Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Riešenie úloh strojového učenia využitím Kubernetes

Bakalárska práca

2022 **Štefan Hadbavný**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Riešenie úloh strojového učenia využitím Kubernetes

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	9.2.1. Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	Marek Ružička
Konzultant:	Marcel Vološin

Košice 2022 **Štefan Hadbavný**

Abstrakt v SJ

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisissem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi necante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultriciesvel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero utmetus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit ametante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Kľúčové slová v SJ

LaTeX, programovanie, sadzba textu

Abstrakt v AJ

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisissem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi necante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultriciesvel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero utmetus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit ametante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Kľúčové slová v AJ

LaTeX, programming, typesetting

Bibliografická citácia

HADBAVNÝ, Štefan. *Riešenie úloh strojového učenia využitím Kubernetes*. Košice: Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky, 2022. 24s. Vedúci práce: Marek Ružička

Tu vložte zadávací list pomocou príkazu
`\thesisspec{cesta/k/suboru/so/zadavacim.listom}`
v preambule dokumentu.
Kópiu zadávacieho listu skenujte čiernobielo (v odtieňoch sivej) na 200 až 300 DPI! Nezabudnite do jednej práce vložiť originál zadávacieho listu!

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval(a) samostatne s použitím uvedenej odbornej literatúry.

Košice, 13.5.2022 .....

Vlastnoručný podpis

Poďakovanie

Na tomto mieste by som rád poďakoval svojmu vedúcemu práce za jeho čas a odborné vedenie počas riešenia mojej záverečnej práce.

Rovnako by som sa rád poďakoval svojim rodičom a priateľom za ich podporu a povzbudzovanie počas celého môjho štúdia.

V neposlednom rade by som sa rád poďakoval pánom *Donaldovi E. Knuthovi* a *Leslie Lamportovi* za typografický systém LaTeX, s ktorým som strávil množstvo nezabudnuteľných večerov.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Úvod práce stručne opisuje stanovený problém, kontext problému a motiváciu pre riešenie problému. Z úvodu by malo byť jasné, že stanovený problém doposiaľ nie je vyriešený a má zmysel ho riešiť. V úvode neuvádzajte štruktúru práce, t.j. o čom je ktorá kapitola. Rozsah úvodu je minimálne 2 celé strany (vrátane formulácie úlohy).

Ďalšie užitočné informácie môžete nájsť v Pokynoch pre vypracovanie záverečných prác[1].

Formulácia úlohy

Text záverečnej práce musí obsahovať sekciu s formuláciou úlohy resp. úloh riešených v rámci záverečnej práce. V tejto časti autor rozvedie spôsob, akým budú riešené úlohy a tézy formulované v zadaní práce. Taktiež uvedie prehľad podmienok riešenia.

[1] https://moodle.fei.tuke.sk/pluginfile.php/27971/mod_resource/content/16/Instructions_v15.pdf

1 Analytická časť

Pri strojovom učení a experimentoch je niekedy potreba na krátku dobu vyšší výkon. Pravidelné migrovanie medzi strojmi by bolo veľmi zdĺhavé. S tým nám pomôže platforma kubernetes.

Kubernetes je platforma, ktorá je veľmi robustná v nasadení, správe a orchestrácií kontajnerov. Dokáže rozdeliť súčasne záťaž medzi jednotlivými strojmi. Tato platforma v súčasnosti vytlačila predošle platformy a stala sa už štandardom. Nasadenie Kubernetes je najefektívnejšie, aj keď existuje dnes veľa podobných technológii a mnoho významných poskytovateľov ponúka klastre Kubernetes.

Podľa mojich uvážení je na našu problematiku vhodný Kubeflow.

1.1 Kubernetes

Spôsob, akým je Kubernetes navrhnutý, je to, čo ho robí výnimočným. Kubernetes má základnú architektúru klienta a servera. Kubernetes má schopnosť vykonávať priebežné aktualizácie, prispôsobuje sa aj ďalším pracovným zaťaženiam automatickým škálovaním uzlov, ak je to potrebné, a môže sa tiež opraviť v prípade rozpadu podu. Poskytujú obrovskú výhodu v tom, že aplikácie nebudú mať žiadne výpadky.

Ako orchestrátor sa stará o prácu s plánovaním kontajnerov v klastri a tiež spravuje pracovné zaťaženia. Takmer všetko v Kubernetes používa deklaratívne konštrukty, ktoré popisujú, ako sa aplikácie skladajú, ako interagujú a ako sú spravované.

Táto platforma je vhodná na riešenie úloh strojového učenia, aj keď priamo na to je vyvinutý kubeflow, Je najrýchlejšie rastúcim projektom z Open Source softvéru. Kubernetes je vhodný na riešenie hlavne kvôli týmto výhodám:

Prenosnosť

Ponúka prenosnosť, rýchlejšie a jednoduchšie nasadenie. To znamená, že v prípade potreby využívať výhody viacerých cloudov alebo serverov a môžu sa rýchlo

rozvíjať bez toho aby sa musela meniť infraštruktúru.

Škálovateľnosť

Ma schopnosť spúšťať kontajnery na jednom alebo viacerých verejných cloudových prostrediach, vo virtuálnych strojoch, čiže je ho možné nasadiť takmer kdekoľvek. A keďže Kubernetes zásadne zmenil spôsob vývoja a nasadzovania, je možné škálovať oveľa rýchlejšie, než v minulosti.

Dostupnosť

Rieši vysokú dostupnosť na úrovni aplikácie aj infraštruktúry. Pridanie spoľahlivej úložnej vrstvy do Kubernetes zaisťuje vysokú dostupnosť stavových úloh. Okrem toho môžu byť hlavné komponenty klastra nakonfigurované na multi-node replikáciu (multi-master), čo tiež zabezpečuje vyššiu dostupnosť.

1.1.1 Architektúra

Pracovné uzly (worker), hlavný uzol (master) a spolu s API, tvoria architektúru Kubernetes. V tejto časti je opísaná architektúra ako tieto uzly fungujú a načo slúži API.

Je zložený z niekoľkých častí, ktoré sú potrebné pre jeho správne fungovanie a efektivitu. Pre správne pochopenie ako Kubernetes funguje je treba sa zoznámiť najmä s niektorými termínmi.

V hlavnom uzle nájdeme komponenty, ktoré riadia klaster spolu s údajmi o stave a konfigurácii klastra. Tieto základne komponenty pracujú tak aby dokázali zabezpečiť prácu tak aby kontajnery fungovali v dostatočnom počte a spotrebnými zdrojmi. Hlavný uzol je neustále v kontakte s jednotlivými strojmi, ktoré vykonávajú prácu. Hlavný uzol sa postará o klaster, tak ako sme ho nakonfigurovali.

Mikroslužby

Pre mikroslužby neexistuje presná definícia. Často sa používajú pri cloudových službách a aplikácii, ktoré sa nasadzujú pomocou kontajnerov.

Uzol

Uzol môže byt virtuálny stroj alebo fyzicky počítač, kde sú nasadené kontajnery. Za prijímanie a spúšťanie pracovných zaťažení a externých zdrojov sú zodpo-

vedne uzly. Kubernetes spúšťa aplikácie a služby v kontajneroch na pomoc s izoláciou, správou a flexibilitou. Každý uzol musí mat modul tzv. runtime kontajnera. Uzol prijíma pracovne pokyny od hlavného uzla a podlá toho vytvára alebo ruší kontajnery a zároveň prispôsobuje pravidla sieťovej prevádzky.

Pod

Pod je skupina jedného alebo viacerých kontajnerov so zdieľaným úložiskom, sieťou a špecifikáciou ako majú kontajnery fungovať. Pri necloudových sú spúšťane na rovnakom fyzickom alebo virtuálnom stroji a pri cloudových na rovnakom logickom hostiteľovi. Všetky kontajnery v pode môžu medzi sebou komunikovať aj sú na samostatných uzloch. Sú vytvorené na základe pracovného zaťaženia nazývanými ovládače, ktoré riadia vytváranie, kopírovanie a stav podov v klastri. Ak napríklad zlyhá uzol v klastri, ovládač zisti, že moduly v tomto uzle nereagujú a vytvorí náhradne moduly na iných uzloch.

Klaster

V klastroch sa spúšťajú kontajnerové aplikácie, ktoré spravuje Kubernetes. Klaster je v podstate séria uzlov spojených dohromady. Spojením uzlov zhromažďujú svoje zdroje(CPU, RAM, atď.). Klaster je v tom prípade oveľa výkonnejší ako jednotlivé stroje. Kubernetes presúva pody po klastri, pri pridávaní alebo odstraňovaní uzlov.[1] Sú to navzájom prepojene počítače, ktoré vykonávajú určitú činnosť.

Kontajner

Kontajner je podobný virtuálnemu stroju ale kontajner je viac efektívnejší, podobne ako virtuálny stroj ma vlastný súborový systém, zdieľaný procesor, operačnú pamäť a ďalšie. Sú funkčne na rôznych operačných systémoch, pretože sa oddeľujú od základnej infraštruktúry.[2] Výrazne zvyšujú efektivitu, vyžadujú menej systémových prostriedkov, pretože neobsahujú obraz operačného systému. Zabezpečujú lepší vývoj aplikácii a lepšiu prenosnosť.

Kube-apiserver

Tento komponent odhaľuje API hlavnému uzlu. V podstate funguje ako frontend k informáciám o stave klastra a premosťuje API s inými objektmi Kubernetes, napríklad modulmi, radičmi a službami. Server API určuje, ci je požiadavka platná,

a ak áno, spracuje ju. K API sa pristupuje prostredníctvom Rest, cez rozhranie príkazového riadka kubectl alebo prostredníctvom iných nástrojov napríklad kubeadm. Zabezpečuje všetku interakciu medzi komponentmi.[3]

Kube-controller-manager

Na hlavnom uzle riadi všetky ovládače. Každý ovládač je vlastne samostatný individuálny proces. Pre zjednodušenie správy klastrov sú však všetky skompilovane do jedného procesu, Za tuto kompiláciu je zodpovedný kube-controller-manager. Jeden ovládač konzultuje plánovač a uisťuje sa, že beží správny počet podov. Ak pod spadne, iný ovládač si to všimne a zareaguje. Ovládač pripája služby k podom, takže požiadavky smerujú do správneho koncového bodu. Existujú aj ovládače na vytváranie účtov a prístupových tokenov API.[4]

Etcd

Ukladá informácie o konfigurácii pre veľké distribuovane systémy Kubernetes teda používa etcd ako úložisko hodnôt jednotlivých kľúčov. Etcd modul musí byt stále dostupný, aby sa zabezpečilo správne fungovanie služieb. Údaje Etcd sú veľmi doležíte a odporúča sa vytvoriť si zálohu.

Plánovač

Z názvu môžeme posúdiť, že slúži na plánovanie rozhodnutí pre novovytvorené pody. Keď je pod vytvorený, musí mu byť priradený uzol, na ktorom sa má spustiť. Plánovač prijíma informácie z API o požiadavkách a špecifikáciách podov a IT zdrojoch na dostupných uzloch. Potom priradí každý pod k príslušnému uzlu. Ak plánovač nemôže nájsť vhodný uzol, pod zostane nenaplánovaný a plánovač zopakuje proces, kým nebude dostupný uzol.

Komponent, ktorý sa nasadzuje ako prvý je takzvaný runtime kontajnera. Zvyčajne sa inštaluje spustením Dockera, ale dostupné sú aj alternatívy ako rkt a runc. Runtime kontajnera je zodpovedný za spustenie a správu kontajnerov, aplikácií zapuzdrených v relatívne izolovanom, ale ľahkom ako keby operačnom prostredí.

Každá jednotka práce v klastri je na svojej základnej úrovni implementovaná ako jeden alebo viac kontajnerov, ktoré je potrebné nasadiť.

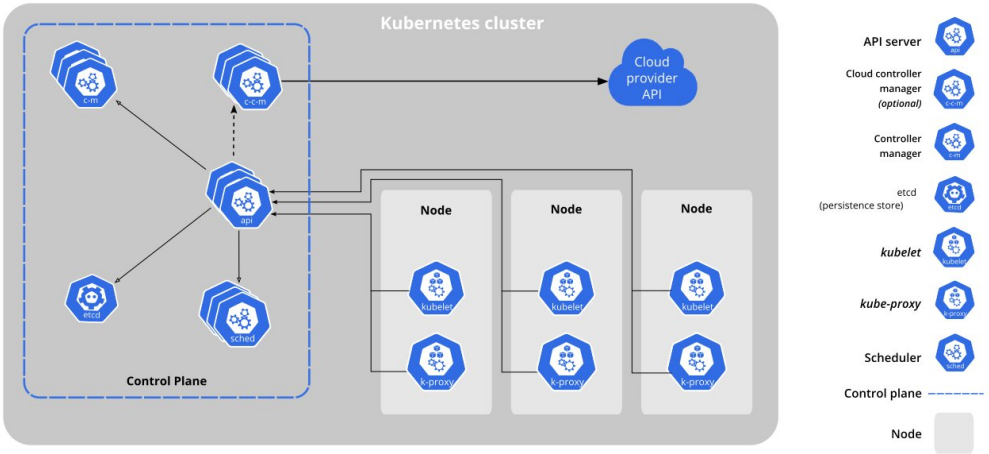
Kubelet

Kubelet je agent, ktorý je spustený na každom pracovnom uzle v klastri. Je to dôležitý komponent, pretože prijíma inštrukcie z hlavného uzla. Kubelet v podstate riadi pody. Zabezpečuje, že všetky kontajnery bežia v pode a že tieto pody sú v poriadku a bežia v správnych časových intervaloch. Číže vytvára a odstraňuje moduly, na základe pokynov od hlavného uzla, ktoré moduly je potrebné pridať alebo odstrániť. Keď riadiaci uzol potrebuje, aby sa niečo vykonalo v uzle, kubelet vykoná tuto akciu.

Kube-proxy

Na správu jednotlivých hostiteľských podsietí a sprístupnenie služieb iným komponentom je na každom uzlovom serveri spustená malá proxy služba s názvom kube-proxy . Tento proces preposiela požiadavky správnym kontajnerom, môže vykonávať primitívne vyvažovanie záťaže a je vo všeobecnosti zodpovedný za zabezpečenie toho, aby bolo sieťové prostredie predvídateľné a dostupné, ale v prípade potreby izolované. Používa proxy UDP, TCP a SCTP, ale nerozumie HTTP.

Na obrázku môžeme vidieť jednotlivé komponenty:



Obr. 1.1: Architektúra kubernetes [5]

1.1.2 Nástroje

Nástroje slúžia na vytváranie lokálneho klastra a následne pracovanie s týmto klastrom. V tejto časti si porovnane nástroje aké majú odlišnosti a čo ponúkajú.

minikube

Ide o spôsob vytvárania virtuálneho počítača, ktorý je v podstate klastrom Kubernetes s jedným uzlom. Vďaka podpore množstva hypervízorov ho možno použiť na všetkých hlavných operačných systémoch. To tiež umožňuje vytvárať viacero inštancií paralelne.

Z užívateľského hľadiska je minikube veľmi priateľský nástroj pre začiatočníkov. Klaster sa spúšťa pomocou minikube start, následne kubectl je pripravený na použitie. Na určenie verzie Kubernetes sa používa príkaz –kubernetes-version.

Pre nových používateľov Kubernetes, môže výrazne pomôcť dashboard, ktorý minikube ponúka. Dashboard sa jednoducho otvorí a poskytuje pekný prehľad o všetkom, čo sa deje v klastri. Minikube taktiež umožňuje pridávať rôzne rozšírenia, ktoré pomáhajú integrovať napríklad dashboard, grafické karty od Nvidie a oveľa viac. Pomocou minikube sa vykonávajú príkazy ako vytvoriť, aktualizovať, odstrániť klaster lokálne na počítači. Je užitočný, pre rýchle vytvorenie malého testovacieho klastera. Používa celkom jednoduché príkazy na vytvorenie a odstránenie klastra.

kind

Ako už názov napovedá, presúva klaster do kontajnerov Docker. To vedie k výrazne vyššej rýchlosti spustenia v porovnaní s vytváraním virtualizácie.

Vytvorenie klastra je veľmi podobné ako pri minikube. Použitím príkazu kind create cluster, vytvárame klaster. Použitím rôznych názvov –name, kind umožňuje vytvárať viaceré inštancie paralelne.

Jedna funkcia, ktorá sa mi osobne páči, je možnosť načítať obrazy kontajnerov priamo do klastra. To ušetrí niekoľko ďalších krokov pri nastavovaní registra a načítavaní môjho obrazu, keď chcem vyskúšať nejaké zmeny. S jednoduchým kind load docker-image my-app:latest je kontajnerový obraz k dispozícii na použitie v mojom klastri.

k3s

K3s je verzia Kubernetes vyvinutá spoločnosťou Rancher Labs. Odstránením postradateľných funkcií (legacy, in-tree pluginy) a použitím ľahkých komponentov (napr. sqlite3 namiesto etcd3) dosiahli výrazné zmenšenie. Výsledkom je jeden binárny súbor s veľkosťou približne 60 MB.

Aplikácia je rozdelená na server K3s a agenta. Prvý pôsobí ako manažér, zatiaľ čo druhý zodpovedá za zvládnutie skutočného pracovného zaťaženia. Upozorňu-

jem, že pri spúšťaní na počítací môže dôjsť k výraznému neporiadku v lokálnom súborovom systéme. Namiesto toho sa vkladá k3s do kontajnera (napr. pomocou rancher/k3s), ktorý tiež umožňuje ľahko spustiť niekoľko nezávislých inštancií.

Jedna funkcia, ktorá vyniká, sa nazýva automatické nasadenie. Umožňuje nasadiť manifesty Kubernetes a grafy Helm ich umiestnením do konkrétneho adresára. K3s sleduje zmeny a stará sa o ich aplikáciu bez akejkoľvek ďalšej interakcie. Je to užitočné najmä pre CI pipelines a zariadenia lot. Stačí vytvoriť/aktualizovať konfiguráciu a K3s sa postará o to, aby boli nasadenia (deployments) aktuálne.

MicroK8s

Sťahuje sa ako inštalačný balík, ktorý nainštaluje jednouzlový (samostatný) klaster K8s za menej ako 60 sekúnd. Da sa použiť aj na vytvorenie klastra s viacerými uzlami pomocou niekoľkých príkazov. MicroK8s má všetky základné komponenty Kubernetes, ako napríklad DNS a Dashboard, sú dostupné použitím jediného príkazu.

MicroK8s je k dispozícii pre väčšinu populárnych distribúcií Linuxu a tiež pre pracovné stanice Windows a Mac prostredníctvom natívneho inštalátora pre oba operačné systémy. V systéme Windows je tiež možnosť získať MicroK8s na WSL.

Používa snap packaging mechanizmus, čož je naozaj pohodlné, pretože prináša automatické aktualizácie. To znamená, že akonáhle bude k dispozícii nová stabilná verzia Kubernetes, váš klaster MicroK8s sa automaticky aktualizuje. Podobne sa získavajú všetky dostupné bezpečnostné záplaty pre Kubernetes.[6]

Každý z týchto nástrojov poskytuje ľahko použiteľné prostredie Kubernetes pre viacero platforiem, no odlišujú ich niekoľko faktorov.

K3s napríklad ponúka prostredie Kubernetes založené na virtualizácii. Pre nastavenie viacero serverov Kubernetes, je potreba manuálne nakonfigurovať ďalšie virtuálne stroje alebo uzly, čo môže byť dosť náročné. Je však navrhnutý na použitie pri nasadzovaní, čo z neho robí jednu z najlepších možností na lokálne simulovanie skutočného nasadzovacieho prostredia.

Aj keď je minikube vo všeobecnosti skvelou voľbou pre lokálne spúšťanie Kubernetes, jednou z hlavných nevýhod je, že môže spustiť iba jeden uzol v miestnom klastri, čo zas vyúsťuje k tomu, zem je o niečo ďalej k produkčnému multiuzlovému prostrediu.

Na rozdiel od miniKube môže microK8S prevádzkovať viacero uzlov v lokálnom klastri.

Inštalácia microK8S na počítače so systémom Linux, ktoré nepodporujú balík snap, je náročná v porovnaní s inými nástrojmi v tomto zozname. microK8S používa balík snap vytvorený spoločnosťou canonical na inštaláciu strojového nástroja Linux, čo sťažuje spustenie na distribúciách Linuxu, ktoré ho nepodporujú. miniKube sa tiež inštaluje na viacero platforiem pomocou virtualizácie.

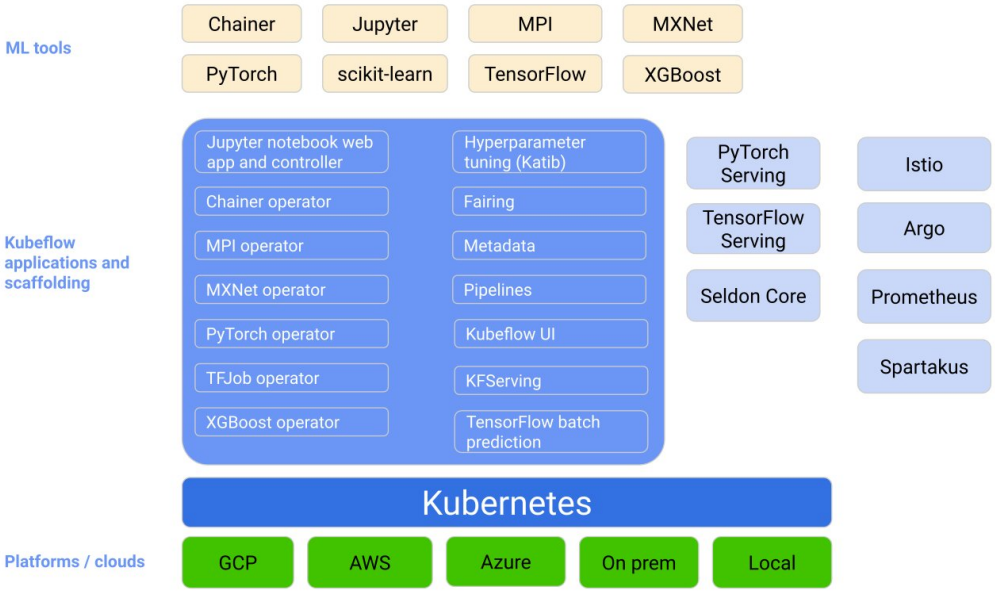
1.2 Kubeflow

Kubeflow ako platforma, je vhodná na nasadenie a vývoj strojového učenia. Primárne slúži pre inžinierov a vedcov, ktorý pracujú s dátami. Obsahuje viaceré komponenty, z ktorých si vývojári môžu vybrať, čo je pre ich používateľov najlepšie, čo znamená, že na nasadzovanie nie je potrebný každý jeden komponent.[7]

1.2.1 Architektúra

Stavia na platforme Kubernetes ako systéme na nasadenie, škálovanie a správu zložitých systémov. Pomocou konfiguračných rozhraní Kubeflow, môžeme špecifikovať nástroje strojového učenia, potrebné pre náš pracovný postup. Potom môžeme nasadiť pracovný postup do rôznych cloudov, miestnych platforiem na experimentovanie a na produkčné použitie.[7]

Na nasledujúcom obrázku môžte vidieť architektúru Kubeflow.



Obr. 1.2: Architektúra kubeflow [7]

1.2.2 Pracovný postup

Postup pozostáva z viacerých krokov, taktiež z iterácie, ktorá je hlavným prvkom pri vyvíjaní sýtemu strojového učenia. Pri tomto postupe je potrebné vykonávať zmeny v parametroch aby sme dosiahli požadované výsledky.

Následne si povieme niečo viac o týchto postupoch.

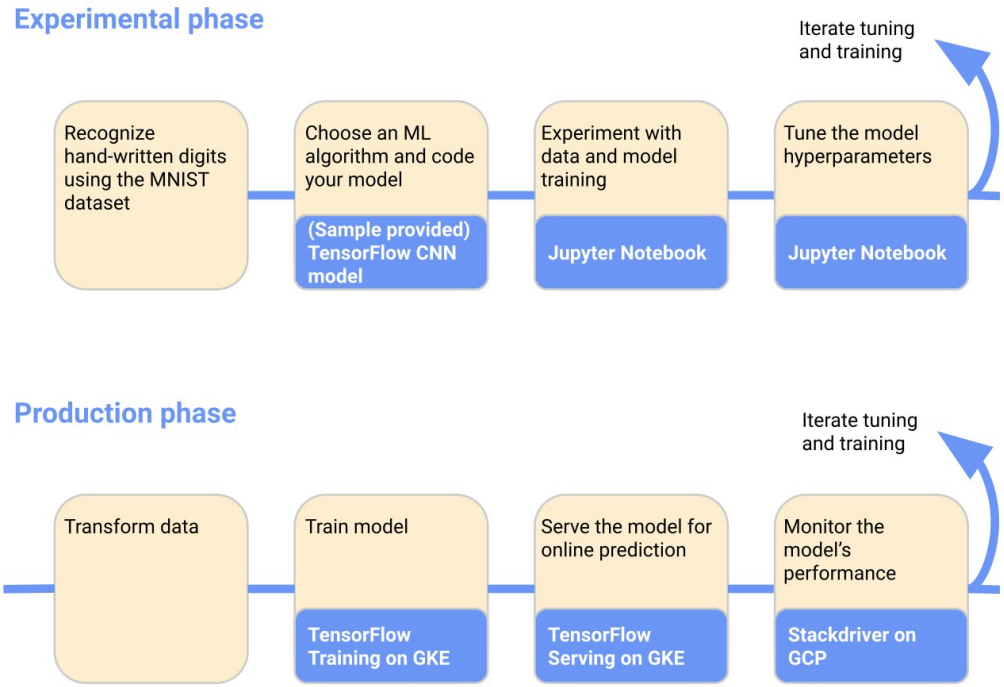
Ako prvé by sme mali vyvíjať model na základe predpokladov a testov. Môžeme ho opísať v nasledujúcich bodoch:[7]

- Identifikovanie problému, ktorý ma systém vyriešiť.
- Zbieranie dát, ktoré potrebujeme na trénovanie modelu.
- Vyberanie algoritmu a nakódovanie modelu.
- Experiment s údajmi a trénovanie modelu.
- Vyladenie parametrov.

Ďalej môžeme nasadiť systém, ktorý bude vykonávať tieto procesy:

- Transformáciu údajov, ktoré náš systém potrebuje.
- Trénovanie modelu.
- Podanie modelu na online prevádzku.
- Monitorovanie výsledkov na úpravu a zmenu modelu.

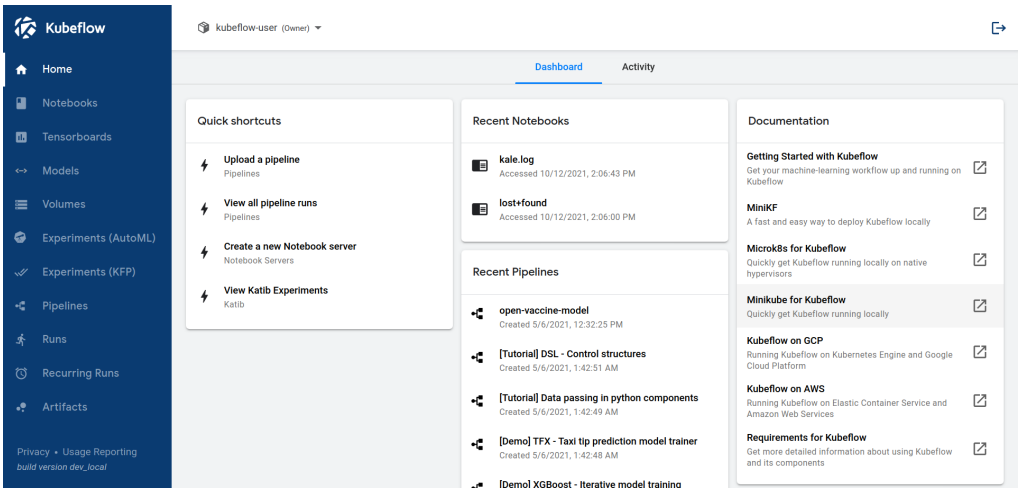
jkkjjkh



Obr. 1.3: Pracovné postupy

1.2.3 Rozhrania

dopisaaat neskoor



Obr. 1.4: Rozhranie

1.3 Kontajnerizácia

Kontajnery predstavujú metódu virtualizácie operačného systému, ktorá vám umožňuje spúšťať aplikáciu a jej závislosti v procesoch izolovaných na zdrojoch. Kontajnery nám umožňujú jednoducho zbaliť kód aplikácie, konfigurácie do ľahko použiteľných stavebných blokov, ktoré poskytujú environmentálnu konzistentnosť, prevádzkovú efektivitu, produktivitu vývojárov a kontrolu verzií. Kontajnery môžu pomôcť zabezpečiť rýchle, spoľahlivé a konzistentné nasadenie aplikácií bez ohľadu na prostredie nasadenia. Kontajnery tiež poskytujú podrobnejšiu kontrolu nad zdrojmi, čo zvyšuje efektivitu vašej infraštruktúry.

V minulosti sa aplikácie spúšťali na fyzických serveroch. Neexistovala možnosť ako nastaviť hranice pre aplikácie na serveroch a to spôsobovalo problémy. Napríklad, ak viaceré aplikácie bežali na jednom serveri, tak existovali tam inštancie, ktoré využívali väčšinu výkonu servera a ďalšie, ktoré nemali dostatočný výkon na vykonávanie akcii. To samozrejme sa už v dnešných časoch veľmi nevyužíva, tato metóda bola príliš nákladná na údržbu mnohých fyzických serverov.

Nasadzovanie aplikácie na virtualizovaných serveroch bolo predstavené ako riešenie. Umožňuje spustiť viacero virtuálnych strojov na jeden fyzicky server. Virtualizácia umožňuje izolovať aplikácie od seba a zabezpečiť tak, zem k informáciám jednej aplikácie sa nedostane druha aplikácia. Virtualizácia umožňuje lepšie využitie výkonu na serveri, aplikáciu umožňuje pridať, aktualizovať a taktiež znížene náklady na hardvér.

Kontajnery sú podobné virtuálnym počítačom, ale majú uvoľnené izolačné vlastnosti na zdieľanie operačného systému medzi aplikáciami. Preto sa kontajnery považujú za jednoduchšie. Podobne ako virtualizácia, kontajner má svoj vlastný súborový systém, zdieľané CPU, pamäte, procesného priestoru a ďalšie. Keďže sú oddelené od základnej infraštruktúry, sú prenosné cez cloudy a distribúcie operačných systémov.

Stali sa obľúbenými, pretože poskytujú veľa výhod:

- Vyššia rýchlosť dodávania vylepšení. Kontajnerovanie monolitických aplikácií pomocou mikroslužieb pomáha vývojovým tímom vytvárať funkcie s vlastným životným cyklom a zásadami škálovania.
- Vylepšená bezpečnosť izoláciou aplikácií od hostiteľského systému a od seba navzájom.
- Nepretržitý vývoj, integrácia a nasadzovanie: poskytuje spoľahlivé a časté vytváranie a nasadzovanie obrazu kontajnera.

- Prenosnosť cloudu a distribúcie operačných systémov, na veľkých verejných cloudoch a kdekoľvek inde.

- Menšia záťaž. Kontajnery vyžadujú menej systémových prostriedkov ako tradičné alebo hardvérové prostredia virtuálnych strojov, pretože nezahŕňajú operačne systémy.

1.3.1 Kontajnerový obraz, runtime a orchestrácia

Súbory kontajnerového obrazu sú statické a spustiteľné verzie aplikácie alebo služby a líšia sa od jednej technológie k druhej. Obraz sa skladá z viacerých vrstiev, ktoré začínajú základným obrazom, ktorý obsahuje všetky závislosti potrebné na spustenie kódu v kontajneri. Každý obraz má na vrchu statických nemenných vrstiev čitateľnú a zapisovateľnú vrstvu. Pretože každý kontajner má svoju vlastnú špecifickú vrstvu kontajnera, ktorá prispôsobuje tento konkrétny kontajner, podkladové vrstvy obrazu možno uložiť a znova použiť vo viacerých kontajneroch. Pre otvorený kontajner sa obraz skladá z manifestu, vrstiev súborového systému a konfigurácií. Pre správne fungovanie kontajnera, musí mať runtime a špecifikáciu obrazu. Špecifikácie runtime predstavujú fungovanie súborového systému, čo sú súbory obsahujúce všetky potrebné údaje pre chod a runtime. Špecifikácia obrazu obsahuje informácie potrebné na spustenie aplikácie alebo služby v kontajneri.[8]

Kontajnerový systém spúšťa obrazy a väčšinou sa používajú na správu nasadení plánovač kontajnerov alebo v našom prípade technológiu orchestrácie, ako je Kubernetes. Kontajnery majú vysokú prenosnosť, pretože každý obraz obsahuje závislosti potrebné na spustenie kódu v kontajneri. Používatelia kontajnera môžu napríklad počas testu spustiť rovnaký obraz na rôznej cloudovej platformy bez zmeny kódu aplikácie v kontajneri.

Orchestrácia kontajnerov umožňuje vývojárom nasadiť veľké množstvo kontajnerov a spravovať ich vo veľkom meradle pomocou konceptu klastrov kontajnerov. Orchestrátor pomáha správcom IT automatizovať proces spúšťania inštancií kontajnerov, poskytovania hostiteľov a spájania kontajnerov do funkčných skupín. S kontajnerovou orchestráciou je možné riadiť životný cyklus aplikácií, pozostávajúci z veľkého počtu kontajnerov.

Orchestrácia ma tieto privilégiá:

- Automaticky nasadzuje kontajnery na základe politík, zaťaženia aplikácií a metrík prostredia.

- Identifikujte neúspešné kontajnery alebo zhluky a opravuje ich.
- Spravuje konfiguráciu aplikácie.
- Pripája kontajnery k úložisku a spravuje sieť.
- Zlepšuje bezpečnosť obmedzením prístupu medzi kontajnermi a externými systémami.

Príklady orchestrátorov sú napríklad Kubernetes, OpenShift, Kubeflow atď.

1.3.2 Architektúra mikroslužieb

Architektúra mikroslužieb rozdeľuje aplikáciu na viacero nezávislých služieb. Každý z nich má svoj vlastný kanál a môže byť nasadený do produkcie kedykoľvek, bez závislosti od iných mikroslužieb.

Bežný spôsob vytvárania a nasadenia mikroslužieb je v kontajneroch. Celá aplikácia mikroslužieb môže byť nasadená ako klaster pomocou kontajnerového orchestrátora. Existuje niekoľko výhod používania kontajnerov pre mikroslužby, na rozdiel od úplných virtuálnych strojov alebo fyzických počítačových serverov:

- Sú jednoduché, čo umožňuje spúšťať viac inštancií mikroslužieb na jednom fyzickom hostiteľovi.
- Kontajnery sa dajú ľahko automatizovať a úzko sa integrovať s pracovnými postupmi.
- Kontajnery sú nemenné, vďaka čomu je ľahké vymazať a nahradiť inštancie mikroslužieb pri vydaní nových verzií.
- Kontajnery sú ľahko prenosné medzi miestnymi vývojovými prostrediami, lokálnymi dátovými centrami a cloudovými prostrediami, čo umožňuje vyvinúť mikroslužby v jednom prostredí a nasadiť ich do iného.

1.3.3 Virtualizácia a kontajner

V dobe, kedy výkony a parametre serverov boli na menšej úrovni a postupne sa zvyšovali zrodili sa takzvane virtuálne stroje, kde bolo možne spustením softvéru nad fyzickými servermi na emuláciu konkrétneho hardvérového systému. Hypervizor je softvér, firmvér alebo hardvér, ktorý vytvára a spúšťa virtuálne počítače. Je to to, čo sa nachádza medzi hardvérom a virtuálnym strojom a je potrebný na virtualizáciu servera.

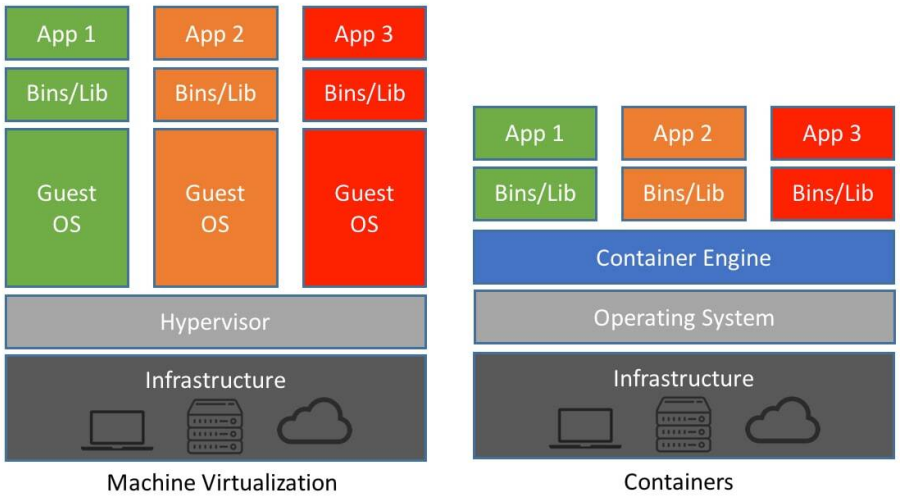
V rámci každého virtuálneho počítača beží jedinečný hosťujúci operačný systém. Virtuálne počítače s rôznymi operačnými systémami môžu bežať na rovnakom fyzickom serveri – VM so systémom UNIX môže byť vedľa virtuálneho počítača so systémom Linux atď. Každý VM má svoje vlastné binárne súbory, knižnice a aplikácie, ktoré obsluhuje, a VM môže mať veľkosť mnoho gigabajtov

Vývoj tiež profitoval z tejto fyzickej konsolidácie, pretože väčšie využitie na väčších a rýchlejších serveroch uvoľnilo následne nepoužívané servery na opätovné použitie na kontrolu kvality, vývoj alebo laboratórne vybavenie. Ale tento prístup mal aj svoje nevýhody. Každý virtuálny stroj obsahuje samostatný obraz operačného systému, ktorý zvyšuje nároky na pamäť a úložný priestor. Ako sa ukázalo, tento problém pridáva na zložitosti všetkým fázam životného cyklu vývoja softvéru, od vývoja až po testovanie na výrobu a obnovu po havárii. Tento prístup tiež výrazne obmedzuje prenosnosť aplikácií medzi verejnými cloudmi, súkromnými cloudmi a tradičnými dátovými centrami. Virtualizácia operačných systémov v poslednom desaťročí narástla na popularite, aby umožnila softvéru predvídateľne a dobre fungovať pri presune z jedného serverového prostredia do druhého. Kontajnery však poskytujú spôsob, ako spustiť tieto izolované systémy na jednom serveri.

Kontajnery sú umiestnené na vrchole fyzického servera a jeho hostiteľského operačného systému napríklad Linux alebo Windows. Každý kontajner zdieľa jadro hostiteľského operačného systému a zvyčajne aj binárne súbory a knižnice. Zdieľané komponenty sú len na čítanie. Kontajnery sú teda výnimočne v tom, že majú veľkosť iba niekoľko megabajtov a ich spustenie trvá len niekoľko sekúnd, na rozdiel od gigabajtov a minút pre virtualizáciu.

Kontajnery tiež znižujú riadenie, pretože zdieľajú spoločný operačný systém, iba jeden operačný systém potrebuje starostlivosť a zásobovanie pre opravy chýb, záplaty atď. Stručne povedané, kontajnery sú jednoduchšie a prenosnejšie ako virtuálne počítače.

Virtuálne stroje a kontajnery sa líšia niekoľkými spôsobmi, ale hlavný rozdiel je v tom, že kontajnery poskytujú spôsob virtualizácie operačného systému, takže na jednej inštancii operačného systému môže bežať viacero pracovných zaťažení. V prípade virtuálnych počítačov sa hardvér virtualizuje, aby bolo možné spustiť viacero inštancií operačného systému. Rýchlosť, svižnosť a prenosnosť kontajnerov z nich robí ďalší nástroj, ktorý pomáha zefektívniť vývoj softvéru.



Obr. 1.5: Porovnanie infraštruktúry kontajnera a virtualizácie [9]

1.3.4 Systemový a aplikačný kontajner

Aplikačné kontajnery, ako napríklad Docker, zapuzdrujú súbory a knižnice aplikácie, ktoré sa majú spustiť na operačnom systéme. Aplikačné kontajnery umožňujú používateľovi vytvoriť a spustiť samostatný kontajner pre viacero nezávislých aplikácií alebo viacero služieb, ktoré tvoria jednu aplikáciu. Napríklad aplikačný kontajner by bol vhodný pre aplikáciu mikroslužieb, kde každá služba, ktorá tvorí aplikáciu, beží nezávisle jedna od druhej.

Systémové kontajnery, sú technologicky podobné kontajnerom aplikácií aj virtuálnym strojom. Systémový kontajner môže spúšťať operačný systém, podobne ako by operačný systém bežal zapuzdrený na VM. Systémové kontajnery však neemulujú hardvér systému, namiesto toho fungujú podobne ako aplikačné kontajnery a používateľ si môže nainštalovať rôzne knižnice, jazyky a systémové databázy. [10]

Takže všeobecne, keď chcete distribuovať aplikáciu ako komponenty, aplikačné kontajnery sú skvela voľba. Zatiaľ čo ak chcete iba operačný systém, do ktorého môžete nainštalovať rôzne knižnice, jazyky, databázy atď. vhodnejšie sú systémové kontajnery.

- v knihe [11] autor prezentuje naozaj odvážne myšlienky
- nemenej zaujímavé výsledky publikuje ďalší autor v článku [12]
- v konferenčnom príspevku [13] sú uvedené tiež zaujímavé veci
- LaTeX[1] je typografický jazyk

Given a set of numbers, there are elementary methods to compute its Greatest Common Divisor, which is abbreviated GCD. This process is similar to that used for the Least Common Multiple (LCM).

1.3.5 Donec vehicula consequat

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisissem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi necante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultriciesvel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero utmetus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit ametante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

1.3.6 Nullam in mauris consectetur

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisissem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi necante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultriciesvel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero utmetus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit ametante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Zdrojový kód 1.1: Program, ktorý pozdraví celý svet

[1] `https://www.latex-project.org/`

```
#include <stdio.h>
int main() {
    /* Print Hello, World! */
    printf("Hello, World!\n");
    return 0;
}
```

1.3.7 Vestibulum tristique elementum varius

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisissem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi necante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultriciesvel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero utmetus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit ametante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Tabuľka 1.1: Country list

Country List			
Country Name or Area Name	ISO ALPHA 2 Code	ISO ALPHA 3 Code	ISO numeric Code
Afghanistan	AF	AFG	004
Aland Islands	AX	ALA	248
Albania	AL	ALB	008
Algeria	DZ	DZA	012
American Samoa	AS	ASM	016
Andorra	AD	AND	020
Angola	AO	AGO	024

1.4 Phasellus id pretium neque

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisissem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi ne-

cante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultriciesvel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero utmetus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit ametante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisissem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi necante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultriciesvel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero utmetus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit ametante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

2 Syntetická časť

Syntetická časť opisuje metódy použité na syntézu riešenia a opisuje syntézu samotného riešenia (zvyčajne je to návrh/implementácia softvérového resp. hardvérového riešenia), pričom sa opiera o závery analytickej časti práce. Začína od toho, ako sa bude riešenie používať: najdôležitejšie scenáre používania a používateľské rozhranie, ktoré bude tieto scenáre efektívne podporovať. Až potom je na rade vnútorná architektúra alebo použité technológie. Syntetická časť tvorí zvyčajne ½ jadra práce.

Syntetickú časť práce vhodne rozdeľte do kapitol a pomenujte ich podľa toho, čomu sú venované.

3 Vyhodnotenie

Vyhodnocovacia časť je kľúčovou časťou záverečnej práce. Tato časť obsahuje vyhodnotenie navrhnutého (vytvoreného) riešenia. Uprednostňované je objektívne vyhodnotenie výsledkov práce, ktoré sa opiera o meranie a štatistické metódy, prípadne matematické dôkazy. V prípade nameraných hodnôt musí autor opísať metódu merania, priebeh merania, výsledky a interpretáciu výsledkov v kontexte riešeného problému a stanovených cieľov. Na základe vyhodnotenia riešenia autor opíše prínosy svojej práce. Vyhodnocovacia časť tvorí zvyčajne ¼ jadra práce.

4 Záver

Záver práce obsahuje zhrnutie výsledkov práce s jasným opisom prínosov a pôvodných (vlastných) výsledkov autora a vyhodnotenie splnenia stanovených cieľov. Je to stručné zhrnutie informácií uvedených v záverečnej práci. Záver by nemal obsahovať nové informácie.

V závere by mal tiež autor poukázať na prípadné otvorené otázky, ktoré sú nad rámec rozsahu práce a mal by odporučiť ďalšie aktivity na pokračovanie pri riešení problému. Rozsah záveru je minimálne 1 celá strana.

Literatúra

1. WWW.DOCS.BYTEMARK.CO.UK. *https://docs.bytemark.co.uk/article/kubernetes-terminology-glossary/the-basics*. 2021.
2. WWW.KUBERNETES.IO. *https://kubernetes.io/docs/concepts/overview/what-is-kubernetes*. 2021.
3. WWW.NEWRELIC.COM. *https://newrelic.com/blog/how-to-relic/what-is-kubernetes*. 2021.
4. WWW.KUMARGAURAV1247.MEDIUM.COM. *https://kumargaurav1247.medium.com/com of-kubernetes-architecture-6feea4d5c712*. 2021.
5. WWW.DEV.TO. *https://dev.to/inductor/wait-docker-is-deprecated-in-kubernetes-now-what-do-i-do-e4m*. 2021.
6. WWW.BLOG.FLANT.COM. *https://blog.flant.com/small-local-kubernetes-comparison/*. 2021.
7. WWW.KUBEFLOW.ORG. *https://www.kubeflow.org/docs/started/kubeflow-overview*. 2021.
8. WWW.VMWARE.COM. *https://www.vmware.com/topics/glossary/content/container-orchestration.html*. 2021.
9. WWW.NETAPP.COM. *https://www.netapp.com/blog/containers-vs-vms/*. 2021.
10. WWW.BLOG.RISINGSTACK.COM. *https://blog.risingstack.com/operating-system-containers-vs-application-containers/*. 2021.
11. BABINGTON, Peter. *The title of the work*. 3. vyd. The address: The name of the publisher, 1993. 10. ISBN 3257227892. An optional note.
12. ADAMS, Peter. The title of the work. *The name of the journal*. 1993, roč. 4, č. 2, s. 201–213. An optional note.
13. DRAPER, Peter. The title of the work. In: EDITOR, The (ed.). *The title of the book*. The address of the publisher: The publisher, 1993, zv. 4, s. 213. 5. An optional note.

Zoznam skratiek

GCD Greatest Common Divisor.

LCM Least Common Multiple.

Zoznam príloh

Príloha A Karel Language Reference

Príloha B CD médium – záverečná práca v elektronickej podobe,

Príloha C Používateľská príručka

Príloha D Systémová príručka

A Karel Language Reference

Karel's Primitives

- `void movek()` - Moves *Karel* one intersection forward.
- `void turn_left()` - Pivots *Karel* 90 degrees left.
- `void pick_beeper()` - Takes a beeper from the current intersection and puts it in the beeper bag.
- `void put_beeper()` - Takes a beeper from the beeper bag and puts it at the current intersection.
- `void turn_on(char* path)` - Turns *Karel* on.
- `void turn_off()` - Turns *Karel* off.

Karel's Sensors

- `int front_is_clear()` - Returns `1` if there is no wall directly in front of *Karel*. `0` if there is.
- `int right_is_clear()` - Returns `1` if there is no wall immediately to *Karel*'s right. `0` if there is.
- `int beepers_present()` - Returns `1` if *Karel* is standing at an intersection that has a beeper. `0` otherwise.
- `int facing_north()` - Returns `1` if *Karel* is facing north. `0` otherwise.
- `int beepers_in_bag()` - Returns `1` if there is at least one beeper in *Karel*'s beeper bag. `0` if the beeper bag is empty.

Misc Functions

- `void set_step_delay(int)` - Sets delay of one *Karel*'s step in miliseconds.
- `loop(int)` - Repeats *Karel*'s instruction in a loop.