………….. 10

2.2 Inventory…………………………………………………………………………………... 10

2.3 Ad-Hoc Commands…………………………………………………………………………10

2.4 Playbooks………………………………………………………………………………….. 10

2.5 Variables…………………………………………………………………………………... 10

2.6 Facts……………………………………………………………………………………….. 10

2.7 Task Control………………………………………………………………………………. 10

2.8 Templates………………………………………………………………………………….. 10

2.9 Roles……………………………………………………………………………………….. 10

2.10 Συμπεράσματα…………………………………………………………………………….. 10

**3 Άλλα εργαλεία για διαχείριση διαμόρφωσης…………………………………………… 11**

3.1 CFEngine…………………………………………………………………………………... 11

3.2 Puppet……………………………………………………………………………………... 11

3.3 Chef………………………………………………………………………………………... 11

3.4 Jenkins…………………………………………………………………………………….. 11

3.5 GitLab CI………………………………………………………………………………….. 11

3.6 SaltStack…………………………………………………………………………………… 11

3.7 Γιατί Επιλέγουμε την Ansible……………………………………………………………… 11

**4 Τρίτη ενότητα……………………………………………………………………………... 12**

4.1 Υποενότητα

**5 Επίλογος…………………………………………………………………………………… 13**

**6 Παράρτημα Εικόνων……………………………………………………………………... 14**

**7 Αναφορές / Links…………………………………………………………………………..15**

**8 Κατάλογος Πινάκων: …………………………………………………………………….. 16**

**9 Πίνακας Εικόνων: ………………………………………………………………………... 16**

**10 Κατάλογος Εννοιών………………………………………………………………………..16**

**Πρακτικές Αυτοματοποίησης ΙΤ με Αnsible**

**Πτυχιακή Εργασία**

**1. Εισαγωγή**

**1.1 Αντικείμενο της πτυχιακής εργασίας**

Σκοπός και στόχοι της πτυχιακής εργασία.

**1.2 Μεθοδολογία**

Η μεθοδολογία που ακολουθήθηκε.

**1.3 Δομή**

Δομή της πτυχιακής εργασίας.

**2.** **Ansible**

Στην ενότητα αυτή θα δούμε την Ansible ως εργαλείο διαχείρισης διαμόρφωσης στον κλάδο του ΙΤ. Θα αναλυθούν βασικές έννοιες στην χρήση στου συγκεκριμένου εργαλείου, όπως το Inventory τα Ad-Hoc Commands, τα Playbooks με τα οποία θα ασχοληθούμε και εκτενέστερα αργότερα και τα Variables δηλαδή τις μεταβλητές που χρησιμοποιούνται στην Ansible. Θα εξηγηθούν παράλληλα έννοιες, όπως τα facts που είναι χρήσιμα στην λειτουργία των playbooks, το task control τα roles και τα templates.

**2.1 Εισαγωγή**

H Ansible είναι ένα εργαλείο λογισμικού ανοιχτού κώδικα, διαχείρισης παραμέτρων και ανάπτυξης εφαρμογών που επιτρέπουν την διαχείριση και παροχή IaC[1] (Infrastructure as Code) συστημάτων, δηλαδή υπολογιστικών συστημάτων, τα οποία δεν είναι διαχειρίσιμα από φυσικό υλικό (hardware) ή από διαδραστικά εργαλεία διαμόρφωσης, αλλά μέσω αρχείων ορισμού (definition files) αναγνώσιμα από τον υπολογιστή. Λειτουργεί σε πολλά συστήματα όμοια με Unix και μπορεί να διαμορφώσει τόσο αυτά όσο και Microsoft Windows συστήματα. Περιλαμβάνει τη δική του δηλωτική γλώσσα για την περιγραφή της διαμόρφωσης του συστήματος. Η Ansible κυκλοφόρησε το 2012 από τον Michael DeHaan και σε συνεργασία με τον Saïd Ziouani ιδρύθηκε η εταιρία Ansible, Inc. η οποία και αργότερα αποκτήθηκε από τη Red Hat το 2015. Η Ansible δεν χρειάζεται κάποιον agent, καθώς συνδέεται προσωρινά απομακρυσμένα μέσω SSH[2] (Secure Shell Protocol) ή μέσω της απομακρυσμένης διαχείρισης των Windows (επιτρέποντας την απομακρυσμένη εκτέλεση PowerShell) για να κάνει τις εργασίες του. H Ansible απαιτεί να υπάρχει εγκατεστημένη η Python σε όλα τα μηχανήματα διαχείρισης καθώς και ο pip μαζί με λογισμικό διαμόρφωσης-διαχείρισης και τα εξαρτώμενα πακέτα του. Οι βασικοί στόχοι της Ansible είναι να είναι:

* Σαφής. Η Ansible χρησιμοποιεί απλή σύνταξη (YAML[3] και Jinja[4] πρότυπα) η οποία είναι εύκολη στην κατανόηση από μεγάλο εύρος του IT κλάδου. Το ίδιο ισχύει και για τα APIs (Application Programming Interfaces) της Ansible.
* Γρήγορη. H Ansible είναι γρήγορη στην κατανόηση, και την εγκατάσταση καθώς δεν απαιτεί την εγκατάσταση επιπλέον agent και daemons στα μηχανήματα που θα υποστούν διαχείριση.
* Ολοκληρωμένη. Η Ansible καταφέρνει να συνδυάζει τρεις λειτουργίες σε μία και ότι χρειάζεται για μια απομακρυσμένη αυτοματοποιημένη διαχείριση συστήματος υλοποιείται με μόνο ένα πακέτο
* Αποδοτική. Το γεγονός πως η Ansible δεν απαιτεί επιπλέον υλικό εγκατεστημένο στους servers συνεπάγεται την απελευθέρωση επιπλέον πόρων για διαχείριση άλλων εφαρμογών από τους servers αυτούς. Παράλληλα τα modules της Ansible λειτουργούν μέσω JSON[5] (JavaScript Object Notation), το οποίο δίνει την ευελιξία στον χρήστη να την χρησιμοποιήσει με modules στην γλώσσα προγραμματισμού που ήδη γνωρίζει
* Ασφαλής. Εφόσον λειτουργεί μέσω SSH, Δεν απαιτείται το άνοιγμα επιπλέον ports ή η χρήση ευπαθών daemons[6] στους servers.

Αξίζει να σημειωθεί πως για να πετύχει αυτούς τους στόχους το project, έχουν συνεισφέρει και εκατοντάδες μεμονωμένοι developers οι οποία ασχολούνται με την έκδοση patches για την γλώσσα στο GitHub. Αυτή τη στιγμή η Ansible αποτελεί ένα από τα δημοφιλέστερα εργαλεία διαχείρισης υποδομών (με πάνω από 50.000 αστέρια του project στο GitHub την στιγμή που γράφεται αυτό).

**3. Άλλα εργαλεία για διαχείριση διαμόρφωσης**

Πέρα από την Ansible και πριν από την κυκλοφορία της, υπήρχαν επιπλέον εργαλεία τα οποία χρησιμοποιούν ακόμα και σήμερα System Administrators για διαχείριση διαμόρφωσης. Τα πιο σημαντικά τα οποία θα δούμε και πιο αναλυτικά παρακάτω είναι τα Pupper, Chef, Jenkins, GitLab CI και SaltStack. Όλα αυτά τα εργαλεία όμως δεν έχουν την ίδια ευελιξία και δυνατότητες με την Ansible σε αρκετούς σημαντικούς τομείς και για αυτό το λόγο η Ansible όπως θα δούμε επικρατεί και είναι η επικρατέστερη επιλογή στον κλάδο του IT πλέον.

**3.1 CFEngine**

Τα εργαλεία αυτοματοποίησης που είναι διαθέσιμα συνεχώς αυξάνονται, με το καθένα να έχει περιπτώσεις στις οποίες υπερτερεί, και άλλες που είναι πιο αδύναμο. Εκτός από την Ansible που είδαμε παραπάνω, το Puppet το Chef και το SaltStack είναι μερικά ακόμη που έχουν φτιαχτεί με παρόμοιο σκοπό, προσφέροντας πολλές δυνατότητες. Η αυτοματοποιημένη διαχείριση παραμετροποίησης ξεκίνησε με το CFEngine, το 1993. Γραμμένο σε γλώσσα C, το CFEngine μπορεί να αξιοποιηθεί σε πολλές πλατφόρμες και αρχιτεκτονικές, από τις μικρότερες ενσωματωμένες συσκευές, σε διακομιστές, σε cloud ακόμα και σε mainframes, με εύκολο χειρισμό δεκάδων και εκατοντάδων χιλιάδων κεντρικών υπολογιστών. Το CFEngine αποτέλεσε έμπνευση για πολλές δυνατότητες που τα νεότερα εργαλεία υιοθέτησαν. Σε σύγκριση με την Ansible όμως, βλέπουμε αρκετές αντιθέσεις. Το CFEngine λειτουργεί μόνο με την εγκατάσταση ενός προγράμματος σε κάθε μηχάνημα στην υποδομή μας. Ο agent που χρησιμοποιεί, ελέγχει και επαληθεύει τις πολιτικές που έχει εφαρμοσμένες ανά τακτά χρονικά διαστήματα, βασιζόμενο σε ένα κεντροποιημένο υπολογιστή που τρέχει το CFEngine, και εκεί καταλήγουν όλα τα αποτελέσματα από την τήρηση - και μη - των πολιτικών (policies) που έχει δημιουργήσει ο System Administrator της εταιρείας. (βλ. Εικόνα 1)

Η ενημέρωση των πολιτικών πραγματοποιείται μόνο όταν υπάρχει κάποια νέα πολιτική διαθέσιμη στο hub, ώστε το κάθε μηχάνημα να μην μεταφέρει όλα τα αρχεία πολιτικών, δημιουργώντας περιττή κλιμάκωση στο δίκτυο της υποδομής. Σε περίπτωση που το CFEngine hub δεν είναι διαθέσιμο στο δίκτυο, τότε τα υπόλοιπα μηχανήματα μπορούν να αντιγράψουν την πιο πρόσφατη τρέχουσα πολιτική από κάποιον εξυπηρετητή (server) που διατηρεί αντίγραφα ασφαλείας. Αν δεν υπάρχει εξυπηρετητής αντιγράφων ασφαλείας, τότε η ενημέρωση των πολιτικών δεν θα πραγματοποιηθεί.

Παρακάτω βλέπουμε ένα παράδειγμα βασικής χρήσης του CFEngine για την

αυτοματοποίηση της εγκατάστασης των πακέτων ntp και lynx:

**body** **common** **control**

{

**bundlesequence** => { "install\_packages" };

**inputs** => { "libraries/cfengine\_stdlib.cf" };

}

**bundle** **agent** install\_packages

{

**vars**:

"desired\_packages"

slist => { *# list of packages we want*

"ntp",

"lynx"

};

**packages**:

"**$(desired\_packages)**" *# operate on listed packages*

**package\_policy** => "add", *# What to do with packages: install them.* **package\_method** => generic; *# Infer package manager (e.g. apt, yum) from the OS.*

}

Μέχρι στιγμής είδαμε μία σημαντική διαφορά σε σχέση με την Ansible - την ανάγκη ενός agent. Αντίθετα, η ανάγκη ενός κεντροποιημένου μηχανήματος, ακολουθεί με παρόμοια λογική την Ansible, όπου κάποιο μηχάνημα πρέπει να τρέξει playbook ή ad-hoc command για να συνδεθεί στους υπόλοιπους.

Είναι αξιοσημείωτος ο τρόπος που επιβάλλονται οι πολιτικές με το CFEngine. Αφού ελεγχθεί η τρέχουσα κατάσταση του συστήματος, πραγματοποιούνται αλλαγές χωρίς να υπάρχει ανάγκη διατήρησης ανοιχτής σύνδεσης με το κεντροποιημένο hub. Έτσι, η επιβολή μπορεί να γίνει με το μηχάνημα εκτός σύνδεσης στο δίκτυο ή ακόμα και όταν το hub δεν γίνεται να είναι διαθέσιμο (πχ λόγω επανεκκίνησης μετά από εγκατάσταση αναβαθμίσεων). Ακόμα, η συχνή επιβολή πολιτικών είναι πιθανή, ακόμα και αν δεν επιθυμούμε να στέλνονται όλες οι αναφορές στο κεντροποιημένο hub. Έτσι, δεν αυξάνουμε την κίνηση στο δίκτυο αν δεν χρειάζεται να αποκτήσουμε πρόσβαση στις αναφορές.

Οι αναφορές είναι άλλη μία σημαντική διαφορά με την Ansible. Όπως θα δούμε και σε επόμενο κεφάλαιο, η Ansible κατά την εκτέλεση ενός playbook, δεν εστιάζει στην παράδοση λεπτομερών αναφορών, ανά task ούτε ανά play. Όσο ο κώδικας που εκτελεί δεν παρουσιάζει κάποιο σφάλμα στα μηχανήματα του inventory μας, η Ansible δεν θα τυπώσει καμία αναφορά. Αντίθετα, θα δούμε στο τέλος τον αριθμό των tasks που εκτελέστηκαν απροβλημάτιστα, αν χρειάστηκε να πραγματοποιηθούν αλλαγές κατά την εκτέλεση του, η αν το μηχάνημα βρισκόταν ήδη στην επιθυμητή κατάσταση.

Ακόμα, το CFEngine θα μεταφέρει μέσω του δικτύου μόνο τις διαφορές σε σχέση με τις πολιτικές που ήδη επιβάλλει. Η Ansible αντίθετα θα προσπαθήσει να εκτελέσει όλο το playbook κάθε φορά που συνδέεται σε ένα μηχάνημα.

Ο βασικός λόγος που η Ansible μας εξυπηρετεί περισσότερο, είναι η "ανικανότητα" της - ή αλλιώς το idempotency. Αυτό σημαίνει πως όσες φορές και να τρέξουμε το ίδιο playbook, θα προσπαθήσει να φέρει το κάθε μηχάνημα στην ίδια ακριβώς κατάσταση. Αν οι απαιτούμενες αλλαγές έχουν ήδη γίνει, το playbook δεν βρίσκει κάποιο σφάλμα σε αυτό. Θεωρεί απλώς πως τα μηχανήματα μας βρίσκονται ήδη στην επιθυμητή κατάσταση. Αυτό είναι σημαντικό καθώς (όπως θα δούμε με τη βοήθεια του Tower), μπορούμε να οργανώσουμε τις ανάγκες της υποδομής μας με μερικά playbooks και να αφήσουμε να τα τρέχουν άτομα που δεν χρειάζεται να γνωρίζουν να γράφουν οι ίδιοι Ansible. Ακόμα, στην περίπτωση που τρέξουμε κάποιο playbook μέσω του Ansible Tower, τότε κάθε εκτέλεση playbook, θα παρέχει και εκτενείς αναφορές μετά την εκτέλεση κάθε task.

Η Ansible, με μοναδική απαίτηση τη χρήση μιας SSH σύνδεσης, φανερώνει τον τομέα στον οποίο ξεχωρίζει. Είναι μία γλώσσα που μπορούμε να εντάξουμε στην υποδομή μας αρκετά εύκολα, και ακόμα και κάποιος System Administrator μπορεί να την χρησιμοποιήσει χωρίς να γράψει κώδικα, με την χρήση μόνο ad-hoc tasks.

**3.2** **Puppet**

Αν υπήρξε ποτέ ένα πανταχού παρόν εργαλείο για την ανάπτυξη λογισμικού DevOps, αυτό είναι το Puppet. Ακόμα κι αν ένας οργανισμός δεν κάνει DevOps, μάλλον χρησιμοποιεί το Puppet κάπου. Το Puppet είναι πολύ καλό στο να αναπτύσσει εφαρμογές σε

προκατασκευασμένη υποδομή. Για όσους μόλις ξεκινούν με την αυτοματοποίηση ανάπτυξης, το Puppet είναι ένα καλό μέρος για να ξεκινήσουν. Αλλά δεν είναι μόνο για αρχάριους. Σύμφωνα με την εταιρεία, περισσότερες από 3 στις 4 εταιρείες του Fortune 100 χρησιμοποιούν το Puppet για την ανάπτυξη εφαρμογών.

Το Puppet αυτοματοποιεί την εγκατάσταση και τη διαμόρφωση εφαρμογών σε διακομιστές προορισμού. Γράφοντας σενάρια που καθορίζουν και εγκαθιστούν τις προϋποθέσεις

εφαρμογής, ορίζοντας τις μεταβλητές διακομιστή που απαιτεί η εφαρμογή και γράφοντας σενάρια ρύθμισης παραμέτρων για εφαρμογές και daemons, μια ομάδα μπορεί απλώς να εκτελέσει ξανά το σενάριο για επανεγκατάσταση. Μια μεγάλη κοινότητα χρηστών έχει ήδη γράψει πολλές εγκαταστάσεις εφαρμογών. Για μεγάλο αριθμό εφαρμογών λογισμικού, ιδιαίτερα στο πεδίο ανοιχτού κώδικα, οι χρήστες χρειάζεται απλώς να κατεβάσουν τα σχετικά Puppet script files και να αλλάξουν αυτό που είναι γραμμένο στα αρχεία διαμόρφωσης για να αντικατοπτρίζει το project τους.

H Ansible διαφέρει από το Puppet στο ότι λειτουργεί χωρίς agents. Η Ansible μπορεί να ρυθμιστεί και να λειτουργήσει χωρίς εγκατάσταση λογισμικού σε κάθε μηχάνημα. Αυτό εξοικονομεί πολύτιμο χρόνο για τους χρήστες της Ansible. Επίσης, η Ansible χρησιμοποιεί την αναγνώσιμη από τον άνθρωπο γλώσσα σειριοποίησης δεδομένων YAML για να ορίσει εγκαταστάσεις. Διαφορετικοί χρήστες βρίσκουν ότι αυτό είναι είτε ένα συν είτε ένα μείον σε σύγκριση με το Puppet Script, και επειδή η συγγραφή σεναρίων είναι μεγάλο μέρος της χρήσης και των δύο εργαλείων, αξίζει να προσδιοριστεί η προτίμηση του εκάστοτε οργανισμού.

Επίσης, μετά την εξαγορά της από την Red Hat, η Ansible προσφέρει τώρα πιο αυστηρές ενσωματώσεις στο stack[7] Red Hat Enterprise Linux (RHEL) που θα μπορούσαν να το κάνουν μια ευνοϊκή επιλογή.

Για τους οργανισμούς που δεν έχουν ακόμη υιοθετήσει την αυτοματοποίηση ανάπτυξης, αξίζει η προσπάθεια να εξετάσουν όλα τα εργαλεία αυτοματισμού ανάπτυξης και να καθορίσουν ποια θα λειτουργήσουν καλύτερα για την επιχείρησή τους. Πολλά από αυτά τα εργαλεία αυτοματισμού είναι επίσης ικανά να παρέχουν είτε ένα πλήρες CI, CD και ARA pipeline[8] είτε ενσωματώσεις με δημοφιλή εργαλεία σε αυτήν την περιοχή.

**5. Επίλογος**

Κείμενο επιλόγου.

**6. Παράρτημα Εικόνων**

Οι προηγούμενες εικόνες βρίσκονται εντός του κυρίου κειμένου.



Εικόνα 1. Περιγραφή της εικόνας (με παραπομπή αν ελήφθη από κάπου) [22].

**7. Αναφορές / Links**

[1] B. Silk, “Teaching probability to electrical engineering students”, IEEE Transactions on Education, Vol. 1 No. 4, 2005, pp. 476-483.

[2] Τίτλος πηγής αναφοράς και λοιποί προσδιορισμοί, URL: <http://www.site.org>, προσπελάστηκε στις 18/11/2011.

{**Οδηγίες περί του τρόπου γραφής των Αναφορών:** Όταν αναφερόμαστε σε μια έννοια για πρώτη φορά πρέπει να βάλουμε αναφορά. Η αναφορά μπαίνει μέσα σε αγκύλη με το αντίστοιχο νούμερο της [1], αμέσως μετά τη λέξη ή στο τέλος της πρότασης / παραγράφου. Στο τέλος της εργασίας πρέπει να υπάρχει πίνακας αναφορών. Ο σωστός τρόπος παρουσίασης των αναφορών είναι:

*Για αναφορά σε άρθρο περιοδικού:* Πχ.:

[1] B. Silk, “Teaching probability to electrical engineering students”, IEEE Transactions on Education, Vol. 1 No. 4, 2005, pp. 476-483.

[2] G. Nag, “Classification of examples for teaching probability to electrical engineering students”, IEEE Transactions on Education, Vol. 561 No. 4, 2008, pp. 34-58, [http://www.talis.com/nodalities/pdf/nodalities issue9.pdf](http://www.talis.com/nodalities/pdf/nodalities%20issue9.pdf) .

*Για αναφορά σε άρθρο Συνεδρίου:* Πχ.:

[3] A. Pig, “An experience in renewable energy”, in Proceedings of the 3rd IEEE International Conference on E-learning in Industrial Electronics, 3-5 Nov. 2009, pp. 135-140.

*Για αναφορά σε βιβλίο:* Πχ.:

[4] G. Yev, “Principles of Microelectronics”, English translation, Publisher, 1974, pp. 100-110, ISBN 5023 546.

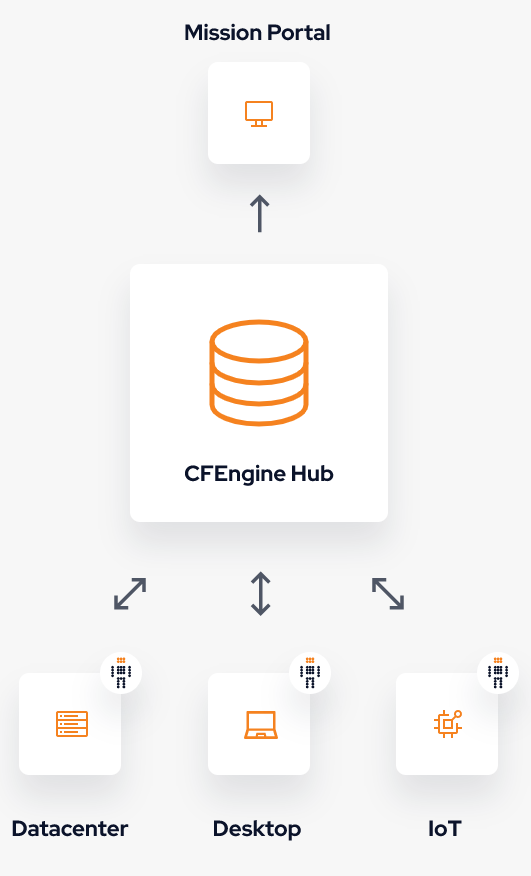
*Για αναφορά σε ιστοσελίδα (link):* Πχ.:

[5] World Wide Web Consortium (W3C), URL: <http://www.w3.org>, προσπελάστηκε στις 8/10/2015.

(οδηγία για την εύρεση του τίτλου ιστοσελίδας: Ο τίτλος προκύπτει από αυτό που γράφει η σελίδα πάνω στο παράθυρο του web browser όταν εμφανίζεται.)}

**8. Κατάλογος Πινάκων:**

**9. Πίνακας Εικόνων:**



Εικόνα 1. Δομή λειτουργίας CFEngine (https://cfengine.com/product-overview/).

**10. Κατάλογος Εννοιών:**

[1] IaC: Η υποδομή ως κώδικας (IaC) είναι η διαδικασία διαχείρισης και παροχής κέντρων δεδομένων υπολογιστή μέσω αρχείων ορισμού αναγνώσιμα από μηχανή, αντί για διαμόρφωση φυσικού υλικού ή διαδραστικών εργαλείων διαμόρφωσης. Η υποδομή πληροφορικής που διαχειρίζεται αυτή τη διαδικασία περιλαμβάνει τόσο φυσικό εξοπλισμό, όπως φυσικούς servers, όσο και virtual machines. Οι ορισμοί μπορεί να είναι σε ένα σύστημα ελέγχου έκδοσης. Ο κώδικας στα αρχεία ορισμού μπορεί να χρησιμοποιεί είτε δέσμες ενεργειών είτε δηλωτικούς ορισμούς, αντί να διατηρεί τον κώδικα μέσω μη αυτόματων διεργασιών, αλλά το IaC χρησιμοποιεί πιο συχνά δηλωτικές προσεγγίσεις.

[2] SSH: Το Secure Shell Protocol (SSH) είναι ένα κρυπτογραφικό πρωτόκολλο δικτύου για την ασφαλή λειτουργία των υπηρεσιών δικτύου μέσα σε ένα μη ασφαλές δίκτυο. Οι πιο αξιοσημείωτες εφαρμογές του είναι η απομακρυσμένη σύνδεση και η εκτέλεση γραμμής εντολών.

[3] YAML: Η YAML είναι μια φιλική προς τον άνθρωπο γλώσσα σειριοποίησης δεδομένων για όλες τις γλώσσες προγραμματισμού.

[4] Jinja template engine: Το Jinja είναι μια μηχανή προτύπων Ιστού για τη γλώσσα προγραμματισμού Python. Δημιουργήθηκε από τον Armin Ronacher και έχει άδεια χρήσης βάσει άδειας BSD. Το Jinja είναι παρόμοιο με τη μηχανή προτύπων Django, αλλά παρέχει εκφράσεις όμοιες της Python, διασφαλίζοντας ταυτόχρονα ότι τα πρότυπα αξιολογούνται σε ένα sandbox. Είναι μια γλώσσα προτύπου που βασίζεται σε κείμενο και επομένως μπορεί να χρησιμοποιηθεί για τη δημιουργία οποιασδήποτε σήμανσης καθώς και για τον πηγαίο κώδικα.

[5] JSON: Το JavaScript Object Notation (JSON) είναι μια τυπική μορφή που βασίζεται σε κείμενο για την αναπαράσταση δομημένων δεδομένων με βάση τη σύνταξη αντικειμένου JavaScript. Χρησιμοποιείται συνήθως για τη μετάδοση δεδομένων σε διαδικτυακές εφαρμογές.

[6] Daemon: Ο δαίμονας είναι ένα πρόγραμμα Unix/Linux που εκτελείται στο παρασκήνιο έτοιμο να εκτελέσει μια λειτουργία όταν απαιτείται. Λειτουργώντας σαν επέκταση του λειτουργικού συστήματος, ένας δαίμονας είναι συνήθως μια διαδικασία χωρίς επίβλεψη που ξεκινά κατά την εκκίνηση του υπολογιστή.

[7] Stack: Ο όρος στοίβα λογισμικού αναφέρεται στο σύνολο των στοιχείων που συνεργάζονται για να υποστηρίξουν την εκτέλεση της εφαρμογής. Ορισμένα στοιχεία λογισμικού τροφοδοτούν διεργασίες back-end, μερικά χρησιμοποιούνται για την εκτέλεση υπολογισμών και μερικά χρησιμοποιούνται στο επίπεδο παρουσίασης για να ενεργοποιήσουν τη διεπαφή χρήστη. Σε κάθε περίπτωση, τα στοιχεία μιας στοίβας λογισμικού λειτουργούν παράλληλα για να παρέχουν αποτελεσματικά υπηρεσίες εφαρμογών στον τελικό χρήστη

[8] CI/CD Pipeline: Ένα CI/CD pipeline είναι μια σειρά βημάτων που πρέπει να εκτελεστούν προκειμένου να παραδοθεί μια νέα έκδοση λογισμικού. Οι αγωγοί συνεχούς ενοποίησης/συνεχούς παράδοσης (CI/CD) είναι μια πρακτική που επικεντρώνεται στη βελτίωση της παράδοσης λογισμικού χρησιμοποιώντας προσέγγιση είτε DevOps είτε SRE (Site Reliability Engineering).

[9] Promise: