Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Orchestrace pomocí nástroje Ansible

David Andrš

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

David Andrš

Informatika

Název práce

Orchestrace pomocí nástroje Ansible

Název anglicky

Orchestration using Ansible tool

Cíle práce

Hlavním cílem práce je analýza možností automatizace správy serverů pomocí nástroje Ansible. Dílčími cíli jsou provést orchestraci procesů na konkrétních příkladech, dále ověřit stav instalace nebo konfigurace.

Metodika

Metodika bakalářské práce je založena na analýze odborné a vědecké literatury a internetových zdrojů. Dle výsledků syntézy budou navrženy skripty pro řešení vybraných úloh pomocí orchestrace a bude ověřena jejich časová náročnost. Na základě získaných informací bude syntetizován závěr práce.

Doporučený rozsah práce

30 – 40 stran

Klíčová slova

Orchestration, Automation IT, Ansible, Secure Shell (SSH), YAML, Linux

---

Doporučené zdroje informací

GEERLING, Jeff. Ansible for DevOps: Server and configuration management for humans. St. Louis: Midwestern Mac, 2015. ISBN 978-0986393419.

HOCHSTEIN, Lorin. Ansible: Up and Running: Automating Configuration Management and Deployment the Easy Way. Sebastopol: O'Reilly, 2015. ISBN 978-1491915325.

KEATING, Jesse. Mastering Ansible. Birmingham: Packt Publishing, 2015. ISBN 978-1784395483.

RATAN, Abhishek. Practical Network Automation: Leverage the power of Python and Ansible to optimize your network. Birmingham: Packt Publishing, 2017. ISBN 978-1788299466.

SHAH, Gourav. Ansible Playbook Essentials. Birmingham: Packt Publishing, 2015. ISBN 978-1784398293.

Předběžný termín obhajoby

2018/19 LS – PEF

Vedoucí práce

Ing. Alexandr Vasilenko, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 11. 9. 2018

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 10. 2018

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 01. 11. 2018

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Orchestrace pomocí nástroje Ansible" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30.11.2019 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Alexandru Vasilenkovi, Ph.D., za vedení mé bakalářské práce a za cenné rady a připomínky, které mi při zpracování práce poskytl.

Orchestrace pomocí nástroje Ansible

Abstrakt

Bakalářská práce se věnuje přístupům orchestrace pomocí nástroje Ansible. V teoretické části je popsána aplikační architektura tohoto orchestračního nástroje. Dále tato část práce seznamuje s možností implementace vlastního modulu, jakožto možnosti rozšíření přístupu orchestrace tohoto nástroje. V praktické části jsou využity získané informace a nástrojem Ansible je zvolená komplexní úloha orchestrována, na platformě Linux, s nativními moduly a v kombinaci s vlastním modulem napsaného v Bash. Na základě zjištěných výsledků je uvedeno doporučení pro použití daného přístupu orchestrace.

Klíčová slova: Orchestrace, Automatizace IT, Ansible, Secure Shell (SSH), YAML, Linux, Bash, Wildfly

Orchestration using Ansible tool

Abstract

This bachelor thesis covers the tool Ansible, and its possibilities to implement automation of IT processes using native and own custom modules. The theoretical part of the thesis will acquaint the reader with the application architecture of this tool. It characterizes the parts of its architecture, and their settings. The next part of the thesis deals with the analysis of the implementation of own module in bash script. The final part presents some practical example of selected automated processes using native and own custom module on the Linux platform. Based on the results, recommendation is introduced for use of the given orchestration's approach.

Keywords: Orchestration, Automation IT, Ansible, Secure Shell (SSH), YAML, Linux, Bash, Wildfly

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Stále se vyvíjející IT infrastruktury vyžadují proces automatizace. Správci ve větších firmách čítající i několik tisíc zaměstnanců spravují aplikace běžící až na tisících serverech. V takovém případě je nutné akce zautomatizovat, aby správci byli schopni plnit práci, která je po nich požadována. Správci v menších firmách mohou zvážit, zda se jim činnosti spojené s aplikacemi vyplatí provést manuálně nebo automatizovaně.

Možnost automatizace procesů nabízí orchestrační nástroj Ansible na platformě Linux, kterým se zabývá tato bakalářská práce. Procesy zde jsou zaměřeny na instalaci a konfiguraci aplikačního serveru Wildfly, což pro správce z pohledu manuálního provedení představuje spoustu času. Z ekonomického hlediska je nutné tyto procesy provést co nejefektivněji. Ideální stav by byl takový, že jeden správce obstará celý proces v co nejkratším čase a bez defektu.

Teoretická část práce seznamuje s aplikační architekturou nástroje Ansible. Dále je v této části zpracována možnost implementace vlastního modulu vytvořeného Bash skriptem. Praktická část práce získané informace využije pro orchestraci komplexní úlohy provedené ve dvou příkladech. Jedním bude implementace všech procesů pomocí nativních modulů a ve druhém pomocí vlastního modulu v kombinaci s nativními. Výsledky obou příkladů budou vyhodnoceny v závěru práce.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem práce je zanalyzovat možnosti orchestrace správy serverů pomocí nástroje Ansible a vyhodnotit vhodnost jejich použití v různých situacích.

Dílčími cíli jsou:

- Charakterizovat orchestrační nástroje Ansible.
- Zhodnotit možnosti implementace vlastního modulu ve zvoleném skriptovacím jazyce jako rozšíření možnosti přístupu orchestrace.
- Navrhnout komplexní úlohu, která bude orchestrována.
- Navrhnout testovací prostředí.
- Navrhnout konfiguraci Ansible s použitím nativních modulů a dále v kombinaci s vlastním modulem.

2.2 Metodika

Metodika bakalářské práce je založena na analýze odborné a vědecké literatury a internetových zdrojů. Vlastní řešení je realizováno konfigurací testovacího prostředí na platformě Linux a implementací procesu orchestrace realizovanou v Ansible pomocí nativních modulů a dále v kombinaci s vlastním modulem. Orchestrace bude provedena na komplexní úloze využívající technologie Java-OpenJDK[1] a aplikační server Wildfly[2]. Časový údaj o zpracování obou přístupů orchestrace bude získán pomocí nativního časovače v orchestračním nástroji. Na základě syntézy získaných výsledků bude vypracován závěr bakalářské práce.

[1] Java-OpenJDK je bezplatná a open-source Java platforma. [15]
[2] Wildfly AS je aplikační server na platformě Java. [16]

3 Teoretická část

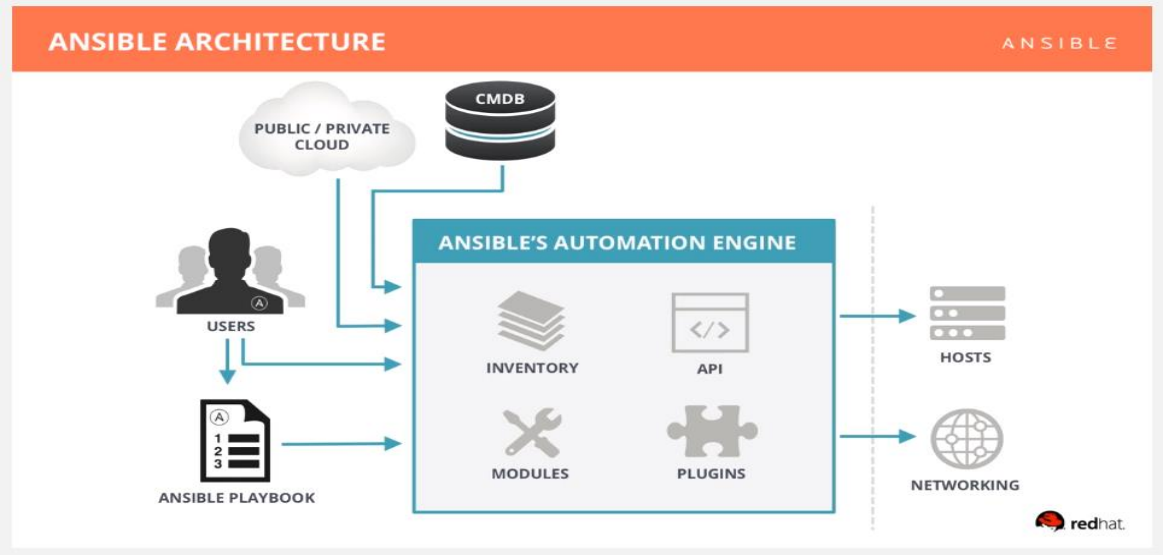
Ansible je orchestrační nástroj, který poskytuje východisko pro činnosti jako shromažďování informací, správu konfigurací a nasazování aplikací paralelně na vícero stanicích. Na rozdíl od nástrojů jako Puppet nebo Chef není nutné předinstalování agentů, protože pro správu stanic po síti je defaultně použit protokol SSH (Secure Shell protokol), který je využíván mezi linuxovými a unixovými stanicemi, nebo službu WinRM (Windows Remote Management umožňuje komunikaci s platformou Windows). Potřeba softwaru pro úspěšné spuštění Ansiblu není veliká, pro MS Windows postačí PowerShell nebo Python v případě systému Linux. Open source verze je konfigurovatelná pomocí konzolového rozhraní a grafické prostředí je součástí placené verze Tower a to pomocí webového řešení.

Původním autorem tohoto nástroje je Michael DeHaan, který je také autorem Cobbleru, softwaru automatizující síťově založené instalace několika operačních systémů z centrálního bodu pomocí služeb, jako jsou DHCP, TFTP a DNS. Ansible jako projekt začal někdy v únoru 2012 a původní společnost AnsibleWorks, Inc., která komerčně podporovala a sponzorovala Ansible, byla v říjnu 2015 převzata společností Red Hat.[1][17][18] Nejnovější stabilní verzí Ansiblu byla v době psaní této práce verze 2.8.5.

3.1 Architektura Ansiblu

Ansible je jednoduchý nástroj na automatizaci IT, jehož základní princip plní soubor několika knihoven a skriptů napsaných v jazyce Python. Samotné provádění konfigurací pomocí zmíněných skriptů se z pohledu architektury (obrázek 1) tohoto enginu užívá pojem „modul“.

Obrázek 1 – Architektura Ansiblu



Zdroj: [31]

Moduly

Mohou být spouštěny přímo na spravovaných hostech tzv. „ad-hoc“ nebo pomocí playbooků, které jsou charakterizovány v této práci ve článku „Playbooky“.[2] Pro zadávání modulů je použita abstrakce, která low-level příkazy[3] odděluje od uživatelů. Díky deklarativní syntaxi se parametry a stavy pro řízení systémových komponent předkládají jako seznamy. To vše lze deklarovat pomocí syntaxe jazyka YAML, který pro člověka velmi přehledný a jednoduchý na pochopení. Ansible obsahuje předinstalovanou knihovnou modulů, která se poskytuje možnosti od správy základních systémových zdrojů až po složitější jako jsou odesílání oznámení, provádění cloudové integrace a jiné. [4] Knihovna všech dostupných modulů je přístupna na stránkách Ansiblu [3].

Moduly a idempotence

Idempotence je důležitou charakteristikou modulů, která znamená, že při opakovaném spuštění se v daném systému vrátí stejný výsledek jako při prvním spuštění. Umožňuje to provádět stejný úkol, aniž by došlo k chybnému stavu. Příkladem využití idempotence může být instalace programu Nginx[4] pomocí modulu apt[5]. Níže uvedené případy znázorňují chování modulů, pokud jsou spuštěny opakovaně:

[3] low-level příkazy jsou charakteristikou pro nízkoúrovňové programovací jazyky a poskytují malou nebo žádnou abstrakci mezi daným programovacím jazykem a strojovými instrukcemi procesoru. [25]
[4] Nginx je softwarový webový server.
[5] apt je balíčkovací systém používaný v Debian GNU/Linuxu a jeho derivátech. [24]

- Modul apt porovná, co bylo uvedeno v parametrech oproti současnému stavu balíku Nginx v daném systému. Pokud modul běží poprvé a Nginx není nainstalován, bude pokračovat v instalaci.
- U každého následného běhu přeskočí instalační část, pokud není dostupná nová verze balíku v repozitáři.

Je tedy důležité, aby uživatelé vytvářející takovéto moduly dbali na zachování této vlastnosti. Většina modulů Ansiblu je idempotentní, s výjimkou některých příkazů a shell modulů. [4]

Pluginy

Jsou psány v jazyce Python a jejich cílem je flexibilně obohatit funkcionalitu Ansiblu. Pro specifické úlohy je možnost napsat vlastní. Některé typy pluginů jsou již součástí Ansiblu. [5] A kromě vlastních lze rozšířit existující např. vyhledávací pluginy mohou být upraveny pro vyhledávání dat z externích zdrojů mimo Ansible. [6]

Inventory

Inventář je zdroj obsahující seznam všech hostů, kteří mohou být rozděleny do skupin. Defaultně se tento seznam ukládá v /etc/ansible/hosts (Linux a Unix) nebo může být specifikován v jiném uložišti. Zahrnuto může být více souborů pro inventory současně a to z dynamického nebo cloudového zdroje či jiného formátu (YAML, ini , aj.). [7] Zápis hostů může být také rozčleněn do složitějších struktur tvořených skupinami. Jeden node může být zapsán také ve více skupinách (obrázek 2).

Obrázek 2 – Hosts – zápis pomocí skupin

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
    east:
      hosts:
        foo.example.com:
        one.example.com:
        two.example.com:
    west:
      hosts:
        bar.example.com:
        three.example.com:
    prod:
      hosts:
        foo.example.com:
        one.example.com:
        two.example.com:
    test:
      hosts:
        bar.example.com:
        three.example.com:
```

Zdroj: [35]

Playbooky

Kromě zadávání ad-hoc příkazů nabízí Ansible pro provádění komplexních změn koncept nazvaný playbook nebo v češtině „scénář“. Použitím scénářů můžeme provádět mnoho akcí najednou a napříč několika systémy. Poskytují způsob řízeného deploymentu nebo zajištění konzistence konfigurace. Scénáře jsou stejně jako moduly psány ve formátu YAML. Při zachování idempotence všech použitých modulů se tato vlastnost přenese i na scénáře a po několikerém spuštění by případné změny měli být provedeny pouze v případě, zda jsou splněny podmínky. Scénáře obsahují jednu nebo více tzv. play (her) (Obrázek 4).

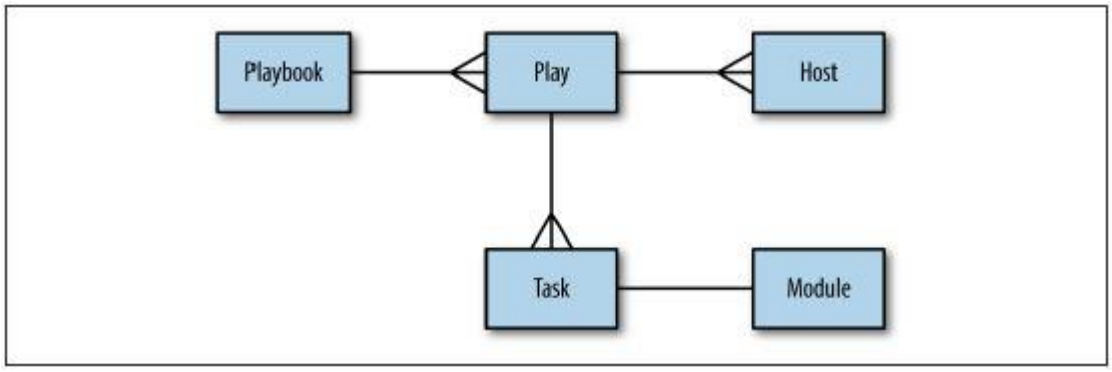
Play se skládá ze tří sekcí:

- Hosts: definuje hostitele, na kterých bude play spuštěna a jakým způsobem bude spuštěna. Zde je také možnost explicitně nastavit uživatelské jméno a další nastavení pro SSH odlišné od defaultních hodnot.

- Vars: definuje proměnné, které budou k dispozici při běhu play.
- Tasks: obsahuje seznam všech modulů v pořadí, v jakém chceme, aby byly Ansiblem spuštěny.

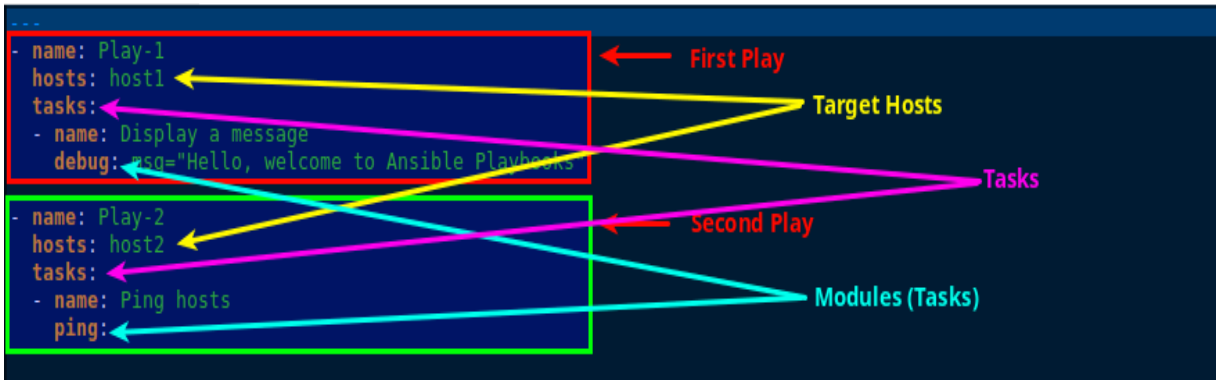
Diagram (Obrázek 3) playbooku ukazující vztahy mezi entitami.

Obrázek 3 - Playbook a vztah mezi jeho entitami



Zdroj: [8]

Obrázek 4 – Zapsání více play v playbooku



Zdroj: [22]

Soubory „scénářů“ se na začátku uvádějí znaky --- a obsahují mnoho klíčových hodnot a seznamů. V řádcích formátu YAML je použito odsazení, které parser využívá pro indikaci vnořených hodnot, což také usnadňuje čtení souboru. Následující úryvek kódu reprezentuje příklad play (hry) v Ansiblu.

```
---
- hosts: webservers
  user: root
  vars:
    apache_version: 2.6
    motd_warning: 'WARNING: Use by ACME Employees ONLY'
    testserver: yes
  tasks:
    - name: setup a MOTD
      copy:
        dest: /etc/motd
        content: "{{ motd_warning }}"
```

Kód 3.1
[6]

Výše uvedený kód na začátku oznamuje, na kterých hostech budou následující příkazy spuštěny, v tomto příkladu se jedná o skupinu *webservers*. Následně se pokračuje zadáním proměnných, jako například zpráva *motd*, která volána v modulu *copy*. Úkolem je v tomto scénáři je zkopírovat definovaný obsah do cílového souboru.

Playbooky a orchestrace

Ansible je popisován jako automatizační a orchestrační nástroj. Tímto je třeba vysvětlit, co konkrétně znamenají slova automatizace a orchestrace v pojetí Ansiblu.

Automatizace uskutečňuje jediný úkol, který bude prováděn samostatně, tj. automatizovat jednu akci. Jedinou akcí může být cokoli od spuštění webového serveru, zastavení služby nebo integrace webové aplikace. Spuštění jediného úkolu v Ansiblu je provedeno pomocí příslušného modulu. Orchestrace již zahrnuje automatizované uspořádání, koordinaci jednotlivých úkolů, které například provádějí správu počítačových systémů, middlewaru a služeb. Tyto komplexní úkoly jsou v Ansiblu prováděny pomocí Playbooků. [36]

Role

V praxi jsou často konfigurovány skupiny např. webových, databázových, middlewarových a jiných serverů, na které při pohledu z větší perspektivy dojde k povšimnutí, že se některé konfigurace opakovaně provádí nad skupinami identických serverů. Tím nejúčinnějším řešením řízení takové infrastruktury je použití takové abstrakce, která nám umožní definovat potřebnou konfiguraci v každé z výše uvedených skupin. Příslušné konfigurace se vyvolají pomocí názvu příslušné role. Role tak poskytují způsob, jak vytvořit modulární kód, který může být dále sdílen a znovu použit. [4]

Složková struktura role

Příklad:

```
site.yml
webservers.yml
fooservers.yml
roles/
   common/
     tasks/
     handlers/
     files/
     templates/
     vars/
     defaults/
     meta/
   webservers/
     tasks/
     defaults/
     meta/
```

Kód 3.2

Je vyžadováno, aby soubory podle svého účelu použití byly vkládány do adresářů s odpovídajícím názvem. Role musí obsahovat alespoň jeden z adresářů uvedených v příkladu složkové struktury role. Seznam adresářů:

- `tasks` - obsahuje hlavní seznam modulů, které má vykonat role.
- `handlers` - obsahuje handlery, které mohou být používány v dané roli nebo dokonce kdekoli mimo ni.
- `defaults` – výchozí proměnné pro roli.
- `vars` – ostatní proměnné pro roli.
- `files` - obsahuje soubory, které lze pomocí dané role nasadit.
- `templates` - obsahuje šablony, které lze nasadit prostřednictvím dané role.
- `meta` - definuje některé metadata pro danou roli.

[9]

Použití rolí

Klasický způsob použití rolí je pomocí volby roles: pro danou play(hru). Níže uvedený kód je součástí souboru side.yml z příkladu kód 3.2 a zobrazuje, jak jednotlivé role volat.

```
---
- hosts: webservers
  roles:
     - common
     - webservers
```

Kód 3.3
[9]

3.2 Implementace vlastních modulů

Nástroj Ansible dokáže rozšířit svoje možnosti pomocí implementace vlastních modulů, které jsou napsány často ve skriptovacích jazycích Perl, Python nebo PowerShell. Protože podpora Perlu je klesající, tak pro Linux přichází v úvahu Python, který jako svobodný software získává na vzrůstající popularitě a jeho hlavní využití je v automatizaci. [19] Další nejmenovaná varianta pro Linux je Bash shell: příkazový interpret, jehož hlavním účelem je umožnit interakci s operačním systémem, což obvykle znamená spouštění programů, které byly uživatelem předloženy shellu ke spuštění. Kromě samotného spouštění programů se také uživatel může setkat s úkoly zahrnující postup opakovaných nebo velmi komplikovaných akcí, které pomocí tohoto skriptovacího jazyka je možné automatizovat. Bash díky zahájení v rámci projektu GNU byl volně dostupný software a v kombinaci se stoupající popularitou Linuxu rychle zastínil Korn shell. Výsledkem je, že Bash je výchozí uživatelský shell v každé linuxové distribuci. [13] Proto se tato práce bude soustředit na implementaci vlastních modulů napsaných pomocí Bash.

Příklad bash skriptu:

```
#!/bin/bash
set -e
# This is potentially dangerous
source ${1}
OLDHOSTNAME="$(hostname)"
CHANGED="False"
if [ ! -z "$hostname" -a "${hostname}x" != "${OLDHOSTNAME}x" ];
then
hostname $hostname
OLDHOSTNAME="$hostname"
CHANGED="True"
fi
echo "hostname=${OLDHOSTNAME} changed=${CHANGED}"
exit 0
```

Kód 3.4: Změna hostname serveru

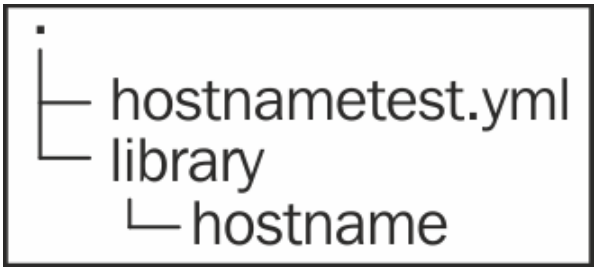
Připravený Bash skript kód 3.4 se použije jako modul a pro jeho načtení se nabízí několik možností. Z pohledu Ansiblu se jedná o možnosti umístění daného modulu, jelikož načítání probíhá postupným prohledáním několika umístění. Nejprve je prohledáno umístění uvedené v parametru *library* v konfiguračním souboru (ansible.cfg), dále umístění uvedené v argumentu *--module-path* prostřednictvím příkazové řádky, následně adresář library umístěný ve stejné složkové úrovni jako spouštěný playbook a nakonec jsou prohledány adresáře library v nastavených rolích.

Ukázkový playbook, který volá Bash skript z příkladu kód 3.4 jako modul:

```
---
- name: Test the hostname file
hosts: testmachine
tasks:
- name: Set the hostname
hostname: hostname=testmachine.example.com
```

Kód 3.5

Obrázek 5 - Struktura adresáře pro playbook



Zdroj: [6]

Obrázek 5 znázorňuje ukázku adresáře, kde se pro načtení modulu zvolila možnost stejného umístění adresáře library a souboru playbooku. Pokud příklad kódu 3.4 vložíme do modulu s názvem hostname a kód 3.5 do scénáře hostnametest.yml, pak v daném umístění spustíme playbook příkazem *ansible-playbook hostnametest.yml*. [6]

3.3 Testování bash modulů

Pro správné fungování modulu je nutné otestovat jeho funkcionalitu. V případě Ansiblu by testování mělo zahrnovat dvě části a to ověření logiky modulu, tedy že napsaný kód v daném skriptu skutečně provádí očekávané zpracování a další část testu by měla ověřit, že výstupy skriptu jsou v korektním formátu.

Testování výstupů

K testování výstupů je možné použít program test-module (Obrázek 6), který je k dispozici ke stažení na GitHubu [10]. Je to jednoduchý program, který umožní otestovat výstupy příslušného modulu. Tento test je možné spouštět na systému bez nainstalovaného Ansible serveru. Zachycené výstupy musí být ve formátu JSON, pokud je vrácen jakýkoli jiný výstup, bude vyhodnocen jako chyba. To znamená, že je nutné ošetřit výstupy stdout a stderr všech příkazů v modulu.

Význam následujících proměnných, které Ansible vyhledává ve výstupech:

- change: Nastavení na hodnotu „*true*“ znamená, že byla provedena změna. Hodnota „*false*“ znamená, že vše již bylo korektně nastaveno.
- failed: Nastavení na „*true*“ znamená, že modul skončil s chybou. Hodnota „*false*“ znamená, že modul je funkční.
- msg: Obsahuje chybovou zprávu, pokud modul selhal, nebo informační zprávu o úspěšném skončení.

Obrázek 6 - Výstup programu test-module

```
ansible@server:~$ ./test-module -m wildfly_work/roles/wildfly-standalone-custom-module/libs/installwildfly -a 'APP=app01'
* including generated source, if any, saving to: /home/ansible/.ansible_module_generated
**********
RAW OUTPUT
{"failed": true, "msg": "Chyba yum install wget."}


**********
PARSED OUTPUT
{
    "failed": true,
    "msg": "Chyba yum install wget."
}
```

Zdroj: Vlastní zpracování

Testování zpracování zdrojového kódu

K ladění kódu příslušného modulu lze provést přímo pomocí Bash shellu a to spuštěním testovaného modulu i s potřebnými parametry příkazem *bash –x*. Výstupem bude kompletní přehled stavů všech příkazů, což umožní ověřit funkčnost celého skriptu. [11] Pokud je nutné ladit konkrétní část kódu, lze vepsat příkaz *set –x*, pro zapnutí xtrace, před problémovou část kódu a za ní pro vypnutí příkaz *set +x* [13] Bude-li využit příklad kód 3.4, který bychom uložili do souboru s názvem sethostname, tak příkaz pro spuštění tohoto kompletního testu bude vypadat následovně: *bash -x sethostname "hostname=testmachine.example.com"*

3.4 Jinja2 – šablonovací systém

Důležitou součástí hnací síly Ansiblu je šablonovací systém, který poskytuje dynamický obsah konfiguračních souborů, nahrazení hodnot modulů proměnnými, podmíněné příkazy a další funkce. Využití šablon přichází do hry s téměř každým aspektem Ansiblu. Šablonovacím systémem Ansiblu je Jinja2 moderní a snadno použitelný šablonovací jazyk pro Python. [26]

Jinja2 šablony

Jinja šablona je normální textový soubor, který může být generován v jakémkoli textovém formátu (HTML, XML, CSV, LaTeX atd.). Šablona obsahuje proměnné nebo výrazy, které se při vykreslení šablony nahradí hodnotami a značky, které řídí logiku šablony. Syntaxe šablony je významně inspirována Djangem[6] a Pythonem. [28] Soubory šablon nemají určenou specifickou příponu, ale jako poznávací prvek se na konci názvu používá extra přípona j2. [27] Umístění a vhodný název šablon j zobrazen ve složce *„templates“* na obrázku 16.

Použití proměnných s Jinja2

Použití šablon oproti statickým souborům přináší možnost dynamičnosti pro konfigurace aplikací. Toho lze využít v případech, kdy se v aplikaci mění pouze určitá pasáž konfigurace. Takovým příkladem může být nastavení dvou instancí, aplikačního serveru Wildfly, běžících na jednom hostu. Bude tedy nutné rozlišit nastavení portů tak, aby se na obě instance bylo možno připojit. Takovéto nastavení instancí může být použito v produkčním prostředí, kdy jedna instance přijímá „ostrý“ provoz a druhá slouží jako testovací z důvodu, že nelze otestovat určité funkce na nižších prostředích.

Pro ukázku definování proměnných lze využít danou pasáž konfigurace ze souboru group_vars/all (obrázek 16):

```
# Porty Wildfly
mgmt_http: 30001
mgmt_https: 30002
ajp: 30003
http: 30004
https: 30005
txn_recovery: 4712
txn_status: 4713
smtp: 25
```

[6] Django je open source webový aplikační framework napsaný v Pythonu, který se volně drží architektury Model-view-controller. [29]

Ukázka aplikování výše definovaných proměnných na dané pasáži konfigurace šablony templates/standalone.xml.j2 (obrázek 16):

```
<socket-binding name="management-http" interface="management" port="{{ mgmt_http }}"/>
<socket-binding name="management-https" interface="management" port="{{ mgmt_https }}"/>
<socket-binding name="ajp" port="{{ ajp }}"/>
<socket-binding name="http" port="{{ http }}"/>
<socket-binding name="https" port="{{ https }}"/>
<socket-binding name="txn-recovery-environment" port="{{ txn_recovery }}"/>
<socket-binding name="txn-status-manager" port="{{ txn_status }}"/>
<outbound-socket-binding name="mail-smtp">
<remote-destination host="localhost" port="{{ smtp }}"/>
```

Filtry

Filtry jsou součástí šablonovacího systému Jinja2. Ansible využívá Jinja2 pro vyhodnocení proměnných a šablon. V rámci vyhodnocení je možné použít filtry definované uvnitř složených závorek {{ … }}, které mohou být použité v daném playbooku nebo uvnitř šablonovacího souboru. Použití filtrů je podobné jako u unixových rour (pipeline)[7]. Jinja2 již obsahuje sadu zabudovaných filtrů, které je možné rozšířit o vlastní filtry. Velice užitečný je defaultní filter:

```
"HOST": "{{ database_host | default('localhost') }}"
```

Bude-li proměnná výše „*database_host“* definovaná, tak hodnota uložená v proměnné bude vyhodnocena. Pokud však nebude definována, tak ve složených závorkách bude vyhodnocen řetězec *„localhost“*. [8]

Filtry pro registrované proměnné

Tyto filtry využívají moduly *change_when* a *failed_when*, které je možno použít k ovlivnění reportování Ansiblu o tom, kdy určitá úloha má za následek změnu nebo selhání. Pro Ansible je obtížné vyhodnotit, zda příkazy zadané pomocí modulů „command“ nebo „shell“ provedli změny, protože budou-li takové příkazy použité bez filtru *change_when*, tak Ansible bude vždy hlásit změnu. Většina modulů hlásí, zda vedly ke změnám korektně, ale je možné toto chování přepsat vyvoláním *change_when*.

[7] Pipeline představuje prostředek realizace přesměrování dat z výstupu jedné entity zpracovávající data na vstup jiné. [30]

Ukázka použití *change_when*:

```
- name: Install dependencies via Composer.
command: "/usr/local/bin/composer global require phpunit/phpunit --prefer-dist"
register: composer
changed_when: "'Nothing to install or update' not in composer.stdout"
```

Modul „*register“* byl použit k uložení výsledků z modulu „command“. Dále bylo zkontrolováno, zda se v registrované proměnné *stdout* nácházel nějaký řetězec. V případě, že Composer neprovedl žádnou akci, tak na výstupu byl zobrazen řetězec „Nothing to install or update“. Použitím tohoto řetězce byl Ansible informován, že daný příkaz vedl ke změně. [27]

3.5 Paralelismus

Ansible se pro každou úlohu připojí k hostitelům paralelně a provede na nich dané úkony. Při větším počtu hostitelů se tento orchestrační nástroj nemusí nutně spojit se všemi hostiteli paralelně. Pokud provozujeme Ansible na dostatečně výkonném serveru, můžeme vhodným nastavením efektivně využít zdroje serveru a zvýšit počet souběžných spojení. Parametr zodpovědný za počet souběžných spojení je defaultně nastaven na hodnotu 5. Tento výchozí parametr je možné změnit jedním ze dvou způsobů.

1. Nastavením proměnné prostředí ANSIBLE_FORKS:

```
export ANSIBLE_FORKS=20
$ ansible-playbook playbook.yml
```

2. V konfiguračním souboru „ansible.cfg“ v části [defaults] je možné upravit parametr „forks“ na požadovanou hodnotu:

```
[defaults]
forks = 20
```

[8]

3.6 AWX: Otevřená varianta Ansible Tower

V této bakalářské práci byl použit Ansible pouze s CLI pod licencí open-source. Pokud však někomu nevyhovuje práce s CLI, protože raději pracuje s přehledným GUI, které navíc dokáže některé informace prezentovat pomocí grafu, tak může využít další varianty. Jednou variantou je Ansible Tower, ale tento nástroj je placený, a proto se jím nebudeme dále v této práci zabývat. Druhou variantou je projekt AWX, jehož vzniku napomohl RedHat tím, že uvolnil kódy Ansible Tower pod open-source licencí. Do vývoje tohoto projektu byla také zapojena komunita. Z AWX je následně odvozována placená verze Ansible Tower. Podobně jako je tomu s Fedorou a RHEL. Dále budou uvedeny funkce, které má AWX navíc oproti čisté verzi Ansiblu.

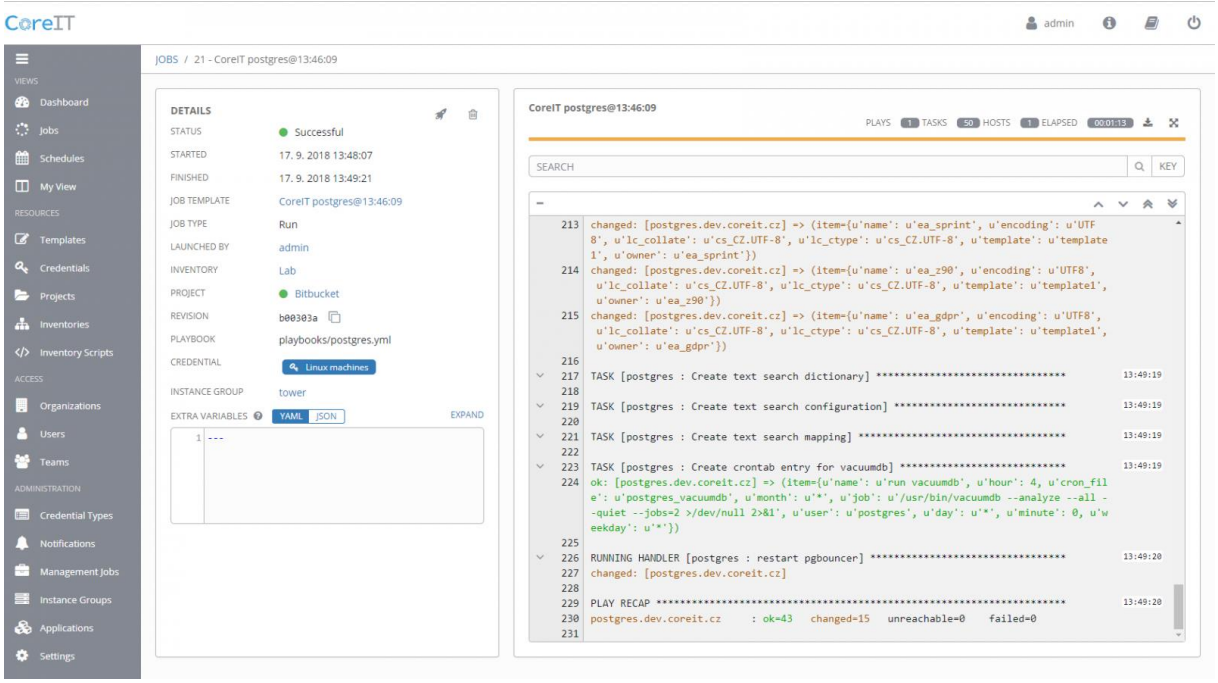
Delegace

Spustí-li se playbook v čisté verzi Ansiblu pod konkrétním uživatelem s přiřazeným inventářem, který bude spravován, tak celý proces bude probíhat s právy daného uživatele. To je skvělé pro rychlou orchestraci, ale tento způsob přestane být komfortní ve chvíli, když bude třeba delegovat takový proces na jiné uživatele. Znamená to poskytnout jim přístupová práva na příslušný inventární soubor a playbook na úrovni operačního systému. Při chybné konfiguraci mohou příslušní uživatele získat vyšší práva a dané soubory upravit. Pro takovou situaci přicházejí do hry delegační funkce AWX [37], který umožňuje vytvářet uživatele a seskupovat je do týmů (obrázek 8). Přístupová oprávnění k inventáři a playbookům lze přiřazovat konkrétním uživatelům nebo týmům. Toto řešení přináší výhodu, že automatizační procesy mohou být spuštěny stisknutím jediného tlačítka. V případě výpadku kritické služby v brzkých ranních hodinách postačí pouze restart této služby a vše začne znovu fungovat. Přesně takový případ nemusí provádět administrátor, nýbrž operátor, kterému bude potřeba vytvořit účet a nastavit práva pouze na playbook provádějící restart dané služby.

Auditing

Velkou výhodou při používání AWX oproti čisté verzi Ansible je možnost sledovat, kdo a kdy spustil jaký playbook, na jaké stroje a jak to dopadlo. AWX přidává do Ansible vlastní callback plugin, který zachytává výstup Ansible v reálném čase. Na dashboardu (obrázek 7) je prezentován přehled, který poskytuje informace o tom, jaké playbooky byly puštěny a jestli se provedly úspěšně nebo ne.

Obrázek 7 – AWX auditing



Zdroj: [32]

Pokud se běh některého playbooku nepovedl, je možno ho kliknutím otevřít a podívat se na detaily a problém odstranit. AWX zachytává výstup Ansiblu a umožňuje ve výpisu tasků kliknout na konkrétní task a nechat si v popup okně zobrazit všechna metadata, která jsou spojená s tím taskem, respektive modulem, který se v něm použil. Všechny tyto informace si AWX ukládá do PostgreSQL[8].

[32]

3.6.1 Koncepty a názvosloví v AWX

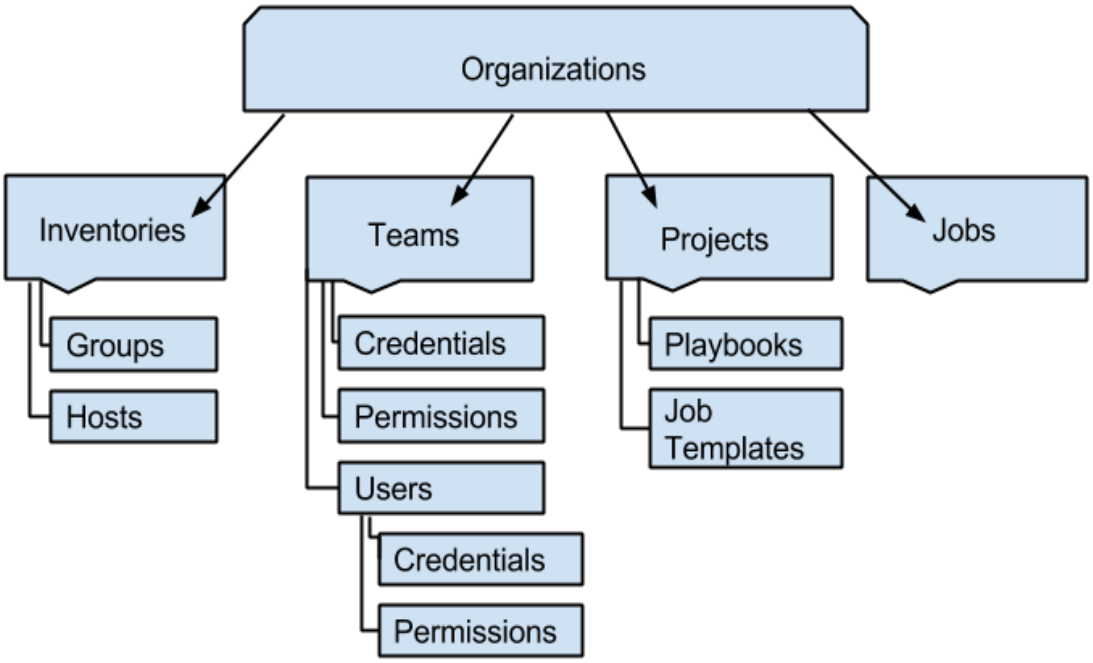
V následujícím textu budou představeny koncepty a názvosloví používané v AWX.

Organizace a týmy

AWX umožňuje vytvářet lokální účty, ale také se umí napojit na externí zdroje. Mezi nimiž najdete Azure, OAuth2, LDAP, Github, RADIUS, SAML, TACACS+. Nastavení připojení k LDAP serveru, je poměrně přehledné a pro člověka, který už někdy konfiguroval připojení k LDAP, by to nebyl problém zvládnout i bez dokumentace. AWX je také tzv. multitenantní. To znamená, že umožňuje spravovat více organizací v jedné instanci a garantuje, že uživatelé jedné organizace uvidí pouze své věci a nedostanou se k datům patřící jiné organizaci.

[8] PostgreSQL je objektově-relační databázový systém

Obrázek 8 – AWX organizace



Zdroj: [33]

Z obrázku 8 lze vyčíst, že objekt organizace je v AWX na nejvyšší úrovni a logicky slučuje objekty úkolů, týmů, projektů a inventářů. [32]

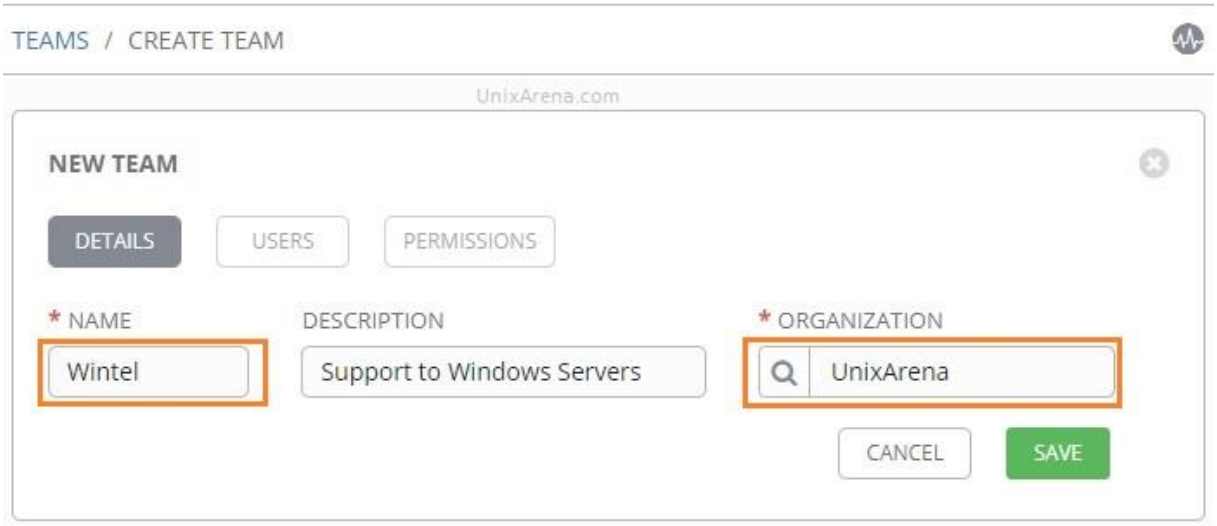
Teams

Tým slouží k seskupení uživatelů, projektů a práv. Skrze týmy jsou řízeny přístupy (RBAC[9]). Bude-li se dodržovat štábní kultura a vše se bude řešit pomocí týmů, tak v nastavení bude dosaženo lepší přehlednosti.

Týmů si můžeme v organizaci vytvořit libovolné množství. Jak už bylo zmíněno, tým může dostat oprávnění stejně jako uživatel. Stejně tak může dostat oprávnění použít konkrétní credentials. I když uživatel nebo tým může použít nějaké credentials, neznamená to, že je může vidět. To nám dává možnost udělit uživateli nebo týmu přístup, aniž bychom jim prozradili uživatelské jméno a heslo. Přehledné GUI usnadní vytvoření týmu (obrázek 9). [32]

[9] RBAC je způsob, jakým se omezují nebo přidělují práva jednotlivým uživatelům systému na základě jim přidělených rolí.

Obrázek 9 – AWX vytvoření týmu



Zdroj: [33]

Credentials

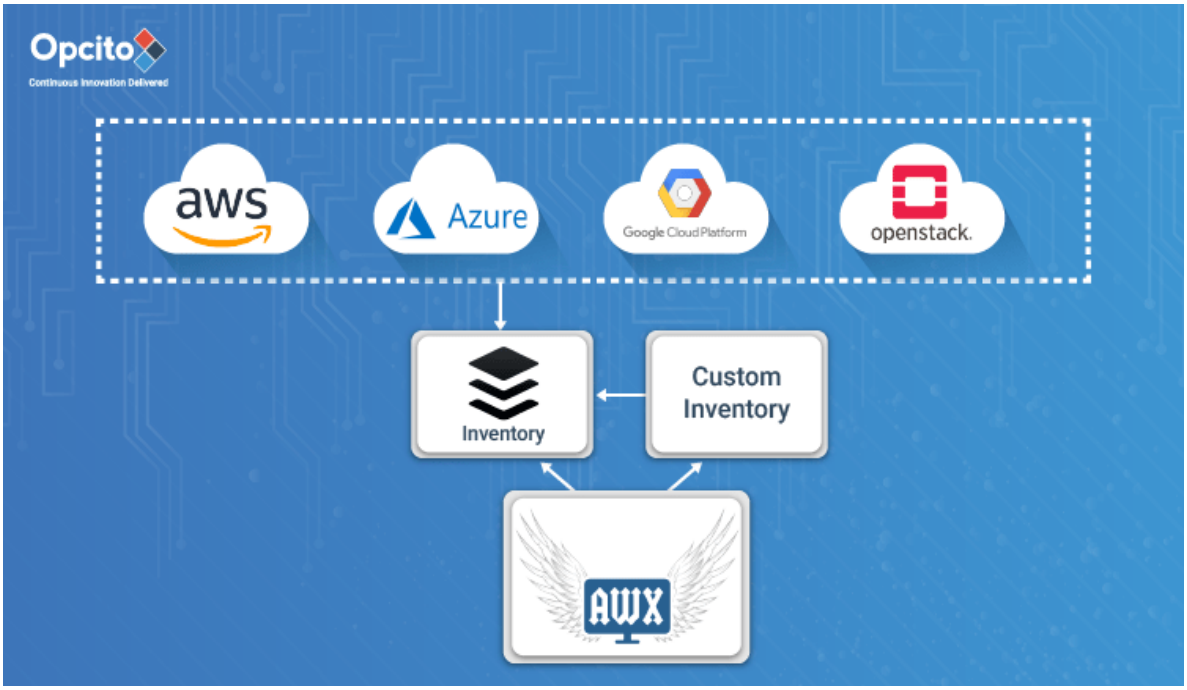
Jsou místo, kam je možno uložit všechny možné přístupy a ty pak použít například v momentě kdy AWX pouští Joby na servery, synchronizuje inventáře z externích zdrojů nebo importuje do projektu data z SCM (source code management). Jak již bylo zmíněno, přístupy je možné přidělovat uživatelům a týmům, avšak oni ho nemohou vidět. Pokud tedy někomu je přidělen přístup a ten člověk odejde z organizace, není třeba měnit jeho heslo. Všechna hesla uchovává AWX v šifrované podobě. [32]

Inventories

Inventář je souhrn hostů, na kterých se spouštějí úkoly, stejně jako v Ansible. To znamená, že můžeme hosty členit do skupin. Informace o hostech a skupinách lze zadat v AWX ručně nebo je naimportovat z jiného zdroje. Ano jedná se vlastně o dynamické inventáře, tak jak je známe z Ansible.

V AWX jsou na výběr nejběžnější poskytovatele cloudových služeb (obrázek 10), ale lze importovat i stávající inventář, který byl již hotový z doby, kdy byl používaný Ansible s CLI. Import z již existujícího projektu je poměrně chytrý a kromě samotných hostů jsou prohledány složky *group_vars* a *host_vars* a všechny nalezené proměnné budou naimportovány k daným hostům. Inventářů můžeme mít i více než jeden. [32]

Obrázek 10 – AWX inventory



Zdroj: [34]

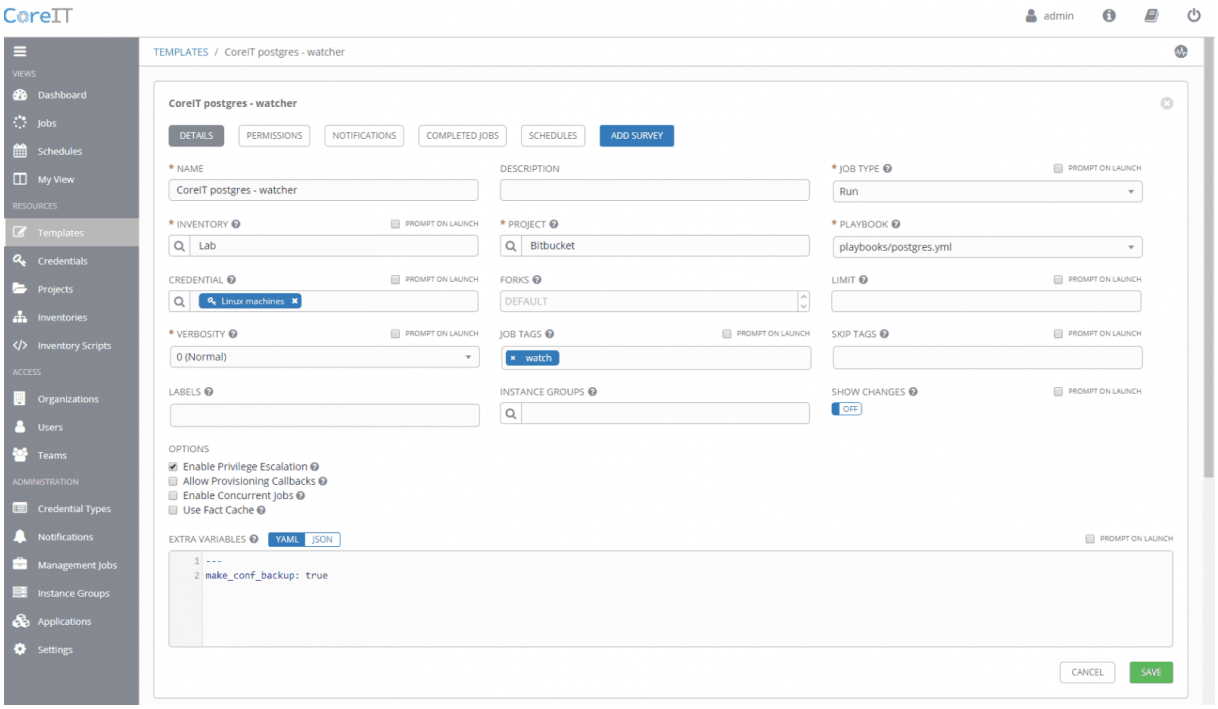
Projects

Projekt je kolekce playbooků, se kterými je možné v AWX pracovat. Samotné playbooky a složky s playbooky lze lokálně umístit na serveru AWX podle Project Base Path parametru, který je defaultně nastaven na /var/lib/awx/projects. Defaultní umístění může být modifikováno administrátorem. Mnohem flexibilnější je použít umístění v SCM. AWX umí pracovat dnes běžně používanými SCM, jako jsou Git, Subversion, and Mercurial. [38]

Jobs

Job je v podstatě instance na AWX serveru, která spouští playbook spravující přidělený inventář hostů. Daná instance běží jako proces na pozadí.

Obrázek 11 – AWX Job Template



Zdroj: [32]

Job Template

Job Template je definice playbooku a další množiny parametrů (obrázek 11), které jsou potřeba k jeho spuštění. Job Template asociuje skrze projekt playbooky, které máme uložené třeba v SCM. Přiřazuje inventory, na kterém se má pustit. Dále credentials, které je potřeba, aby se bylo možno přihlásit například k serveru pomocí SSH. Tohle všechno dohromady tedy vytváří Job Template. [32]

Notifications

AWX umožňuje nastavit různé způsoby notifikací. Například přes e-mail, Slack, Twilio, Pagerduty nebo Hipchat. Pokud nevyhovuje ani jedna z nabízených možností, není třeba zoufat, protože AWX ještě implementuje obecný způsob jakým je webhook. Díky tomu si lze například posílat informace do vlastního monitorovacího systému. Notifikace je možno nechat posílat při úspěšném nebo neúspěšném běhu Jobu. Užitečné je nastavit notifikace pro případ neúspěšného běhu a směřovat tyto notifikace na administrátory Job Template, respektive playbooků. [32]

Provision callbacks

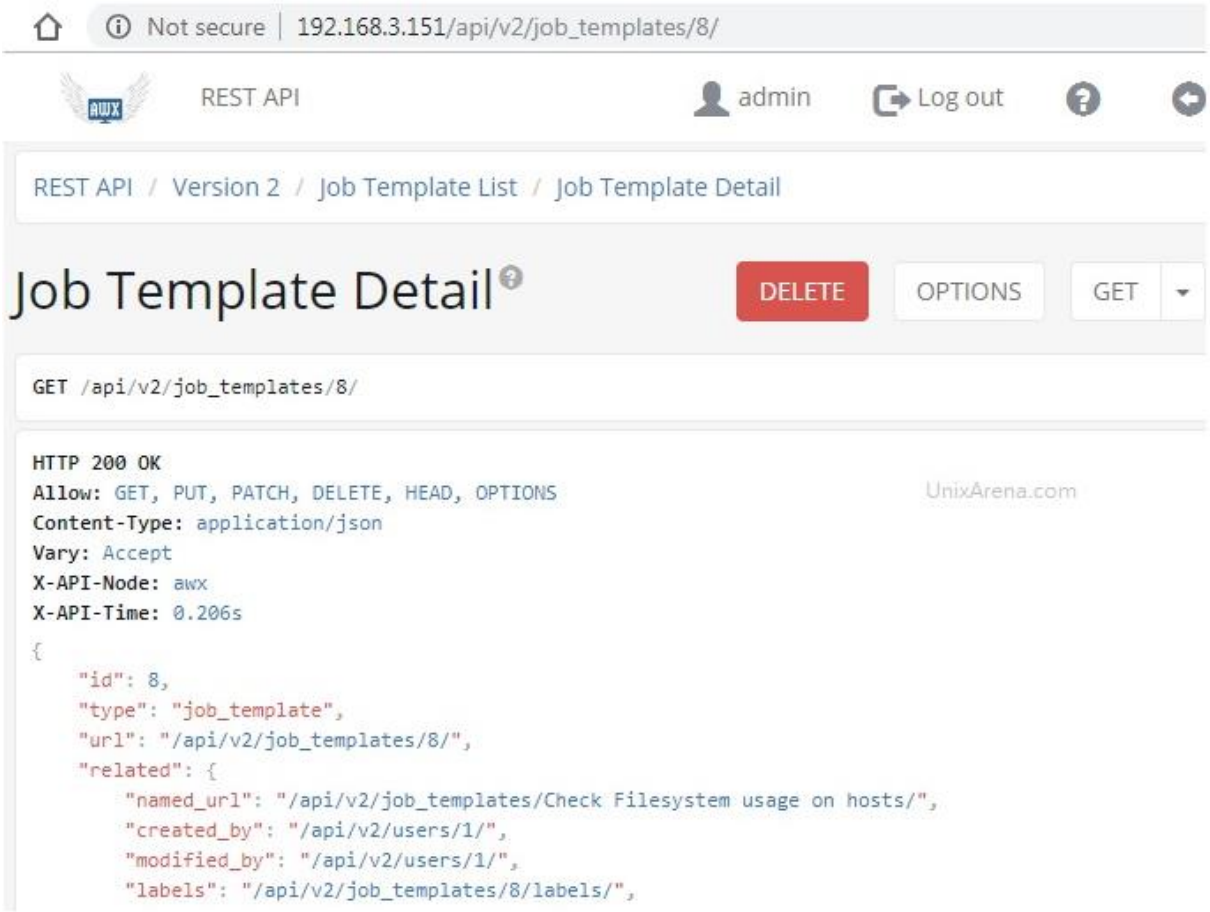
Provision callback je mechanismus, který umožňuje použít Ansible v pull módu. Je tedy možné, aby si o spuštění nějakého Job Template řekl daný host, místo toho aby to musel udělat

uživatel (nebo scheduler) v AWX. Je nutné si uvědomit, že Job Tempate se pustí pouze na onom stroji, který callback zavolal. Tento mechanismus je zde pro to, aby umožnil automatickou konfiguraci strojů, které byly automaticky vytvořeny jiným systémem. Například pomocí AWS auto-scaling. Typicky se toto bude spouštět ve skriptu při prvním bootu nebo z cronu. [32]

API

AWX stejně jako Ansible Tower využívá REST API. Záměrně budou vypsány základní charakteristiky tohoto rozhraní. REST je datově zaměřený, což jinými slovy znamená, že přístup ke zdrojům musí být popsán konkrétními daty. Dále využívá bezestavovosti, která kromě jiného vyžaduje aktivitu připojení od klienta. Pokud se zaměříme na architekturu AWX, všimneme si, že výše uvedené charakteristiky jsou dostupné na všech funkcionalitách ve webovém prostředí a to znamená, že jsou dostupné taky jako API. Na obrázku 12 je zobrazen přístup do Job Template pomocí REST API. [39] [40]

Obrázek 12 – AWX - Přístup pomocí REST API

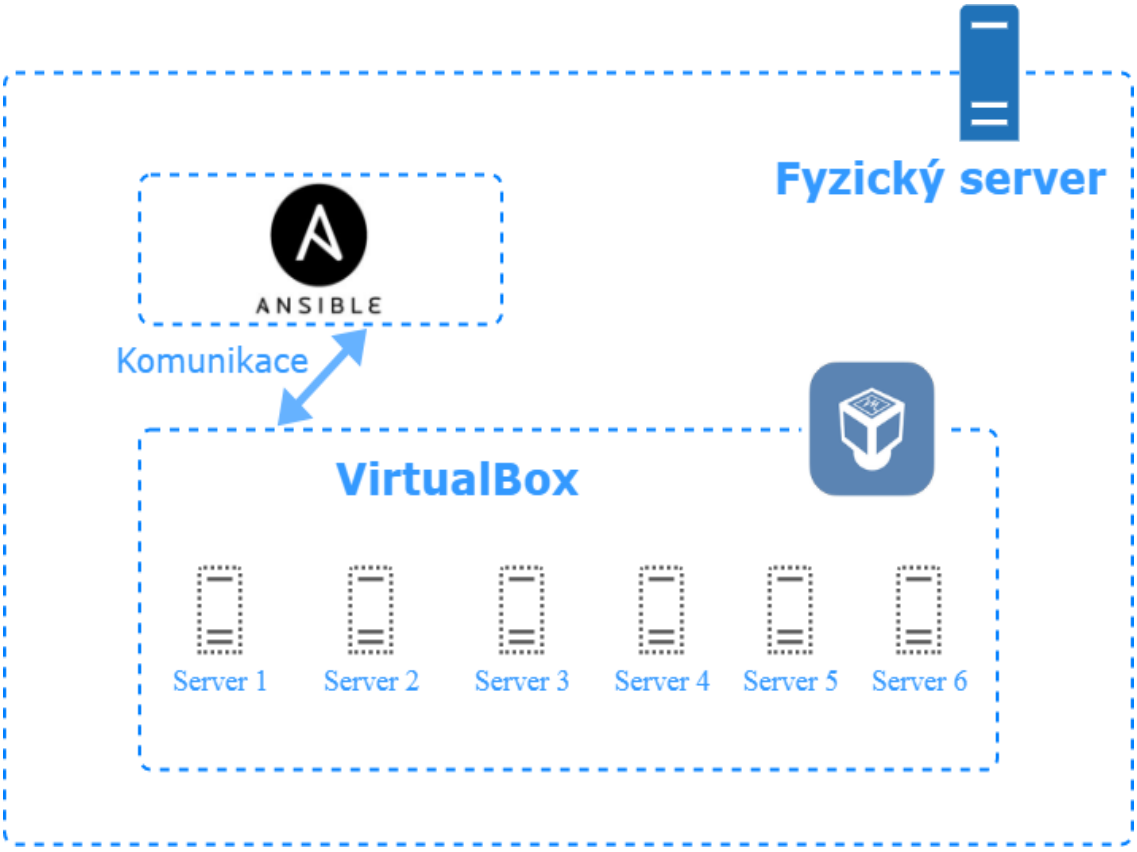


Zdroj: [41]

4 Praktická část

Kapitola pojednává o vlastním návrhu infrastruktury vytvořené softwarem VirtualBox a orchestračním procesům spuštěných na serveru Ansible. Tato kapitola se především věnuje detailnímu popisu jednotlivých procesů zpracovávající komplexní úlohu a také konkrétní konfiguraci softwaru Ansible a modelové infrastruktuře, na které je demonstrován celý návrh.

Obrázek 13 – Návrh infrastruktury



Zdroj: Vlastní zpracování

4.1 Návrh infrastruktury

Testovací infrastruktura byla navržena podle návrhu na obrázku 13. Návrh byl efektivně vytvořen na jednom fyzickém počítači díky možnosti virtualizace, která byla realizována za pomoci softwaru VirtualBox[10]. Samotný orchestrační nástroj Ansible byl nainstalován přímo na fyzickém počítači, jehož specifikace uvedená v tabulce č. 1 napovídá, že je k dispozici dostatek výkonu pro vytvoření šesti virtuálních serverů. Specifikace virtuálních serverů je

[10] VirtualBox je virtualizační nástroj. [14]

uvedena v tabulce č. 2 a výkon těchto serverů byl zvolen tak, aby mohla být provedena analýza časové náročnosti zvolených orchestračních procesů. Pro korektní vyhodnocení daných procesů bylo jejich zpracování provedeno nad homogenní komplexní úlohu, která je popsána níže v této části práce.

Parametry serverů

Tabulka 1 - Parametry fyzického počítače pro testování

ASUS ROG G55VW-S1231H	
CPU	Intel i7 3630QM
RAM	8GB
HDD	SSD 250GB
Operační systém	Ubuntu 18.04.2 LTS
Orchestrační nástroj	Ansible 2.8.5
Nástroj pro virtualizaci	VirtualBox 5.2.32

Zdroj: Vlastní zpracování

V tabulce č. 1 jsou uvedeny reálné hodnoty tohoto stroje. Tím, že na něm běží i virtuální stroj, je třeba zohlednit, že při běhu obou strojů současně bude část výkonu fyzického stroje zkonzumována.

Tabulka 2 - Parametry virtuálních serverů pro testování

CPU	1 Core
RAM	1GB
HDD	6GB
Operační systém	CentOs 7.5.1804
Python	2.7.5

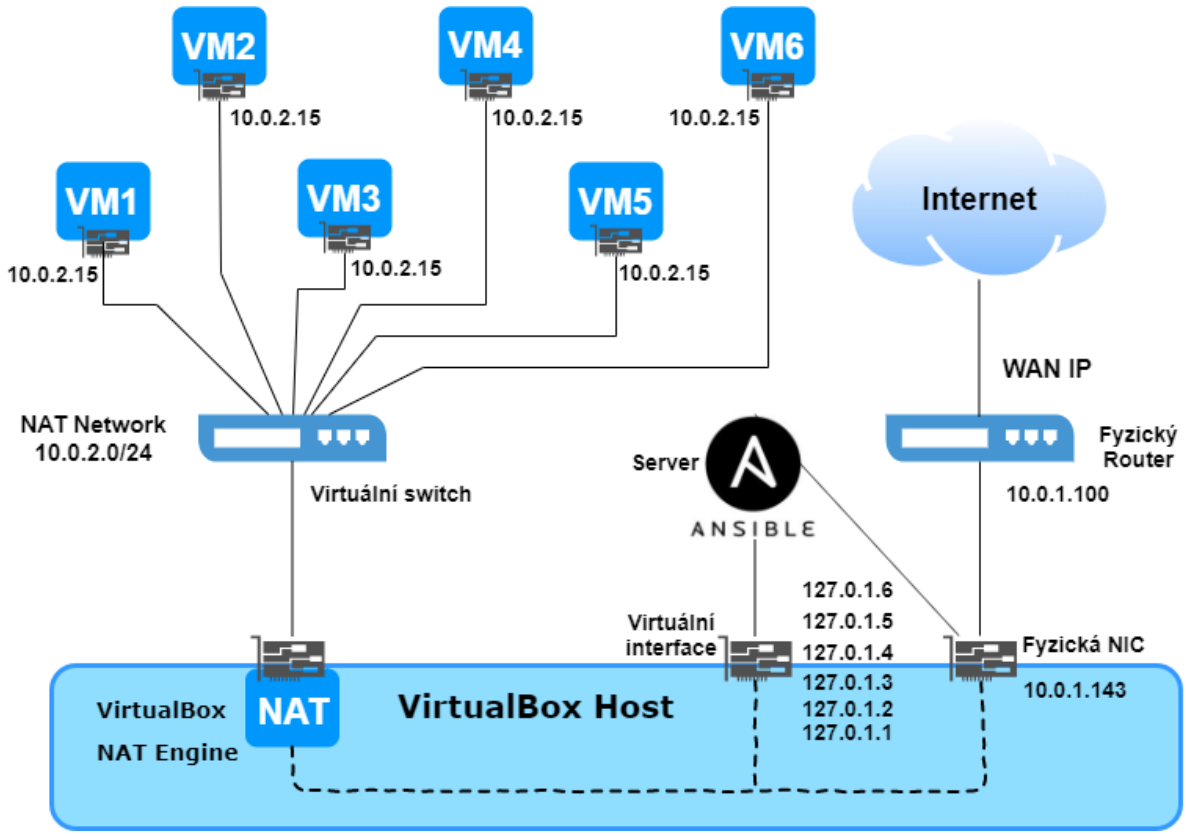
Zdroj: Vlastní zpracování

4.1.1 Návrh síťového propojení

Komunikace mezi Ansible serverem, virtuálními servery a internetem je znázorněna na obrázku 14. Podle návrhu si lze všimnout stejné ip adresy 10.0.2.15 na všech virtuálních serverech. To bylo způsobeno při vytváření modelové infrastruktury, kdy byl ve VirtualBoxu vytvořen jeden referenční server s právě uvedenou ip adresou, který byl následně duplikován pomocí funkce „Klonovat“. Navržené zapojení virtuálních serverů také nabádá k myšlence, že má-li několik serverů ve stejném subnetu stejnou ip adresu, musí dojít ke konfliktu ip adres. V režimu NAT [20], ve kterém jsou nastaveny jednotlivé servery, ke konfliktu nedojde, protože tento režim serverům neumožňuje vzájemnou komunikaci a také je každému přiřazen vlastní virtuální router. Za pomoci NAT Engine (obrázek 14) je komunikace mezi servery a Ansiblem zajištěna pomocí překladu portů a ip adres. Na obrázku 15 je zobrazeno nastavení NAT Engine

pro ssh komunikaci mezi fyzickým a virtuálním serverem VM1 (obrázek 14). Kompletní návrh předchozího nastavení je uveden v tabulce č. 3.

Obrázek 14 – Návrh síťového propojení



Zdroj: Vlastní zpracování

Obrázek 15 – VirtualBox nastavení serveru pro ssh komunikaci

Zdroj: Vlastní zpracování

Tabulka 3 – Návrh ip adres a portů pro virtuální servery

Virtuální server	IP adresa a port hostitele	Port hosta
VM1	127.0.1.1:10122	22
VM2	127.0.1.2:10222	22
VM3	127.0.1.3:10322	22
VM4	127.0.1.4:10422	22
VM5	127.0.1.5:10522	22
VM6	127.0.1.6:10622	22

Zdroj: Vlastní zpracování

4.2 Návrh orchestračních procesů

Byli vytvořeny dva návrhy orchestračních procesů. Jeden návrh byl vytvořen pouze pomocí nativních modulů dostupných v nástroji Ansible. Druhý je vytvořen také pomocí nativních modulů v kombinaci s vlastním modulem napsaného ve skriptovacím jazyce Bash. Za účelem odpovídajícího vyhodnocení doby zpracování byla pro oba návrhy vytvořena homogenní komplexní úloha, která popsána níže. Úloha byla zvolena tak, aby tvořila ucelený postup dílčích úkonů, které provedou konfiguraci a spuštění aplikačního serveru na systému Linux. Úloha byla komplexně navržena za účelem zachycení dostatečně velkého časového úseku, aby bylo možno s určitostí porovnat dobu zpracování obou variant.

Použitý software

Tabulka 4 – Použitý software pro aplikační server

Komponenta	Verze
Aplikační server	Wildfly-15.0.1.Final
Java	Openjdk-11.0.2_linux-x64
WAR[11] soubor	Staženo z internetu [23]

Zdroj: Vlastní zpracování

Dílčí úkony komplexní úlohy

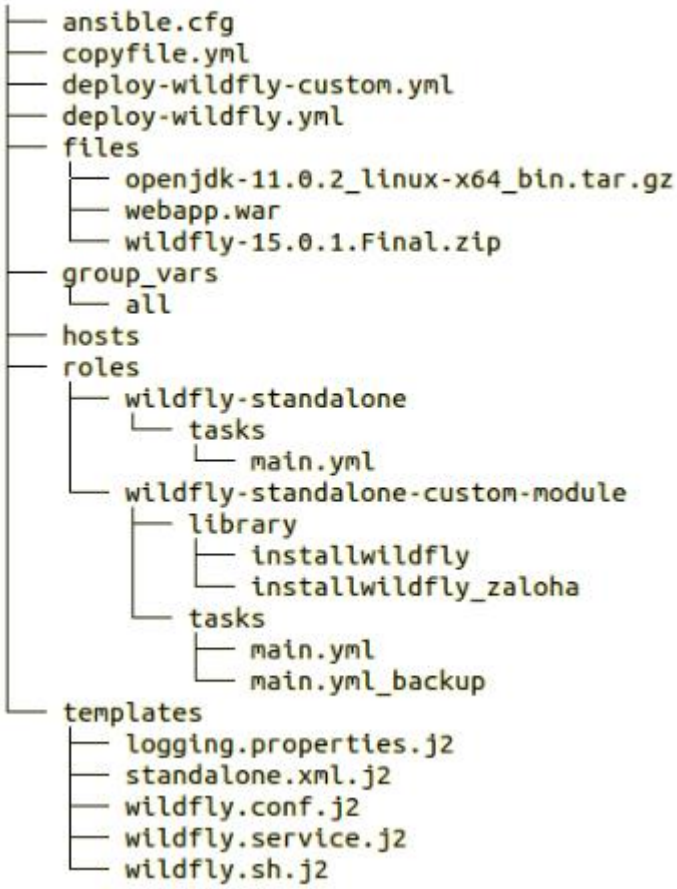
Pro test byly zvoleny následující úkony:

1. Stažení potřebných balíků z repozitáře.
2. Vytvoření uživatelů a skupin.
3. Vytvoření složkové struktury.
4. Kopírování archivačních souborů Wildfly a OpenJDK z Ansible serveru (obrázek 16) do dočasné složky na straně hosta.
5. Kopírování template z Ansiblu serveru (obrázek 16) do hosta.
6. Rozbalení archivů Wildfly a OpenJDK z dočasné složky do složky /opt/wildfly.
7. Změna vlastníka a skupiny složky s OpenJDK.
8. Vytvoření symlinků pro složky s Wildfly a OpenJDK.
9. Kopírování složky standalone na straně hosta obsahující defaultní konfiguraci pro aplikační server Wildfly.
10. Kopírování souborů z dočasné složky do umístění obsahující konfiguraci aplikačního serveru Wildfly.

[11] WAR je zkratka pro webový archiv. [23]

11. Nasazení WAR souboru obsahující aplikaci do aplikačního serveru.
12. Rekurzivní změna vlastníka a skupiny složky obsahující konfiguraci aplikačního serveru.
13. Smazání dočasné složky.
14. Start procesu aplikačního serveru pomocí systemd.

Obrázek 16 – Návrh konfigurace Ansible serveru



Zdroj: Vlastní zpracování

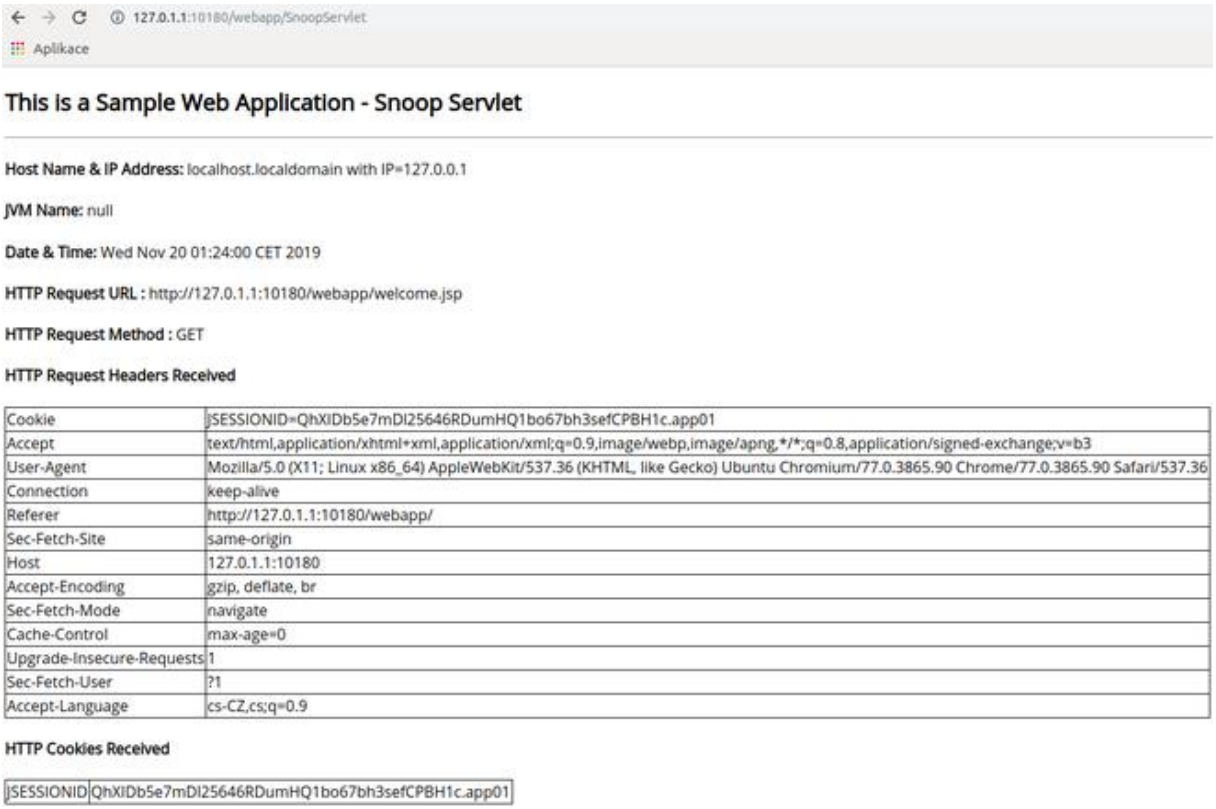
Kombinace vlastního modulu s nativními

Orchestrační proces s vlastním modulem používá nativní moduly v komplexní úloze v bodech 4 a 5. Navíc v rámci bodu 4 byl také stažen soubor war, jehož nasazení do aplikačního serveru již řeší vlastní modul. Tato dílčí konfigurace byla vytvořena za účelem využít funkce Ansiblu pro přenos souborů bez dodatečných programů a dosáhnout toho, že oba orchestrační procesy budou přenos souborů na dané hosty provádět totožným způsobem. Celkový čas zpracování daných procesů bude tedy nejvíce ovlivněn samotnou efektivností zvolených přístupů.

Ukázka spuštěné webové aplikace

Spuštěná webová aplikace (obrázek č. 17) na serveru VM1 (obrázek č. 14), kde proběhla orchestrace komplexní úlohy.

Obrázek 17 – Spuštěná webová aplikace

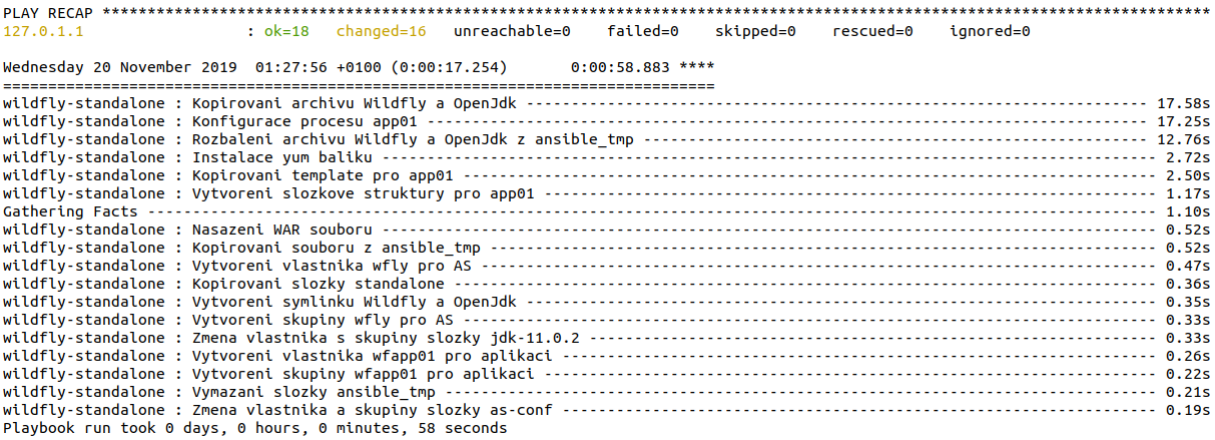


Zdroj: Vlastní zpracování

4.3 Získání časového údaje o době zpracování

Pro získání časového údaje pro každou roli bylo využito samotného Ansible, který disponuje integrovanými časovači. Přidáním hodnot timer nebo profile_tasks do volby „callback_whitelist“ v ansible.cfg (obrázek 16) se časovače aktivují. Hodnota timer poskytuje základní výstup, který zobrazen na posledním řádku v obrázku č. 18. Hodnota profile_tasks poskytuje podrobnější výstup s dobou zpracování jednotlivých modulů (obrázek 18). Výstup časovačů je vždy zobrazen po dokončení příslušného playbooku.

Obrázek 18 - Časovač implementovaný v Ansiblu



Zdroj: Vlastní zpracování

5 Výsledky a diskuse

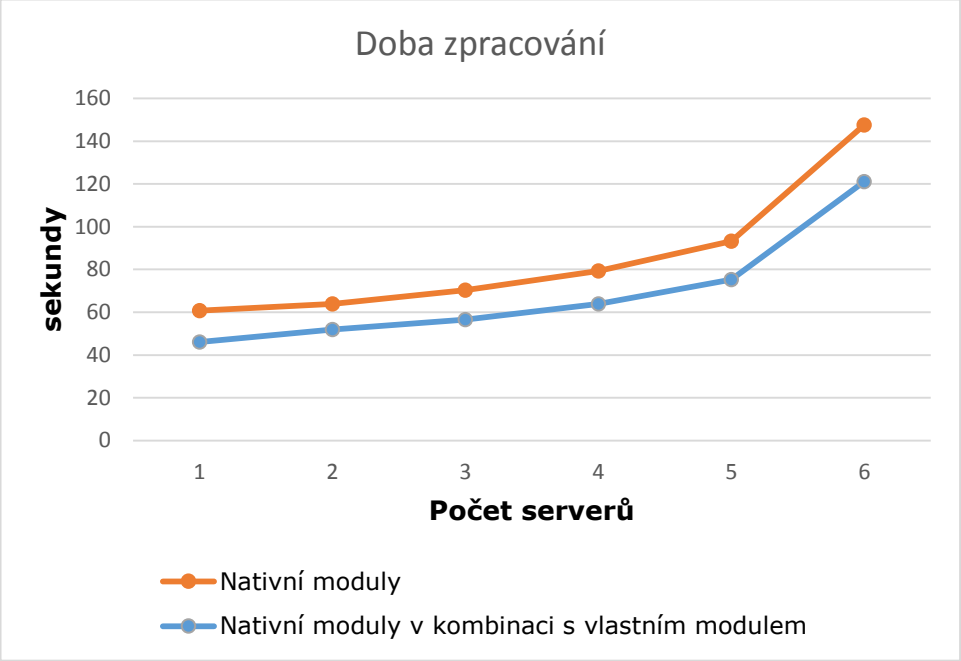
Výsledkem této bakalářské práce je funkční návrh infrastruktury, která slouží pro vyhodnocení časové náročnosti orchestračních procesů spuštěných v nástroji Ansible. V návrhu byla využita virtualizace, která přispěla k vytvoření infrastruktury bez použití fyzických serverů a síťových prvků. Doba zpracování orchestračních procesů je zachycena v tabulce č. 5, do které byli hodnoty z Ansiblu převedeny do jednotek sekund. Pro lepší přehlednost byly hodnoty také zobrazeny v grafu č. 1. Z grafického zobrazení je patrné, že zpracování komplexní úlohy se postupným přidáváním serverů prodlužuje u obou orchestračních procesů. V celém měření dominoval proces realizovaný vlastním modulem, který byl rychleji zpracován než ten s čistě nativními moduly. Ansible pomocí protokolu SSH přesouvá moduly na spravované hosty, kde jsou následně spuštěny. To znamená, že každá úloha představující jeden nativní modul bude jednotlivě zaslaná na hosta a spuštěna. Vlastní modul bude také přenesen v jednom volání, může však obsahovat více úloh a to je výhoda, která se projeví rychlejším zpracování.

Tabulka 5 – Doba zpracování orchestračních procesů

Počet serverů	Nativní moduly	Nativní moduly v kombinaci s vlastním modulem
1	60,718s	46,055s
2	63,828s	51,974s
3	70,326s	56,605s
4	79,282s	63,906s
5	93,226s	75,277s
6	147,556s	121,15s

Zdroj: Vlastní zpracování

Graf 1 - Doba zpracování orchestračních procesů



Zdroj: Vlastní zpracování

Pomocí statistického nástroje SAS 9.4 [21] byli oba naměřené soubory porovnány. Nejdříve byl proveden Shapiro-Wilkův test normality. Vyhodnocení testu je v tabulce č. 6, kde hodnota p obou souborů je větší než obvyklá referenční hladina významnosti α=0,05. Znamená to, že test nezamítá hypotézu o normalitě rozdělení těchto souborů. Tímto je možno provést dvouvýběrový t-test, který je vyhodnocen v tabulce č. 7. Zvolená hladina významnosti je stejná jako v předchozím testu. Hodnota p 0.7201 v sekci Equality of Variable znamená, že není důvod k zamítnutí předpokladu o rovnosti rozptylů. Bylo tedy vhodné pokračovat v testu při stejných rozptylech, který je uveden na řádce Equal v tabulce č. 7. Protože hodnota p je 0.3592, můžeme konstatovat, že se doba zpracování u obou orchestračních procesů statisticky významně neliší.

Tabulka 6 – Test normality

Test normality pro nativní moduly				
Test	**Statistic**		**p Value**	
Shapiro-Wilk	W	0.799779	Pr < W	0.0585
Test normality pro vlastní modul				
Test	**Statistic**		**p Value**	
Shapiro-Wilk	W	0.819452	Pr < W	0.0873

Zdroj: Vlastní zpracování

Tabulka 7 – Dvouvýběrový t-test

Method	Variances	DF	t Value	Pr > |t|
Pooled	Equal	10	-0.96	0.3592
Satterthwaite	Unequal	9.728	-0.96	0.3598
Equality of Variances				
Method	**Num DF**	**Den DF**	**F Value**	**Pr > F**
Folded F	5	5	1.40	0.7201

Zdroj: Vlastní zpracování

5.1 Celkové náklady softwaru s placenou podporou

V této kapitole bude provedena kalkulace pro jednotlivý software, který byl použit v návrhu praktické části této bakalářské práce. V návrh této práce byl použit všechen software v open-source verzích z důvodu vysokých nákladů podporovaných verzí. Bude-li tento návrh použit v produkčním prostředí s vysokým nárokem na dostupnost, tak přichází v úvahu použití verzí softwaru s jejich placenou podporou. Uvedené ceny (tabulka č. 8) nemusí být aktuální z několika důvodů. Jedním důvodem je dosavadní aktuálnost cen k listopadu 2019 a druhým důvodem je aktuálnost kurzu České koruny 23,151,- vůči Americkému dolaru ke stejnému datu jako v předchozím případě. Přepočet cen je proveden, protože vydavatelé softwaru uvádějí ceny v Americkém dolaru. Do ceny se zahrnují pouze položky za software, není zde započítána koupě nebo provoz serverů. Placená podpora bude navržena podle předpokládané infrastruktury, která obsahuje 15 virtuálních serverů se systémem Ubuntu. Každý server má procesor s 10 Core.

Tabulka 8 – Placená podpora softwaru

Položka	open-source položka	Položka s placenou podporou	Typ podpory	Délka podpory	Počet podporovaných serverů	Cena za rok
OS	Ubuntu/Centos	Ubuntu	Virtual Server Essential	1 rok	Za 1 virtuální server	(75*15) $1125,-
Ansible	Ansible	Ansible Tower	Standard	1 rok	Až do 100 serverů	$10,-
Java	OpenJdk 11	Oracle JDK 11	Java SE Subscription	1 měsíc	1-99 serverů	(25*12) $300,-
Aplikační server	Wildfly	JBoss EAP	n/a	1 rok	Za 150 Core	$105,-
Celkem v Kč						32572,54,-

Zdroj: Vlastní zpracování

6 Závěr

Bakalářská práce ze získaných poznatků poukazuje na několik scénářů využití jednotlivých přístupů orchestrace nástroje Ansible. Prvotním předpokladem bylo, že z provedené analýzy doby zpracování orchestračního přístupu vyplyne, že jeden z přístupů bude trvat výrazně kratší dobu než ten druhý. Následnou syntézou nebyl prvotní předpoklad potvrzen. Výsledkem statistické analýzy je, že se oba orchestrační přístupy statisticky významně neliší. Bylo však zjištěno, že použití jednotlivých přístupů má svůj účel v několika případech.

Prvním případem je, kdy v určitých podmínkách rozhodují ve zpracování jednotky sekund. V tomto případě podle grafu č. 1 dominuje přístup s vlastním modulem, který je napsán v některém skriptovacím jazyce a provádí na straně spravovaného uzlu více úloh, které by jinak musely být napsány jednotlivě pomocí nativních modulů. Dalším případem je situace, kdy je již vytvořen skript plnící komplexní úlohu, kterou by přepsat do varianty používající pouze nativní moduly, bylo časově náročné nebo nefunkční. Doporučením je implementovat skript jako modul. Posledním případem je provádění základních úloh např. založení uživatele, změna hesla. Takové úlohy je výhodné implementovat pomocí nativních modulů.

Orchestrační nástroj Ansible doporučuji díky jeho možnostem, jak orchestrovat procesy a možnosti získat tento nástroj jako bezplatný svobodný software.

7 Seznam použitých zdrojů

1. Ansible blog. The origins of Ansible [online]. [cit. 2018-09-28]. Dostupné z: https://www.ansible.com/blog/2013/12/08/the-origins-of-ansible
2. Ansible documentation. Working with Modules [online]. [cit. 2018-09-28]. Dostupné z: https://docs.ansible.com/ansible/2.6/user_guide/modules.html
3. Ansible documentation. *All modules* [online]. [cit. 2018-09-28]. Dostupné z: https://docs.ansible.com/ansible/latest/modules/list_of_all_modules.html
4. SHAH, Gourav. *Ansible Playbook Essentials*. Birmingham: Packt Publishing, 2015. ISBN 978-1784398293.
5. Ansible documentation. Working with Plugins [online]. [cit. 2018-09-28]. Dostupné z: https://docs.ansible.com/ansible/2.6/plugins/plugins.html
6. HALL, Daniel. *Ansible Configuration Management - Second Edition*. Second Edition. Birmingham: Packt Publishing, 2015. ISBN 978-1785282300.
7. Ansible documentation. Working with Inventory [online]. [cit. 2018-09-28]. Dostupné z: https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html
8. HOCHSTEIN, Lorin. *Ansible: Up and Running: Automating Configuration Management and Deployment the Easy Way*. Sebastopol: O'Reilly, 2015. ISBN 978-1491915325
9. Ansible documentation. *Roles* [online]. [cit. 2018-10-17]. Dostupné z: https://docs.ansible.com/ansible/2.5/user_guide/playbooks_reuse_roles.html#roles
10. GitHub. *ansible/hacking* [online]. [cit. 2019-02-22]. Dostupné z: https://github.com/ansible/ansible/tree/devel/hacking
11. GitHub. *Writing Ansible Modules in Bash* [online]. [cit. 2019-02-22]. Dostupné z: https://github.com/pmarkham/writing-ansible-modules-in-bash/blob/master/ansible_bash_modules.md
12. Ansible documentation. *Module Index* [online]. [cit. 2019-02-24]. Dostupné z: https://docs.ansible.com/ansible/latest/modules/modules_by_category.html
13. ALBING, Carl, J. P. VOSSEN a Cameron NEWHAM. *Bash cookbook*. Sebastopol, CA: O'Reilly, c2007. ISBN 978-0-596-52678-8.
14. VirtualBox. [online]. [cit. 2019-03-11]. Dostupné z: https://www.virtualbox.org/
15. Java-OpenJDK. [online]. [cit. 2019-03-11]. Dostupné z: https://openjdk.java.net/
16. Wildfly AS. [online]. [cit. 2019-03-11]. Dostupné z: https://www.wildfly.org/
17. Wikipedia. Cobbler(software) [online]. [cit. 2019-10-23]. Dostupné z: https://en.wikipedia.org/wiki/Cobbler_(software)
18. Red Hat to Acquire IT Automation and DevOps Leader Ansible [online]. [cit. 2019-10-23]. Dostupné z: https://www.redhat.com/en/about/press-releases/red-hat-acquire-it-automation-and-devops-leader-ansible
19. RATAN, Abhishek. *Practical Network Automation: Leverage the power of Python and Ansible to optimize your network*. Birmingham: Packt Publishing, 2017. ISBN 978-1788299466.
20. VirtualBox networking explained. [online]. [cit. 2019 -11 18]. Dostupné z: https://technology.amis.nl/2018/07/27/virtualbox-networking-explained/
21. Statistický software SAS. [online]. [cit. 2019-11-18]. Dostupné z: https://www.sas.com/cs_cz/academic-program.html

22. Introduction to Ansible. [online]. [cit. 2019-11-18]. Dostupné z: https://www.polarsparc.com/xhtml/Ansible-2.html
23. War file to Deploy and Test. [online]. [cit. 2019-11-18]. Dostupné z: https://www.middlewareinventory.com/blog/sample-web-application-war-file-download/
24. Advanced Packing Tool. [online]. [cit. 2019-11-18]. Dostupné z: https://cs.wikipedia.org/wiki/Advanced_Packaging_Tool
25. Nízkoúrovňové programovací jazyky. [online]. [cit. 2019-11-18]. Dostupné z: https://cs.wikipedia.org/wiki/Ni%C5%BE%C5%A1%C3%AD_programovac%C3%AD_jazyk
26. KEATING, Jesse. *Mastering Ansible*. Birmingham: Packt Publishing, 2015. ISBN 978-1784395483.
27. GEERLING, Jeff. *Ansible for DevOps: Server and configuration management for humans*. St. Louis: Midwestern Mac, 2015. ISBN 978-0986393419.
28. Template Designer Documentation. [online]. [cit. 2019-11-18]. Dostupné z: https://jinja.palletsprojects.com/en/2.10.x/templates/#builtin-filters
29. Django. [online]. [cit. 2019-11-18]. Dostupné z: https://cs.wikipedia.org/wiki/Django
30. Roura (UNIX). [cit. 2019-11-18]. Dostupné z: https://cs.wikipedia.org/wiki/Roura_(Unix)
31. What is ansible. [online]. [cit. 2019-11-18]. Dostupné z: https://www.educba.com/what-is-ansible/
32. AWX. [online]. [cit. 2019-11-18]. Dostupné z: https://www.root.cz/clanky/awx-otevrena-varianta-ansible-tower-pro-automatizaci-spravy/
33. Ansible Tower/AWX. [online]. [cit. 2019-11-18]. Dostupné z: https://www.unixarena.com/2018/11/ansible-tower-awx-organization-team-users-hierarchy.html/
34. Custom inventory management using Ansible AWX/Tower. [online]. [cit. 2019-11-18]. Dostupné z: https://www.opcito.com/blogs/custom-inventory-management-using-ansible-awx-tower
35. How to build your inventory. [online]. [cit. 2019-11-18]. Dostupné z: https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html
36. Bmc blogs. *Defining Automation and Orchestration*. [online]. [cit. 2019-11-18]. Dostupné z: https://www.bmc.com/blogs/it-orchestration-vs-automation-whats-the-difference
37. Security and Delegation with Ansible Tower. [online]. [cit. 2019-11-18]. Dostupné z: https://www.ansible.com/blog/security-and-delegation-with-ansible-tower-part-1
38. Projects. [online]. [cit. 2019-11-18]. Dostupné z: https://docs.ansible.com/ansible-tower/3.0/html/userguide/projects.html
39. REST. [online]. [cit. 2019-11-18]. Dostupné z: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/
40. Ansible Tower Tools. [online]. [cit. 2019-11-18]. Dostupné z: https://docs.ansible.com/ansible-tower/latest/html/towerapi/tools.html
41. AWX REST API. [online]. [cit. 2019-11-18]. Dostupné z: https://www.unixarena.com/2019/03/ansible-tower-awx-trigger-ansible-job-using-rest-api.html/