Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Deployment a správa aplikácií v Cloude pomocou Ansiblu a Terraformu

BAKALÁŘSKÁ PRÁCE

Studijní program: Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Autor: Ľubomír Bielik

Vedoucí bakalářské práce: Ing. Luboš Pavlíček

Praha, máj 2020

Prehlásenie

Prehlasujem, že som bakalársku prácu Deployment a správa aplikácií v Cloude pomocou Ansiblu a Terraformu vypracoval samostatne, za použitia v práci uvedených prameňov a literatúry.

V Prahe, dňa 10. mája 2020 .....

Ľubomír Bielik

Poďakovanie

Chcel by som sa poďakovať môjmu vedúcemu práce, Ing. Lubošovi Pavlíčkovi, za to, že si našiel čas na konzultácie a nápomocné rady k štruktúre práce, i v komplikovanej dobe karantény. Zároveň chcem poďakovať Lucií Šatkovej, za pomoc pri úprave textu.

Abstrakt

V mojej bakalárskej práci by som sa chcel venovať jednej z najaktuálnejších tém IT prostredia, a to Cloudu a jeho správe. Cloudová infraštruktúra sa stala za relatívne krátky čas veľmi rozšírená, vzhľadom na jej výhody. V niektorých prípadoch je pre spoločnosti výhodnejšia ako udržiavať si vlastné datacentrum, a s tým spojené komplementárne náklady a služby.

Každá infraštruktúra potrebuje správu, ktorú vykonávajú administrátori. Pri práci so štandardnou serverovou infraštruktúrou je táto správa zdĺhavá, a vyžaduje veľkú technickú odbornosť administrátorov. S príchodom Cloudu sa však požiadavky na technické zručnosti pracovníkov znižujú, pretože o technické zázemie sa stará správca Cloudu. Existuje veľké množstvo úrovní Cloudu, ktoré vyžadujú iné znalosti od ľudí, ktorí s nimi pracujú. Začínajú na IaaS (Infrastructure as a Service), pokračujú cez PaaS (Platform as a Service), až k SaaS (Software as a Service). V tejto práci sa bližšie pozriem na Cloud ako službu, a pokúsim sa predstaviť služby Cloudu, ktoré sú najviac využívané. Ako ďalší cieľ práce je priblížiť dva populárne nástroje používané na správu Cloudu, Terraform a Ansible. Sú používané v takzvaných DevOps nasadzovacích pipelinách, ktoré urýchľujú vývojový proces aplikácií a programov. Každý z týchto nástrojov má vlastné využitie, ktoré priblížim v prvej polovici práce. Taktiež sa bližšie pozriem na široké možnosti využitia Ansiblu. V praktickej časti ukážem výhody spojenia týchto dvoch nástrojov pri správe Cloudového prostredia AWS, a zároveň oba tieto nástroje porovnám pri deploymente jednoduchej infraštruktúry v Cloude.

Kľúčové slová

Ansible, Terraform, Cloud, DevOps, nasadenie, administrácia, správa, infraštruktúra

Abstract

In my bachelor thesis, I would like to address one of the latest subjects of IT scene, Cloud and its administration. Cloud infrastructure has become very popular in a relatively short time, due to its advantages. In some cases, it is more efficient for a company to have Cloud infrastructure, than owning a datacentre, with its complementary costs.

Every infrastructure needs administration, done by administrators. While working with standard server infrastructure, it can be tedious, and it often requires certain level of technical skills. With arrival of Cloud, these skill requirements are lowering, because Cloud provider is taking care of technical background. Number of Cloud levels exist, and each level requires a bit different skillset from people working with it. It begins with IaaS (Infrastructure as a Service), continues through PaaS (Platform as a Service), up to SaaS (Software as a Service). In my thesis, I would like to have a closer look at Cloud itself, and introduce most used Cloud services. Another goal of my thesis is to have a closer look at two popular tools used to manage Cloud, Terraform and Ansible. They are being used in so called DevOps deployment pipelines, which are being used to accelerate developing processes of applications. Each one of these tools has its own use, which I am going to address in first half of my thesis. Also, I would like to have a closer look on broad use-cases of Ansible. In practical part, I would like to show advantages of connecting these two tools in management of AWS Cloud environment, and I will compare these tools using an example of deployment of simple Cloud infrastructure.

Keywords

Ansible, Terraform, Cloud, DevOps, deployment, administration, management, infrastructure

Obsah

Úvod 8
1 Rešerš 9
2 O Cloude 11
2.1 Bežné služby Cloudu 12
2.1.1 Výpočetný výkon 13
2.1.2 Úložisko 14
2.1.3 Zálohovanie 14
2.1.4 Vlastné nastavenie siete 15
2.1.5 Databáza v Cloude 15
2.1.6 Dátová analýza 16
2.1.7 Kontajnerizácia 16
2.1.8 Governance a správa nákladov 17
2.1.9 Ostatné 17
2.2 Životný cyklus aplikácie v Cloude 18
3 Ansible 21
3.1 Ansible playbook 22
3.1.1 Inventár 23
3.1.2 Premenné 24
3.1.3 Tasky a Role 25
3.1.4 Handlery 26
3.2 Využitie Ansiblu 26
3.2.1 Nastavenie prostredia 26
3.2.2 Konfiguračný management 27
3.2.3 Deployment aplikácií 28
3.2.4 Continuous delivery 28
3.2.5 Automatizácia zabezpečenia 29
3.2.6 Orchestrácia 29
4 O Terraforme 30
4.1 Moduly 30
4.2 Premenné 31
5 Spojenie Ansiblu a Terraformu 33
5.1 Spoločný deployment aplikácií 34
6 Porovnanie využitia Ansiblu a Terraformu 42

6.1 Terraform 43
6.2 Ansible 45
6.3 Vyhodnotenie porovnania 46
Záver 48
Použitá literatúra 49
Prílohy I
Příloha A: Název první přílohy ..... **Error! Bookmark not defined.**
Příloha B: Název druhé přílohy ..... **Error! Bookmark not defined.**

Úvod

Náročnosť administrácie a udržiavania on-premise infraštruktúry sa môže stať výzvou pre všetky typy podnikov. Pred niekoľkými rokmi však bol svetu predstavený verejný Cloud, ktorý sa snaží zjednodušiť tieto úlohy, spojené s prevádzkovaním vlastnej serverovej infraštruktúry. Tému Cloudu a administrácie infraštruktúry som si vybral, pretože náplňou mojej práce je administrácia on-premise infraštruktúry a v určitých prípadoch aj Cloudu Microsoft Azure.

Spolu s tým prichádza množstvo nástrojov, ktoré je možné s Cloudovými prostrediami prepojiť alebo sú priamo na administráciu Cloudu určené. Dva z týchto nástrojov sú Ansible a Terraform. Tieto nástroje som vybral, pretože s Ansiblom už mám mierne skúsenosti, a Terraform je nástroj ktorý plánujeme v práci zaviesť pri administrovaní Cloudu. Tu vzniká aj inšpirácia na tému mojej práce. Zavedenie Terraformu spolu s Ansiblom by vedelo zjednodušiť administráciu Cloudu, a urýchliť vývojový proces aplikácií. V neposlednom rade, spojenie nasadzovania Cloudových zdrojov a následného okamžitého konfiguračného managementu by mohlo urýchliť proces nasadzovania produktov, ktoré využívajú naši zákazníci. Spojenie týchto dvoch aplikácií však nie je nutné, keďže každá z nich dokáže ako vytvárať zdroje v Cloude, tak ich aj ďalej spravovať. Preto sa chcem pozrieť aj na porovnanie pridanej hodnoty jednotlivých aplikácií počas rovnakej úlohy, a to samostatného nasadenia zdrojov.

1 Rešerš

Na rešerš som použil niekoľko internetových vyhľadávačov a databáz vedeckých článkov. Vzhľadom na to, že obe technológie sú relatívne mladé, nepodarilo sa mi nájsť toľko zdrojov ako som očakával. Vyhľadávače mi vracali malý počet relevantných výsledkov, no ako sa po bližšom skúmaní týchto zdrojov ukázalo, nie všetky boli vyhovujúce. Vyhľadávače ich zobrazili vďaka výskytu kľúčového slova v práci.

Ako prvý by som spomenul ACM Digital Library, do ktorej mám prístup vďaka VŠE. Svoje vyhľadávanie som bližšie špecifikoval reťazcom "+ansible +terraform", ktorý vo vyhľadávaní vynútil prítomnosť oboch slov vo výsledkoch. Touto cestou som získal 8 výsledkov, nanešťastie však ani jeden nebol pre mňa prínosný, pretože žiadny z nájdených článkov sa nezaoberal využitím a funkcionalitou týchto nástrojov a boli v nich spomenuté len ako príklady nástrojov na orchestráciu.

Po neúspechu v ACM Digital Library som hľadal na stránke Theses.cz, kde som pri hľadaní reťazca "ansible terraform" dostal 8 výsledkov, z ktorých dva najrelevantnejšie k mojej téme práce boli **Cloud and on-premise deployment platform for YSoft SafeQ** (M.Šalata, 2018, Diplomová práca, Masarykova univerzita - Fakulta Informatiky), kde bolo využitie Terraformu ako aplikácie deployujúcej virtuálne stroje v cloude, ktoré boli následne orchestrované Chefom. Práca sa však hlavne zaoberala aplikáciou Ysoft SafeQ a jej nasadením. (1) Druhá práca bola **Automated Infrastructure Deployment on Public and Private Cloud as a Part of Continuous Integration Process** (Š. Obetko, 2019, Diplomová práca, Masarykova univerzita - Fakulta Informatiky), ktorá sa zaoberala podobnou tematikou, a to deployment aplikácií v cloude. Práca sa však zaoberá deploymentom ako vo verejnom, tak v súkromnom Cloude a vychádza z prvej spomínanej (Cloud and on-premise deployment platform for YSoft SafeQ), z ktorej prevzala Chef ako orchestračný nástroj. (2) V práci **Cloud and on-premise deployment platform for YSoft SafeQ** bol spomenutý Ansible, no vzhľadom na to, že práca sa zaoberala výberom aplikácie na konfiguračný management, Ansible tam bol ako jeden z možných kandidátov na ďalšie použitie. Autor práce však uprednostnil nástroj Chef, ako som už spomínal. Vo vyhľadávači som našiel aj tri diplomové práce z Univerzity Hradec Králové, avšak k samotným prácam sa mi nepodarilo dostať vzhľadom na to, že žiadna z nich nebola dostupná v online archíve spomínanej univerzity.

Následne som využíval službu Google Scholar, v ktorej som sa snažil nájsť články a vedecké práce zaoberajúce sa spoločným využitím spomínaných technológií. Opäť som vyhľadával reťazec “ansible terraform”, s navráteným počtom výsledkov 295. Našiel som knihu, ktorá je k mojej téme relevantná, s názvom **DevOps for SharePoint: With Packer, Terraform, Ansible, and Vagrant** (2018, ISBN: 978-1-4842-3688-1). V kapitole 5: Provisioning the SharePoint Farm to AWS Using Terraform and Ansible sa nachádza podrobný návod na nasadenie SharePoint farmy do AWS. Obe aplikácie zastávajú odlišnú rolu pri nasadzovaní s tým, že Terraform je využitý na vytvorenie samotnej infraštruktúry v AWS ako sieť, databázy a samotné stroje, zatiaľ čo Ansible sa stará o inštaláciu a nastavenie samotného SharePointu do predom vytvoreného prostredia. (3) Zvyšné výsledky vyhľadávania opäť postrádali relevanciu k mojej téme, pretože na to, aby ich vyhľadávač vrátil, stačilo, aby mali v sebe kľúčové slovo. Našiel som však knihy, ktoré súvisia s jednotlivými technológiami použitými v tejto práci.

Pri opise samotných technológií som pristúpil k oficiálnej dokumentácií a k nasledujúcim knihám, ktoré sa mi podarilo dohľadať pomocou Google Scholar: **Ansible: Up and Running** (2. edícia, 2017, ISBN: 978-1-491-97980-8), z ktorej som čerpal informácie na popísanie samotného Ansiblu, jeho využitia a konceptov. Ďalšou knihou, z ktorej som čerpal je **Terraform: Up & Running: Writing Infrastructure as Code** (1. edícia - Early Release, 2016, ISBN: 978-1-491-97703-3). Z tejto knihy som získal informácie, ako je popis Terraformu a jeho definícia. Zároveň som sa z nej snažil načerpať poznatky o Terraforme a jeho používaní.

2 O Cloude

Cloud je moderná platforma na vzdialené využívanie výpočetného výkonu a služieb pomocou internetu od verejných prevádzkovateľov. Jedna z najdôležitejších vecí, čo sa týka Cloudu je to, že všetky dáta a operácie s nimi sa vykonávajú na vzdialených serveroch, ktoré sú umiestnené v datacentrách providerov. Tí pomocou internetu dodávajú svojim zákazníkom okrem výpočetného výkonu a úložiska aj širokú paletu iných služieb ponúkaných v Cloude. Cloud by sa dal preto popísať aj ako virtuálna skupina rôznych zdrojov, využiteľných na spracovanie a úložisko dát, aj chod aplikácií, ktoré sú dostupné na požiadanie zákazníka. Predaj týchto zdrojov je automatizovaný, čím je zaistená flexibilita a ich vysoká dostupnosť.

Cloud sa dá rozdeliť podľa niekoľkých vlastností. Ako prvé rozdelenie je rozdelenie na verejný, hybridný a súkromný Cloud. **Verejný Cloud** je dostupný pre všetkých pomocou internetu a verejného portálu, v ktorom si zákazníci objednávajú svoje zdroje, využívajú ich a služby na ne viazané a platia za ne. Stačí sa zaregistrovať do portálu spoločnosti ponúkajúcej Cloud a okamžite je možné nakupovať a využívať tieto služby. Zároveň je to najrozšírenejšia forma Cloudu. Verejné cloudy sú asi tie najznámejšie, napríklad Amazon Web Services (AWS), Microsoft Azure, IBM Cloud a iné.

Na druhej strane stojí **súkromný Cloud**, ktorý sa nevída tak často. Z istého uhla pohľadu sa ani nemusí považovať za Cloud. Je to služba prevádzkovaná na vlastnom hardware zákazníka, ktorá je dodávaná externou firmou. V niektorých prípadoch toto riešenie môže byť interné v rámci jednej spoločnosti, čo znamená, že ako hardware tak software, na ňom nainštalovaný, sú pod správou spoločnosti, ktorá tieto prvky využíva. Toto riešenie ako Cloudové v skutočnosti vnímajú len koncoví užívatelia. VMware Software Defined Data Center je asi najlepší príklad riešenia pre súkromný Cloud.

Hybridný Cloud je spojenie dvoch predchádzajúcich riešení do jedného celku, v ktorom je využívaná infraštruktúra zákazníka, zdroje od verejného poskytovateľa Cloudu a to všetko je mierené na užívateľov, ktorí túto službu vnímajú ako Cloudovú. Ako príklad by som si dovolil uviesť službu Plus4U od spoločnosti Unicorn, ktorá funguje ako v súkromnom Cloude, tak v Microsoft Azure a je využívaná ako interný informačný systém. (4, 5)

Ďalšie rozdelenie je podľa typu ponúkaných služieb. Asi každý súčasný poskytovateľ ponúka buď všetky alebo drtivú väčšinu týchto typov služieb, pre čo najširšie pokrytie trhu a zisky. Štandardne sa služby delia na IaaS (Infrastructure as a Service), PaaS (Platform as a Service)

a SaaS (Software as a Service), no v súčasnosti sa pridávajú aj iné typy služieb, ako napríklad FaaS (Function as a Service).

Infrastructure as a Service funguje na princípe prenájmu infraštruktúry ako sú servery, virtuálne stroje, privátne siete, úložisko a služby s tým spojené, napríklad monitoring, databázy a podobne. Existujú dva spôsoby platby za tieto služby a to buď mesačný paušálny poplatok alebo pay-as-you-go systém, kde zákazník zaplatí len za čas (prípadne kapacitu, alebo inú jednotku), ktorý reálne využije.

Platform as a Service znamená, že zákazník si prenajíma pripravené prostredie, v ktorom poskytovateľ dopredu všetko nastaví. Tieto prostredia sa využívajú napríklad na vývoj aplikácií, ich testovanie, beh a podobne. Pri PaaS sa zákazník nemusí vôbec starať o základnú infraštruktúru, ktorá je potrebná pre chod jeho aplikácie ako napríklad korektné nastavenie siete, operačný systém, súborový systém a mnohe iné. Len sa pripojí na pripravené prostredie a môže ho hneď začať využívať. Možnosti platby za služby PaaS sú rovnaké ako v prípade Iaas.

Software as a Service predstavuje hosting a kompletnú správu softwaru a infraštruktúry s ním spojenej od poskytovateľa. Užívateľ si vo väčšine prípadov len nainštaluje klientskú aplikáciu (vo veľkom množstve prípadov je možné využiť internetový prehliadač) a pripojí sa k serverom poskytovateľa. Pomocou klientskej aplikácie využíva všetky funkcie softwaru. Príklad tohto softwaru je Office 365 od spoločnosti Microsoft. V tomto prípade prevláda platba mesačného paušálneho poplatku pred pay-as-you-go systémom. (5)

2.1 Bežné služby Cloudu

Cloud sa rozšíril medzi zákazníkov ako platforma ponúkajúca štandardnú paletu služieb, aké si dokážeme predstaviť v On-Premise datacentrách. Klasické servery ponúkajúce výpočetný výkon, diskové polia zastrešujúce úložný priestor diskov, niekedy hardwarové bezpečnostné prvky nasadené na ochranu, napríklad IDS a IPS systémy na vlastných strojoch, Firewally na vlastných strojoch, a samozrejme, sieťové prvky spájajúce všetky tieto komponenty dohromady s okolitým svetom. Je však potrebné spomenúť, že na serveri s dostupným výpočetným výkonom sa dajú nainštalovať rôzne aplikácie a nástroje, a tým ich premeniť takmer v hocičo, od spomínaných IDS a IPS systémov, až po stroje poskytujúce základňu neurónovým sieťam a umelej inteligencií. Všetok tento majetok si však vyžaduje

obrovské množstvo zdrojov na zaobstaranie a údržbu, či už jednorazových nákladov pri zakúpení alebo tých pravidelných, ako sú mesačné náklady na elektrinu, chladenie, zamestnancov, ktorí sa o tieto zariadenia a softvér starajú a podobne. Veľké množstvo spoločností si preto nemohlo dovoliť udržiavať tak veľký výpočetný výkon. Toto všetko sa zmenilo s príchodom Cloudu, ktorý všetky tieto základné služby sprístupnil oveľa širšej palete zákazníkov ako kedy predtým.

Verejné Cloudové služby existujú na trhu už od roku 2006, kedy spoločnosť Amazon prišla s produktami Simple Storage Service (V skratke S3) a Elastic Compute Cloud (V skratke EC2). Služba S3 je od počiatku stavaná a plánovaná ako služba zaoberajúca sa ukladaním dát a jeho následným spracovaním. V Cloudovej infraštruktúre zastupuje miesto diskových polí, na ktoré sa ukladajú dáta. Narozdiel od S3, EC2 je služba ponúkajúca výpočetný výkon na chod rôznych aplikácií, operačných systémov, funkcií a tak ďalej. Táto služba nahrádza výpočetný výkon klasických serverov. Od roku 2006 však ubehlo 14 rokov a Amazon nie je jediná spoločnosť ponúkajúca Cloudové služby. Trh sa rozšíril ako počtom konkurenčných firiem, tak aj počtom služieb a funkcií, ktoré tieto spoločnosti ponúkajú. V tejto kapitole by som sa rád pozrel na služby, ktoré AWS ponúka a zhodujú sa s portfóliom konkurenčných spoločností. (6)
Pôvodne som chcel nájsť zoradenie služieb podľa ich popularity, no nepodarilo sa mi nájsť zdroj, ktorý by som považoval za dôveryhodný. Zdroje na túto tému ktoré som našiel boli väčšinou blogy, ktoré sa líšili ako zoradením jednotlivých služieb, tak časom v ktorom boli publikované. Pomocou služieb zhodných s konkurenciou by som chcel vytvoriť prienik a ním vymedziť produkty, ktoré sa dajú považovať za bežné služby Cloudu. Prienik som hľadal medzi 4 spoločnosťami, ktorých Cloudové služby by sa dali vyhlásiť ako najväčšie a najznámejšie. Sú nimi Amazon AWS, Microsoft Azure, Google Cloud Platform a IBM Cloud. Snažil som sa ku všetkým službám uviesť aj konkrétny príklad, no ku niektorým to bolo obtiažne vzhľadom na ich obšírnosť.

2.1.1 Výpočetný výkon

Ako prvá a základná služba Cloudu ktorú popíšem je služba výpočetného výkonu. Každý provider má túto službu pomenovanú inak: U AWS je to Amazon Elastic Compute Cloud (EC2), u Microsoft Azure to je Virtual Machines, u Google Cloud Platform to je Compute Engine a IBM Cloud ju pomenoval Cloud Virtual Servers. Táto služba sa radí do triedy IaaS (Infrastructure as a Service). Najčastejšie sa poskytuje vo forme virtuálnych strojov, pri čom má každý virtuálny stroj špecifikovaný hardware, s ktorým bude dostupný. Štandardne je to počet jadier CPU, veľkosť pamäte RAM, typ pripojeného disku, rýchlosť a počet sieťových

pripojení. Veľmi často sa dajú pri vytváraní instancie virtuálneho stroja lepšie špecifikovať aj ďalšie nastavenia. Medzi nimi sú sieťové nastavenia, ktoré umožňujú napríklad zakazovať otvorenie sieťových portov alebo vytvorenie verejnej IP adresy pre okamžitú dostupnosť stroja do internetu. Pri vytváraní je taktiež potrebné zvoliť image operačného systému, s ktorým bude stroj fungovať. Všetky Cloudy ponúkajú veľkú paletu operačných systémov rôznych verzií a to ako Linux, tak aj Windows Server. Po konečnej konfigurácií a nasadení virtuálneho stroja je možné sa k nemu pripojiť štandardizovanými protokolmi ako je SSH a RDP. Po pripojení je možné ich začať okamžite využívať a inštalovať na ne aplikácie aké si užívateľ praje a využívať potenciál výpočetného výkonu, ktorý mu je k dispozícií. (7, 8, 9, 10)

2.1.2 Úložisko

Ďalšia zo základných služieb je úložisko. Po tom, čo zákazník bude mať k dispozícií výpočetný výkon, musí spracované dáta niekam ukladať. Na to slúži služba úložiska dát, ktorá sa v AWS nazýva Simple Storage Service (S3), v Microsoft Azure to je Storage Account, v Google Cloud Platform je pomenovanie pre túto službu Cloud Storage a v IBM Cloude je to Object Storage. Úložiská služieb nerozlišujú medzi formou dát, ktoré boli do nich uložené, čo znamená, že sa dajú využiť na veľmi veľa použití naraz. Od obsahu pre webové stránky, obsahu mobilných aplikácií, štrukturovaných alebo neštrukturovaných dát, až po zálohy zmenšené kompresnými algoritmami. Tieto dáta môžu mať krátky životný cyklus alebo môžu byť naopak určené na archiváciu. Zároveň je štandardom možnosť definovať prístupovú politiku k dátam, čím sa dajú ochrániť pred zneužitím a zároveň sa dá umožniť požadovaný prístup oprávneným osobám. (11, 12, 13, 14)

2.1.3 Zálohovanie

Spolu s úložiskom a ukladaním dát mi napadne otázka zálohovania. Zálohovanie je jedno z najdôležitejších bezpečnostných opatrení, ktoré sa momentálne používa. Najväčšou výhodou, pre ktorú sa stalo jedným z kľúčových nástrojov ochrany dát je to, že sa vytvára bod obnovy, do ktorého je možné sa vrátiť. Pri zlom spracovaní dát, pri nechcenom vymazaní, pri poškodení pamäťového média, kde sa dáta nachádzajú a pri podobných prípadoch je často obnova dát zo zálohy najrýchlejšie a najspoľahlivejšie riešenie. Nevýhoda je však zaplnenie až niekoľko násobného objemu ukladacieho priestoru pre potrebu uchovania záloh. Každý z poskytovateľov Cloudu ponúka zálohovanie buď ako samostatnú službu alebo ako súčasť inej dátovej služby. Spoločnosť Google ponúka zálohovanie ako

súčasť služby Cloud Storage, respektíve ako jedno z možných využití tejto služby. Zvyšné spoločnosti majú zálohovanie ako samostatný produkt, konkrétne AWS ako AWS Backup, Microsoft Azure ako Azure Backup a IBM ako IBM Cloud Backup. (15, 16, 17, 18)

2.1.4 Vlastné nastavenie siete

Po zálohovaní by som rád spomenul nastavenie siete. V súčasnom svete internetu je sieť jedna z kľúčových súčastí každej infraštruktúry. Inak to nie je ani v cloude. Každý cloudový provider ponúka detailné nastavenie siete, pomocou ktorej je možné spájať zdroje vytvorené v cloude. Od nastavenie jednotlivých podsietí, pravidiel na týchto podsieťach, až po nastavenie špecifických portov. Štandardné sieťovanie sa však v Cloudoch berie ako samozrejmosť. Zaujímavosťou, akú Cloudový provideri ponúkajú, je však využitie ich vlastných sieťových prepojení datacentier, takzvaných Content Delivery Networks (CDN), čo sú vyhradené sieťové prepoje medzi datacentrami s pridanými vstupnými bodmi mimo datacentier pre čo najväčšiu dostupnosť. Tieto siete sľubujú väčšiu rýchlosť, stabilitu a bezpečnosť ako verejné internetové siete, pretože sú vyhradené priamo pre zákazníkov Cloudu. Touto cestou je možné znížiť latenciu a zrýchliť presúvanie dát medzi jednotlivými strojmi naprieč zemeguľou. Každý z mnou spomínaných providerov túto možnosť ponúka, a to menovite: AWS ako Amazon CloudFront, Microsoft Azure ako Content Delivery Network, Google ako Cloud CDN a IBM taktiež ako Content Delivery Network. (19, 20, 21, 22)

2.1.5 Databáza v Cloude

Nasledujúcou službou bežne ponúkanou v Cloude, a taktiež požadovanou zákazníkmi, je databáza. Databáz v súčasnosti existuje niekoľko druhov. Okrem klasického SQL, ktoré musí mať presne zadefinovanú štruktúru a na dotazovanie sa používa striktný query jazyk, je to aj NoSQL, čo vo väčšine prípadov znamená ukladanie dát ako objektov. Táto databáza nemusí mať nijako prísne zadefinovanú štruktúru, keďže tá sa definuje až pri ukladaní dát. Dôležité je len dodržať formát vstupného objektu. Najčastejšie sa v týchto databázach využíva definovanie objektov vo formáte JSON. Asi najpopulárnejšia z NoSQL databáz je MongoDB. Každý z providerov ponúka niekoľko možností databáz, ktoré bývajú často odlišné. Preto som zmienil ich typy, ktoré podporujú všetci poskytovatelia, čo znamená že či bude mať zákazník záujem o relačné SQL alebo objektové NoSQL, každá cloudová spoločnosť mu vyjde v ústrety s priamou podporou týchto technológií. (23)

2.1.6 Dátová analýza

K databázam sa viaže spracovávanie dát, preto by som rád nadviazal programom na dátovú analýzu s názvom Apache Spark. Využíva sa na spracovávanie veľkého množstva dát, z ktorých sa dajú následne vytvárať rôzne analýzy, čo už záleží od obsahu samotných dát. Dátová analýza je kľúčová pre všetky veľké firmy. Je to nástroj využívaný v bankovníctve, energetike, umelej inteligencií, IoT (Internet Vecí) a rôznych iných odvetviach, a v spoločnostiach, ktoré potrebujú vykonávať rýchle spracovávanie veľkého objemu dát, vytvárať z nich grafy, reporty, predpovede a podobne. Apache Spark je kompatibilný s veľkým množstvom dostupných technológií, medzi ktoré patrí SQL, Python, Java, jazyk R a iné. Vzhľadom na popularitu a schopnosti tohto nástroja je natívne podporovaný všetkými spomínanými cloudovými poskytovateľmi, v AWS to je Amazon EMR, v Microsoft Azure má modul názov Azure Databricks, v Google Cloude sa volá Dataproc a v spoločnosti IBM má názov IBM Analytics for Apache Spark for Cloud. (24, 25, 26, 27)

2.1.7 Kontajnerizácia

Po dátovej analýze by som prešiel k možnosti kontajnerizácie. Kontajnerizácia ako funkcia je vo svete IT ešte mladšia ako samotný Cloud alebo jeho myšlienka. Kontajnerizácia znamená oddelenie aplikácií bežiacich na serveri do samostatných kontajnerov. Množstvu užívateľov to môže znieť podobne ako virtualizácia, avšak koncepty týchto dvoch technológií sú iné, nezameniteľné. Vo virtualizácií je simulovaný úplne celý počítač od tej najnižšej hardwarovej vrstvy. Na virtuálny stroj je potrebné nainštalovať operačný systém, po čom sa začne správať rovnako ako fyzický stroj, s rovnakými špecifikáciami a rovnakým operačným systémom. Je to uzavretá sústava, v ktorej musí byť prítomný už spomínaný operačný systém, ktorého knižnice a súčasti sú distribuované medzi všetky aplikácie, ktoré sa tam nainštalujú. Je možné vytvoriť niekoľko virtuálnych strojov na jednom host stroji. Každý z týchto virtuálov bude mať vlastný operačný systém a bude predstavovať vlastné pracovná prostredie. Oproti tomu kontajnery obsahujú len samotné aplikácie a knižnice nutné k ich chodu čerpajú z operačného systému hosta. Kontajnery sa vyznačujú oveľa menšou veľkosťou a záťažou na hardware hosta pri ich behu. Najčastejšie sa používajú na rozklad jednotlivých služieb do menších celkov a prácu v moderných IT metodikách ako je DevOps. Rozklad na kontajnery je užitočný pri rýchlom presúvaní a rozdeľovaní práce na veľkom projekte do menších celkov. Napríklad webový server fungujúci v samostatnom kontajneri a databáza, na ktorú je pripojený rovnako v samostatnom kontajneri. Pri problémoch s hociktorou z týchto súčastí je možné kontajner jednoducho reštartovať s minimálnym dopadom na druhú službu. Rozšírenosti kontajnerizácie napomáha aj metodika DevOps, čo

je spojenie vývoja s nasadzovaním a testovaním. V tejto metodike sa veľmi často využíva kontajnerizácia spolu s automatickým nasadzovaním a konfiguráciou kontajnerov. Vzhľadom na veľmi rýchlo rastúcu popularitu týchto nástrojov sa stali kontajnerizačné nástroje súčasťou Cloudu. Podporovaná aplikácia Kubernetes zvláda okrem štandardnej kontajnerizácie aj spájanie kontajnerov do clustrov, podporuje ich orchestráciu a automatizáciu. Nachádza sa v štandardnej ponuke všetkých poskytovateľov, u AWS ako Amazon Elastic Kubernetes Service (EKS), v Microsoft Azure ako Azure Kubernetes Service (AKS), Google Cloud ho ponúka s pomenovaním Google Kubernetes Engine (GKE) a v IBM sa nazýva IBM Cloud™ Kubernetes Service. (28, 29, 30, 31, 32)

2.1.8 Governance a správa nákladov

Posledná služba, ktorú by som rád bližšie špecifikoval, patrí medzi tie najdôležitejšie. Ide o Cost Management a Governance. Obe tieto služby sú prirodzene zakomponované vo všetkých mnou spomínaných cloudových platformách, dovolím si povedať že sú zakomponované v úplne všetkých Cloudoch. Cost Management je služba, ktorá je spojená s backendom Cloudu a stará sa o výpočet nákladov za služby, ktoré zákazník využíva. Najbežnejší model účtovania, ktorý táto služba využíva je pay-as-you-go, čo v praxi znamená, že ceny za služby sú účtované za čas ich využívania, prípadne za objem dát, ktorý spracujú. Governance je rovnako jedna z koreňových služieb napojených na backendové systémy Cloudov. V praxi je to služba kontrolujúca všetky procesy a zdroje v Cloude, ktorá sa stará o ich správne využívanie podľa cieľu, ktorý chce zákazník dosiahnuť s prihliadnutím na všetky možné typy riskov. V Cloude je to súbor politík, skladajúcich sa z rôznych pravidiel, ktoré nastavujú práva jednotlivým používateľom na zdroje v Cloude a ich využívanie. Governance je v Cloude dôležitý vzhľadom na rýchle tempo a decentralizáciu rozhodovania o jednotlivých zdrojoch. Takto je možné predísť vysokým nákladom v jednom oddelení alebo zbytočným bezpečnostným rizikám naprieč celou spoločnosťou. (33, 34)

2.1.9 Ostatné

Zvyšné služby, ktoré ponúkajú Cloudy nie sú toľko využívané, respektíve nie všetky spoločnosti pre ne dokážu nájsť využitie. Niektoré z nich môžu byť užitočné pre relatívne široké spektrum zákazníkov, ako napríklad LoadBalancing využívaný na High Availability nasadenie aplikácií, s ktorým je možné docieliť vysoké SLA. Iné sú oveľa užšej špecializácie, ako je napríklad Blockchain, umelá inteligencia, vývojové prostredie pre virtuálnu realitu a podobne. Takto široké pôsobisko spoločností poskytujúcich Cloud je výhodou pre malé

spoločnosti, ktoré sa môžu od začiatku špecializovať na tieto odvetvia informatiky s nižšími počiatočnými nákladmi. (35)

2.2 Životný cyklus aplikácie v Cloude

Ako každá súčasť IT sveta, aj cloudová infraštruktúra má svoj životný cyklus. Životný cyklus cloudovej infraštruktúry sa odvíja od jej primárneho použitia, čo je distribúcia zdrojov ku koncovému užívateľovi. Každý užívateľ Cloudu si ho zvolil pre jednu z výhod, ktoré ponúka, preto je dôležité mať tieto vlastnosti na pamäti aj pri životnom cykle. Niektoré vlastnosti sa však viažu k iným typom Cloudu – IaaS, PaaS, SaaS. Cloud sa snaží svojím zákazníkom prinášať hlavne:

- **Flexibilitu (IaaS, SaaS)** - Možnosť hocikedy si objednať a vytvoriť nové virtuálne zdroje na požiadanie s minimálnym časovým oneskorením. Veľa Cloudových providerov ponúka škálovanie ako horizontálne (počet strojov), tak vertikálne (výkon jednotlivých strojov), čo umožňuje koncovému užívateľovi prispôsobovať svoju cloudovú infraštruktúru aktuálnym podmienkam trhu, prípadne sa snažiť tieto podmienky predvídať. Horizontálne škálovanie spomínam v mojej práci aj ja v kapitole 6, kde porovnávam možnosti horizontálneho škálovania Terraformu a Ansiblu. (36)
- **Dostupnosť (IaaS, PaaS, SaaS)** - Nové zdroje sa vytvárajú pomocou samoobslužného ovládacieho portálu, zväčša voľne prístupného z internetu, na ktorom je možné si zdroje objednať nonstop. Touto cestou sa zároveň sprístupňujú stroje s vysokým výpočetným výkonom malým firmám, prípadne jednotlivým užívateľom, ktorí by inak nemali z finančných dôvodov prístup k obdobne výkonnej On-Premise infraštruktúre. (37)
- **Automatizáciu (IaaS, SaaS)** - Pomocou cloudových orchestračných nástrojov, ako Terraform a Ansible, je možné zautomatizovať deploy infraštruktúry v Cloude, a tým urýchliť vývojové cykly alebo plynule zavádzať DevOps metodológiu na vývoj softwaru a aplikácií. (38)
- **Úsporu nákladov (IaaS)** - Koncovým užívateľom cloudu odpadne povinnosť vysokých vstupných nákladov pri zakúpení si vlastnej serverovej infraštruktúry. To je však len začiatok do prevádzkovania vlastnej infraštruktúry je potrebné zarátať aj náklady spojené so zapojením a počiatočným nastavením serverov a im pridružených zdrojov. Tento náklad sa dá najlepšie vyjadriť ako cena pracovníkov, ktorí by to mali na starosť. Zároveň je potrebné brať do úvahy elektrickú spotrebu

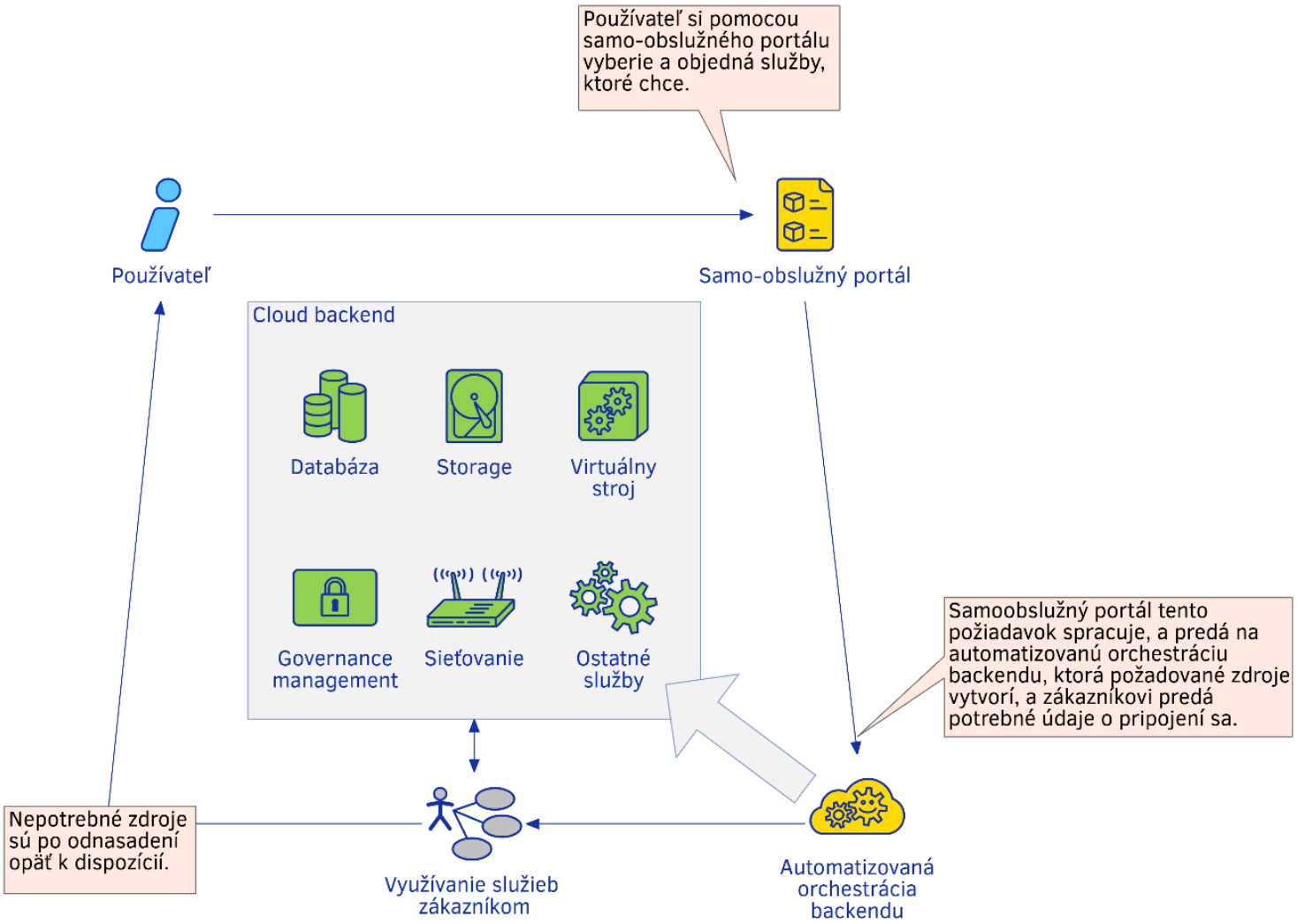
týchto zariadení, ktorá je často nemalá a ďalšiu hodinovú cenu pracovníkov, ktorí sa budú musieť o zariadenia starať. (39)

- **Využitie kapacity (IaaS)** - V neposlednom rade je dôležité myslieť aj na využitie kapacity serverov a ostatných prvkov infraštruktúry, ako sieťové prvky a úložiská. Pri kúpe zariadenia On-Premise je bežným javom, že kapacita stroja nie je stále naplno využitá, z čoho by sa dalo povedať že investícia do zariadení nie je naplno využívaná, čím vzniká strata. V Cloude sa koncový zákazník nemusí obávať o nevyužitú kapacitu, keďže o tú sa stará dodávateľ Cloudu, čím sa užívateľ môže vyhnúť ďalšej finančnej strate. (39)
- **Audit (IaaS, PaaS, SaaS)** - Poskytovatelia Cloudu sa radia medzi veľké spoločnosti s dlhoročnými skúsenosťami, čo znamená že pojem audit pre nich nie je neznámy. Z toho dôvodu sa snažia svojim zákazníkom už v základných subskripciách poskytovať veľké množstvo informácií, ktoré je možné použiť na audit infraštruktúry, aplikácií, ktoré sú prevádzkované, ceny, ktorá sa platí, dostupnosti, ktorú zákazník dostáva a podobne. Pri vyžiadaní každý poskytovateľ môže svojmu zákazníkovi poskytnúť konkrétne logy pre podrobné vyhľadanie informácií o konkrétnom subjekte auditu. (40)

Životný cyklus Cloudovej infraštruktúry štandardne začína pri zákazníkovi, ktorý sa zaregistruje na samoobslužnom portáli poskytovateľa. Po zadaní potrebných údajov na registráciu a platby je možné ihneď vytvárať objednávky na zdroje v Cloude. Zákazník samozrejme nie je limitovaný len systémom na automatické vytváranie objednávok, môže využiť aj nástroje na správu, ako Terraform, Ansible a iné. Výhoda týchto nástrojov je kombinácia objednávania infraštruktúry a hneď po jej vytvorení začiatok nasadzovania žiadaných aplikácií. Vytvorené objednávky automatizovaný systém poskytovateľa spracuje a ihneď potom ich začne vytvárať v zvolených datacentrách. Podľa toho aké služby si zákazník zvolil, jeho infraštruktúra sa môže replikovať v ďalších datacentrách, ktoré budú mať odlišnú geografickú polohu. Vytvorenie týchto zdrojov trvá v priemere od niekoľkých minút, v extrémnych prípadoch až do niekoľkých hodín. Všetko záleží od veľkosti a zložitosti infraštruktúry, ktorú si zákazník praje nasadiť. Po nasadení zdrojov dostane zákazník informácie o pripojení a ich stave prostredníctvom rovnakého portálu, akým si služby objednal a môže ich začať okamžite využívať. Po skončení potreby ich opäť prostredníctvom portálu zákazník môže odstrániť spomedzi zdrojov, ktoré využíva, čím mu odpadne

povinnosť za ne platiť a výpočetná kapacita na fyzických serveroch sa uvoľní pre ďalšie použitie, viď Diagram 1.

Diagram 1 - Životný cyklus Cloudových služieb



3 Ansible

Ansible sa radí medzi nástroje konfiguračného managementu. Nástroje zaoberajúce sa konfiguračným managementom majú najčastejšie na starosti nastavenie serverov a popis tohto nastavenia. Okrem nastavenia serverov sa tieto nástroje využívajú aj na nasadzovanie rôznych aplikácií, prípadne agilnom vývoji a sú dosť skloňované v súvislosti s metodikou DevOps. Na nasadenie sa najčastejšie používajú v prípadoch nasadenia aplikácií vyvíjaných takzvane "in-house", pod čím si môžeme predstaviť nasadenie aplikácie vyvinutej pre interné potreby spoločnosti na niekoľko serverov a jej udržiavanie. Pri agilnom vývoji a DevOps sa konfiguračný management najčastejšie využíva v testovacích alebo nasadzovacích postupoch, kde je najskôr vytvorený virtuálny stroj (alebo kontajner), do ktorého je umiestnená aplikácia na testovanie, alebo používanie, ktorá je zároveň nastavená podľa potreby a je možné ju hneď začať využívať. Samozrejme, celý tento postup je riadený nástrojom na konfiguračný management pre jeho urýchlenie, elimináciu chýb a zaistenie jednotných výsledkov pri opakovaní. Proces konfigurovania serverov pozostáva z niekoľkých krokov. Ako prvé je dôležité vybrať si ten správny nástroj, od čoho sa odvíja ďalší postup. Existuje niekoľko druhov týchto aplikácií a okrem Ansiblu sú využívané nástroje ako Chef alebo Puppet. Hlavný rozdiel medzi týmito nástrojmi je v princípe ich fungovania. (41)

Môžu byť na takzvanej "push" alebo "pull" báze. Voľný preklad týchto názvov už sám napovedá, akým princípom bude konfigurácia fungovať. "Push" báza znamená, že nástroj využíva konfiguračný súbor, v ktorom je definované čo má byť na serveroch vykonané, nástroj sa po spustení pripojí na všetky servery a vykoná všetko, čo bolo popísané v konfiguračnom súbore, s ktorým bol spustený. Týmto spôsobom bola konfigurácia z hlavného stroja "pretlačená" na ostatné, odtiaľ pochádza názov "push". Konfigurácia týmto spôsobom potrebuje len nainštalovaný nástroj managementu na stroji s prístupom na všetky servery, ktoré bude konfigurovať. Na druhej strane, "pull" báza funguje na princípe "ťahania" konfigurácie z hlavného servera. V tom prípade je nutné okrem nainštalovania hlavnej aplikácie s prístupom na manažované servery, inštalovať aj klientské aplikácie (takzvaných agentov), ktoré sa budú pravidelne pripájať na hlavný server a kontrolovať aktuálnosť konfigurácie. Oba prístupy sa odlišujú a majú svoje výhody aj nevýhody. Aplikácie využívajúce "pull" prístup sú oveľa náročnejšie na počiatočnú inštaláciu a pri pridaní každého servera, ktorý bude aplikáciou manažovaný je nutné naň nainštalovať agenta a prepojiť ho so serverom. Manažovanie samotných serverov je však oveľa jednoduchšie, pretože korektné spojenie servera s klientmi sa zaistí už pri inštalácií tohto

nástroja, a následne už sa len píšu súbory zaisťujúce konfiguráciu a umiestňujú sa na server. O distribúciu sa postarajú agenti pravidelne kontrolujúci a aplikujúci zmeny v nastaveniach. Nástroje založené na "push" princípe však bývajú oveľa jednoduchšie na počiatočnú inštaláciu, čo však vyvažuje udržiavanie aktuálnosti konfigurácie manuálne. Vždy pri zmene v nastaveniach je potrebné spustiť aplikáciu, ktorá sa pripojí na klientov, a vykoná zmeny v nastaveniach. (41)

Ansible sa radí medzi nástroje využívajúce "push" bázu fungovania a je takzvane "state-driven", čo znamená že sa orientuje na výsledok. Zjednodušene, v Ansible sa definuje stav, v akom sa majú jednotlivé servery nachádzať a Ansible sa po spustení postará o vykonanie potrebných úkonov na to, aby bol tento stav splnený. Ďalšia z jeho hlavných vlastností je idempotencia, čo znamená že aj pri opakovanom spustení sa dosiahne rovnakého stavu, aký je definovaný v konfiguračných súboroch Ansiblu. Na svoje fungovanie využíva SSH pripojenie na Linuxové servery (WinRM na stroje s Windowsom). Pomocou SSH je zaistená široká kompatibilita s Linuxovými systémami a zároveň je spojenie bezpečné a nenáročné na prevádzku a udržiavanie. Na korektné fungovanie po pripojení je nutné, aby na stroji bol nainštalovaný Python vo verzií 2.5 (prípadne Python 2.4 s knižnicou simplejson). Ansible využíva Python na celý priebeh konfigurácie. Z konfiguračného súboru sa vytvorí Python skript, v ktorom sú definované príkazy, ten je odoslaný na klientský stroj cez SSH, kde je nakoniec pomocou Pythonu spustený a vykoná všetky príkazy. Ako som už spomínal, nie je potrebná inštalácia žiadneho klientského nástroja. Po zaistení SSH pripojenia a vhodnej verzie Pythonu už je možné prejsť k samotnému používaniu Ansiblu. (41)

3.1 Ansible playbook

Všetky súbory Ansiblu sú písané jazykom YAML, ktorý je navrhnutý s ohľadom na jednoduchú čitateľnosť a používanie. Každý YAML dokument popisuje štruktúrované dáta. Vyznačuje sa odsadením dvoma (prípadne štyrmi) medzerami a neobmedzenými úrovňami vnorenia. Základná súčasť je hlavný konfiguračný súbor, takzvaný playbook, s prednastaveným menom playbook.yml. V ňom je popísané, na ktorých serveroch a v akom poradí sa budú vykonávať zmeny. Playbook sa nachádza v zložke, v ktorej je umiestnený samotný konfiguračný súbor (viď. Diagram 2), hodnoty premenných, zápis hostov,

jednotlivé úlohy, ktoré budú vykonané Ansiblom (takzvané tasky) a na záver tu môžu byť aj handlery, čo sú úlohy, ktoré majú špeciálne určenie a poviem o nich viac neskôr. (41)

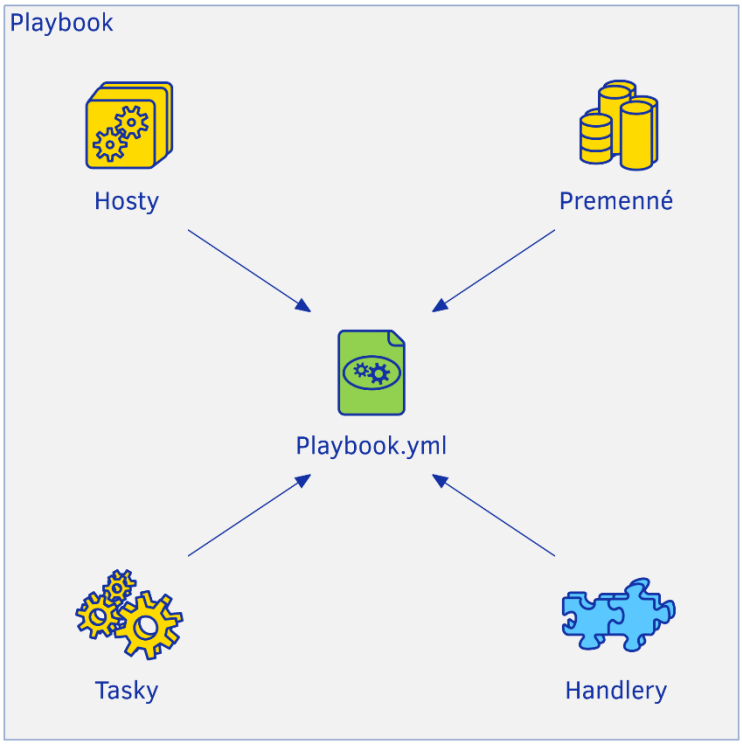


Diagram 2: Obsah zložky Ansible Playbooku

3.1.1 Inventár

Základným kameňom pre fungovanie Ansiblu je definícia hostov, na ktorých sa bude pripájať a vykonávať na nich zmeny. Na toto slúži súbor, ktorý sa nazýva inventory file. Tu sa definujú adresy systémov, na ktoré bude Ansible pristupovať. Je možné tieto systémy spájať do skupín, ktoré sa dajú použiť v playbooku ako cieľová skupina hostov, pričom sa na všetkých strojoch definovaných v skupine vykonajú akcie z playbooku. Samozrejme sa dá jeden host zaradiť do viacerých skupín a skupiny je možné spájať s inými skupinami. Takto je možné oddeliť napríklad databázové stroje od strojov, kde pobežia webové servery, bezpečnostné prvky infraštruktúry, celé časti vývojových prostredí atď. V inventároch sa dá využiť jednoduchá iterácia, pre skrátenie zápisu strojov s podobnými menami. Napríklad na zápis 10 strojov do inventára s DNS menom db01.example.com, až db10.example.com je vhodnejší zápis db[01:10].example.com. V inventároch je tiež možné k hostom pripísať dodatočné premenné. Premenné je možné definovať aj pre celú skupinu, kde bude použitá

táto hodnota pre každého hosta zo skupiny. Príklad na Ansible inventár so všetkými možnosťami konfigurácie je možné vidieť vo Výpise 1.1 nižšie. (41)

Výpis 1.1: Príklad Ansible inventory file so všetkými možnosťami konfigurácie.

```
mail.example.com    - jednoduchý zápis hosta

[mail]
mail01.example.com
mail02.example.com    - zápis svoch hostov do skupiny “mail”

[production]
web01.example.com
db[01:05].example.com    - zápis 5 strojov pomocou jednoduchej iterácie
web02.example.com state=inactive    - zápis premennej patriacej jednému hostovi
mail01.example.com    - host patriaci do dvoch skupín naraz

[production:vars]
root_remote_access=denied    - skupinová premenná

[customer1:children]
mail
production    - skupina združujúca iné skupiny
```

3.1.2 Premenné

Po Inventári sú dôležité premenné používané v playbooku. Premenné sa dajú definovať priamo v inventároch, no to môže viesť k zbytočnému nafúknutiu a komplikovanosti inventárov pri veľkom počte premenných. Na premenné slúži zložka group_vars, prípadne zložka host_vars. Premenné v týchto zložkách sa viažu priamo na skupiny alebo hosty, ako ich názov napovedá. Pomenovanie premenných má svoje obmedzenia. Môže obsahovať len písmená, číslice alebo podtržník a musia začínať písmenom. Premenné nemusia mať len jednoduchý formát typu kľúč-hodnota. Jazyk YAML umožňuje vytvárať slovníky, a tým premenné hierarchizovať alebo iterovať celým slovníkom. (41)

Špeciálne premenné, ktoré Ansible získava automaticky, sa volajú Fakty (Facts). Na začiatku každého playbooku, po úspešnom pripojení sa Ansible pozbiera detaily o stroji, na ktorý sa pripája a tie uloží ako JSON pod týmto týmto názvom. Fakty štandardne obsahujú veľké množstvo dát s detailami o konkrétnom stroji ako napríklad IP adresy, čas, dátum, detaily o diskoch, detaily operačného systému, hodnoty premenných prostredia

(environment variables), detaily o sieťových rozhraniach, o procesore, pamäti, vlastne komplexne o stave daného stroja,a tieto premenné sa dajú použiť v playbooku. Používateľ môže toto zbieranie detailov o stroji zrušiť, napísaním parametra gather_facts: False, na začiatok playbooku. (41)

3.1.3 Tasky a Role

Pokračovať budem opisom samotných úloh, ktoré Ansible využíva na vykonávanie konfiguračného managementu. Volajú sa Tasky a sú základnou súčasťou playbookov. V taskoch sú priamo definované všetky akcie, ktoré bude Ansible vykonávať na serveri. Na vykonanie akcie tasky používajú moduly. Moduly sú skripty, ktoré Ansible používa na vykonanie konkrétnej akcie na cieľovom stroji. Takmer každý z nich používa parametre na nastavenia, s ktorými vykonajú potrebné akcie. Napríklad modul file, ktorý spracúva súbory a zložky dokáže zmeniť vlastníctvo súboru, vytvoriť zložku, vytvoriť linky na súbory a zároveň súbory mazať. Všetko pomocou správneho doplnenia parametrov. Veľké množstvo už je v Ansible pripravené, no je možnosť si vytvoriť aj vlastné, použitím programovacieho jazyka Python, v ktorom je napísaný aj samotný Ansible. (41)

Tasky sa najčastejšie združujú do Rolí. Pomocou nich sa dá playbook rozlížiť na menšie časti, čím sa dá zjednodušiť. Môžeme ich prirovnať k funkciám v programovacích jazykoch, ktoré si programátor napíše a spätne ich vo svojom kóde používa. Jedna rola sa dá zároveň použiť v niekoľkých playbookoch, čím sa dá vytváranie playbookov urýchliť. Rola predstavuje zložku rozčlenenú do viacerých podzložiek, v ktorých by sa mali nachádzať adekvátne súbory pre správne fungovanie role. Samotné role majú predpísanú štruktúru, do ktorej majú byť rozčlenené, viď Výpis 1.2. Na diagrame je zložka roles, ktorá združuje všetky role a rola s názvom “role_1”. Nasleduje rozčlenenie role do podzložiek, kde sa nachádzajú tasky, premenné, templaty, súbory, premenné ktoré môžu byť prepísané, v názvosloví Ansiblu “defaults”, meta informácie o roli a handlery. (41)

Výpis 1.2: Členenie podzložiek Ansible role.

```
roles/
    role_1/
        tasks/
        vars/
        templates/
        files/
        defaults/
        meta/
        handlers/
```

3.1.4 Handlery

Dostávam sa k spomínaným Handlerom. Sú to akcie podobné taskom, ktoré sú spúšťané špeciálnou podmienkou po dobehnutí Tasku, v ktorom je špecifikovaná. V Taskoch sa Handler pridá parametrom notify a konkrétnym menom handleru. Handlery sa veľmi často využívajú napríklad na reštart služieb pri zmene ich konfigurácie. Vo Výpise 1.3 je príklad použitie handleru po skopírovaní konfiguračného súboru do webového servera Apache je zavolaný handler s názvom "restart apache", ktorý sa postará o reštart webového servera, čím sa aplikuje nanovo skopírovaný konfiguračný súbor. (41)

Handlery sa štandardne spúšťajú na konci playbooku, keď všetky tasky úspešne dobehli. Sú spustené len raz, i keď boli zavolanie viac krát. Ak je zavolaných viacero handlerov, vykonajú sa v poradí, v akom boli v playbooku volané. (41)

Výpis 1.3: Využitie handleru na reštart webového servera Apache, v tasku na skopírovanie konfiguračného súboru.

```
- name: copy Apache configuration file and apply its settings to Apache
  copy:
    src: files/website.conf
    dest: /etc/httpd/conf/website.conf
    owner: root
    group: root
    mode: 0600
  notify: restart apache
```

3.2 Využitie Ansiblu

Ansible sa považuje za automatizačnú platformu, ktorá svojou jednoduchosťou a silou zvláda široké spektrum problémov pri systémovej administrácií. Od zabezpečenia prostredia pre aplikácie, cez konfiguračný management, až po orchestráciu, jeho využiteľnosť nájde uplatnenie vo veľkom množstve use-casov, ktoré by som rád priblížil v tejto kapitole.

3.2.1 Nastavenie prostredia

Zabezpečenie a nastavenie prostredia na vývoj alebo prevádzku aplikácií je v IT svete jedna z najdôležitejších služieb. Ansible zvláda managovanie a zabezpečenie prostredia od štandardných serverov (takzvane bare-metal), až po managovanie cloudovej infraštruktúry, virtuálnych strojov v cloude alebo on-premise, samotných hypervizorov, úložiska a iných. Po vytvorení jednotlivých instancií s nimi Ansible dokáže ďalej pracovať ako pripojiť k nim

úložisko, nasadiť ich v sieti za LoadBalancer, nainštalovať aplikácie, aktualizovať operačný systém a iné úlohy, ktoré môžu byť vykonávané rôznymi tímami. Ansible je schopný spájať procesy a postupne prechádzať z deploymentu infraštruktúry do konfiguračného managementu, orchestrácie až po deploy a management jednotlivých aplikácií. (42)

Správa fyzických serverov je umožnená integrovanými modulmi vytvorenými komunitou alebo priamo spoločnosťou Red Hat na správu nástrojov ako Red Hat Satellite, Stacki a podporou správy širokej palety operačných systémov ako Linux, Windows, UNIX a OS X. (42)

Integrovanie nástrojov na správu virtualizácie ako VMware, Red Hat Virtualization, Vagrant a iné, umožňuje administrátorom automatizovať vytváranie, mazanie a ďalšiu správu virtuálnych strojov a prostredí. Zároveň užívatelia môžu konfigurovať siete a fyzické sieťové zariadenia od spoločností ako Cisco, Juniper a iné. Ansible zvláda automatizované vytváranie a konfiguráciu sietí, ich testovanie a následnú správu. Zároveň je možné spravovať úložiská, a to ako na báze softwaru, tak hardwaru. Tieto všetky vlastnosti robia z Ansible mocný nástroj na spravovanie prostredia datacentier on-premise, kde Ansible dokáže zjednodušiť a urýchliť takmer celú správu. (42)

Tieto všetky vlastnosti sú tak isto aplikovateľné aj na Cloudovú infraštruktúru. Ansible obsahuje veľké množstvo modulov pre integráciu a správu rôznych Cloudov, a to od verejných, ako je napríklad AWS a Azure, až po správu súkromných Cloudov, ako je OpenStack private cloud. Všetky schopnosti a možnosti, ktoré má Ansible s on-premise infraštruktúrou sú dostupné aj pre Cloud, čo znamená vytváranie a správa virtualizácie, sietí a úložiska. Vzhľadom na širokú ponuku Cloudových poskytovateľov Ansible nie je obmedzený len na tieto úlohy. Je schopný pracovať s oveľa širšou paletou funkcionalít, ktorá záleží individuálne od poskytovateľa a modulov podporovaných Ansiblom. (42)

3.2.2 Konfiguračný management

Bežne zaužívaný konfiguračný management pozostáva zo skriptov, ktoré sa môžu starať o správu celého systému. Toto riešenie je však už zastaralé a s rastom strojov, ktoré treba spravovať to môže byť problematické. Automatizované nástroje preberajú kontrolu nad touto oblasťou pôsobenia. Je však veľa nástrojov, ktoré sú príliš komplexné a so správou moc nepomáhajú. Ansible sa prezentuje ako jednoduché riešenie bez agentov na klientských systémoch, využívajúc len SSH dostupné na väčšine systémov (Pre Windows je to Windows Remote Management, alebo WinRM). Toto zjednodušuje používanie a deploy samotného Ansiblu. Zároveň súbory používané na konfiguráciu sú jednoduché, dalo by sa povedať, že popisujú infraštruktúru a stav, v akom sa má nachádzať. Nezáleží na stave, v akom sa systém

nachádza, Ansible ho pretransformuje do stavu, ktorý je definovaný v playbookoch. Toto zvyšuje spoľahlivosť v porovnaní so skriptami a umožňuje opakovane vytvárať rovnaké stroje a prostredia. Základná distribúcia Ansiblu obsahuje viac ako 1300 modulov, ktoré je možné rozšíriť o niekoľko násobne väčšie množstvo modulov vytvorených komunitou Ansiblu. Moduly sú prepravené na cieľový systém, kde sú spúšťané, aby bol dosiahnutý cieľový stav. Sú spúšťané pod predom definovaným používateľským účtom, ktorý nepotrebuje mať root práva. Podpora pre eskaláciu práv na úroveň sudo je, samozrejme, pre kontrolu súčastí systému, ktoré to vyžadujú. Zároveň je podporovaná Kerberos autentizácia. (43)

3.2.3 Deployment aplikácií

Ansible poskytuje jednoduchý a konzistentný spôsob nasadzovania aplikácií pomocou jeho jednotnej štruktúry. Namiesto zložitých skriptov je využitý jednoduchý jazyk YAML na definovanie playbookov, ktorý je pochopiteľný už pri prvom čítaní. Pomocou playbookov sa definuje stav, v akom sa má systém nachádzať a Ansible vykoná všetky potrebné operácie, aby systém do toho stavu priviedol. Playbooky umožňujú, aby boli inštalácie, upgrady a každodenný management opakovateľný a spoľahlivý. So správnym skomponovaním playbookov je možné dosiahnuť nulového výpadku pri nasadení alebo upgrade aplikácie pomocou rozsiahleho množstva ovládacích modulov, ktoré sú pre Ansible dostupné. Pomocou týchto modulov je možné pri deploymente aplikácie vytvoriť presný workflow, podľa ktorého by inak administrátori museli postupovať. Zároveň Ansible dokáže komunikovať s REST API rozhraniami alebo odoslať informatívnu správu o stave deploymentu formou e-mailu, prípadne inej podporovanej platformy na odosielanie správ. (44)

3.2.4 Continuous delivery

Continuous delivery metodika sa na automatizačné a orchestračné nástroje spolieha. Sú potrebné pre rýchle automatické vytvorenia prostredí a okamžitého nasadenia aplikácie hneď potom. Ansible pomocou jeho “push based” formy dovoľuje vysoký stupeň kontroly nad rôznymi operáciami a konfiguráciou. Môže pracovať s load balancermi, nastavovať servery, vytvárať virtuálne servery, cloud, webové služby a mnohé iné. S Ansiblom je možné rozdeliť stroje do skupín, ktoré budú managované postupne a každá môže byť využívaná na niečo iné. Jednotlivé úkony vykonávané v skupinách a poradie v ktorých sa majú vykonávať je prísne kontrolované Ansiblom a pomocou spájania playbookov je možné definovať celú Delivery pipelinu od vytvorenia stroja, až po nasadenie aplikácie. Ansible je zároveň schopný ovládať sieťovanie, load balancing, monitorovacie systémy, čím je možné práve

upgradovaný stroj vylúčiť z load balancing poolu a monitorovania, kým nebude nasadenie alebo upgrade hotový. (45)

3.2.5 Automatizácia zabezpečenia

Pomocou Ansiblu je možné zároveň posilniť zabezpečenie prostredí a serverov. Reagovanie na bezpečnostné hrozby a útoky môže byť veľmi pomalé a nie vždy presne také, ako by bolo potrebné. S pomocou automatizačných nástrojov je možné automatizovať a integrovať riešenia, ktoré môžu reagovať na hrozby skrz celé prostredie koordinovaným spôsobom za použitia správnych playbookov. Pomocou playbookov tvorených adekvátnymi modulmi je možné urýchliť akcie pri podozrivých akciách ako je zaradenie IP adresy na blacklist alebo izolovanie podozrivých dát na bližšie skúmanie. (46)

3.2.6 Orchestrácia

Už dávno sa neinštaluje jedna služba na jeden server, v modernom IT svete sú stroje a technológie využívané na maximum ich potenciálu. Aplikácie s komplexnými závislosťami a aplikácie nasadzované do clusterov nie sú ničím výnimočným. Všetky tieto služby je potrebné správne nastaviť pre ich korektné fungovanie a spoluprácu. V niektorých prípadoch je taktiež potrebné, aby boli určité služby spustené a nastavené skôr ako iné. Napríklad databáza by mala byť spustena a nakonfigurovaná skôr ako sa k nej pripojí server webového backendu. Pri nasadzovaní a správe týchto aplikácií sa často využívajú orchestračné nástroje. Ansible sa dá využiť okrem inštalácie na správne nastavenie celého softwarového prostredia na orchestráciu. Je možné v ňom špecifikovať deployment komponentov, ktoré sa hneď po deploymente správne nakonfigurujú a budú pracovať tak ako je od nich očakávané. Táto sada komponentov sa dá zároveň deploynúť v každom prostredí, kde majú komponenty a Ansible podporu a po deployi bude vyzerať všade rovnako vďaka idempotencií Ansiblu. Zároveň je možné orchestrovať komponenty naprieč viacerými prostrediami a viacerými strojmi naraz, vďaka už spomínanej širokej podpore Ansible modulov na správu. (47)

4 O Terraforme

Terraform sa radí medzi nástroje manažujúce infraštruktúru - anglicky provisioning. Spôsob, akým sa o to snaží sa nazýva Infrastructure as Code, čo znamená, že všetky časti infraštruktúry a ich prepojenia sú vyjadrené kódom. Je to open source nástroj napísaný v jazyku Go. Terraform je po skompilovaní len jeden binárny súbor a nie je potrebné ho inštalovať ani pridávať žiadnu extra infraštruktúru, stačí ho skopírovať na stroj, na ktorom ho chceme využívať. Jednoduchosť Terraformu spočíva v tom, že po autentizácií prakticky len posiela požiadavky na API endpointy Cloudových poskytovateľov, čím vytvára infraštruktúru a zisťuje jej stav. Nepotrebuje prestupný server na pripájanie sa ku Cloudu a zároveň na už existujúcich instanciách infraštruktúry nepotrebuje mať nainštalovaného klienta. Kompletná obsluha Cloudov prebieha výhradne pomocou už spomínaných API požiadavok na Cloud. To oveľa zjednodušuje jeho používanie, pretože nie je potreba zložitá inštalácia ani prítomnosť agentov na spravovaných serveroch. Výhoda Terraformu je jeho široká podpora poskytovateľov Cloudu a možnosť deploymentu kompletnej infraštruktúry pomocou vhodne napísaného modulu a použitím jedného príkazu. Pre svoje fungovanie Terraform potrebuje mať vytvorené moduly, v ktorých je popísaný stav, v akom sa má Cloud nachádzať a súbor s premennými, z ktorých čerpá informácie pri nasadzovaní. (48)

4.1 Moduly

Terraform využíva moduly ako konfiguračné súbory a sú písané v konfiguračnom jazyku vytvorenom spoločnosťou stojacou za Terraformom - HashiCorp Configuration Language, ktorý pripomína JSON. Bol navrhnutý s dôrazom na jednoduchosť a čitateľnosť, aby bolo možné Terraform používať čo najskôr. Je to deklaratívny jazyk, pomocou ktorého sa definuje stav, v akom sa má infraštruktúra nachádzať. Skladá sa z blokov, ktoré majú v sebe definované vlastnosti vo forme “kľúč = hodnota”, viď Výpis 2.1. Na začiatku je dôležité definovať poskytovateľa, ku ktorému sa má Terraform pripojiť a to blokom s názvom “provider”. V ňom je popísaný región, do ktorého sa všetky súčasti infraštruktúry popísané ďalej nasadia. Nasleduje popis samotnej infraštruktúry. Na to sa využívajú bloky pomenované ako “resource”. Tie sú univerzálne použité na definíciu všetkých nasadzovaných súčastí. Rozlišuje sa medzi nimi názvom v úvodzovkách, ktorý nasleduje hneď za názvom bloku. Hlavičku tohto bloku uzatvára užívateľský názov vytváraného zdroja, opäť v úvodzovkách. V tomto konkrétnom prípade ide o nasadenie virtuálneho stroja

v AWS. Nami použité v príklade, je ID imagu operačného systému, ktorý bude použitý v stroji, a instance_type je typ stroja, ktorý bude vytvorený. (49)

Výpis 2.1: Definícia virtuálneho stroja v AWS

```
provider “aws” {
  region = “us-east-1”
}

resource “aws_instance” “priklad” {
  ami = “ami-12345”
  instance_type = “t2.micro”
}
```

V moduloch je možné definovať celú infraštruktúru spolu s väzbami jednotlivých súčastí ako servery, databázy, siete a iné.

Po vytvorení modulu ponúka Terraform možnosť skontrolovať aké zmeny budú vykonané v infraštruktúre bez toho, aby ich reálne vykonal a užívateľ mohol skontrolovať svoje chyby v konfigurácií. Na tento, takzvaný “beh na nečisto”, slúži príkaz terraform plan, ktorý ukáže na výstupe ako budú ovplyvnené zdroje v Cloude. Zdroje, pred ktorými bude znak plus (+) budú nanovo vytvorené, zdroje pred ktorými bude znak mínus (-) budú odstránené a zdroje, pred ktorými bude znak vlnovky (~) budú nejakým spôsobom pozmenené. (49)

Keď si je užívateľ istý tým, čo chce vykonať, stačí len spustiť Terraform príkazom terraform apply a zo všetkých definícií z modulu sa vytvoria API príkazy, ktoré budú odoslané na server, následne vykonané a užívateľ bude informovaný o ich výsledkoch. Terraform zároveň sleduje zmeny, ktoré vykonal pomocou súboru s príponou “.tfstate”. Pri zmene v infraštruktúre stačí vykonať zmenu v Terraform module, opäť spustiť Terraform s týmto modulom a všetky úpravy sa aplikujú na Cloud. Modifikácie môžu byť od triviálnych ako pridanie disku do VM, až po deploy alebo odstránenie celého prostredia. (49)

4.2 Premenné

Premenné slúžia ako parametre pre moduly, vďaka ktorým je možné upravovať vlastnosti modulov bez zmeny ich kódu. Každá premenná, ktorá je zahrnutá v module v ňom musí byť deklarovaná blokom “variable”. Po deklarovaní bloku nasleduje špecifikácia názvu, ktorá musí byť ohraničená úvodzovkami. Mená premenných musia byť unikátne. V deklarácií premennej môže byť špecifikovaný jej dátový typ argumentom “type”. Podporované typy sú: string, number a bool. Deklarácia premennej môže zároveň obsahovať prednastavenú

hodnotu premennej argumentom “default”. Prístup k premennej z modulu je prostredníctvom výrazu var.<NÁZOV PREMENNEJ>. Hodnota premennej je prístupná pomocou výrazu len v module, kde bola premenná deklarovaná. Premenné sa dajú definovať aj do samostatného súboru s príponou .tfvars, .tfvars.json, .auto.tfvars a .auto.tfvars.json. Súbory s presným menom “terraform.tfvars” a “terraform.tfvars.json” sú Terraformom automaticky nahrávané a premenné z nich sa môžu použiť v moduloch. Rovnako sú automaticky nahrávané a spracovávané všetky súbory s príponami .auto.tfvars a .auto.tfvars.json. V praktickej časti mojej práce som využil automatický spôsob nahrávania premenných pomocou súboru terraform.tfvars. (50)

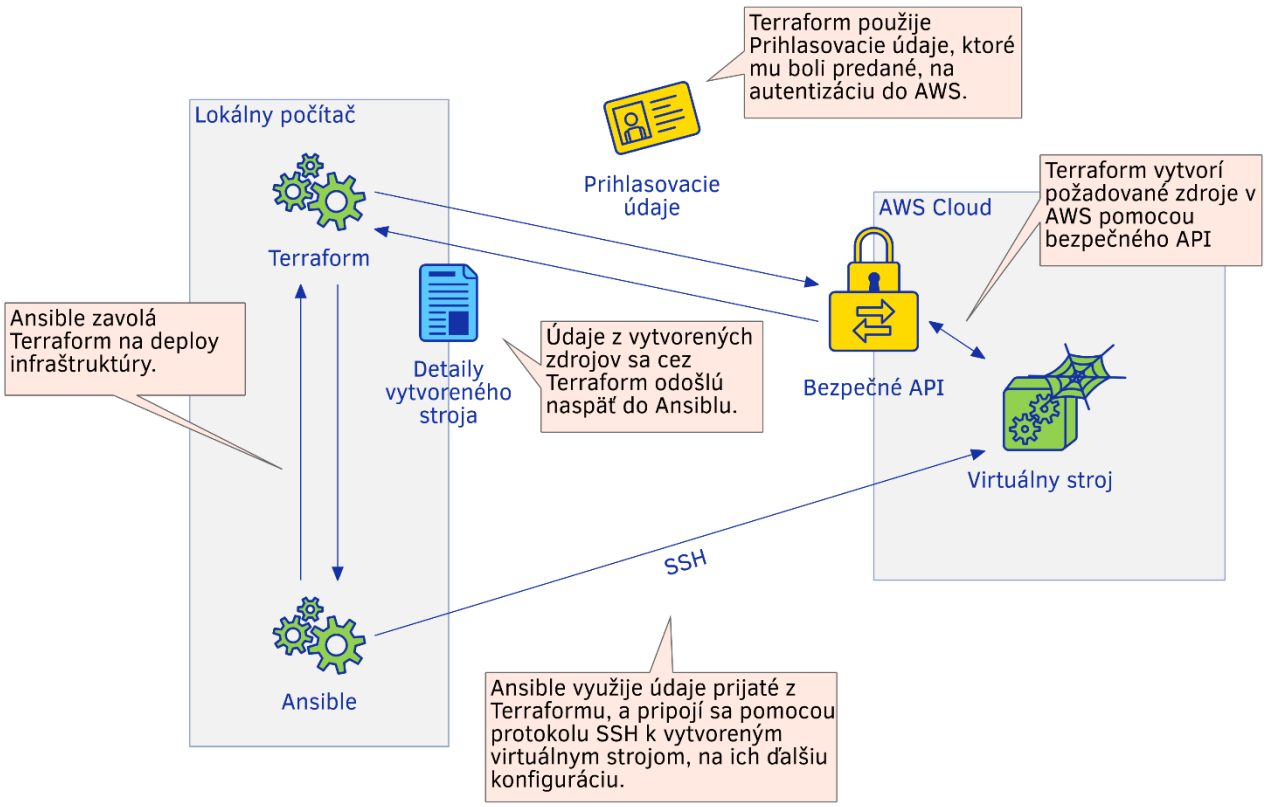
Ako som už spomínal, Terraform patrí medzi nástroje poskytujúce provisioning. Tieto nástroje sa často miešajú s nástrojmi na konfiguračný management. Hlavný dôvod pre toto miešanie je prekrývanie funkcionality jednotlivých aplikácií, čo znamená, že programy starajúce sa primárne o provisioning sú schopné vykonávať aj konfiguračný management a naopak. Napríklad Terraform dokáže na vytvorených serveroch spúšťať príkazy a skripty, ktoré naň predtým nahral, no na komplexnejší konfiguračný management sú vhodnejšie nástroje priamo na to určené, ako napríklad Ansible. Na druhú stranu, Ansible je schopný vytvoriť virtuálne zdroje vo veľkom množstve Cloudov, no je oveľa zložitejšie tieto zdroje ďalej sledovať a vykonávať zmeny v už vytvorenej infraštruktúre. Pre toto je vhodnejší Terraform, ako ukážem v druhej polovici praktickej časti mojej práce.

5 Spojenie Ansiblu a Terraformu

V nasledujúcej kapitole chcem ukázať výhody spojenia Ansiblu s Terraformom pri nasadení a konfigurácií zdrojov v Amazon Web Services Cloude (AWS). Zameriam sa na miniatúrnu DevOps pipelinu, ktorá bude začínať vytvorením a spustením infraštruktúry pomocou Terraformu, od čoho sa bude odvíjať Ansible, ktorý na vytvorenú infraštruktúru nasadí webový server Apache s niekoľkými testovacími stránkami. Tento postup sa dá s miernymi úpravami použiť na väčšinu prípadov administrácie Cloudu a jeho zdrojov, ako pri nasadzovaní, tak pri dodatočnej administrácií. V Diagrame 3 je nákres spolupráce týchto dvoch aplikácií a Cloudu.

Celá ukážka sa spúšťa pomocou jedného Ansible playbooku. Ten na začiatku využije Terraform,a v ňom špecifikovaný modul na deployment infraštruktúry v AWS. Terraform použije prihlasovacie údaje do AWS na vytvorenie bezpečnostnej skupiny, v ktorej budú povolené porty 22 pre službu SSH, 80 pre HTTP protokol a 443 pre HTTPS protokol. Po úspešnom vytvorení uloží výstup do premenných. Ansible využije tieto premenné na pripojenie sa k stroju už cez internet a SSH, kde nainštaluje webový server Apache a zobrazí jednoduchú testovaciu stránku, ktorá potvrdí, že celé nasadenie prebehlo v poriadku.

Diagram 3 - Využitie Ansiblu a Terraformu pri administrácií Cloudu



Praktickú časť práce som vykonával v stroji s operačným systémom CentOS 8.1.1911 (Kernel 4.18.0-147.5.1.el8_1.x86_64), ktorý som prebral z oficiálnej webovej stránky projektu CentOS (https://www.centos.org/).

Príprava AWS spočívala v niekoľkých krokoch, počínajúc vytvorením užívateľského účtu, ktorý som používal skrz celú prácu. Po vytvorení účtu a prihlásení sa, som vytvoril bezpečnostné údaje na ovládanie AWS strojovo, respektíve pomocou Terraformu a AWS API. Tieto údaje som vytvoril v sekcií **My Security Credentials**, kde som na stránke Users vytvoril nového používateľa. Ako prístupový typ som zvolil **„Programmatic access“**, čo mi ďalej umožnilo vytvoriť údaje prístupový kľúč, ktorým Terraform používa na pripojenie sa do Cloudu. V ďalšom kroku som vytvoril používateľskú skupinu s administrátorským prístupom do mojej AWS subskripcie a užívateľa som do tejto skupiny priradil. Nasledovala definícia značiek (takzvaných „tagov“), ktorá nie je povinná, no ja som ich niekoľko vytvoril ako meno, projekt, škola, lokalita a subjekt. Po zadefinovaí tagov bola stránka sumarizujúca mnou vyplnené údaje na kontrolu ich správnosti a tlačidlo na vytvorenie užívateľa. Po vytvorení užívateľa sa zobrazili prístupové kľúče. Tajný prístupový kľúč (Secret access key) sa však zobrazuje len tento jedenkrát, preto je dôležité si ho poznamenať. V prípade straty je nutné revokovať kľúče a vystaviť nové. To bolo kompletné nastavenie, ktoré som vykonal vo webovom ovládacom paneli AWS. Všetky ďalšie úpravy som vykonával programami Ansible a Terraform, o ktorých píšem v nasledujúcich podkapitolách.

5.1 Spoločný deployment aplikácií

V kapitole venujúcej sa spoločnému deploymentu ukážem spoluprácu Terraformu a Ansiblu na deploymente virtuálneho stroja v AWS, na ktorý neskôr nainštalujem a nakonfigurujem Apache web server s virtuálnym serverom a testovacou html index stránkou.

Základné údaje som do Ansiblu predal pomocou YAML súborov, kde boli deklarované premenné. Premenné som rozdelil do dvoch súborov podľa ich dôležitosti. Snažil som sa využiť dostupných možností technológií a priblížiť sa k štandardu používanému v komerčných spoločnostiach. V súbore nachádzajúcom sa v ceste group_vars/all/vars.yml som definoval premenné, ktoré majú menšiu senzitivitu a priradil im hodnoty. Tieto premenné obsahovali dáta ako názov Cloudového providera, ktorý bol použitý, identifikačný kód imagu použitého na vytvorenie virtuálneho stroja v Cloude, región, v ktorom sa bude stroj nachádzať, názov bezpečnostného profilu, číslo portu, ktorý bude

používať webový server a podobne. To znamená, že prípadné odcudzenie dát alebo ich prečítanie neoprávnenou osobou by nemalo zásadný dopad na moju prácu. V druhom súbore, nachádzajúcom sa v ceste group_vars/all/vault.yml som definoval premenné ktoré sú dôležité pre pripájanie sa k AWS API na vytvorenie stroja použitého pre webový server a to ID prístupového kľúča vytvoreného pomocou webovej konzole AWS tajnú časť spomínaného prístupového kľúča a verejnú časť SSH kľúča, ktorá bude skopírovaná do adekvátneho súboru (/home/centos/.ssh/authorized_keys) nanovo vytvorenom stroji v Cloude, pre umožnenie SSH prístupu bez zadávania hesla. S prihliadnutím na citlivosť týchto údajov som využil funkcionalitu Ansible-vault a zašifroval som celý obsah spomínaného súboru s pomocou hesla.

Vo Výpise 3.1 rozoberiem adresárovú štruktúru prvej polovice praktickej časti. Súbory, ktoré sú zvýraznené podčiarknutím sú dynamické súbory, generované za behu playbooku.

Výpis 3.1: Adresárová štruktúra Ansible playbooku, použitého na spoločný deployment aplikácií.

```
ansible-terraform/
├─ ansible.cfg
├─ playbook.yml
├─ group_vars/
│   ├─ dynamic.yml
│   └─ all/
│       ├─ vars.yml
│       └─ vault.yml
├─ inventory
│  └─ hosts
├─ roles/
│  └─ webserver/
│      ├─ tasks/
│      │  └─ main.yml
│      └─ templates/
│         ├─ bcPage.conf.j2
│         └─ index.html.j2
├─ templates/
│  └─ terraform.tfvars.j2
└─ terraform/
   ├─ main.tf
   ├─ terraform.tfstate
   ├─ terraform.tfstate.backup
   └─ terraform.tfvars
```

Riešenie som sa snažil zabaliť do jedného Ansible playbooku, aby bolo spúšťanie čo najjednoduchšie, respektíve jedným príkazom (viď Výpis 3.2).

Výpis 3.2: Súbor playbook.yml, použitý na spoločný deployment pomocu Ansiblu a Terraformu.

```
---

# Playbook sa spusta prikazom: ansible-playbook playbook.yml --private-key=~/.ssh/BcAWS.pem --ask-vault-pass --extra-vars="aws_instance_state=absent"

- hosts: localhost
  connection: local
  gather_facts: no
  tasks:
    - name: Vytvor subor s premennymi terraform.tfvars v zlozke Terraformu.
      template:
        src: "{{ playbook_dir }}/templates/terraform.tfvars.j2"
        dest: "{{ playbook_dir }}/terraform/terraform.tfvars"
    - name: Spusti Terraform skript na deploy AWS stroja.
      terraform:
        project_path: "{{ playbook_dir }}/terraform/"
        state: "{{ aws_instance_state }}"
        force_init: true
        variables:
          aws_region: "{{ aws_region }}"
          aws_access_key_id: "{{ aws_access_key_id }}"
          aws_secret_access_key: "{{ aws_secret_access_key }}"
          aws_ami_id: "{{ aws_ami_id }}"
          public_key: "{{ public_key }}"
      register: terraform_output
    - name: Vypis vystup Terraform skriptu.
      debug:
        var: terraform_output
    - name: Prirad IP adresu vytvoreneho stroja do premennej.
      set_fact:
        vm_ip_web: "{{ terraform_output.outputs.public_ip_web.value }}"
      when:
        - terraform_output.outputs.state_web is defined
    - name: Dynamicky vytvor inventar a subor s premennymi pre Ansible.
      block:
        - name: Uisti sa ze neexistuje dynamicky subor s premennymi pre Ansible.
          file:
            path: group_vars/dynamic.yml
            state: absent
        - name: Vytvor novy dynamicky subor s premennymi.
          file:
            path: group_vars/dynamic.yml
            state: touch
        - name: Uisti sa ze existuje zlozka, kde bude inventar.
          file:
            path: inventory/
            state: directory
        - name: Uisti sa ze neexistuje subor s hostami.
          file:
            path: inventory/hosts
            state: absent
        - name: Vytvor novy subor s hostami.
          file:
            path: inventory/hosts
            state: touch
        - name: Pridaj IP adresu a premennu na eskalovanie prav do suboru.
          blockinfile:
            path: group_vars/dynamic.yml
```

```
            marker: ""
            block: |
              ---
              ansible_host: AWS_WEB
              ansible_user: centos
              become: true
        - name: Pridaj IP adresu servera do suboru hostov.
          blockinfile:
            path: inventory/hosts
            marker: ""
            block: |
              [AWS_WEB]
              {{ vm_ip_web }}
        - name: Pridaj hosta do skupiny s nazvom AWS_WEB.
          add_host:
            hostname: "{{ vm_ip_web }}"
            groupname: AWS_WEB
            remote_user: "centos"
      when:
        - terraform_output.outputs.state_web is defined
    - name: Pockaj 30 sekund na otvorenie portu 22, a skus v odpovedi najst
string "OpenSSH"
      wait_for:
        port: 22
        host: '{{ vm_ip_web }}'
        search_regex: OpenSSH
        delay: 30
      when: vm_ip_web is defined

### ANSIBLE CAST NA AWS STROJI
- hosts: AWS_WEB
  remote_user: centos
  tasks:
    - name: Nainstaluj webovy server a skopiruj sablonu webovej stranky.
      include_role:
        name: webserver
```

Playbook pozostáva z dvoch hlavných častí. Prvá časť, kde je ako host špecifikovaný localhost (stroj, na ktorom je playbook spustený) vytvorí súbor s premennými pre Terraform skript zo súborov group_vars/all/vars.yml, a group_vars/all/vault.yml. Tento postup som zvolil pre možnosť znovu použitia spomínaných premenných ako v Ansible, tak v Terraforme.

Nasleduje spustenie samotného skriptu, pri ktorom sa môže zdať že Ansible zamrzol, no nie je tomu tak. Pri spúšťaní skriptu je vynútená opätovná inicializácia Terraformu. Vynútenie inicializácie spôsobí obnovenie konfigurácie Terraformu v prípade nových nastavení. Po dokončení skriptu sa zaregistruje premenná terraform_output s obsahom jeho výstupu v JSON formáte. Výstup je hneď potom vypísaný Ansible taskom na to určeným.

Ďalší Ansible task zaregistruje verejnú IP adresu z výstupu Terraformu, ktorý je v spomínanej premennej terraform_output. S prihliadnutím na JSON štruktúru výstupu skriptu je možné ho prechádzať relatívne jednoducho slovníkovou formou.

Nasleduje blok taskov, ktoré pripravujú súbory Ansiblu na konfiguráciu konkrétneho webového servera v AWS, ktorý sa spustí len v prípade, ak je definovaná premenná z výstupu Terraform skriptu, vypovedajúca o stave stroja, ktorý mal byť vytvorený v AWS. V opačnom prípade Ansible celý blok úloh preskočí.

Každej úlohe vytvárania súborov predchádza preventívna úloha, ktorá sa postará o to, že v danej zložke už neexistuje rovnaký súbor obsahujúci staré hodnoty premenných, a tým zamedzí možným chybám pri pripájaní sa k vytvorenému stroju alebo konfigurácií samotného stroja. Ako prvý súbor, ktorý sa vymaže a nanovo vytvorí je súbor nachádzajúci sa v ceste group_vars/dynamic.yml. V nasledujúcej úlohe vytváram zložku inventory/, v ktorej sa bude nachádzať inventár. V prípade že táto zložka už existuje, Ansible túto úlohu preskočí a bude pokračovať vo vykonávaní playbooku.

Playbook pokračuje úlohou, ktorá preventívne odstráni inventár, v ktorom sa nachádza definícia IP adresy AWS stroja, na ktorom sa bude inštalovať webový server. Hneď potom sa vytvorí prázdny súbor s hostami, inventory/hosts.

Po vytvorení prázdnych súborov do nich začnem nahrávať čerstvé dáta. Ako prvý sa naplní dátami súboru group_vars/dynamic.yml, v ktorom je obsiahnutý názov skupiny, do ktorej bude zaradený vytvorený stroj, používateľské meno na pripojenie k virtuálnemu stroju a nastavenie o eskalovaní užívateľských práv Ansiblu po pripojení sa k stroju. Hneď potom bude naplnený súbor inventory/hosts, kde bude špecifikovaná skupina, do ktorej bude zaradený. Zároveň rovnakého hosta so skupinou a užívateľom na pripojenie definujem v inventári, ktorý má Ansible uložený v pamäti po spustení playbooku pomocou modulu add_host.

Nasledujúca úloha otestuje pripojenie k vytvorenému stroju pomocou SSH. Úloha s pokusom o pripojenie vyčká 30 sekúnd pre elimináciu prípadného oneskorenia deploymentu stroja, ktorý sa v cloude bežne vyskytuje. Po ubehnutí oneskorenia sa Ansible pokúsi o pripojenie a v návratovej hodnote nájsť reťazec “OpenSSH”, čím sa dá skontrolovať, že stroj je úspešne zapnutý a na porte 22 funguje SSH služba.

V prípade úspešného otestovania SSH služby z predchádzajúceho kroku bude playbook pokračovať do svojej druhej časti, kde je zahrnutá rola na inštaláciu a nastavenie webového servera. Rolu som do playbooku zaradil pomocou modulu include_role. Tento modul role načíta do playbooku až keď príde na radu ich spustenie. Týmto spôsobom sa dá obmedziť

využitie zdrojov Ansiblu, ako je pamäť a swap, v prípade, že stroj nebude v Cloude úspešne deploynutý.

V mojej modelovej situácií je to vzhľadom na nízku spotrebu zdrojov irelevantné, no v prípade, že Ansible by bol spúšťaný na stroji, ktorý by slúžil len na management strojov a ich následné nastavenie, mal by obmedzené zdroje a orchestrácia by bola omnoho náročnejšia ako z pohľadu deploya virtuálnych strojov, tak z pohľadu orchestrácie týchto strojov, úspora pamäte a swapu by určite nebola zanedbateľná položka. Zároveň je možné že sa CPU Load vyšplhá do vysokých úrovní a stroj prestane reagovať.

Pokračoval by som popisom Terraform skriptu, ktorý je v Ansible playbooku použitý na deployment stroja v AWS, na ktorom bude spustený webový server (viď Výpis 3.3).

Výpis 3.3: Súbor main.tf, použitý Terraformom na deploy webového server v AWS.

```
variable "aws_region" {}
variable "profile" {}
variable "server_port" {}
variable "key_name" {}
variable "public_key" {}
variable "private_key" {}
variable "aws_access_key_id" {}
variable "aws_secret_access_key" {}
variable "aws_availability_zone" {}
variable "aws_ami_id" {}


provider "aws" {
  profile    = var.profile
  region     = var.aws_region
  access_key = var.aws_access_key_id
  secret_key = var.aws_secret_access_key
}

resource "aws_key_pair" "deployer" {
  key_name   = var.key_name
  public_key = var.public_key
}


resource "aws_security_group" "BcTest" {
  name = "BcTest-SG"
  ingress {
    from_port   = var.server_port
    to_port     = var.server_port
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port   = 22
```

```
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}


resource "aws_instance" "BcTestWeb" {
  ami = var.aws_ami_id
  instance_type = "t2.micro"
  key_name = var.key_name
  vpc_security_group_ids = [aws_security_group.BcTest.id]
  tags = {
    Name = "terraform-BcTestWeb"
  }
  root_block_device {
    delete_on_termination = true
  }


}

output "public_ip_web" {
  value       = aws_instance.BcTestWeb.public_ip
  description = "Verejna IP adresa mojho modeloveho web servera."
}

output "name_web" {
  value       = aws_instance.BcTestWeb.tags.Name
  description = "Nazov mojho modeloveho web servera."
}

output "state_web" {
  value       = aws_instance.BcTestWeb.instance_state
  description = "Status mojho modeloveho web servera."
}
```

Na spúšanie skriptu používam natívny Ansible modul - terraform. Touto cestou je jednoduchšie spúšťať Terraform pomocou definície zložky, kde sa nachádza celý projekt.

V Terraform projekte sa dá použiť súbor s premennými. Túto funkcionalitu som využil v spojení s Ansiblom a súbor s premennými terraform/terraform.tfvars, v Terraform projekte generujem. V samotnom skripte použitom na deployment infraštruktúry terraform/main.tf,

sú premenné deklarované hneď na začiatku. Ich hodnoty sa automaticky importujú zo súboru terraform.tfvars.

Po premenných je v skripte deklarovaný poskytovateľ Cloudu, v mojom prípade AWS, blokom provider. V ňom je okrem poskytovateľa cloudu definované meno bezpečnostného profilu z AWS, ktorý bude použitý na pripojenie ku cloudu, región, v ktorom sa budú zdroje nasadzovať a adekvátny pár kľúčov na prístup k AWS bezpečnostnému profilu.

Nasleduje definícia verejného kľúča, ktorý bude nahraný do novovzniknutého stroja a s adekvátnym privátnym kľúčom bude možné pripojenie. Po kľúči už začína deklarácia jednotlivých súčastí infraštruktúry. Ako prvú definujem bezpečnostnú skupinu (anglicky, security group), do ktorej bude vzniknutý stroj patriť. V tejto skupine sú popísané pravidlá pre prichádzajúcu a odchádzajúcu sieťovú komunikáciu. S prihliadnutím na fakt, že vytváram webový server, na ktorý budem pristupovať pomocou SSH, som povolil prichádzajúcu komunikáciu na portoch 80 a 22 a odchádzajúcu komunikáciu na portoch 80 a 443. Všetka komunikácia je povolená na tcp protokole a nie je limitovaná IP adresou druhej strany.

Následne definujem samotný stroj pomocou bloku aws_instance. V bloku sa nachádzajú definície AMI (ID imagu operačného systému, ktorý bude použitý vo vytvorenom stroji), typ instancie (v mojom prípade je to t2.micro, keďže využívam takzvaný AWS free tier, ktorý poskytuje základné cloudové služby zdarma), meno verejného kľúča definovaného predtým, ID bezpečnostnej skupiny, ktorú som si definoval predtým, samotné meno instancie, upravím root_block_device, čo je root partícia disku obsiahnutá v imagi. V tejto partícií vynútim jej vymazanie pri zastavení stroja, pretože stroje, ktoré v mojom modelovom projekte vytváram neslúžia na uchovávanie dát na dlhšiu dobu.

Na záver definujem v troch blokoch typu output verejnú IP adresu, meno a stav deployovaného webového servera. Tieto údaje ďalej spracovávam v Ansibli, ako som popisoval skôr v tejto kapitole.

6 Porovnanie využitia Ansiblu a Terraformu

Pri porovnaní deploymentu Cloudových zdrojov by som sa chcel venovať vlastnostiam Ansiblu a Terraformu, ktoré majú spoločné a to je možnosť nasadenia zdrojov v Cloude. Porovnávať som chcel samotné nástroje bez pridania externých skriptov alebo iných aplikácií do procesu nasadenia zdrojov. V porovnaní mali oba nástroje za úlohu vytvoriť rovnakú sadu zdrojov v AWS Cloude, a to verejný kľúč, ktorým je možné sa pripojiť do vytvorených instancií, podsieť, do ktorej budú všetky vytvorené instancie patriť a nakoniec 5 strojov veľkosti t2.micro, viď Diagram 4.

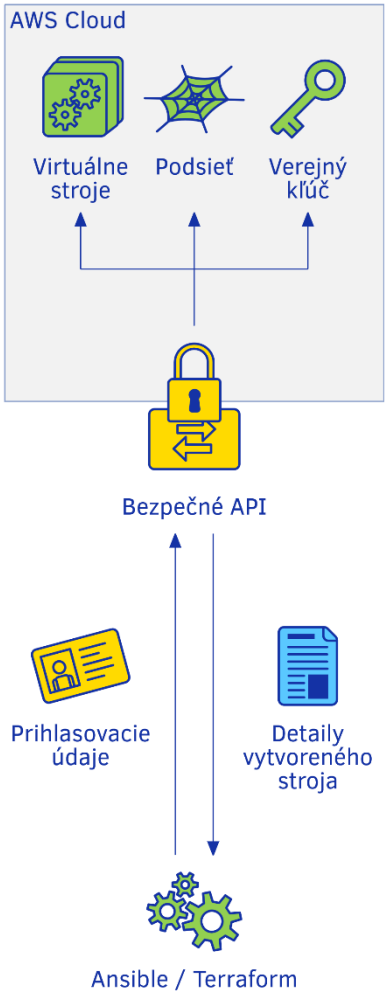


Diagram 4 - Postup deploymentu, pri ktorom boli testované oba nástroje

Definovanie tejto infraštruktúry do konfiguračných súborov (V Terraforme hlavný modul, v Ansible playbook) je na porovnateľnej úrovni čo sa týka zložitosti, no v syntaxi sú rozdiely.

Terraform je stavový, čo znamená, že si uchováva informácie o instanciách, ktoré nasadil. Ansible je naopak bezstavový, takže si informácie neuchováva a k tomu je idempotentný. To znamená, že pri každom spustení Ansiblu s rovnakými parametrami bude jeho výstup rovnaký. Pri škálovaní infraštruktúry je dôležité myslieť na tieto dve vlastnosti. Pri znovupoužití rovnakých konfiguračných súborov na nasadenie nového počtu instancií, Terraform si podľa uloženého posledného stavu vypočíta koľko zdrojov musí nasadiť, aby bol ich počet zhodný s konfiguráciou, s ktorou bol spustený a vytvorí len chýbajúce zdroje pri zachovaní zdrojov už vytvorených. Na druhej strane Ansible, vzhľadom na to, že si stav vytvorených zdrojov nikde neukladá, opätovne nasadí úplne všetky definované zdroje v konfigurácií. To znamená, že sa k pôvodným zdrojom pridajú nové zdroje v počte, ktorý je definovaný v konfigurácií, čím vznikne nežiaduci prebytok. Takémuto správaniu v prípade Ansiblu sa však dá zamedziť a to zakomponovaním modulu ec2_instance_facts do playbooku, ktorým je možné zistiť detaily o instanciách virtuálnych strojov, ktoré bežia v AWS Cloude.

S Terraformom je zároveň jednoduchšie tieto zdroje odnasadiť. Pri odnasadení celej infraštruktúry projektu, ktorý je definovaný v konfiguračných súboroch, stačí zmeniť príkaz a po potvrdení Terraform automaticky odnasadí všetky zdroje špecifikované v súboroch. Toto je umožnené vďaka jeho stavovosti a priamemu prispôsobeniu na prácu s Cloudom. Na druhej strane, Ansible potrebuje mať špecifikované stroje v playbooku v dostupných premenných alebo musí byť využitý dynamický inventár vytváraný pomocou sktipov na to, aby ich vedel odnasadiť vďaka jeho bezstavovosti. Toto komplikuje proces odnasadenia, pretože je potrebné uchovávať dáta o vytvorených instanciách alebo zistiť identifikačné údaje strojov, ktoré chceme odstrániť. To sa dá buď cez AWS webové rozhranie alebo opäť pomocou modulu ec2_instance_facts.

6.1 Terraform

Ako prvý by som popísal hlavný modul Terraformu (viď Výpis 4.1). Ten začína definovaním všetkých premenných, ktoré sú síce prázdne, no ich hodnota sa automaticky načítava z priloženého súboru s premennými. Nasleduje blok, v ktorom je definovaný provider, ktorým je AWS. Týmto blokom sa Terraform prihlasuje do AWS, na čo musí mať zadefinované premenné ako profil, región, ID prístupového kľúča a tajná časť prístupového kľúča. Ďalej sa nachádza blok, ktorý definuje verejný kľúč, ktorý sa nahrá do vytvorených virtuálnych strojov, aby bolo možné sa ku nim pripojiť. Tento blok obsahuje premenné ako názov

vytvoreného verejného kľúča a samotná hodnota tohto kľúča. Po verejnom kľúči sa vytvorí podsieť. V bloku podsiete je na začiatku definované ID VPC (Virtual Private Cloud). VPC je časť Amazon Cloudu, v ktorom má užívateľ pod kontrolou sieťovanie a môže v ňom nastaviť napríklad podsiete, vlastné pravidlá pre komunikáciu medzi strojmi a s okolitým svetom a podobne. Blok pokračuje definíciou CIDR bloku samotnej podsiete. Na záver som pridal tag s menom pre jednoduchšiu identifikáciu siete. V poslednom bloku sa vytvárajú samotné instancie virtuálnych strojov. Blok začínam definíciou počtu týchto strojov. Tá je dôležitá pre ukázanie rozdielu medzi jednotlivými konfiguračnými aplikáciami, ktorým sa chcem v tejto časti práce zaoberať. Po definícií počtu nasleduje AMI ID (ID Imagu použitého na vytvorenie stroja), typ stroja, názov verejného kľúča vytvoreného v predchádzajúcich blokoch, ID podsiete, ktorá bola taktiež vytvorená v predošlých blokoch a nakoniec špecifikácia odstránenia root partície pri zastavení stroja.

Výpis 4.1: Súbor main.tf, použitý Terraformom na deploy 5 virtuálnych strojov v AWS.

```
variable "aws_region" {}
variable "profile" {}
variable "key_name" {}
variable "public_key" {}
variable "private_key" {}
variable "aws_access_key_id" {}
variable "aws_secret_access_key" {}
variable "aws_availability_zone" {}
variable "aws_ami_id" {}
variable "aws_vpc_id" {}


provider "aws" {
  profile    = var.profile
  region     = var.aws_region
  access_key = var.aws_access_key_id
  secret_key = var.aws_secret_access_key
}

resource "aws_key_pair" "deployer_part2" {
  key_name   = var.key_name
  public_key = var.public_key
}

resource "aws_subnet" "subnet_part2" {
  vpc_id     = var.aws_vpc_id
  cidr_block = "172.31.50.0/24"

  tags = {
    Name = "Main"
  }
}

resource "aws_instance" "BcTest_part2" {
  count = 5
  ami = var.aws_ami_id
```

```
  instance_type = "t2.micro"
  key_name = var.key_name
  subnet_id = aws_subnet.subnet_part2.id
  root_block_device {
    delete_on_termination = true
  }
}
```

6.2 Ansible

Po zadefinovaní Terraform modulu by som prešiel na Ansible playbook (viď Výpis 4.2). Ten však začína definícou strojov, na ktoré sa má samotná aplikácia pripojiť, aby vykonala konfiguráciu. Vzhľadom na to, že som vytváral nové stroje, ale na žiadny som sa nepripájal, do hodnoty hosts som zadal hodnotu "localhost", čo je adresa miestneho počítača. Môj nasledujúci krok bol vypnúť zhromažďovanie faktov, ktoré je v tomto prípade zbytočné a len by spomaľovalo chod playbooku. Neskôr už sa nachádzajú len jednotlivé tasky, ktoré využívam na vybudovanie infraštruktúry v Cloude. Sú takmer identické s blokmi v hlavnom module Terraformu. Rozdiel je v tom, že každý jeden task, pripájajúci sa na Cloud má ako prvé špecifikované prihlasovacie údaje do AWS, pretože sa potrebuje autorizovať predtým než vykoná zmeny. Druhou zmenou oproti Terraformu je prítomnosť úloh, ktoré sa starajú o vypísanie návratových hodnôt z vytvárania zdrojov. Každá úloha, ktorá vytvára zdroje v Cloude musí byť nasledovaná ďalšou, ktorá vypisuje výsledok vytvárania týchto zdrojov predtým registrovaných do premennej, inak by som nemal prehľad o stave zdrojov, ktoré som chcel vytvoriť. Terraform vytvára tento výpis automaticky. Čo sa týka samotných zdrojov, vytváram ich v rovnakom poradí ako v Terraform module. Začínam vytvorením verejného kľúča. Do tejto úlohy vkladám parametre ako meno kľúča a jeho hodnotu. Nasleduje vytvorenie podsiete kam opäť vkladám hodnoty VPC ID a CIDR blok použitý pre podsieť, no na záver je hodnota, ktorá špecifikuje žiaduci stav, v akom sa má podsieť nachádzať. Poslednou úlohou, ktorá bude vytvárať zdroje, je úloha na vytvorenie virtuálnych strojov. V nej špecifikujem meno verejného kľúča a podsiete, ktoré som vytvoril, typ stroja, AMI ID (ID imagu použitého na vytvorenie stroja), región, stav, v akom sa majú stroje nachádzať, premenná wait, ktorá vynúti Ansible na počkanie, než sa virtuálne stroje dostanú do označeného stavu, než bude pokračovať a počet instancií, ktoré chcem vytvoriť.

Výpis 4.2: Súbor playbook.yml, použitý Ansiblom na deploy 5 virtuálnych strojov v AWS.

```
---
# Playbook sa spusta prikazom: ansible-playbook --ask-vault-pass playbook.yml

- hosts: localhost
  gather_facts: False
```

```
tasks:
  - name: Vytvor verejny kluc v AWS.
    ec2_key:
      aws_access_key: "{{ aws_access_key_id_part2 }}"
      aws_secret_key: "{{ aws_secret_key_part2 }}"
      name: "{{ aws_key_name_part2 }}"
      key_material: "{{ public_key }}"
    register: aws_key_result
  - name: Vypis hodnotu premennej aws_key_result z vytvarania verejneho kluca.
    debug:
      msg: "{{ aws_key_result }}"
  - name: Vytvor podsiet do ktorej budu patrit vytvorene instancie.
    ec2_vpc_subnet:
      aws_access_key: "{{ aws_access_key_id_part2 }}"
      aws_secret_key: "{{ aws_secret_key_part2 }}"
      state: present
      vpc_id: "{{ aws_vpc_id_part2 }}"
      cidr: 172.31.50.0/24
    register: subnet_aws
  - name: Vypis hodnotu premennej subnet_aws z deploya podsiete.
    debug:
      msg: "{{ subnet_aws }}"
  - name: Vytvor 5 strojov v AWS.
    ec2:
      aws_access_key: "{{ aws_access_key_id_part2 }}"
      aws_secret_key: "{{ aws_secret_key_part2 }}"
      key_name: "{{ aws_key_name_part2 }}"
      vpc_subnet_id: "{{ subnet_aws.subnet.id }}"
      instance_type: t2.micro
      image: "{{ aws_ami_id_part2 }}"
      region: "{{ aws_region_part2 }}"
      state: present
      wait: true
      count: 5
    register: ec2
  - name: Vypis hodnotu premennej ec2 z deploya strojov.
    debug:
      msg: "{{ ec2 }}"
```

6.3 Vyhodnotenie porovnania

Ako Terraform, tak Ansible dokážu nasadiť Cloudovú infraštruktúru v AWS Cloude. Terraform je priamo prispôsobený na nasadzovanie týchto zdrojov, takže stačí jednoduchšia konfigurácia, s ktorou ponúka väčšiu pridanú hodnotu ako je stráženie stavu nasadených zdrojov a rýchlejšia úprava už nasadených zdrojov. Neponúka však konfiguračný management, takže je zložité pomocou neho inštalovať iné programy alebo inak upravovať konfiguráciu strojov. Na druhú stranu, Ansible dokáže ako nasadzovať zdroje v Cloude, tak ich aj ďalej konfigurovať, inštalovať aplikácie alebo inak upravovať. Je však menej intuitívny

v spravovaní Cloudových zdrojov ako Terraform. Pri konfiguračných súboroch rovnakej veľkosti a zamerania, ponúka menšiu funkcionalitu oproti Terraformu a to konkrétne sledovanie stavu nasadených zdrojov alebo ich následná úprava. Pre túto funkcionalitu by bolo nutné rozšíriť playbook o ďalšie moduly, ktoré by skomplikovali playbook. S ohľadom na to, že je to univerzálnejší nástroj by som však doporučoval na spravovanie Cloudu skôr Ansible. Administrácia samotného Cloudu je síce o niečo zložitejšia, no následné možnosti konfiguračného managementu, ktoré ponúka Ansible a rozobral som ich v teoretickej časti, ho predurčujú na komplexnejšiu správu a deployment prostredí, kdežto Terraform sa obmedzuje len na správo samotných služieb Cloudu.

Záver

Cieľom tejto práce bolo priblížiť platformu AWS a Cloudu obecne s predstavením najviac využívaných služieb Cloudu. Spolu s Cloudom som chcel priblížiť používanie technológií Ansible a Terraform a ich základné prvky, ktoré som popísal v teoretickej časti. Zároveň som chcel viac rozvinúť užívateľské spektrum nástroja Ansible s typickými príkladmi jeho použitia. Tento cieľ sa mi podarilo splniť. V mojej práci približujem spôsobom vhodným pre laického a stredne pokročilého čitateľa všetky tieto témy v teoretickej časti.

V praktickej časti mojej práce som sa snažil ukázať možnosti spolupráce Ansiblu a Terraformu pri nasadení Cloudových zdrojov v AWS a okamžitej správe týchto zdrojov. V druhej polovici praktickej časti som chcel porovnať Ansible a Terraform ako samostatné nástroje na deployment Cloudových služieb v AWS. Cieľ ukázania spolupráce Ansiblu a Terraformu sa mi podarilo splniť pomocou vytvorenia jednoduchej pipeliny zaobalenej v Ansible playbooku. Tento playbook najskôr vytvoril pomocou Terraformu potrebné zdroje v AWS Cloude, z ktorých získal prístupové údaje na pripojenie sa na konkrétny stroj, kde následne nainštaloval webový server a nakonfiguroval testovaciu webovú stránku.

Cieľ druhej časti sa mi taktiež podarilo splniť, i keď s menšími obmedzeniami. Pri použití Ansiblu som nevyužil všetky možnosti, ktoré Ansible ponúka pri správe Cloudu a dynamického inventára. Bolo to z dôvodu udržať konfiguračné súbory v približne rovnakej rovine a Ansible by pri využití modulov potrebných pre dorovnanie schopností Terraformu prevýšil ako veľkosť, tak zložitosť konfiguračného súboru. V závere druhej časti som zhrnul porovnanie týchto nástrojov, z ktorého vyplýva, že Terraform je lepší ako samostatný správca Cloudových služieb, ale Ansible ho prekonáva v prípade komplexnej správy Cloudu a administrácie prostredia v ňom nasadeného.

Použitá literatúra

(1) ŠALATA, M., 2018. *Cloud and on-premise deployment platform for YSoft SafeQ*. Brno: Masarykova univerzita, Fakulta informatiky. [cit. 2020-03-13]. Dostupné z: https://is.muni.cz/th/bhjeb/

(2) OBETKO, Š., 2019. *Automated Infrastructure Deployment on Public and Private Cloud as a Part of Continuous Integration Process*. Brno: Diplomová práce. Masarykova univerzita, Fakulta informatiky. [cit. 2020-03-13]. Dostupné z: https://is.muni.cz/th/frrux/

(3) MEDINA, O., SCHUMANN, E., 2018. *DevOps for SharePoint*. California: Apress Media, LLC. ISBN 978-1-4842-3688-8

(4) KNORR, Eric. What is cloud computing? Everything you need to know now. In: *InfoWorld* [online]. 2018-10-02 [vid. 2020-04-30]. Dostupné z: https://www.infoworld.com/article/2683784/what-is-cloud-computing.html

(5) What is cloud computing? *Microsoft* [online]. [vid. 2020-04-30]. Dostupné z: https://azure.microsoft.com/en-in/overview/what-is-cloud-computing/

(6) CAREY, Scott. The history of AWS: A timeline of defining moments from 2002 to now. In: *Computerworld* [online]. 2019-03-12 [vid. 2020-04-30]. Dostupné z: https://www.computerworld.com/article/3412382/the-history-of-aws--a-timeline-of-defining-moments-from-2002-to-now.html

(7) Amazon EC2. *Amazon Web Services* [online]. [vid. 2020-05-01]. Dostupné z: https://aws.amazon.com/ec2/

(8) Virtual Machines. *Microsoft* [online]. [vid. 2020-05-01]. Dostupné z: https://azure.microsoft.com/en-us/services/virtual-machines/

(9) Compute Engine. *Google* [online]. [vid. 2020-05-01]. Dostupné z: https://cloud.google.com/compute

(10) IBM Cloud Virtual Servers. *IBM* [online]. [vid. 2020-05-01]. Dostupné z: https://www.ibm.com/cloud/virtual-servers-cloud/

(11) Amazon S3. *Amazon Web Services* [online]. [vid. 2020-05-01]. Dostupné z: https://aws.amazon.com/s3/

(12) Storage Account Overview. *Microsoft* [online]. [vid. 2020-05-01]. Dostupné z: https://docs.microsoft.com/en-us/azure/storage/common/storage-account-overview

(13) Cloud Storage. *Google* [online]. [vid. 2020-05-01]. Dostupné z: https://cloud.google.com/storage

(14) IBM Cloud Object Storage. *IBM* [online]. [vid. 2020-05-01]. Dostupné z: https://www.ibm.com/cloud/object-storage

(15) AWS Backup. *Amazon Web Services* [online]. [vid. 2020-05-01]. Dostupné z: https://aws.amazon.com/backup/

(16) Azure Backup. *Microsoft* [online]. [vid. 2020-05-01]. Dostupné z: https://azure.microsoft.com/en-us/services/backup/

(17) Cloud Storage for data archiving. *Google* [online]. [vid. 2020-05-01]. Dostupné z: https://cloud.google.com/storage/archival

(18) IBM Cloud Backup. *IBM* [online]. [vid. 2020-05-01]. Dostupné z: https://www.ibm.com/cloud/backup

(19) Amazon CloudFront. *Amazon Web Services* [online]. [vid. 2020-05-01]. Dostupné z: https://aws.amazon.com/cloudfront/

(20) Content Delivery Network. *Microsoft* [online]. [vid. 2020-05-01]. Dostupné z: https://azure.microsoft.com/en-us/services/cdn/

(21) Cloud CDN. *Google* [online]. [vid. 2020-05-01]. Dostupné z: https://cloud.google.com/cdn

(22) IBM Cloud Content Delivery Network. *IBM* [online]. [vid. 2020-05-01]. Dostupné z: https://www.ibm.com/cloud/cdn

(23) SMALLCOMBE. Mark. SQL vs. NoSQL: How Are They Different and What Are the Best SQL and NoSQL Database Systems? In: *Xplenty* [online]. 2019-08-23 [vid. 2020-05-01]. Dostupné z: https://www.xplenty.com/blog/the-sql-vs-nosql-difference/

(24) Amazon EMR. *Amazon Web Services* [online]. [vid. 2020-05-01]. Dostupné z: https://aws.amazon.com/emr/

(25) Azure Databricks. *Microsoft* [online]. [vid. 2020-05-01]. Dostupné z: https://azure.microsoft.com/en-us/services/databricks/

(26) Dataproc. *Google* [online]. [vid. 2020-05-01]. Dostupné z: https://cloud.google.com/dataproc

(27) IBM Analytics for Apache Spark. *IBM* [online]. [vid. 2020-05-01]. Dostupné z: https://www.ibm.com/analytics/ca/en/technology/cloud-data-services/spark-as-a-service/

(28) Red Hat, Inc. 2020, *Containers vs VMs*. [cit. 18.4.2020]. Dostupné z: https://www.redhat.com/en/topics/containers/containers-vs-vms

(29) Amazon Elastic Kubernetes Service. *Amazon Web Services* [online]. [vid. 2020-05-01]. Dostupné z: https://aws.amazon.com/eks/

(30) Azure Kubernetes Service. *Microsoft* [online]. [vid. 2020-05-01]. Dostupné z: https://azure.microsoft.com/en-us/services/kubernetes-service/

(31) Google Kubernetes Engine. *Google* [online]. [vid. 2020-05-01]. Dostupné z: https://cloud.google.com/kubernetes-engine

(32) IBM Cloud Kubernetes Service. *IBM* [online]. [vid. 2020-05-01]. Dostupné z: https://www.ibm.com/cloud/container-service/

(33) Cloud Financial Management with AWS. *Amazon Web Services* [online]. [vid. 2020-05-01]. Dostupné z: https://aws.amazon.com/aws-cost-management/

(34) Management and Governance on AWS. *Amazon Web Services* [online]. [vid. 2020-05-01]. Dostupné z: https://aws.amazon.com/products/management-tools/

(35) Cloud Products. *Amazon Web Services* [online]. [vid. 2020-05-01]. Dostupné z: https://aws.amazon.com/products/

(36) AWS Auto Scaling. *Amazon Web Services* [online]. [vid. 2020-04-08]. Dostupné z: https://aws.amazon.com/autoscaling/

(37) AWS Smart Business. *Amazon Web Services* [online]. [vid. 2020-04-08]. Dostupné z: https://aws.amazon.com/campaigns/smart-business/

(38) CAMPBELL, Josh. Terraform: Beyond the Basics with AWS. In: *Amazon Web Services* [online]. 2016-02-04 [vid. 2020-04-08]. Dostupné z: https://aws.amazon.com/blogs/apn/terraform-beyond-the-basics-with-aws/

(39) PHAPHOOM, Nattakarn a OZA, Nilay a WANG, Xiaofeng. Does cloud computing deliver the promised benefits for IT industry? *Proceedings of the WICSA/ECSA 2012 Companion Volume* [online]. August 2012, 45-52 [vid. 2020-04-08]. ISBN 978-1-4503-1568-5. Dostupné prostredníctvom ACM Digital Library z: https://dl-acm-org.zdroje.vse.cz/doi/abs/10.1145/2361999.2362007

(40) Azure security logging and auditing. *Microsoft* [online]. [vid. 2020-04-08]. Dostupné z: https://docs.microsoft.com/en-us/azure/security/fundamentals/log-audit

(41) HOCHSTEIN, Lorin a MOSER, René. *Ansible: Up and Running 2nd Edition*. Sebastopol: O'Reilly Media, 2017. ISBN 978-1-491-97980-8.

(42) USE CASE: Provisionig. *Red Hat* [online]. [vid. 2020-05-01]. Dostupné z: https://www.ansible.com/use-cases/provisioning

(43) USE CASE CONFIGURATION MANAGEMENT. *Red Hat* [online]. [vid. 2020-05-01]. Dostupné z: https://www.ansible.com/use-cases/configuration-management

(44) USE CASE: Application Deployment. *Red Hat* [online]. [vid. 2020-05-01]. Dostupné z: https://www.ansible.com/use-cases/application-deployment

(45) USE CASE: Continuous Delivery. *Red Hat* [online]. [vid. 2020-05-01]. Dostupné z: https://www.ansible.com/use-cases/continuous-delivery

(46) Security Automation. *Red Hat* [online]. [vid. 2020-05-01]. Dostupné z: https://www.ansible.com/use-cases/security-automation

(47) USE CASE: Orchestration. *Red Hat* [online]. [vid. 2020-05-01]. Dostupné z: https://www.ansible.com/use-cases/orchestration

(48) BRIKMAN, Yevgeniy. *Terraform: Up and Running*. Sebastopol: O'Reilly Media, 2017. ISBN 978-1-491-97708-8

(49) Modules. *HashiCorp* [online]. [vid. 2020-05-02]. Dostupné z: https://www.terraform.io/docs/configuration/modules.html

(50) Input Variables. *HashiCorp* [online]. [vid. 2020-05-02]. Dostupné z: https://www.terraform.io/docs/configuration/variables.html

Prílohy

K práci je priložený .zip súbor so zdrojovými kódmi použitými v praktickej časti práce.

Příloha A: Praktická časť

Priložený archív „prakticka_cast.zip“ obsahuje pracovnú zložku, ktorú som využíval v praktickej časti. Tá sa ďalej delí na dve zložky, „ansible-terraform“, v ktorej sú súbory demonštrujúce spoluprácu Ansiblu a Terraformu z prvej časti bakalárskej práce, a „compare“, ktorá obsahuje súbory využité na porovnanie deploymentu pomocou Ansiblu a Terraformu v Cloude. Súbor **vault.yml**, v ktorom sú definované premenné Ansiblu na pripojenie sa k AWS je zabezpečený heslom, ktoré je “bakalarska_praca”. Tento súbor je umiestnený v praktickej časti dvakrát, prvýkrát v zložke **prakticka_cast.zip\ansible-terraform\group_vars\all**, kde je použitý na spoluprácu Ansiblu a Terraformu. Druhýkrát je v zložke **prakticka_cast.zip\compare\ansible\group_vars\all**, kde je využitý pre samostatné fungovanie Ansiblu. Vo Výpise 5.1 nasleduje popis adresárovej štruktúry archívu “prakticka_cast.zip”.

Výpis 5.1: Adresárová štruktúra archívu prakticka_cast.zip, ktorý obsahuje všetky súbory použité v praktickej časti

```
prakticka_cast.zip
├─ ansible-terraform/
│  ├─ ansible.cfg
│  ├─ playbook.yml
│  ├─ group_vars/
│  │   └─ all/
│  │      ├─ vars.yml
│  │      └─ vault.yml
│  ├─ inventory/
│  ├─ roles/
│  │  └─ webserver/
│  │     ├─ tasks/
│  │     │  └─ main.yml
│  │     └─ templates/
│  │        ├─ bcPage.conf.j2
│  │        └─ index.html.j2
│  ├─ templates/
│  │  └─ terraform.tfvars.j2
│  └─ terraform/
│     └─ main.tf
└─ compare/
   ├─ ansible/
   │  ├─ playbook.yml
   │  ├─ group_vars/
   │  │   └─ all/
   │  │      ├─ vars.yml
   │  │      └─ vault.yml
   │  └─ inventory/
   │     └─ hosts
   └─ terraform/
      ├─ terraform.tfvars
      └─ main.tf
```